600212899 Solutions Manual for Statistical Computing With R Rizzo

SOLUTIONS MANUAL FOR

Statistical Computing with R

by

Maria L. Rizzo

Chapman & Hall/CRC
Taylor & Francis Group

SOLUTIONS MANUAL FOR

Statistical Computing with R

by

Maria L. Rizzo

Chapman & Hall/CRC
Taylor & Francis Group
Boca Raton London New York

Chapman & Hall/CRC is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-13: 978-1-4200-7696-7 (Softcover)

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Contents

Preface

This manual contains solutions to all of the exercises in the book "Statistical Computing with R" including R code if programming was required in the problem. The solution manual was prepared with `Sweave` so any output shown is unedited. The chunks of R code in solutions have been extracted using `Stangle`, and the resulting source files are available upon request. The solutions have been checked in the latest release of R (R-2.6.0) with the latest package versions available at that release date. Also see `personal.bgsu.edu/~mrizzo` for updates and the R code for examples in the book. Comments, corrections, and suggestions are always welcome.

Maria L. Rizzo
Department of Mathematics and Statistics
Bowling Green State University

CHAPTER 3

Methods for Generating Random Variables

3.1 *Write a function that will generate and return a random sample of size n from the two-parameter exponential distribution Exp(λ, η) for arbitrary n, λ and η.*

The cdf is

$$F(x) = 1 - e^{-\lambda(x-\eta)}, \qquad x \geq \eta.$$

Apply the inverse transformation. Generate a random $U \sim$ Uniform(0,1). Then

$$U = 1 - e^{-\lambda(X-\eta)} \Rightarrow -\log(1-U) = \lambda(X-\eta) \Rightarrow X = \eta - \frac{1}{\lambda}\log(1-U).$$

If $U \sim$ Uniform(0,1) then $1-U \stackrel{D}{=} U$ so it is equivalent to generate $X = \eta - \frac{1}{\lambda}\log(U)$. Recall that the quantiles are given by

$$x_\alpha = -\frac{1}{\lambda}\log(1-\alpha) + \eta.$$

```
> rexp2 <- function(n, lambda, eta) {
+     u <- runif(n)
+     x <- eta - log(u)/lambda
+     return(x)
+ }
```

The function is applied below to generate a sample of 1000 variates from Exp(λ, η) and we compare the sample quantiles with the theoretical quantiles in a table.

```
> lambda <- 2
> eta <- 1
> x <- rexp2(1000, lambda, eta)
> p <- seq(0.1, 0.9, 0.1)
> Q <- -log(1 - p)/lambda + eta
> xq <- quantile(x, p)
> print(round(rbind(Q, xq), 3))
     10%   20%   30%   40%   50%   60%   70%   80%   90%
Q  1.053 1.112 1.178 1.255 1.347 1.458 1.602 1.805 2.151
xq 1.053 1.115 1.183 1.261 1.348 1.479 1.626 1.860 2.190
```

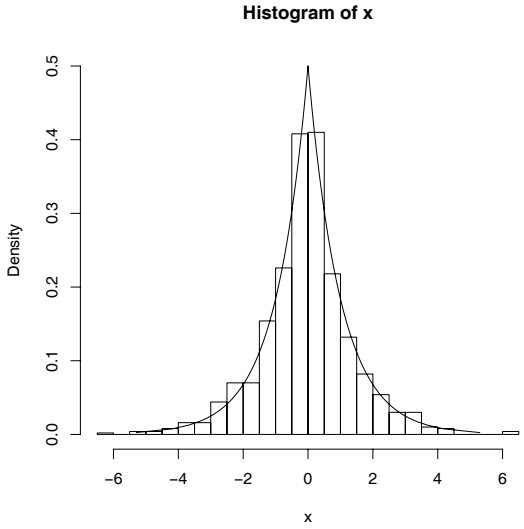
3.2 *The standard Laplace distribution has density $f(x) = \frac{1}{2}e^{-|x|}$, $, x \in R$. Use the inverse transform method to generate a random sample of size 1000 from this distribution.*

Generate a random u from Uniform(0, 1). To compute the inverse transform, consider the cases $u < \frac{1}{2}$ and $u \geq \frac{1}{2}$ separately. If $u \geq \frac{1}{2}$ then $u = \int_{-\infty}^{x} f(t)dt =$

$\frac{1}{2} + \frac{1}{2}(1 - e^{-x})$. If $u < \frac{1}{2}$ then $u = \int_{-\infty}^{x} f(t)dt = \frac{1}{2} - \frac{1}{2}(1 - e^{-x}) = \frac{1}{2}e^{-x}$. Deliver

$$x = F^{-1}(u) = \begin{cases} -\log(2u-1), & \frac{1}{2} \le u < 1; \\ \log(2u), & 0 < u < \frac{1}{2}. \end{cases}$$

```
> n <- 1000
> u <- runif(n)
> i <- which(u >= 0.5)
> x <- c(-log(2 * u[i] - 1), log(2 * u[-i]))
> a <- c(0, qexp(ppoints(100), rate = 1))
> b <- -rev(a)

> hist(x, breaks = "Scott", prob = TRUE, ylim = c(0, 0.5))
> lines(a, 0.5 * exp(-a))
> lines(b, 0.5 * exp(b))
```



3.3 *The Pareto(a, b) distribution has cdf*

$$F(x) = 1 - \left(\frac{b}{x}\right)^a, \qquad x \ge b > 0, a > 0.$$

Derive the probability inverse transformation $F^{-1}(U)$ and use the inverse transform method to simulate a random sample from the Pareto(2, 2) distribution.

The inverse transform is

$$u = F(x) = 1 - (b/x)^a \Rightarrow x = b(1-u)^{-1/a},$$

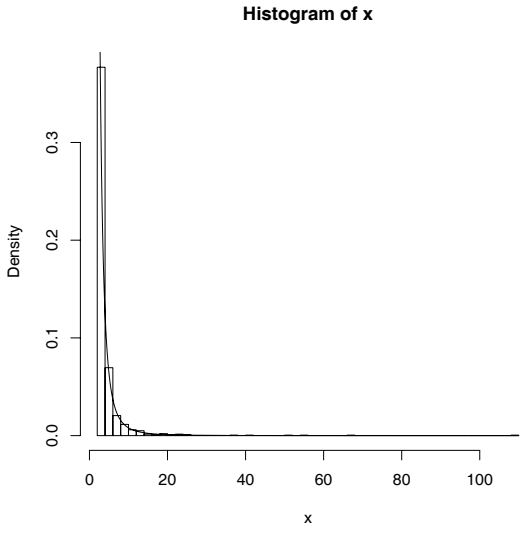
and $U \sim$ Uniform(0,1) has the same distribution as $1 - U$.

```
> a <- b <- 2
> n <- 1000
> u <- runif(n)
> x <- b * u^(-1/a)
> print(summary(x))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.001   2.278   2.787   4.059   3.949 109.900
```

The density of X is $f(x) = F'(x) = ab^a x^{-(a+1)}$, $x \geq b$.

```
> hist(x, breaks = "Scott", prob = TRUE)
> y <- sort(x)
> fy <- a * b^a * y^(-(a + 1))
> lines(y, fy)
```



3.4 *The Rayleigh density*

$$f(x) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)}, \qquad x \geq 0, \sigma > 0.$$

Develop an algorithm to generate random samples from a Rayleigh(σ) distribution.

Show that $F(x) = 1 - \exp(-x^2/(2\sigma^2)$, $x > 0$. Apply the inverse transform

$$u = 1 - e^{-x^2/(2\sigma^2)} \Rightarrow F^{-1}(u) = \sigma(-2\log(1-u))^{1/2}$$

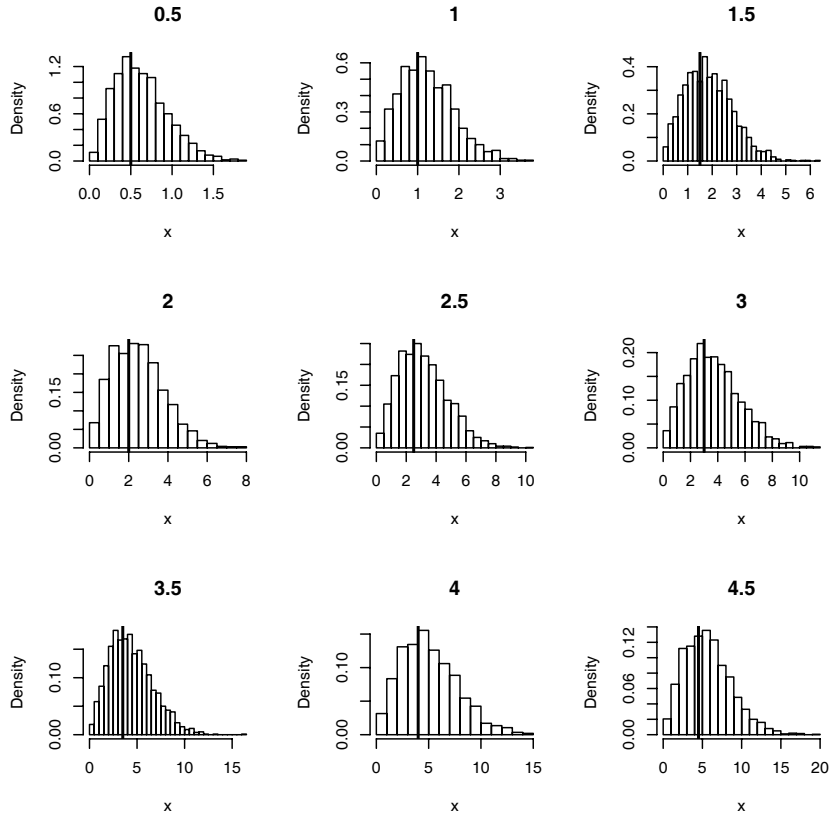
and use the fact that $U \sim$ Uniform(0,1) has the same distribution as $1 - U$.

```
> n <- 2000
> sigma <- seq(0.5, 4.5, length = 9)
```

Below, samples of size $n = 2000$ are generated for several choices of σ. In the histograms below corresponding to $\sigma = 0.5, \ldots, 4.5$ the mode of the sample is close to the theoretical mode σ.

```
> par(mfrow = c(3, 3))
> for (i in 1:length(sigma)) {
+     u <- runif(n)
+     x <- sigma[i] * sqrt(-2 * log(u))
+     hist(x, breaks = "Scott", prob = TRUE, main = sigma[i])
+     abline(v = sigma[i], lwd = 2)
```

```
+ }
> par(mfrow = c(1, 1))
```



3.5 *A discrete random variable X has probability mass function*

x	0	1	2	3	4
$p(x)$	0.1	0.2	0.2	0.2	0.3

Use the inverse transform method to generate a random sample of size 1000 from the distribution of X.

Method 1:

```
> n <- 1000
> p <- c(0.1, 0.2, 0.2, 0.2, 0.3)
> cdf <- cumsum(p)
> x <- numeric(n)
> for (i in 1:n) x[i] <- sum(as.integer(runif(1) > cdf))
> rbind(table(x)/n, p)
      0     1     2     3     4
  0.118 0.206 0.201 0.191 0.284
p 0.100 0.200 0.200 0.200 0.300
```

(Alternately, use `apply` to find $F^{-1}(u)$.)

The relative sample frequencies agree closely with the theoretical probability distribution (p) in the second row.

Repeat using the R `sample` *function.*

```
> n <- 1000
> p <- c(0.1, 0.2, 0.2, 0.2, 0.3)
> x <- sample(0:4, size = n, prob = p, replace = TRUE)
> rbind(table(x)/n, p)
      0     1     2     3    4
  0.104 0.199 0.211 0.196 0.29
p 0.100 0.200 0.200 0.200 0.30
```

3.6 *Prove that the accepted variates generated by the acceptance-rejection sampling algorithm are a random sample from the target density f_X.*

Suppose that X has density f_X and Y has density f_Y, and there is a constant $c > 0$ such that $f_X(t) \le cf_Y(t)$ for all t in the support set of X. Let A denote the set of accepted candidate points from the distribution of Y. Then

$$P(A) = \int F_U\left(\frac{f_X(t)}{cf_Y(t)}\right) f_Y(t)\,dt = \int \frac{f_X(t)}{c}\,dt = \frac{1}{c},$$

where F_U is the Uniform(0,1) cdf, and

$$\begin{aligned} f_Y(y|A) = \frac{f_{Y,A}(y)}{P(A)} &= \frac{P(A|Y=y)f_Y(y)}{1/c} \\ &= \frac{P[U < f_X(y)/(cf_Y(y))]f_Y(y)}{1/c} \\ &= \frac{f_X(y)/(cf_Y(y))f_Y(y)}{1/c} = f_X(y). \end{aligned}$$

The candidate points $Y_1, Y_2, \ldots$ are independent, and $U_1, U_2, \ldots$ are a random Uniform(0,1) sample independent of $Y_1, Y_2, \ldots$, hence the accepted sample is a random sample from the density $f_{Y|A} = f_X$.

3.7 *Generate a random sample of size 1000 from the Beta(3,2) distribution by acceptance-rejection method.*

Note that if $g(x)$ is the Uniform(0,1) density, then

$$\frac{f(x)}{g(x)} = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\frac{x^{a-1}(1-x)^{b-1}}{1} \le \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}, \qquad 0 < x < 1.$$

The R function below is a generator above for arbitrary parameters (a, b). It can be applied to generate the Beta(3, 2) sample.

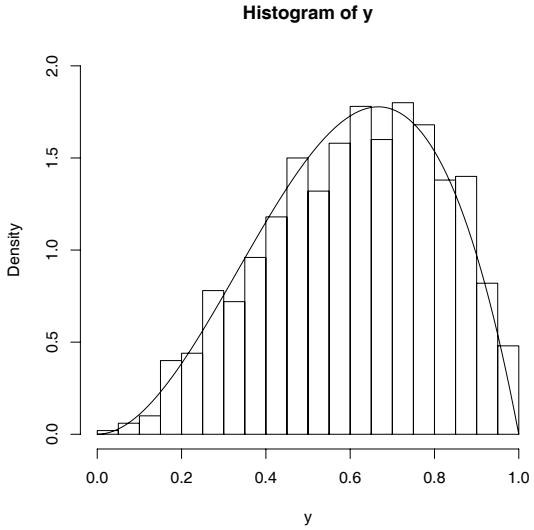
Generate x from $g(x) \sim$ Uniform(0,1) and accept x if $x^{a-1}(1-x)^{b-1} > u$. This generator can be quite inefficient if a or b is large.

```
> rBETA <- function(n, a, b) {
+     n <- 1000
+     k <- 0
+     y <- numeric(n)
+     while (k < n) {
+         u <- runif(1)
```

```
+         x <- runif(1)
+         if (x^(a - 1) * (1 - x)^(b - 1) > u) {
+             k <- k + 1
+             y[k] <- x
+         }
+     }
+     return(y)
+ }
```

The function is applied below to generate 1000 Beta(3, 2) variates and the histogram of the sample is shown with the Beta(3, 2) density superimposed.

```
> y <- rBETA(1000, a = 3, b = 2)
> hist(y, breaks = "Scott", prob = TRUE, ylim = c(0, 2))
> z <- seq(0, 1, 0.01)
> fz <- 12 * z^2 * (1 - z)
> lines(z, fz)
```



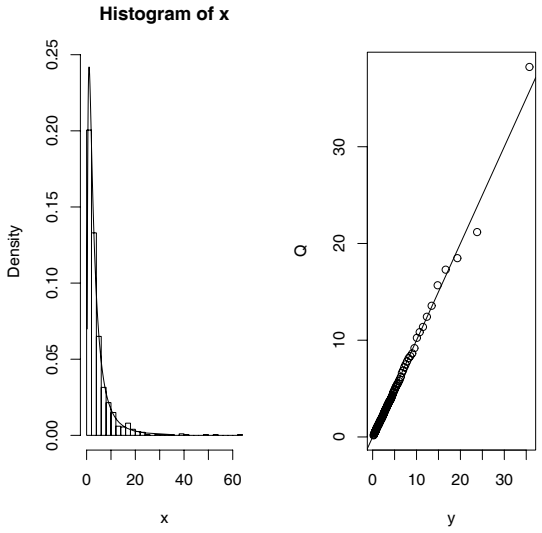
3.8 *Write a function to generate random variates from a Lognormal(μ, σ) distribution using a transformation method.*

If $X \sim \text{Lognormal}(\mu, \sigma^2)$ then $X = e^Y$ where $Y \sim N(\mu, \sigma^2)$.

```
> rLOGN <- function(n, mu, sigma) return(exp(rnorm(n, mu,
+     sigma)))
> x <- rLOGN(1000, 1, 1)
> print(summary(x))
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
 0.07344  1.32400  2.62300  4.37900  5.23300 62.03000
```

The function `rLOGN` is applied to generate a sample of size 1000, and the histogram of the sample with the lognormal density curve superimposed is shown below. Another graphical comparison can be made with a QQ plot.

```
> par(mfrow = c(1, 2))
> hist(x, breaks = "Scott", prob = TRUE, ylim = c(0, dlnorm(1,
+     1, 1)))
> y <- qlnorm(ppoints(100), 1, 1)
> fy <- dlnorm(y, 1, 1)
> lines(y, fy)
> Q <- quantile(x, ppoints(100))
> qqplot(y, Q)
> abline(0, 1)
> par(mfrow = c(1, 1))
```



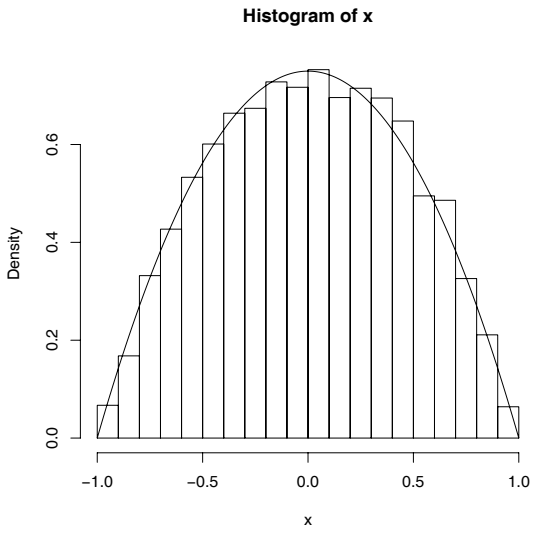
3.9 *The rescaled Epanechnikov kernel is a symmetric density function*

$$f_e(x) = \frac{3}{4}(1 - x^2), \qquad |x| \le 1.$$

Devroye and Györfi give the following algorithm for simulation from this distribution. Generate iid $U_1, U_2, U_3 \sim \text{Uniform}(-1, 1)$. If $|U_1| \ge |U_2|$ and $|U_3| \ge |U_1|$ deliver U_2; otherwise deliver U_3. The function `rEPAN` below uses this algorithm to generate random variates from f_e, and we construct the histogram density estimate of a large simulated random sample.

```
> rEPAN <- function(n) {
+     x <- numeric(n)
+     u1 <- runif(n, -1, 1)
+     u2 <- runif(n, -1, 1)
+     u3 <- runif(n, -1, 1)
+     for (i in 1:n) {
+         if ((abs(u3[i]) >= abs(u2[i])) && (abs(u3[i]) >=
+             abs(u1[i])))
+             x[i] <- u2[i]
+         else x[i] <- u3[i]
+     }
```

```
+     return(x)
+ }
> x <- rEPAN(10000)
> summary(x)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.991400 -0.343500  0.011010  0.007964  0.361100  0.984300

> hist(x, breaks = "Scott", prob = TRUE)
> a <- seq(-1, 1, 0.001)
> fe <- 0.75 * (1 - a^2)
> lines(a, fe)
```



3.10 *Prove that the algorithm given in Exercise 3.9 generates variates from the density f_e.*

The algorithm is equivalent to the following. Generate Y_1, Y_2, Y_3 iid from Uniform(0,1). If Y_3 is max let $Y = Y_2$, otherwise $Y = Y_3$. Thus, Y is the first or second order statistic of the sample Y_1, Y_2, Y_3. Deliver $T = \pm Y$ with probability $1/2, 1/2$.

Recall that the cdf of the k^{th} order statistic when $n = 3$ is given by

$$G_k(y_k) = \sum_{j=k}^{3} \binom{3}{j} [F(y_k)]^j [1 - F(y_k)]^{3-j}.$$

The cdf of Y is

$$\begin{aligned} G(y) &= \frac{1}{2}G_1(y) + \frac{1}{2}G_2(y) \\ &= \frac{1}{2}[(1 - (1-y)^3) + (3y^2(1-y) + y^3)] = \frac{1}{2}[3y - y^3] \end{aligned}$$

and the density of Y is

$$g(y) = G'(y) = \frac{1}{2}(3 - 3y^2) = \frac{3}{2}(1 - y^2), \quad 0 < y < 1.$$

Therefore, the density of T is

$$f_T(t) = \frac{1}{2} \times \frac{3}{2}(1-t^2) = \frac{3}{4}(1-t^2), \quad -1 < t < 1,$$

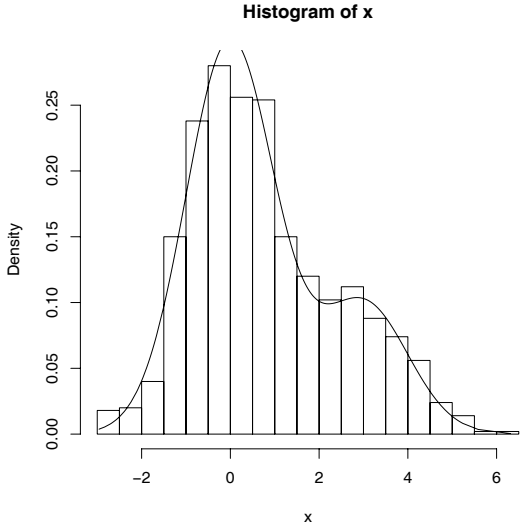
and $f_T(t) = f_e(t)$.

3.11 *Generate a random sample of size 1000 from a normal location mixture. The components of the mixture have $N(0,1)$ and $N(3,1)$ distributions with mixing probabilities p_1 and $p_2 = 1 - p_1$.*

```
> n <- 1000
> p <- 0.75
> mu <- sample(c(0, 3), size = 1000, replace = TRUE, prob = c(p,
+     1 - p))
> x <- rnorm(n, mu, 1)
```

Below is the histogram of the sample with density superimposed, for $p_1 = 0.75$.

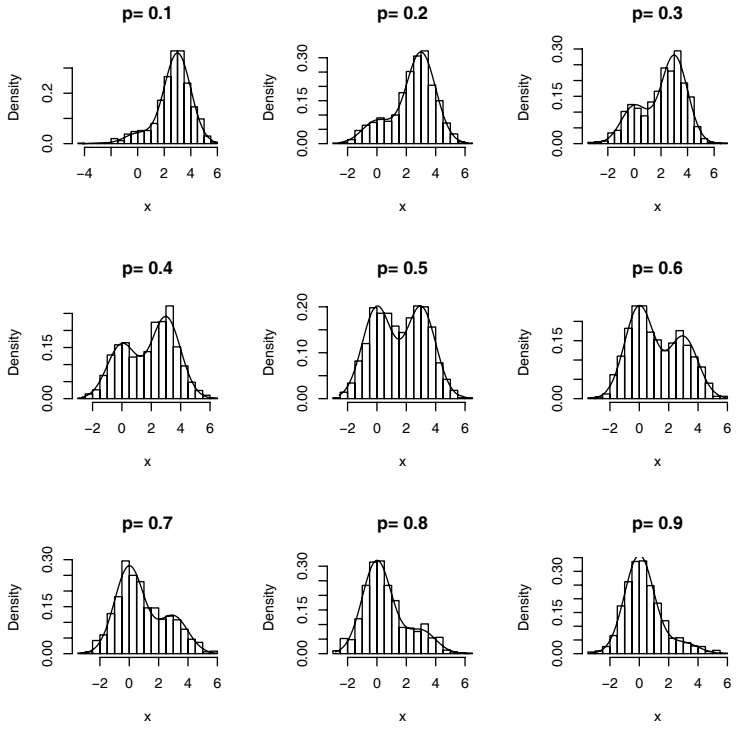
```
> hist(x, breaks = "Scott", prob = TRUE)
> y <- sort(x)
> fy <- p * dnorm(y) + (1 - p) * dnorm(y, 3, 1)
> lines(y, fy)
```



Repeating with different values for p_1:

```
> par(mfrow = c(3, 3))
> p <- seq(0.1, 0.9, length = 9)
> for (i in 1:9) {
+     mu <- sample(c(0, 3), size = 1000, replace = TRUE,
+         prob = c(p[i], 1 - p[i]))
+     x <- rnorm(n, mu, 1)
+     hist(x, breaks = "Scott", prob = TRUE, main = paste("p=",
+         p[i]))
+     y <- sort(x)
```

```
+     fy <- p[i] * dnorm(y) + (1 - p[i]) * dnorm(y, 3,
+         1)
+     lines(y, fy)
+ }
> par(mfrow = c(1, 1))
```



From the graphs, we might conjecture that the mixture is bimodal if $0.2 < p < 0.8$. (Some results characterizing the shape of a normal mixture density are given by I. Eisenberger (1964), "Genesis of Bimodal Distributions," *Technometrics* **6**, 357–363.)

3.12 *Simulate a continuous Exponential-Gamma mixture. Suppose that the rate parameter Λ has Gamma(r, β) distribution and Y has Exp(Λ) distribution. That is, $(Y|\Lambda = \lambda) \sim f_Y(y|\lambda) = \lambda e^{-\lambda y}$. Generate 1000 random observations from this mixture with $r = 4$ and $\beta = 2$.*

Supply the sample of randomly generated λ as the Exponential rate argument in `rexp`.

```
> n <- 1000
> r <- 4
> beta <- 2
> lambda <- rgamma(n, r, beta)
> x <- rexp(n, rate = lambda)
```

3.13 The mixture in Exercise 3.12 has a Pareto distribution with cdf

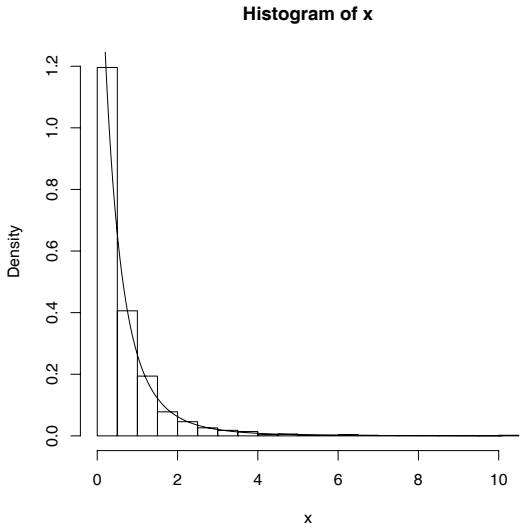
$$F(y) = 1 - \left(\frac{\beta}{\beta + y}\right)^r, \quad y \geq 0.$$

(This is an alternative parameterization of the Pareto cdf given in Exercise 3.) The Pareto density is

$$f(y) = F'(y) = \frac{r\beta^r}{(\beta + y)^{r+1}}, \qquad y \geq 0.$$

Below we generate 1000 random observations from the mixture with $r = 4$ and $\beta = 2$ and compare the empirical and theoretical (Pareto) distributions by graphing the density histogram of the sample and superimposing the Pareto density curve.

```
> hist(x, breaks = "Scott", prob = TRUE)
> y <- sort(x)
> fy <- r * beta^r * (beta + y)^(-r - 1)
> lines(y, fy)
```



3.14 *Generate 200 random observations from the 3-dimensional multivariate normal distribution having mean vector* $\mu = (0, 1, 2)$ *and covariance matrix*

$$\Sigma = \begin{bmatrix} 1.0 & -0.5 & 0.5 \\ -0.5 & 1.0 & -0.5 \\ 0.5 & -0.5 & 1.0 \end{bmatrix}.$$

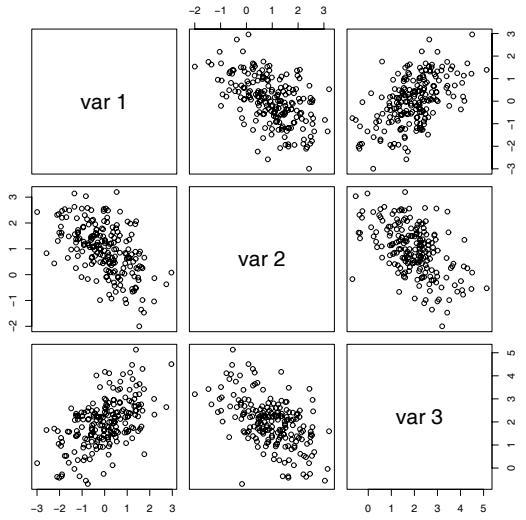
using the Choleski factorization method.

```
> rmvn.Choleski <- function(n, mu, Sigma) {
+     d <- length(mu)
+     Q <- chol(Sigma)
+     Z <- matrix(rnorm(n * d), nrow = n, ncol = d)
+     X <- Z %*% Q + matrix(mu, n, d, byrow = TRUE)
+     X
+ }
```

```
> Sigma <- matrix(c(1, -0.5, 0.5, -0.5, 1, -0.5, 0.5, -0.5,
+     1), 3, 3)
> mu <- c(0, 1, 2)
> x <- rmvn.Choleski(200, mu, Sigma)
> colMeans(x)
[1] 0.01982928 0.95307678 1.96838036
> cor(x)
           [,1]       [,2]       [,3]
[1,]  1.0000000 -0.5232784  0.5413212
[2,] -0.5232784  1.0000000 -0.5135698
[3,]  0.5413212 -0.5135698  1.0000000
```

From the `pairs` plot below it appears that the centers of the distributions agree with the parameters in μ, and the correlations also agree approximately with the parameters in Σ.

```
> pairs(x)
```



3.15 *Write a function that will standardize a multivariate normal sample for arbitrary n and d.*

The transformation used to generate multivariate normal samples by eigen-decomposition was $X = Z\Sigma^{1/2} + J\mu^T$, where $\Sigma^{1/2} = P\Lambda^{1/2}P^T$. To standardize the sample,

$$Y = (X - J\mu^T)\Sigma^{-1/2},$$

where $\Sigma^{-1/2} = P\Lambda^{-1/2}P^T$. To standardize the sample using estimated mean and covariance,

$$Y = (X - J\overline{X}^T)S^{-1/2},$$

where $\overline{X}$ is the vector of sample means and $S^{-1/2}$ is computed from the eigen-decomposition by the same method used to compute $\Sigma^{-1/2}$.

```
> standardize <- function(x) {
+     if (is.vector(x))
+         return(scale(x))
+     x <- as.matrix(x)
+     d <- ncol(x)
+     n <- nrow(x)
+     S <- cov(x)
+     m <- colMeans(x)
+     means <- matrix(m, n, d, byrow = TRUE)
+     ev <- eigen(S, symmetric = TRUE)
+     lambda <- ev$values
+     V <- ev$vectors
+     R <- V %*% diag(sqrt(1/lambda)) %*% t(V)
+     y <- (x - means) %*% R
+     return(y)
+ }
```

To check the function, print the means and covariance matrix of a standardized sample.

```
> x <- rmvn.Choleski(200, mu, Sigma)
> colMeans(x)
[1] -0.1475262  1.1106628  1.9567587
> cov(x)
           [,1]       [,2]       [,3]
[1,]  1.0827541 -0.5786420  0.5830538
[2,] -0.5786420  1.1000760 -0.5739768
[3,]  0.5830538 -0.5739768  1.1177893
> y <- standardize(x)
> colMeans(y)
[1] -2.190956e-17  6.938894e-17  1.103609e-16
> cov(y)
             [,1]         [,2]         [,3]
[1,] 1.000000e+00 8.825479e-17 5.948511e-17
[2,] 8.825479e-17 1.000000e+00 6.039300e-16
[3,] 5.948511e-17 6.039300e-16 1.000000e+00
```

The sample mean vector of the standardized sample is (approximately) the zero vector, and the sample covariance of the standardized sample is (approximately) the identity matrix.

3.16 *Efron and Tibshirani discuss the* `scor (bootstrap)` *test score data on 88 students who took examinations in five subjects. Each row of the data frame is a set of scores* $(x_{i1}, \ldots, x_{i5})$ *for the* i^{th} *student. Standardize the scores by type of exam. That is, standardize the bivariate samples* (X_1, X_2) *(closed book) and the trivariate samples* (X_3, X_4, X_5) *(open book). Compute the covariance matrix of the transformed sample of five scores.*

```
> data(scor, package = "bootstrap")
> cbook <- as.matrix(scor[, 1:2])
> obook <- as.matrix(scor[, 3:5])
```

```
> cstd <- standardize(cbook)
> ostd <- standardize(obook)
> x <- cbind(cstd, ostd)
> round(cov(x), 5)
        [,1]    [,2]    [,3]    [,4]    [,5]
[1,] 1.00000 0.00000 0.35315 0.17654 0.19225
[2,] 0.00000 1.00000 0.37329 0.25521 0.20976
[3,] 0.35315 0.37329 1.00000 0.00000 0.00000
[4,] 0.17654 0.25521 0.00000 1.00000 0.00000
[5,] 0.19225 0.20976 0.00000 0.00000 1.00000
```

3.17 *Compare the performance of the acceptance-rejection beta generator, the ratio of gammas method and the R generator* `rbeta`*. Fix the parameters* $a = 2, b = 2$ *and time each generator on 1000 iterations with sample size 5000.*

Note: For "large" a and b, the `rBETA` generator is very slow. For generating a large sample and large number of iterations, $a = b = 2$ is large.

```
> n <- 500
> N <- 100
> a <- 2
> b <- 2
> set.seed(100)
> system.time(for (i in 1:N) rbeta(n, a, b), gcFirst = TRUE)
   user  system elapsed
   0.05    0.00    0.05
> set.seed(100)
> system.time(for (i in 1:N) {
+     u <- rgamma(n, a, 1)
+     v <- rgamma(n, b, 1)
+     x <- u/(u + v)
+ }, gcFirst = TRUE)
   user  system elapsed
   0.06    0.00    0.06
> set.seed(100)
> system.time(for (i in 1:N) rBETA(n, a, b), gcFirst = TRUE)
   user  system elapsed
  16.11    0.02   16.25
```

There is a bigger difference in the timings when the parameters a and b are larger.

```
> a <- 3
> b <- 2
> set.seed(100)
> system.time(for (i in 1:N) rbeta(n, a, b), gcFirst = TRUE)
   user  system elapsed
   0.05    0.00    0.05
> set.seed(100)
> system.time(for (i in 1:N) {
```

```
+     u <- rgamma(n, a, 1)
+     v <- rgamma(n, b, 1)
+     x <- u/(u + v)
+ }, gcFirst = TRUE)
   user  system elapsed
   0.06    0.00    0.06
> set.seed(100)
> system.time(for (i in 1:N) rBETA(n, a, b), gcFirst = TRUE)
   user  system elapsed
  31.08    0.02   31.14
```

3.18 *Write a function to generate a random sample from a $W_d(\Sigma, n)$ (Wishart) distribution for $n > d + 1 \geq 1$, based on Bartlett's decomposition.*

The following function generates one $W_d(\Sigma, \nu)$ random variate. (Using `for` loops for clarity here; see the `lower.tri` function for a way to avoid the loops.)

```
> rWISH <- function(Sigma, n) {
+     d <- nrow(Sigma)
+     U <- chol(Sigma)
+     y <- matrix(0, d, d)
+     for (j in 1:d) {
+         for (i in j:d) y[i, j] <- rnorm(1)
+         y[j, j] <- sqrt(rchisq(1, n - i + 1))
+     }
+     A <- y %*% t(y)
+     return(t(U) %*% A %*% U)
+ }
```

Try the generator on a few examples:

```
          [,1]      [,2]      [,3]
[1,] 51.029934 -4.224545  7.992898
[2,] -4.224545 53.988282 -2.174543
[3,]  7.992898 -2.174543 48.808521
          [,1]      [,2]      [,3]
[1,] 60.644509 -2.166379 22.816272
[2,] -2.166379 55.098878  9.795274
[3,] 22.816272  9.795274 48.865629
          [,1]      [,2]
[1,]  34.94311 -21.66413
[2,] -21.66413  49.22049
          [,1]      [,2]
[1,]  36.44908 -18.50966
[2,] -18.50966  47.56137
```

Note that the result is a $d \times d$ matrix. To generate a random sample from $W_d(\Sigma, \nu)$, the result can be returned in an array.

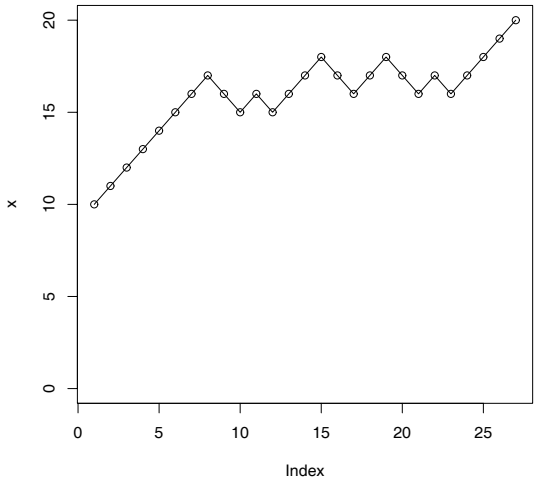
```
> rWish <- function(n, Sigma, nu) {
+     w <- replicate(n, rWISH(Sigma, nu))
+     return(array(w, c(2, 2, n)))
```

```
+ }
> rWish(3, Sigma = s, 50)

, , 1

          [,1]      [,2]
[1,]  19.43331 -10.96212
[2,] -10.96212  36.93807

, , 2

          [,1]      [,2]
[1,]  39.40838 -26.47205
[2,] -26.47205  56.02625

, , 3

          [,1]      [,2]
[1,] 19.061526 -9.897257
[2,] -9.897257 57.900117
```

3.19 *Suppose that A and B each start with a stake of $10, and bet $1 on consecutive coin flips. The game ends when either one of the players has all the money. Let S_n be the fortune of player A at time n. Then $\{S_n, n \geq 0\}$ is a symmetric random walk with absorbing barriers at 0 and 20. Simulate a realization of the process $\{S_n, n \geq 0\}$ and plot S_n vs the time index from time 0 until a barrier is reached.*

```
> A <- 10
> p <- 0.5
> x <- numeric(1000)
> x[1] <- A
> for (i in 2:1000) {
+     incr <- sample(c(-1, 1), size = 1, prob = c(p, 1 -
+         p))
+     x[i] <- incr
+     A <- sum(x)
+     if (isTRUE(all.equal(A, 20)))
+         break
+     if (isTRUE(all.equal(A, 0)))
+         break
+ }
> x <- cumsum(x[1:i])
```

```
> plot(x, type = "l", ylim = c(0, 20))
> points(x)
```



3.20 A *compound Poisson process is a stochastic process* $\{X(t), t \geq 0\}$ *that can be represented as the random sum* $X(t) = \sum_{i=1}^{N(t)} Y_i,\ t \geq 0$, *where* $\{N(t), t \geq 0\}$ *is a Poisson process and* $Y_1, Y_2, \ldots$ *are iid and independent of* $\{N(t), t \geq 0\}$. *Write a program to simulate a compound Poisson*(λ)*–Gamma process (*Y *has a Gamma distribution). Estimate the mean and the variance of* $X(10)$ *for several choices of the parameters and compare with the theoretical values.*

```
> r <- 4
> beta <- 2
> lambda <- 3
> t0 <- 10
> PP <- function(lambda, t0) {
+     Tn <- rexp(1000, lambda)
+     Sn <- cumsum(Tn)
+     stopifnot(Sn[1000] > t0)
+     n <- min(which(Sn > t0))
+     return(n - 1)
+ }
> y <- numeric(1000)
> for (i in 1:1000) {
+     N <- PP(lambda, t0)
+     y[i] <- sum(rgamma(N, shape = r, rate = beta))
+ }
```

Show that $E[X(t)] = \lambda t E[Y_1]$ and $Var(X(t)) = \lambda t E[Y_1^2]$. Then the empirical and theoretical values of the mean are:

```
> mean(y)
[1] 60.22843
> lambda * t0 * r/beta
[1] 60
```

The empirical and theoretical values of the variance are:

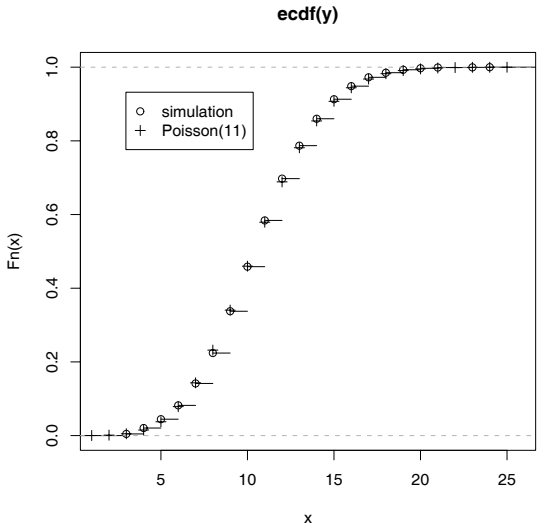
```
> var(y)

[1] 145.2459

> lambda * t0 * (r/beta^2 + (r/beta)^2)

[1] 150
```

3.21 *A nonhomogeneous Poisson process has mean value function* $m(t) = t^2 + 2t, \quad t \geq 0$. *Determine the intensity function* $\lambda(t)$ *of the process, and write a program to simulate the process on the interval* $[4, 5]$. *Compute the probability distribution of* $N(5) - N(4)$, *and compare it to the empirical estimate obtained by replicating the simulation.*

The mean value function is $m(t) = \int_0^t \lambda(t)dt = t^2 + 2t$, so $\lambda(t) = 2t + 2$ and on [4, 5] we have $\lambda(t) \leq 12 = \lambda_0$. The probability distribution of $N(5) - N(4)$ is Poisson with mean $m(5) - m(4) = 5^2 + 2(5) - (4^2 + 2(4)) = 35 - 24 = 11$.

```
> n <- 2000
> lambda <- 12
> upper <- 100
> y <- replicate(n, expr = {
+     N <- rpois(1, lambda * upper)
+     Tn <- rexp(N, lambda)
+     Sn <- cumsum(Tn)
+     Un <- runif(N)
+     keep <- (Un <= (2 * Sn + 2)/lambda)
+     sum(Sn[keep] <= 5 & Sn[keep] > 4)
+ })
> mean(y)

[1] 10.952

> var(y)

[1] 10.67904
```

```
> plot(ecdf(y))
> points(0:25, ppois(0:25, 11), pch = 3)
> legend("topleft", inset = 0.1, legend = c("simulation",
+     "Poisson(11)"), pch = c(1, 3))
```



CHAPTER 4

Visualization of Multivariate Data

4.1 *Generate 200 random observations from the multivariate normal distribution having mean vector* $\mu = (0, 1, 2)$ *and covariance matrix*

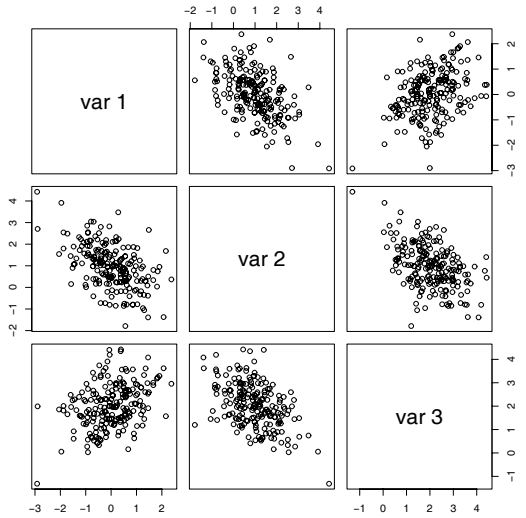
$$\Sigma = \begin{bmatrix} 1.0 & -0.5 & 0.5 \\ -0.5 & 1.0 & -0.5 \\ 0.5 & -0.5 & 1.0 \end{bmatrix}.$$

Use any of the functions `rmvn.eigen`, `rmvn.Choleski` from Chapter 3, `mvrnorm (MASS)` or `rmvnorm (mvtnorm)`.

```
> library(MASS)
> Sigma <- matrix(c(1, -0.5, 0.5, -0.5, 1, -0.5, 0.5, -0.5,
+     1), 3, 3)
> mu <- c(0, 1, 2)
> x <- mvrnorm(200, mu, Sigma)
> colMeans(x)

[1] 0.02626572 0.96678314 2.04179355

> cor(x)

           [,1]       [,2]       [,3]
[1,]  1.0000000 -0.5011990  0.3621894
[2,] -0.5011990  1.0000000 -0.4714206
[3,]  0.3621894 -0.4714206  1.0000000

> detach(package:MASS)
```

From the pairs plot below it appears that the parameters for each plot approximately agree with the parameters of the corresponding bivariate distributions.

```
> pairs(x)
```

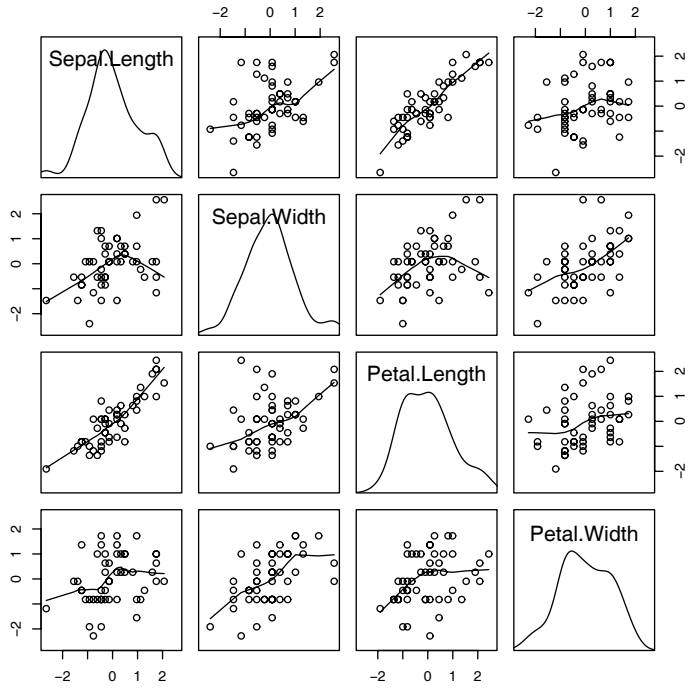


4.2 *Add a fitted smooth curve to each of the iris virginica scatterplots.*

The panel function below is similar to `panel.smooth`, with the options for color removed.

```
> data(iris)
> panel.d <- function(x, ...) {
+     usr <- par("usr")
+     on.exit(par(usr))
+     par(usr = c(usr[1:2], 0, 0.5))
+     lines(density(x))
+ }
> panel.sm <- function(x, y, bg = NA, pch = par("pch"),
+     cex = 1, span = 2/3, iter = 3, ...) {
+     points(x, y, pch = pch, bg = bg, cex = cex)
+     ok <- is.finite(x) & is.finite(y)
+     if (any(ok))
+         lines(stats::lowess(x[ok], y[ok], f = span, iter = iter),
+             ...)
+ }
> x <- scale(iris[101:150, 1:4])
> r <- range(x)
```

```
> pairs(x, panel = panel.sm, diag.panel = panel.d, xlim = r,
+     ylim = r)
```



4.3 *The random variables X and Y are independent and identically distributed with normal mixture distributions. The components of the mixture have $N(0,1)$ and $N(3,1)$ distributions with mixing probabilities p_1 and $p_2 = 1 - p_1$ respectively.*

The code below generates a bivariate random sample from the joint distribution of (X, Y).

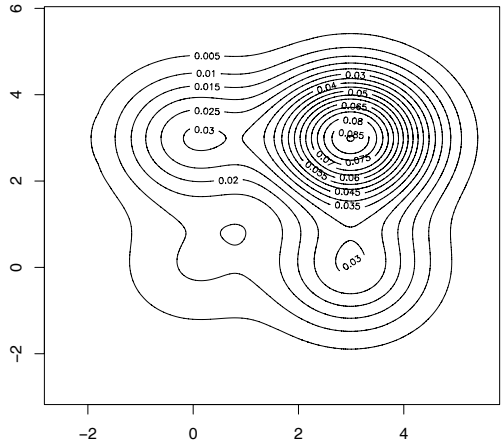
```
> n <- 500
> mu <- c(0, 3)
> p <- 0.25
> m <- sample(mu, size = 2 * n, replace = TRUE, prob = c(p,
+     1 - p))
> X <- matrix(rnorm(2 * n, m), n, 2)
```

For the contour plot, we need the joint density. The random variables are independent so the joint density is the product of the marginals. (If dependent, cannot sort X and Y independently.)

(Generally, when the joint density is available, we would not construct the contour plot from a sample, because we can generate the grid of points directly. When the joint density is not available, a density estimate can provide the z values.)

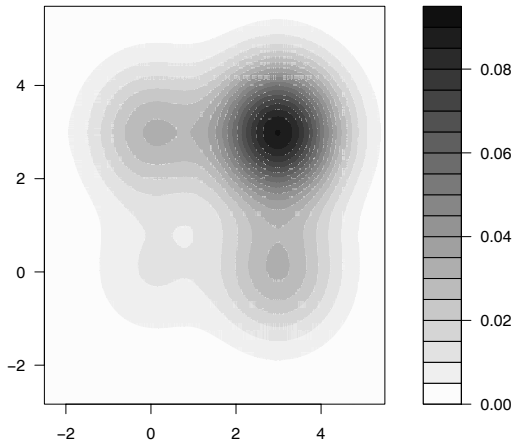
```
> f <- function(x, y) {
+     f1 <- p * dnorm(x, mu[1]) + (1 - p) * dnorm(x, mu[2])
+     f2 <- p * dnorm(y, mu[1]) + (1 - p) * dnorm(y, mu[2])
+     f1 * f2
+ }
> x <- sort(X[, 1])
```

```
> y <- sort(X[, 2])
> z <- outer(x, y, f)

> contour(x, y, z, nlevels = 20)
```



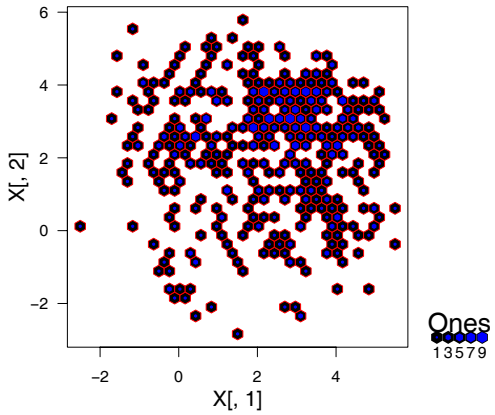
4.4 *Construct a filled contour plot of the bivariate mixture in Exercise 4.3.*

```
> filled.contour(x, y, z, col = gray(seq(0.99, 0.01, length = 20)))
```



Increase n above to get a nicer plot. The default plot of a hexbin object is in grayscale. The plot command below produces the plot in color.

```
> require(hexbin)
> bin <- hexbin(X[, 1], X[, 2])
> plot(bin, style = "nested.lattice")
```

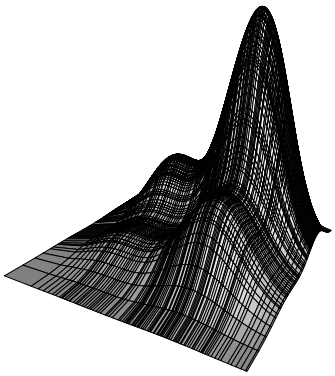


4.5 *Construct a surface plot of the bivariate mixture in Exercise 4.3.*

(First "thinning out" the data because the perspective plot turns out to be quite dark for the printed version.)

```
> i <- seq(1, n, 5)
> u <- x[i]
> v <- y[i]
> w <- z[i, i]
```

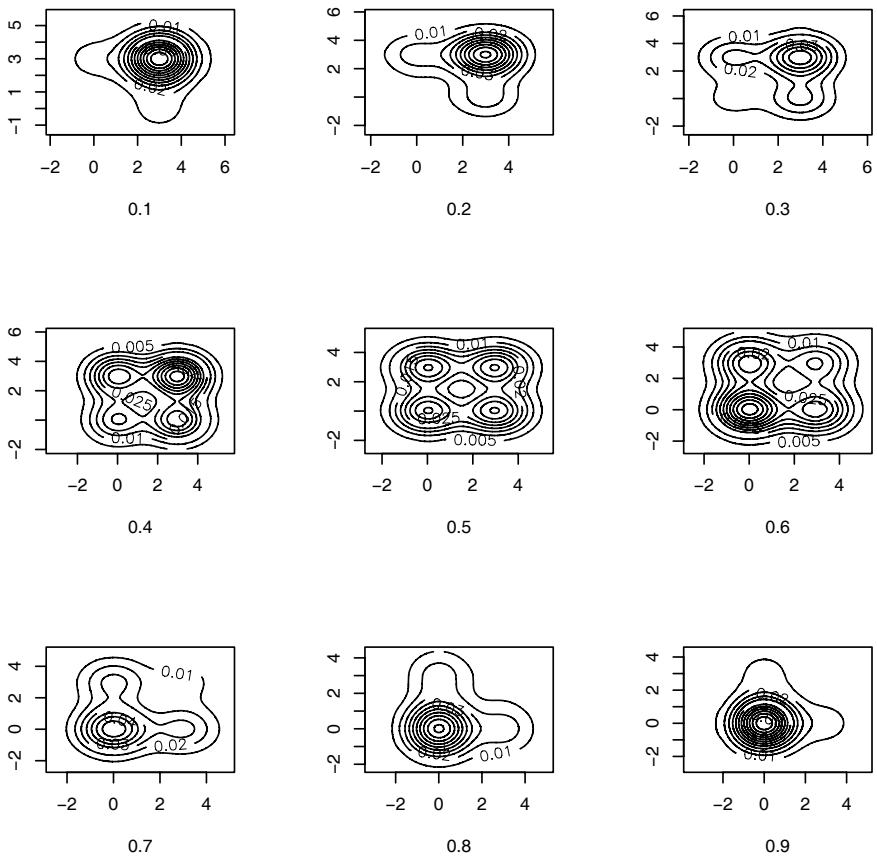
```
> persp(u, v, w, shade = TRUE, theta = 30, ltheta = 30,
+     box = FALSE)
```



4.6 *Repeat Exercise 4.3 for various different choices of the parameters of the mixture model, and compare the distributions through contour plots.*

```
> n <- 200
> mu <- c(0, 3)
> pr <- seq(0.1, 0.9, 0.1)
> f <- function(x, y) {
+     f1 <- p * dnorm(x, mu[1]) + (1 - p) * dnorm(x, mu[2])
+     f2 <- p * dnorm(y, mu[1]) + (1 - p) * dnorm(y, mu[2])
+     f1 * f2
+ }
> x <- matrix(0, n, 9)
> y <- matrix(0, n, 9)
> z <- array(0, c(n, n, 9))
> for (i in 1:9) {
+     p <- pr[i]
+     m <- sample(mu, size = 2 * n, replace = TRUE, prob = c(p,
+         1 - p))
+     X <- matrix(rnorm(2 * n, m), n, 2)
+     x[, i] <- sort(X[, 1])
+     y[, i] <- sort(X[, 2])
+     z[, , i] <- outer(x[, i], y[, i], f)
+ }
```

```
> par(mfrow = c(3, 3))
> for (i in 1:9) contour(x[, i], y[, i], z[, , i], nlevels = 10,
+     xlab = pr[i])
> par(mfrow = c(1, 1))
```



4.7 *Create a parallel coordinates plot of the* `crabs (MASS)` *data using all 200 observations. Compare the plots before and after adjusting the measurements by the size of the crab. Interpret the resulting plots.*

To use lattice: `print(parallel(~crabs[4:8] | sp*sex, crabs))`

Graphs are very similar to the lattice parallel plots in the book. Much of the variability between groups is in overall size. Following the suggestion in Venables and Ripley (2002) we adjust the measurements by the area of the carapace.

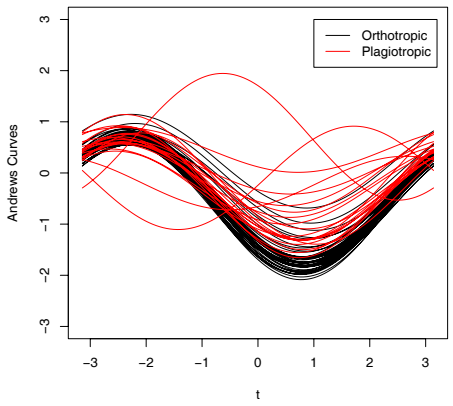
```
a <- crabs$CW * crabs$CL          #area of carapace
```

Orange males have a clear profile that is almost opposite that of blue females and similar on some measurements to orange females. The blue males have the least clear profile and do not appear to match the blue females. In both species the rear width appears to be larger in females than in males. Orange crabs have small carapace width relative to body depth.

4.8 *Create a plot of Andrews curves for the* `leafshape17 (DAAG)` *data, using the logarithms of measurements (logwid, logpet, loglen).*

The code is the same as the example in the book except that the names of the variables are changed.

```
> library(DAAG, warn.conflicts = FALSE)
> attach(leafshape17)
> f <- function(a, v) {
+     v[1]/sqrt(2) + v[2] * sin(a) + v[3] * cos(a)
+ }
> x <- cbind(bladelen, petiole, bladewid)
> n <- nrow(x)
> mins <- apply(x, 2, min)
> maxs <- apply(x, 2, max)
> r <- maxs - mins
> y <- sweep(x, 2, mins)
> y <- sweep(y, 2, r, "/")
> x <- 2 * y - 1

> plot(0, 0, xlim = c(-pi, pi), ylim = c(-3, 3), xlab = "t",
+     ylab = "Andrews Curves", main = "", type = "n")
> a <- seq(-pi, pi, len = 101)
> dim(a) <- length(a)
> for (i in 1:n) {
+     g <- arch[i] + 1
+     y <- apply(a, MARGIN = 1, FUN = f, v = x[i, ])
+     lines(a, y, lty = 1, col = g)
+ }
> legend(3, c("Orthotropic", "Plagiotropic"), lty = 1,
+     col = 1:2)
```




```
> detach(leafshape17)
> detach(package:DAAG)
```

4.9 *Refer to the full* `leafshape (DAAG)` *data set. Produce Andrews curves for each of the six locations. Split the screen into six plotting areas, and display all six plots on one screen. Set line type or color to identify leaf architecture.*

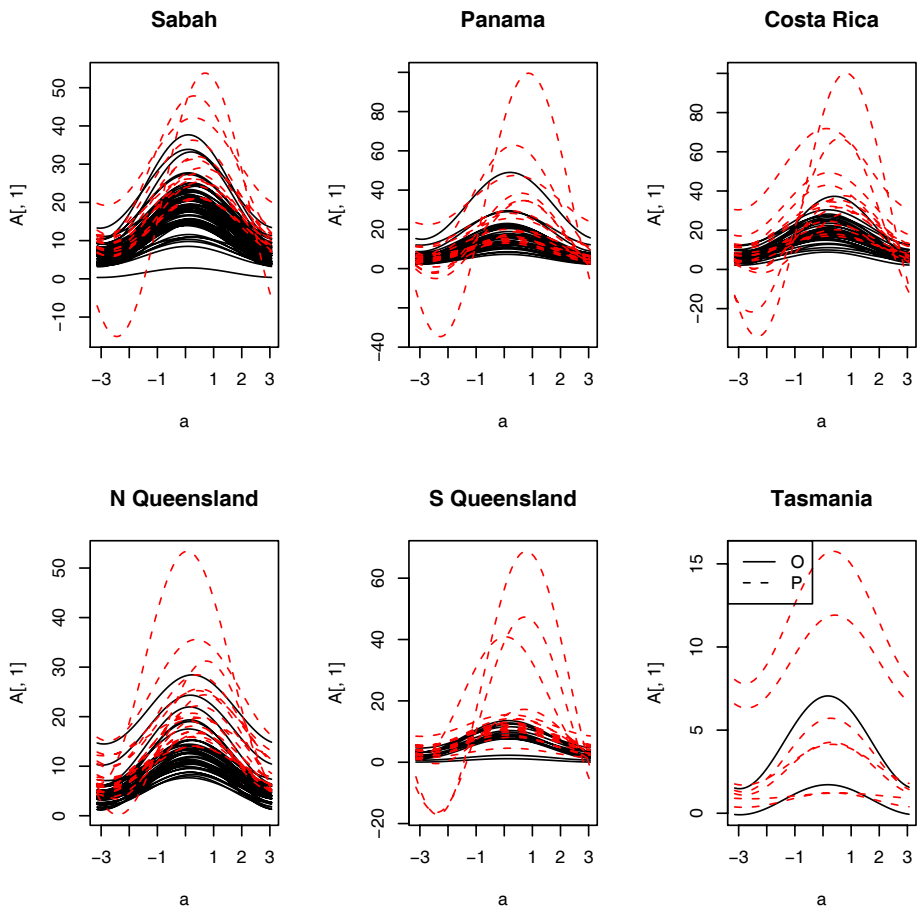
There are a few apparent outliers or unusual observations, but most of the differences appear to be in scale or size of leaf. The second array of plots scales

the measurements to common range. The profiles of the Andrews curves look very similar across all six locations.

```
> library(DAAG, warn.conflicts = FALSE)
> attach(leafshape)
> f <- function(a, v) {
+     v[1]/sqrt(2) + v[2] * sin(a) + v[3] * cos(a)
+ }
> names(leafshape)

[1] "bladelen" "petiole"  "bladewid" "latitude" "logwid"   "logpet"
[7] "loglen"   "arch"     "location"

> L <- levels(location)
> table(location)

location
       Sabah       Panama   Costa Rica N Queensland S Queensland
          80           55           50           61           31
    Tasmania
           9

> loc <- as.integer(location)
> a <- seq(-pi, pi, 0.1)
```

For gray scale below: `palette(gray((1:6) / 8))`

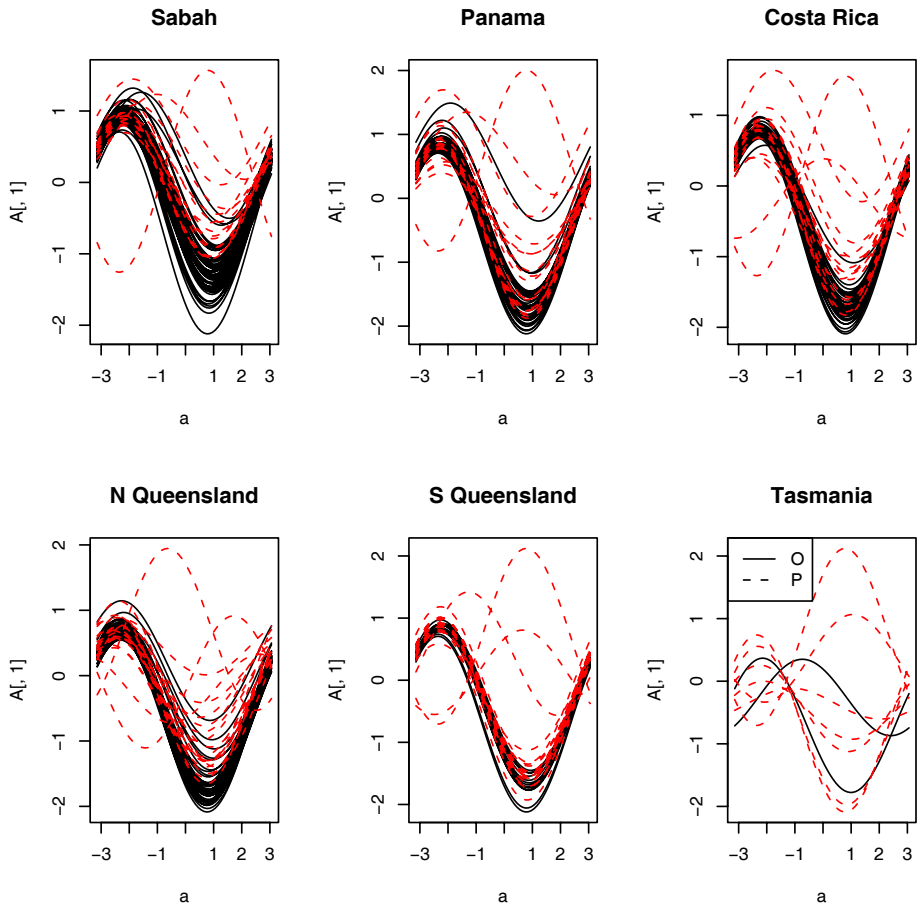
```
> par(mfrow = c(2, 3))
> for (j in 1:6) {
+     y <- subset(leafshape, subset = (location == L[j]))
+     x <- as.matrix(y[, 1:3])
+     ar <- as.integer(y$arch) + 1
+     A <- apply(x, 1, f, a = a)
+     plot(a, A[, 1], ylim = range(A), type = "l", main = L[j])
+     for (i in 2:nrow(x)) lines(a, A[, i], col = ar[i],
+         lty = ar[i])
+ }
> legend("topleft", legend = c("O", "P"), lty = 1:2)
> par(mfrow = c(1, 1))
```



For gray scale below: `palette(gray((1:6) / 8))`

```
> par(mfrow = c(2, 3))
> for (j in 1:6) {
+     y <- subset(leafshape, subset = (location == L[j]))
+     x <- as.matrix(y[, 1:3])
+     n <- nrow(x)
+     mins <- apply(x, 2, min)
+     maxs <- apply(x, 2, max)
+     r <- maxs - mins
+     z <- sweep(x, 2, mins)
+     z <- sweep(z, 2, r, "/")
+     x <- 2 * z - 1
+     ar <- as.integer(y$arch) + 1
+     A <- apply(x, 1, f, a = a)
+     plot(a, A[, 1], ylim = range(A), type = "l", main = L[j])
+     for (i in 2:nrow(x)) lines(a, A[, i], col = ar[i],
+         lty = ar[i])
```

```
+ }
> legend("topleft", legend = c("O", "P"), lty = 1:2)
> par(mfrow = c(1, 1))
```




```
> detach(leafshape)
> detach(package:DAAG)
> palette("default")
```

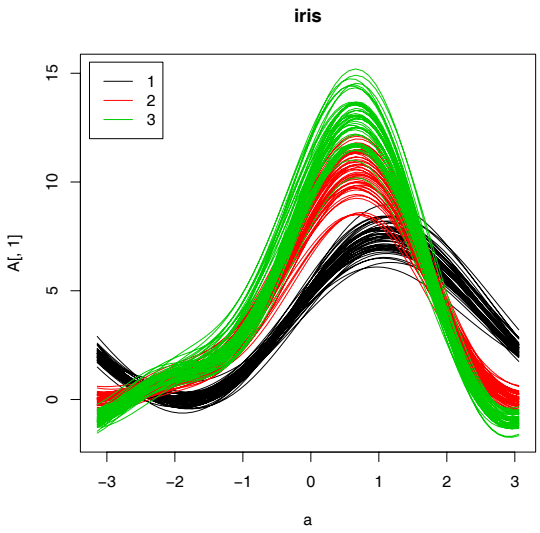
4.10 Generalize the Andrews curve function for vectors in $\mathbb{R}^d$, where the dimension $d \geq 2$ is arbitrary. Test this function by producing Andrews curves for the `iris` data ($d = 4$) and `crabs (MASS)` data ($d = 5$).

```
> f <- function(v, a) {
+     d <- length(v)
+     y <- v[1]/sqrt(2)
+     for (i in 2:d) {
+         j <- i%/%2
+         if (i%%2) {
+             y <- y + v[i] * cos(j * a)
```

```
+         }
+         else {
+             y <- y + v[i] * sin(j * a)
+         }
+     }
+     return(y)
+ }
> a <- seq(-pi, pi, 0.1)
> x <- as.matrix(iris[, 1:4])
> A <- apply(x, 1, f, a = a)
```

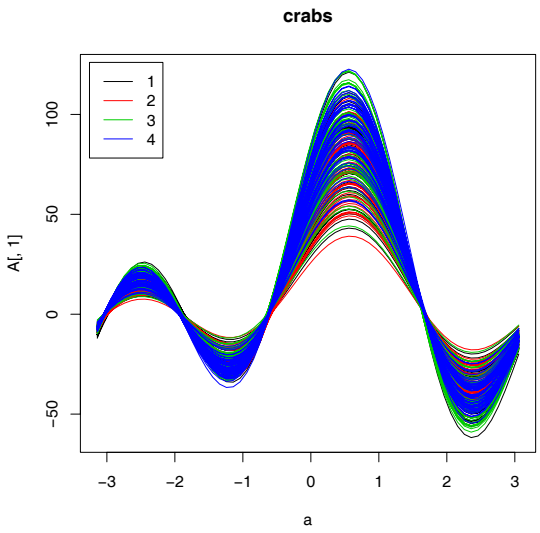
For gray scale below: `palette(gray((1:6) / 8))`

```
> plot(a, A[, 1], ylim = range(A), type = "l", main = "iris")
> s <- as.integer(iris$Species)
> for (i in 2:nrow(x)) lines(a, A[, i], col = s[i], lty = 1)
> legend("topleft", inset = 0.02, legend = 1:3, lty = 1,
+     col = 1:3)
```




```
> library(MASS)
> attach(crabs)
> x <- as.matrix(crabs[, 4:8])
> g <- rep(1:4, each = 50)
> A <- apply(x, 1, f, a = a)
```

```
> plot(a, A[, 1], ylim = range(A), type = "l", main = "crabs")
> for (i in 2:nrow(x)) lines(a, A[, i], col = g[i])
> legend("topleft", inset = 0.02, legend = 1:4, col = 1:4,
+     lty = 1)
```



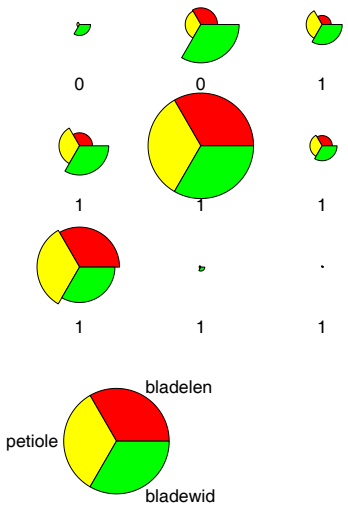

```
> detach(crabs)
> detach(package:MASS)
> palette("default")
```

4.11 *Refer to the full* `leafshape (DAAG)` *data set. Display a segment style stars plot for leaf measurements at latitude 42 (Tasmania). Repeat using the logarithms of the measurements.*

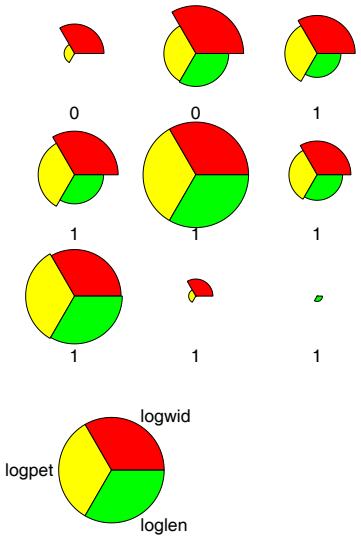
Use for color version: `palette(rainbow(6))`

For gray scale: `palette(gray((1:6) / 8))`

```
> library(DAAG, warn.conflicts = FALSE)
> attach(leafshape)
> names(leafshape)
[1] "bladelen" "petiole"  "bladewid" "latitude" "logwid"   "logpet"
[7] "loglen"   "arch"     "location"
> table(location)
location
       Sabah       Panama   Costa Rica N Queensland S Queensland
          80           55           50           61           31
    Tasmania
           9
> x <- subset(leafshape, subset = (location == "Tasmania"))
> y <- x[, 1:3]
> logy <- x[, 5:7]


> palette(rainbow(6))
> stars(y, draw.segments = TRUE, labels = x$arch, nrow = 3,
+     ylim = c(-2, 10), key.loc = c(3, -1))
```




```
> stars(logy, draw.segments = TRUE, labels = x$arch, nrow = 3,
+     ylim = c(-2, 10), key.loc = c(3, -1))
```




```
> palette("default")
> detach(leafshape)
> detach(package:DAAG)
```

CHAPTER 5

Monte Carlo Integration and Variance Reduction

5.1 *Compute a Monte Carlo estimate of*

$$\int_0^{\pi/3} sin\, t\, dt$$

and compare your estimate with the exact value of the integral.

The simple Monte Carlo estimator is

$$(b-a)\int_a^b g(x)dx = \frac{\pi}{3}\left\{\frac{1}{m}\sum_{i=1}^{m} sin(u)\right\},$$

where u is generated from Uniform$(0, \pi/3)$.

```
> m <- 10000
> x <- runif(m, 0, pi/3)
> theta.hat <- pi/3 * mean(sin(x))
> print(theta.hat)
[1] 0.5030077
```

The exact value of the integral is 0.5. Repeating the estimation 1000 times gives an estimate of the standard error:

```
> y <- replicate(1000, expr = {
+     x <- runif(m, 0, pi/3)
+     theta.hat <- pi/3 * mean(sin(x))
+ })
> mean(y)
[1] 0.4999822
> sd(y)
[1] 0.002743902
```

5.2 *Compute a Monte Carlo estimate of the standard normal cdf, by generating from the Uniform(0,x) distribution. Compare your estimates with the normal cdf function* `pnorm`*. Compute an estimate of the variance of your Monte Carlo estimate of* $\Phi(2)$*, and a 95% confidence interval for* $\Phi(2)$*.*

```
> x <- seq(0.1, 2.5, length = 10)
> m <- 10000
> cdf <- numeric(length(x))
> for (i in 1:length(x)) {
+     u <- runif(m, 0, x[i])
+     g <- x[i] * exp(-(u^2)/2)
+     cdf[i] <- mean(g)/sqrt(2 * pi) + 0.5
```

```
+ }
> Phi <- pnorm(x)
> print(round(rbind(x, cdf, Phi), 3))
    [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
x   0.10 0.367 0.633 0.900 1.167 1.433 1.700 1.967 2.233 2.500
cdf 0.54 0.643 0.737 0.816 0.879 0.924 0.954 0.974 0.988 0.991
Phi 0.54 0.643 0.737 0.816 0.878 0.924 0.955 0.975 0.987 0.994
```

To estimate the variance of the MC estimate of $\Phi(2)$, replicate the experiment. Then apply the CLT to construct an approximate 95% confidence interval for $\Phi(2)$.

```
> est <- replicate(1000, expr = {
+     u <- runif(m, 0, 2)
+     g <- 2 * exp(-(u^2)/2)
+     mean(g)/sqrt(2 * pi) + 0.5
+ })
> pnorm(2)
[1] 0.9772499
> c(mean(est), sd(est))
[1] 0.977209570 0.002317498
> mean(est) + qnorm(c(0.025, 0.975)) * sd(est)
[1] 0.9726674 0.9817518
```

5.3 *Compute a Monte Carlo estimate $\hat{\theta}$ of*

$$\theta = \int_0^{0.5} e^{-x}\, dx$$

by sampling from Uniform(0, 0.5), and estimate the variance of $\hat{\theta}$. Find another Monte Carlo estimator θ^* by sampling from the exponential distribution. Which of the variances (of $\hat{\theta}$ and $\hat{\theta}^*$) is smaller, and why?

[The exact value of the integral is $\theta = 1 - e^{-.5} \doteq 0.3934693$.]

The simple Monte Carlo estimator is

$$\hat{\theta} = (b-a)\int_a^b g(x)dx = \frac{1}{2}\left\{\frac{1}{m}\sum_{i=1}^{m} e^{-u}\right\},$$

where u is generated from Uniform$(0, \frac{1}{2})$.

```
> m <- 10000
> u <- runif(m, 0, 0.5)
> theta <- 0.5 * mean(exp(-u))
> theta
[1] 0.3924437
> est <- replicate(1000, expr = {
+     u <- runif(m, 0, 0.5)
+     theta <- 0.5 * mean(exp(-u))
+ })
> mean(est)
[1] 0.393469
```

```
> c(var(est), sd(est))
[1] 3.140757e-07 5.604246e-04
```

Let

$$\hat{\theta}^* = \frac{1}{m}\sum_{i=1}^{m} I(v < 0.5),$$

where v is generated from standard exponential distribution.

```
> m <- 10000
> v <- rexp(m, 1)
> theta <- mean(v <= 0.5)
> theta
[1] 0.3938
> est1 <- replicate(1000, expr = {
+     v <- rexp(m, 1)
+     theta <- mean(v <= 0.5)
+ })
> mean(est1)
[1] 0.3935925
> c(var(est1), sd(est1))
[1] 2.391485e-05 4.890281e-03
> var(est)/var(est1)
[1] 0.01313308
```

The simulation suggests that $Var(\hat{\theta}) < Var(\hat{\theta}^*)$. In this example we can compute the exact variance of the estimators for comparison.

$$Var(\hat{\theta}^*) = \frac{\theta(1-\theta)}{m} = (1-e^{-1/2})(e^{-1/2})/m \doteq 2.386512e-05.$$

The variance of $g(U)$ is

$$\begin{aligned} Var(e^{-U}) &= \int_0^{1/2} 2e^{-2u}du - \left[\int_0^{1/2} 2e^{-u}du\right]^2 \\ &= 1 - e^{-1} - 4(1-e^{-1/2})^2 \\ &= -e^{-1} - 1 - 4(1 - 2e^{-1/2} + e^{-1}) \\ &= 1 - e^{-1} - 4 + 8e^{-1/2} - 4e^{-1} \\ &= 8e^{-1/2} - 5e^{-1} - 3. \end{aligned}$$

The variance of $\hat{\theta}$ is

$$\frac{Var(\frac{1}{2}g(U))}{m} \doteq \frac{0.01284807}{4m} \doteq 3.212018e-07.$$

Then

$$\frac{Var(\hat{\theta})}{Var(\hat{\theta}^*)} = \frac{0.01284807/4}{(1-e^{-1/2})(e^{-1/2})} \doteq 0.01345905.$$

5.4 *Write a function to compute a Monte Carlo estimate of the Beta(3, 3) cdf, and use the function to estimate $F(x)$ for $x = 0.1, 0.2, \ldots, 0.9$. Compare the estimates with the values returned by the* `pbeta` *function in R.*

This solution uses the "ratio of gammas" Beta(a,b) generator from Chapter 3. The function `mcpBETA` is not set up to handle vector arguments, so all but the first element of the arguments are ignored.

```
> mcpBETA <- function(x, a, b, m = 10000) {
+     x <- x[1]
+     a <- a[1]
+     b <- b[1]
+     u <- rgamma(m, a, 1)
+     v <- rgamma(m, b, 1)
+     y <- u/(u + v)
+     return(mean(y <= x))
+ }
> x <- seq(0.1, 0.9, 0.1)
> k <- length(x)
> p <- numeric(k)
> for (i in 1:k) p[i] <- mcpBETA(x[i], 3, 3)
> round(rbind(x, pbeta(x, 3, 3), p), 3)
   [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
x 0.100 0.200 0.300 0.400 0.500 0.600 0.700 0.800 0.900
  0.009 0.058 0.163 0.317 0.500 0.683 0.837 0.942 0.991
p 0.008 0.059 0.166 0.318 0.506 0.680 0.841 0.943 0.992
```

5.5 *Compute (empirically) the efficiency of the sample mean Monte Carlo method of estimation of the definite integral $\Phi(x)$ relative to the "hit or miss" method.*

Method 1: Generate iid Uniform(0,1) random numbers $u_1, \ldots, u_m$, and compute

$$\hat{\theta}_1 = \overline{g_m(u)} = \frac{1}{m} \sum_{i=1}^{m} x\, e^{-(u_i x)^2/2}.$$

If $x > 0$, the estimate of $\Phi(x)$ is $0.5 + \hat{\theta}/\sqrt{2\pi}$. If $x < 0$ compute $\Phi(x) = 1 - \Phi(-x)$.

Method 2: Generate a random sample $z_1, \ldots, z_m$ from the standard normal distribution, and let

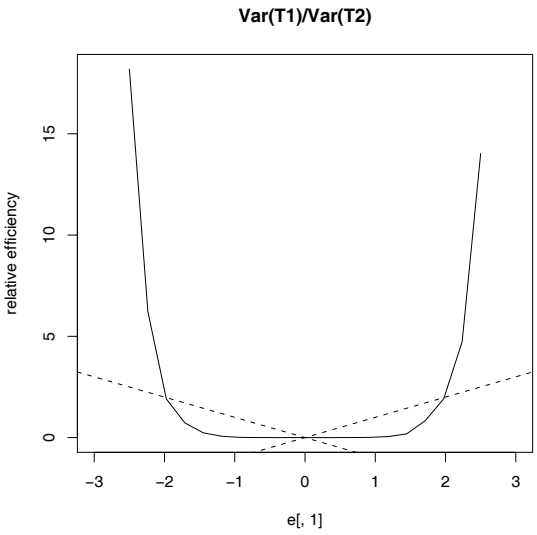
$$\hat{\theta}_2 = \frac{1}{m} \sum_{i=1}^{m} I(z_i \le x).$$

```
> k <- 20
> X <- seq(-2.5, 2.5, length = k)
> e <- matrix(1, nrow = k, ncol = 4)
> m <- 10000
> for (i in 1:k) {
+     x <- X[i]
+     theta1 <- replicate(200, expr = {
+         u <- runif(m)
+         g <- x * exp(-(u * x)^2/2)
+         theta1 <- mean(g)/sqrt(2 * pi) + 0.5
```

```
+     })
+     theta2 <- replicate(200, expr = {
+         z <- rnorm(m)
+         theta2 <- mean(z < x)
+     })
+     eff <- var(theta1)/var(theta2)
+     e[i, ] <- c(x, eff, var(theta1), var(theta2))
+ }
> e
```

```
            [,1]         [,2]         [,3]         [,4]
 [1,] -2.5000000 1.819067e+01 1.102752e-05 6.062188e-07
 [2,] -2.2368421 6.217543e+00 8.393307e-06 1.349939e-06
 [3,] -1.9736842 1.907010e+00 4.551715e-06 2.386833e-06
 [4,] -1.7105263 7.255836e-01 2.837183e-06 3.910209e-06
 [5,] -1.4473684 2.346970e-01 1.540794e-06 6.565033e-06
 [6,] -1.1842105 6.547021e-02 5.774102e-07 8.819435e-06
 [7,] -0.9210526 1.063258e-02 1.447418e-07 1.361304e-05
 [8,] -0.6578947 1.219160e-03 2.292732e-08 1.880583e-05
 [9,] -0.3947368 6.311327e-05 1.431083e-09 2.267483e-05
[10,] -0.1315789 6.074460e-08 1.792422e-12 2.950751e-05
[11,]  0.1315789 7.050335e-08 1.879855e-12 2.666335e-05
[12,]  0.3947368 5.116095e-05 1.101887e-09 2.153765e-05
[13,]  0.6578947 1.228239e-03 2.385682e-08 1.942359e-05
[14,]  0.9210526 9.981724e-03 1.557516e-07 1.560368e-05
[15,]  1.1842105 4.967473e-02 5.713941e-07 1.150271e-05
[16,]  1.4473684 1.845425e-01 1.315797e-06 7.130049e-06
[17,]  1.7105263 8.342219e-01 3.187566e-06 3.821005e-06
[18,]  1.9736842 1.914301e+00 4.839226e-06 2.527933e-06
[19,]  2.2368421 4.738075e+00 6.807637e-06 1.436794e-06
[20,]  2.5000000 1.402469e+01 9.643485e-06 6.876078e-07
```

```
> plot(e[, 1], e[, 2], type = "l", main = "Var(T1)/Var(T2)",
+     ylab = "relative efficiency", xlim = c(-3, 3))
> abline(0, 1, lty = 2)
> abline(0, -1, lty = 2)
```



5.6 *Consider the antithetic variate approach to estimating*

$$\theta = \int_0^1 e^x dx.$$

Compute $Cov(e^U, e^{1-U})$ *and* $Var(e^U + e^{1-U})$*, where* $U \sim$ *Uniform(0,1). What is the percent reduction in variance of* $\hat{\theta}$ *that can be achieved using antithetic variates (compared with simple MC)?*

$$\begin{aligned}
Cov(e^U, e^{1-U}) &= E[e^U e^{1-U}] - E[e^U]E[e^{(1-U}] \\
&= e - (e-1)^2 \doteq -0.2342106; \\
Var(e^U) &= E[e^{2U}] - (E[e^U])^2 = \frac{1}{2}(e^2-1) - (e-1)^2 \doteq 0.2420356; \\
Cor(e^U, e^{1-U}) &= \frac{Cov(e^U, e^{1-U})}{\sqrt{Var(e^U)}\sqrt{Var(e^{1-U})}} = \frac{e-(e-1)^2}{\frac{1}{2}(e^2-1)-(e-1)^2}.
\end{aligned}$$

(The variances of e^U and e^{1-U} are equal because U and $1-U$ are identically distributed.)

Suppose $\hat{\theta}_1$ is the simple MC estimator and $\hat{\theta}_2$ is the antithetic estimator. Then if U and V are iid Uniform (0,1) variables, we have

$$Var(\frac{1}{2}(e^U + e^V)) = \frac{1}{4}2Var(e^U) = \frac{1}{2} \cdot \frac{1}{2}(e^2 - 1 - (e-1)^2) \doteq 0.1210178.$$

If antithetic variables are used,

$$\begin{aligned}
Var(\frac{1}{2}(e^U + e^{1-U})) &= \frac{1}{4}(2Var(e^U) + 2Cov(e^U, e^{1-U})) \\
&= \frac{1}{2}\left(\frac{1}{2}(e^2-1) - (e-1)^2) + e - (e-1)^2\right) \\
&\doteq 0.003912497.
\end{aligned}$$

The reduction in variance is

$$\frac{Var(\hat{\theta}_1) - Var(\hat{\theta}_2)}{Var(\hat{\theta}_1)} = \frac{0.1210178 - 0.003912497}{0.1210178} = 0.96767,$$

or 96.767%.

5.7 *Refer to Exercise 5.6. Use a Monte Carlo simulation to estimate θ by the antithetic variate approach and by the simple Monte Carlo method. Compute an empirical estimate of the percent reduction in variance using the antithetic variate.*

```
> m <- 10000
> mc <- replicate(1000, expr = {
+     mean(exp(runif(m)))
+ })
> anti <- replicate(1000, expr = {
+     u <- runif(m/2)
+     v <- 1 - u
+     mean((exp(u) + exp(v))/2)
+ })
> v1 <- var(mc)
> v2 <- var(anti)
> c(mean(mc), mean(anti))
[1] 1.718507 1.718364
> c(v1, v2)
[1] 2.548294e-05 7.865248e-07
> (v1 - v2)/v1
[1] 0.9691352
```

In this simulation the reduction in variance printed on the last line above is close to the theoretical value 0.96767 from Exercise 5.6.

5.8 *Let $U \sim$ Uniform(0,1), $X = aU$, and $X' = a(1-U)$, where a is a constant. Show that $\rho(X, X') = -1$. Is $\rho(X, X') = -1$ if U is a symmetric beta random variable?*

Let $\mu = E[U]$ and $\sigma^2 = Var(U)$. Then

$$\begin{aligned} VarX &= Var(aU) = a^2\sigma^2; \\ VarX' &= Var(a(1-U)) = a^2 Var(-U) = a^2\sigma^2; \\ Cov(X, X') &= Cov(aU, a(1-U)) = a^2 Cov(U, 1-U) \\ &= a^2(E[U(1-U)] - E[U]E[1-U]) \\ &= a^2(\mu - (\sigma^2 + \mu^2) - \mu(1-\mu)) = -a^2\sigma^2. \end{aligned}$$

Therefore, for all U such that $\sigma^2 = Var(U) < \infty$ and in particular for the symmetric beta distributions we have

$$\rho(X, X') = \frac{Cov(aU, a(1-U))}{\sqrt{Var(aU)Var(a(1-U))}} = \frac{-a^2\sigma^2}{\sqrt{a^2\sigma^2}\sqrt{a^2\sigma^2}} = -1.$$

5.9 *The Rayleigh density is*

$$f(x) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)}, \qquad x \geq 0,\ \sigma > 0.$$

Implement a function to generate samples from a Rayleigh(σ) distribution, using antithetic variables. What is the percent reduction in variance of $\frac{X+X'}{2}$ compared with $\frac{X_1+X_2}{2}$ for independent X_1, X_2?

Here $F(x) = 1 - \exp(-x^2/(2\sigma^2)$, $x > 0$ and

$$u = 1 - e^{-x^2/(2\sigma^2)} \Rightarrow F^{-1}(u) = \sigma(-2\log(1-u))^{1/2}.$$

```
> Ray1 <- function(n, sigma) {
+     u <- runif(n)
+     return(sigma * sqrt(-2 * log(u)))
+ }
> Ray2 <- function(n, sigma) {
+     u <- runif(n/2)
+     x1 <- sigma * sqrt(-2 * log(u))
+     x2 <- sigma * sqrt(-2 * log(1 - u))
+     return(c(x1, x2))
+ }
> m <- 10000
> sigma <- 2
> r1 <- replicate(1000, mean(Ray1(2, sigma)))
> r2 <- replicate(1000, mean(Ray2(2, sigma)))
> var(r1)
[1] 0.8416862
> var(r2)
[1] 0.05339187
```

The approximate percent reduction in variance is

```
> 100 * (var(r1) - var(r2))/var(r1)
[1] 93.65656
```

5.10 Use Monte Carlo integration with antithetic variables to estimate

$$\int_0^1 \frac{e^{-x}}{1+x^2}dx,$$

and find the approximate reduction in variance as a percentage of the variance without variance reduction.

```
> m <- 10000
> mc <- replicate(1000, expr = {
+     u <- runif(m)
+     mean(exp(-u)/(1 + u^2))
+ })
> anti <- replicate(1000, expr = {
+     u <- runif(m/2)
+     x1 <- exp(-u)/(1 + u^2)
+     x2 <- exp(-(1 - u))/(1 + (1 - u)^2)
+     mean(c(x1, x2))
+ })
> var(mc)
[1] 6.059822e-06
```

```
> var(anti)
[1] 2.145301e-07
> mean(mc)
[1] 0.5248207
> mean(anti)
[1] 0.5247966
> 100 * (var(mc) - var(anti))/var(mc)
[1] 96.4598
```

5.11 *If $\hat{\theta}_1$ and $\hat{\theta}_2$ are unbiased estimators of θ, and $\hat{\theta}_1$ and $\hat{\theta}_2$ are antithetic, we derived that $c^* = 1/2$ is the optimal constant that minimizes the variance of $\hat{\theta}_c = c\hat{\theta}_2 + (1-c)\hat{\theta}_2$. Derive c^* for the general case. That is, if $\hat{\theta}_1$ and $\hat{\theta}_2$ are any two unbiased estimators of θ, find the value c^* that minimizes the variance of the estimator $\hat{\theta}_c = c\hat{\theta}_2 + (1-c)\hat{\theta}_2$.*

Suppose that $\hat{\theta}_1$ and $\hat{\theta}_2$ are any two unbiased estimators of θ. Then for every constant c,

$$\hat{\theta}_c = c\hat{\theta}_1 + (1-c)\hat{\theta}_2$$

is also unbiased for θ. The variance of $c\hat{\theta}_1 + (1-c)\hat{\theta}_2$ is

$$Var(\hat{\theta}_2) + c^2 Var(\hat{\theta}_1 - \hat{\theta}_2) + 2c\, Cov(\hat{\theta}_2, \hat{\theta}_1 - \hat{\theta}_2). \tag{1}$$

In the special case of antithetic variables, the estimators $\hat{\theta}_1$ and $\hat{\theta}_2$ are identically distributed and $Cov(\hat{\theta}_1, \hat{\theta}_2) = -1$. Hence, the variance in (1) is

$$Var\hat{\theta}_c = 2c^2 Var(\hat{\theta}_1) - 2cVar(\hat{\theta}_1) + Var(\hat{\theta}_1),$$

and the optimal constant is $c^* = 1/2$. In the general case, (1) is a quadratic function of c, and the value of c that minimizes the variance is

$$\begin{aligned} c^* &= -\frac{Cov(\hat{\theta}_2, \hat{\theta}_1 - \hat{\theta}_2)}{Var(\hat{\theta}_1 - \hat{\theta}_2)} = -\frac{E[\hat{\theta}_2(\hat{\theta}_1 - \hat{\theta}_2)] - E[\hat{\theta}_2]E[\hat{\theta}_1 - \hat{\theta}_2]}{Var(\hat{\theta}_1) + Var(\hat{\theta}_2) - 2\,Cov(\hat{\theta}_1, \hat{\theta}_2)} \\ &= -\frac{E[\hat{\theta}_2\hat{\theta}_1] - E[\hat{\theta}_2^2] - E[\hat{\theta}_2]E[\hat{\theta}_1] + E[\hat{\theta}_2]^2}{Var(\hat{\theta}_1) + Var(\hat{\theta}_2) - 2\,Cov(\hat{\theta}_1, \hat{\theta}_2)} \\ &= \frac{Var(\hat{\theta}_2) - Cov(\hat{\theta}_1, \hat{\theta}_2)}{Var(\hat{\theta}_1) + Var(\hat{\theta}_2) - 2\,Cov(\hat{\theta}_1, \hat{\theta}_2)}. \end{aligned}$$

5.12 *Let $\hat{\theta}_f^{IS}$ be an importance sampling estimator of $\theta = \int g(x)dx$, where the importance function f is a density. Prove that if $g(x)/f(x)$ is bounded, then the variance of the importance sampling estimator $\hat{\theta}_f^{IS}$ is finite.*

Suppose that f is a density, $\theta = \int g(x)dx < \infty$, and $\left|\frac{g(x)}{f(x)}\right| \leq M < \infty$. Let $\hat{\theta} = \hat{\theta}_f^{IS}$. Then

$$\begin{aligned} Var\hat{\theta} = E[\hat{\theta}^2] - (E[\hat{\theta}])^2 &= E\left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{g(X_i)}{f(X_i)}\right)^2 f(X_i)\right] - \theta^2 \\ &= \int \frac{g(x)^2}{f(x)}\,dx - \theta^2 = \int \frac{g(x)}{f(x)} g(x)\,dx - \theta^2 \\ &\leq M\int g(x)dx - \theta^2 = M\theta - \theta^2 < \infty. \end{aligned}$$

5.13 *Find two importance functions f_1 and f_2 that are supported on $(1, \infty)$ and are 'close' to*

$$g(x) = \frac{x^2}{\sqrt{2\pi}}\, e^{-x^2/2}, \qquad x > 1.$$

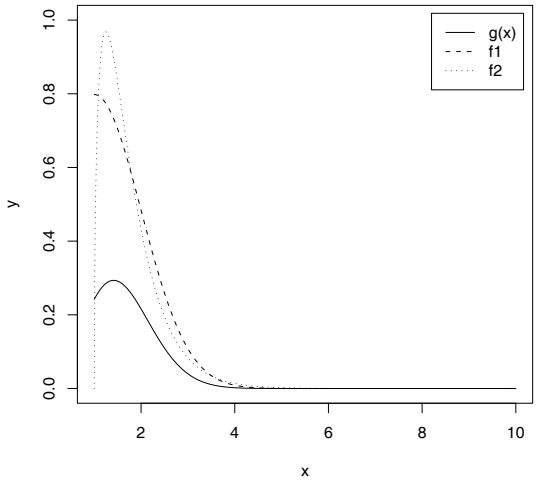
Which of your two importance functions should produce the smaller variance in estimating

$$\int_1^\infty \frac{x^2}{\sqrt{2\pi}}\, e^{-x^2/2}\, dx$$

by importance sampling? Explain.

First display the graph of $g(x)$. From the graph, we might consider a normal distribution or a gamma distribution.

```
> x <- seq(1, 10, 0.01)
> y <- x^2 * exp(-x^2/2)/sqrt(2 * pi)
> plot(x, y, type = "l", ylim = c(0, 1))
> lines(x, 2 * dnorm(x, 1), lty = 2)
> lines(x, dgamma(x - 1, 3/2, 2), lty = 3)
> legend("topright", inset = 0.02, legend = c("g(x)", "f1",
+     "f2"), lty = 1:3)
```



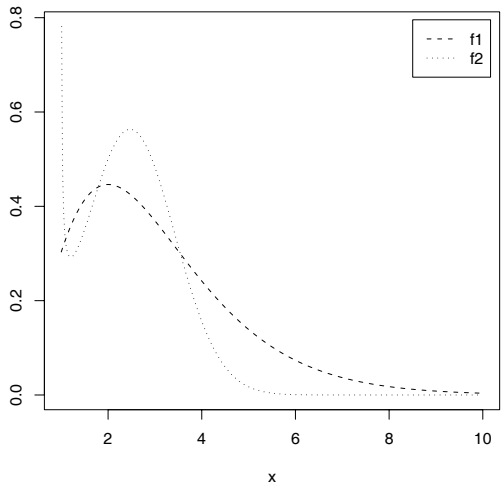
Here f_1 is a $\chi(1)$ variable that is translated to $x > 1$. Thus f_1 is twice the $N(1, 1)$ density. The gamma variable is also translated to $x > 1$.

Many other importance functions could be considered. These two satisfy the conditions that the support set is $(1, \infty)$.

Compare the ratios $g(x)/f(x)$.

From the plot, we might expect the folded normal importance function to produce the smaller variance in estimating the integral, because the ratio $g(x)/f(x)$ is closer to a constant function.

```
> plot(x, y/(dgamma(x - 1, 3/2, 2)), type = "l", lty = 3,
+     ylab = "")
> lines(x, y/(2 * dnorm(x, 1)), lty = 2)
> legend("topright", inset = 0.02, legend = c("f1", "f2"),
+     lty = 2:3)
```



5.14 Obtain a Monte Carlo estimate of

$$\int_1^\infty \frac{x^2}{\sqrt{2\pi}} e^{-x^2/2}\, dx$$

by importance sampling.

```
> m <- 10000
> is1 <- replicate(1000, expr = {
+     x <- sqrt(rchisq(m, 1)) + 1
+     f <- 2 * dnorm(x, 1)
+     g <- x^2 * exp(-x^2/2)/sqrt(2 * pi)
+     mean(g/f)
+ })
> is2 <- replicate(1000, expr = {
+     x <- rgamma(m, 3/2, 2) + 1
+     f <- dgamma(x - 1, 3/2, 2)
+     g <- x^2 * exp(-x^2/2)/sqrt(2 * pi)
```

```
+     mean(g/f)
+ })
> c(mean(is1), mean(is2))
[1] 0.4006183 0.4006317
> c(var(is1), var(is2))
[1] 2.044402e-07 1.054996e-06
> var(is1)/var(is2)
[1] 0.1937829
```

Clearly importance function (1), a shifted Normal(1,1) (folded at x=1) produces the more efficient estimator.

For comparison, we can check the numerical integration result.

```
> g <- function(x) x^2 * exp(-x^2/2)/sqrt(2 * pi)
> integrate(g, lower = 1, upper = Inf)
0.400626 with absolute error < 5.7e-07
```

5.15 Obtain the stratified importance sampling estimate of

$$\theta = \int_0^1 \frac{e^{-x}}{1+x^2}$$

with importance function

$$f(x) = \frac{e^{-x}}{1-e^{-1}}, \qquad 0 < x < 1,$$

on five subintervals, $(j/5, (j+1)/5)$, $j = 0, 1, \ldots, 4$. On the j^{th} subinterval

$$f_j(x) = f_{x|I_j}(x) = \frac{5e^{-x}}{1-e^{-1}}, \qquad \frac{j-1}{5} < x < \frac{j}{5}.$$

```
> M <- 10000
> k <- 5
> m <- M/k
> si <- numeric(k)
> v <- numeric(k)
> g <- function(x) exp(-x)/(1 + x^2)
> f <- function(x) (k/(1 - exp(-1))) * exp(-x)
> for (j in 1:k) {
+     u <- runif(m, (j - 1)/k, j/k)
+     x <- -log(1 - (1 - exp(-1)) * u)
+     fg <- g(x)/f(x)
+     si[j] <- mean(fg)
+     v[j] <- var(fg)
+ }
> sum(si)
[1] 0.5244961
> mean(v)
[1] 1.77338e-05
> sqrt(mean(v))
[1] 0.004211152
```

Without stratification we have a larger variance:

```
> k <- 1
> m <- M/k
> si <- numeric(k)
> v <- numeric(k)
> for (j in 1:k) {
+     u <- runif(m, (j - 1)/k, j/k)
+     x <- -log(1 - (1 - exp(-1)) * u)
+     fg <- g(x)/f(x)
+     si[j] <- mean(fg)
+     v[j] <- var(fg)
+ }
> sum(si)
[1] 0.5242244
> mean(v)
[1] 0.009384273
> sqrt(mean(v))
[1] 0.09687246
```

CHAPTER 6

Monte Carlo Methods in Inference

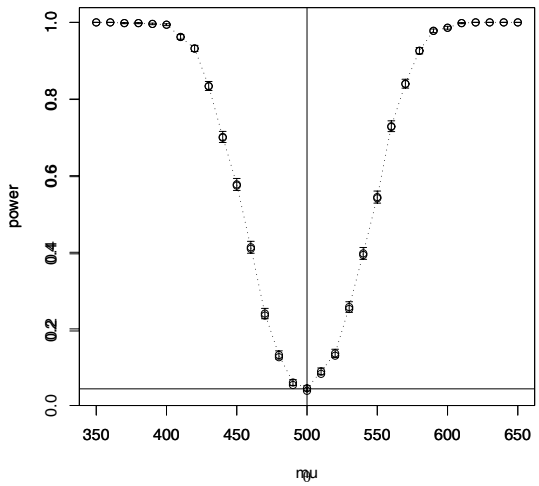
6.1 *Estimate the MSE of the level k trimmed means for random samples of size 20 generated from a standard Cauchy distribution. (The target parameter θ is the center or median; the expected value does not exist.) Summarize the estimates of MSE in a table for $k = 1, 2, \ldots, 9$.*

```
> n <- 20
> K <- n/2 - 1
> m <- 1000
> mse <- matrix(0, n/2, 2)
> trimmed.mse <- function(n, m, k) {
+     tmean <- numeric(m)
+     for (i in 1:m) {
+         x <- sort(rcauchy(n))
+         tmean[i] <- sum(x[(k + 1):(n - k)])/(n - 2 * k)
+     }
+     mse.est <- mean(tmean^2)
+     se.mse <- sqrt(mean((tmean - mean(tmean))^2))/sqrt(m)
+     return(c(mse.est, se.mse))
+ }
> for (k in 0:K) mse[k + 1, 1:2] <- trimmed.mse(n = n, m = m,
+     k = k)
> mse <- as.data.frame(cbind(0:K, mse))
> names(mse) <- list("k", "t-mean", "se")
> print(mse)
   k     t-mean         se
1  0 82.6578971 0.28749176
2  1  1.3422777 0.03663287
3  2  0.3478463 0.01865002
4  3  0.2516950 0.01584669
5  4  0.1708088 0.01305890
6  5  0.1503993 0.01226187
7  6  0.1441091 0.01199713
8  7  0.1352604 0.01161845
9  8  0.1352857 0.01162528
10 9  0.1400206 0.01183221
```

6.2 *Plot the empirical power curve for the t-test, changing the alternative hypothesis to $H_1 : \mu \neq 500$, and keeping the significance level $\alpha = 0.05$.*

```
> n <- 20
> m <- 1000
```

```
> mu0 <- 500
> sigma <- 100
> mu <- c(seq(350, 650, 10))
> M <- length(mu)
> power <- numeric(M)
> for (i in 1:M) {
+     mu1 <- mu[i]
+     pvalues <- replicate(m, expr = {
+         x <- rnorm(n, mean = mu1, sd = sigma)
+         ttest <- t.test(x, alternative = "two.sided", mu = mu0)
+         ttest$p.value
+     })
+     power[i] <- mean(pvalues <= 0.05)
+ }

> library(Hmisc, warn.conflict = FALSE)
> plot(mu, power)
> abline(v = mu0, lty = 1)
> abline(h = 0.05, lty = 1)
> se <- sqrt(power * (1 - power)/m)
> errbar(mu, power, yplus = power + se, yminus = power -
+     se, xlab = bquote(theta))
> lines(mu, power, lty = 3)
> detach(package:Hmisc)
```

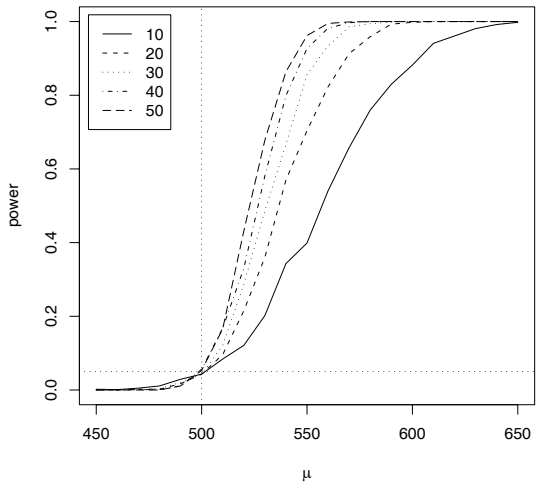


Note: If the error bars do not appear on the printed plot, they should be visible on the screen (an Sweave issue).

6.3 *Plot the power curves for the one-sided t-test for sample sizes 10, 20, 30, 40, and 50, but omit the standard error bars. Plot the curves on the same graph, each in a different color or different line type, and include a legend.*

```
> N <- c(10, 20, 30, 40, 50)
> m <- 1000
> mu0 <- 500
> sigma <- 100
> mu <- c(seq(450, 650, 10))
> M <- length(mu)
> power <- matrix(0, M, 5)
> for (j in 1:5) {
+     n <- N[j]
+     for (i in 1:M) {
+         mu1 <- mu[i]
+         pvalues <- replicate(m, expr = {
+             x <- rnorm(n, mean = mu1, sd = sigma)
+             ttest <- t.test(x, alternative = "greater",
+                 mu = mu0)
+             ttest$p.value
+         })
+         power[i, j] <- mean(pvalues <= 0.05)
+     }
+ }
```

```
> plot(mu, power[, 1], type = "l", ylim = range(power), xlab = bquote(mu),
+     ylab = "power")
> abline(v = mu0, lty = 3)
> abline(h = 0.05, lty = 3)
> for (j in 2:5) lines(mu, power[, j], lty = j)
> legend("topleft", inset = 0.02, legend = N, lty = 1:5)
```



Replace `lty=j` with `col=j` in `lines` and in `legend` for color. The plots show that for a fixed alternative, the power is increasing with sample size.

6.4 *Suppose that $X_1, \ldots, X_n$ are a random sample from a from a lognormal distribution with unknown parameters. Construct a 95% confidence interval for the parameter μ. Use a Monte Carlo method to obtain an empirical estimate of the confidence level.*

Transform X to normal and estimate μ with the sample mean of the transformed sample.

```
> n <- 30
> CI <- replicate(10000, expr = {
+     x <- rlnorm(n)
+     y <- log(x)
+     ybar <- mean(y)
+     se <- sd(y)/sqrt(n)
+     ybar + se * qnorm(c(0.025, 0.975))
+ })
> LCL <- CI[1, ]
> UCL <- CI[2, ]
> sum(LCL < 0 & UCL > 0)
[1] 9380
> mean(LCL < 0 & UCL > 0)
[1] 0.938
```

6.5 *Use a Monte Carlo experiment to estimate the coverage probability of the t-interval for random samples of $\chi^2(2)$ data with sample size $n = 20$.*

```
> n <- 20
> rootn <- sqrt(n)
> t0 <- qt(c(0.025, 0.975), df = n - 1)
> CI <- replicate(10000, expr = {
+     x <- rchisq(n, df = 2)
+     ci <- mean(x) + t0 * sd(x)/rootn
+ })
> LCL <- CI[1, ]
> UCL <- CI[2, ]
> sum(LCL < 2 & UCL > 2)
[1] 9167
> mean(LCL < 2 & UCL > 2)
[1] 0.9167
```

The t-interval is more robust to departures from normality than the interval for variance. For the $\chi^2(2)$ distribution the empirical coverage rate was only 77.3%.

6.6 *Estimate the 0.025, 0.05, 0.95 and 0.975 quantiles of the skewness $\sqrt{b_1}$ under normality by a Monte Carlo experiment. Compute the standard error of the estimates using the normal approximation for the density (with exact variance formula). Compare the estimated quantiles with the quantiles of the large sample approximation $\sqrt{b_1} \approx N(0, 6/n)$.*

Equation (2.14) gives the variance of a sample quantile:

$$Var(\hat{x}_q) = \frac{q(1-q)}{nf(x_q)^2}.$$

Here the density f is the density of the skewness statistic, and the value of n is the number of replicates of the statistic. To estimate $se(\hat{x}_q)$, we are approximating f with the asymptotic normal density.

```
> sk <- function(x) {
+     xbar <- mean(x)
+     m3 <- mean((x - xbar)^3)
+     m2 <- mean((x - xbar)^2)
+     return(m3/m2^1.5)
+ }
> m <- 10000
> n <- 50
> skstats <- replicate(m, expr = {
+     x <- rnorm(n)
+     sk(x)
+ })
> p <- c(0.025, 0.05, 0.95, 0.975)
> q1 <- quantile(skstats, p)
> q2 <- qnorm(p, 0, sqrt(6 * (n - 2)/((n + 1) * (n + 3))))
> q3 <- qnorm(p, 0, sqrt(6/n))
> f <- dnorm(q2, 0, sqrt(6 * (n - 2)/((n + 1) * (n + 3))))
> v <- p * (1 - p)/(m * f^2)
> rbind(p, q1, sqrt(v))
           2.5%          5%        95%       97.5%
p   0.025000000  0.05000000 0.95000000 0.975000000
q1 -0.640460596 -0.53554723 0.54001617 0.638831577
    0.008719622  0.00689781 0.00689781 0.008719622
> rbind(q1, q2, q3)
         2.5%         5%       95%     97.5%
q1 -0.6404606 -0.5355472 0.5400162 0.6388316
q2 -0.6397662 -0.5369087 0.5369087 0.6397662
q3 -0.6789514 -0.5697940 0.5697940 0.6789514
```

The first table shows the sample quantiles of the skewness statistic and standard error of the estimate for sample size 50.

The second table shows the three estimates of quantiles q_1 (sample) q_2 (normal with exact variance) and q_3 normal asymptotic distribution. The estimated quantiles q_2 are closer to the empirical quantiles than the estimates q_3 using the asymptotic variance $6/n$.

6.7 *Estimate the power of the skewness test of normality against symmetric Beta(α, α) distributions and comment on the results.*

```
> alpha <- 0.1
> n <- 30
> m <- 2500
> ab <- 1:10
> N <- length(ab)
> pwr <- numeric(N)
> cv <- qnorm(1 - alpha/2, 0, sqrt(6 * (n - 2)/((n + 1) *
```

```
+     (n + 3))))
> for (j in 1:N) {
+     a <- ab[j]
+     sktests <- numeric(m)
+     for (i in 1:m) {
+         x <- rbeta(n, a, a)
+         sktests[i] <- as.integer(abs(sk(x)) >= cv)
+     }
+     pwr[j] <- mean(sktests)
+ }
> pwr
 [1] 0.0188 0.0120 0.0228 0.0284 0.0444 0.0516 0.0480 0.0548 0.0468
[10] 0.0724
```

The symmetric beta alternatives are not normal, but beta is symmetric. This simulation illustrates that the skewness test of normality is not very effective against light-tailed symmetric alternatives. The empirical power of the test is not higher than the nominal significance level.

Are the results different for heavy-tailed symmetric alternatives such as $t(\nu)$? Yes, the skewness test is more effective against a heavy-tailed symmetric alternative, such as a Student t distribution. Below we repeat the simulation for several choices of degrees of freedom.

```
> alpha <- 0.1
> n <- 30
> m <- 2500
> df <- c(1:5, seq(10, 50, 10))
> N <- length(df)
> pwr <- numeric(N)
> cv <- qnorm(1 - alpha/2, 0, sqrt(6 * (n - 2)/((n + 1) *
+     (n + 3))))
> for (j in 1:N) {
+     nu <- df[j]
+     sktests <- numeric(m)
+     for (i in 1:m) {
+         x <- rt(n, df = nu)
+         sktests[i] <- as.integer(abs(sk(x)) >= cv)
+     }
+     pwr[j] <- mean(sktests)
+ }
> data.frame(df, pwr)
   df    pwr
1   1 0.8700
2   2 0.6448
3   3 0.5232
4   4 0.4116
5   5 0.3392
6  10 0.1980
7  20 0.1476
```

```
8  30 0.1372
9  40 0.1120
10 50 0.1092
```

The skewness test of normality is more powerful when the degrees of freedom are small. As degrees of freedom tend to infinity the t distribution tends to normal, and the power tends to α. One reason that the skewness test is more powerful in this case than against the symmetric beta distributions is that $|\sqrt{b_1}|$ is positively correlated with kurtosis. Kurtosis of beta distribution is less than the normal kurtosis, while kurtosis of t is greater than the normal kurtosis.

6.8 *Repeat the Count Five test power simulation, but also compute the F test of equal variance, at significance level $\hat{\alpha} \doteq 0.055$. Compare the power of the Count Five test and F test for small, medium, and large sample sizes.*

```
> count5test <- function(x, y) {
+     X <- x - mean(x)
+     Y <- y - mean(y)
+     outx <- sum(X > max(Y)) + sum(X < min(Y))
+     outy <- sum(Y > max(X)) + sum(Y < min(X))
+     return(as.integer(max(c(outx, outy)) > 5))
+ }
> sigma1 <- 1
> sigma2 <- 1.5
> m <- 10000
> for (n in c(20, 30, 50, 100, 200, 500)) {
+     tests <- replicate(m, expr = {
+         x <- rnorm(n, 0, sigma1)
+         y <- rnorm(n, 0, sigma2)
+         C5 <- count5test(x, y)
+         Fp <- var.test(x, y)$p.value
+         Ftest <- as.integer(Fp <= 0.055)
+         c(C5, Ftest)
+     })
+     cat(n, rowMeans(tests), "\n")
+ }
20 0.3089 0.4137
30 0.4695 0.5854
50 0.6554 0.8071
100 0.8474 0.9801
200 0.9493 1
500 0.9905 1
```

The simulation results suggest that the F-test for equal variance is more powerful in this case, for all sample sizes compared ($se \leq 0.005$ and $se \approxeq .002$ when $\hat{p}$ is close to 1).

6.9 *Let X be a non-negative random variable with $\mu = E[X] < \infty$. For a random sample $x_1, \ldots, x_n$ from the distribution of X, the Gini ratio is defined by*

$$G = \frac{1}{2n^2\mu} \sum_{j=1}^{n} \sum_{i=1}^{n} |x_i - x_j| = \frac{1}{n^2\mu} \sum_{i=1}^{n} (2i - n - 1) x_{(i)}.$$

Estimate by simulation the mean, median and deciles of $\hat{G}$ if X is standard lognormal. Repeat the procedure for the uniform distribution and Bernoulli(0.1). Also construct density histograms of the replicates in each case.

Occasionally, Bernoulli(0.1) samples may produce a zero mean. In that case, the Gini ratio will return NaN.

```
> G <- function(x) {
+     n <- length(x)
+     x <- sort(x)
+     k <- 2 * (1:n) - n - 1
+     g <- sum(k * x)/(n^2 * mean(x))
+     return(g)
+ }
> n <- 100
> m <- 10000
> g1 <- replicate(m, G(rlnorm(n)))
> g2 <- replicate(m, G(runif(n)))
> g3 <- replicate(m, G(rbinom(n, size = 1, 0.5)))
> g4 <- replicate(m, G(rbinom(n, size = 1, 0.1)))
> p <- seq(0.1, 0.9, 0.1)
> g <- data.frame(g1, g2, g3, g4)
```

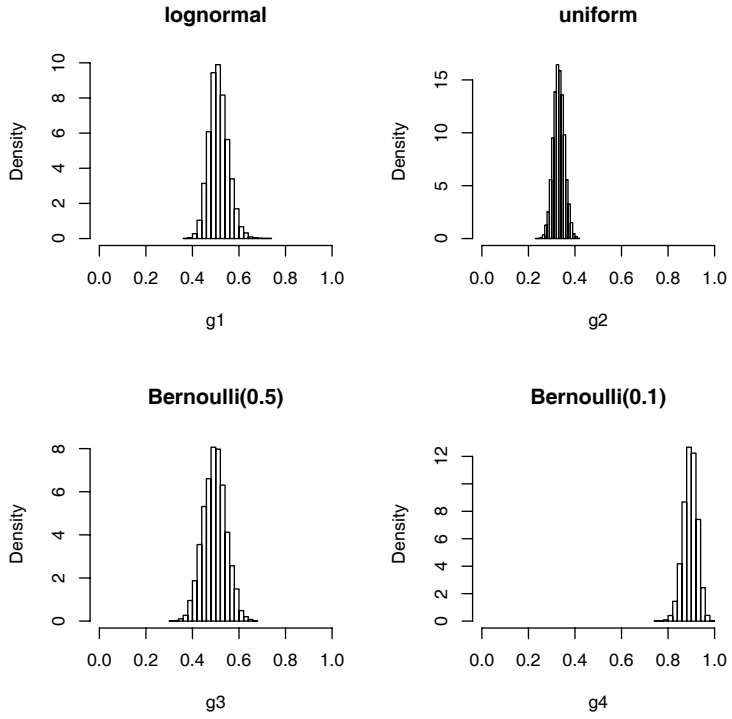
The mean and median are given in the summary below, and the deciles are computed by the quantile function.

```
> summary(g)
       g1              g2              g3              g4
 Min.   :0.3734   Min.   :0.2351   Min.   :0.3000   Min.   :0.75
 1st Qu.:0.4848   1st Qu.:0.3143   1st Qu.:0.4700   1st Qu.:0.88
 Median :0.5096   Median :0.3301   Median :0.5000   Median :0.90
 Mean   :0.5126   Mean   :0.3306   Mean   :0.5001   Mean   :0.90
 3rd Qu.:0.5384   3rd Qu.:0.3468   3rd Qu.:0.5300   3rd Qu.:0.92
 Max.   :0.7296   Max.   :0.4193   Max.   :0.6800   Max.   :0.99
> apply(g, MARGIN = 2, quantile, prob = p)
           g1        g2   g3   g4
10% 0.4618236 0.3002079 0.44 0.86
20% 0.4784673 0.3104826 0.46 0.88
30% 0.4904896 0.3177272 0.47 0.89
40% 0.4999647 0.3244480 0.49 0.89
50% 0.5096416 0.3301368 0.50 0.90
60% 0.5201558 0.3362826 0.51 0.91
70% 0.5318989 0.3431508 0.53 0.92
80% 0.5462120 0.3508530 0.54 0.93
90% 0.5659326 0.3615012 0.56 0.94
```

G measures inequality in income distribution, so the uniform distribution has the smallest mean. The most extreme disparity among these four is the Bernoulli(0.1) example, corresponding to large G.

```
> par(mfrow = c(2, 2))
> hist(g1, prob = TRUE, main = "lognormal", xlim = c(0, 1))
```

```
> hist(g2, prob = TRUE, main = "uniform", xlim = c(0, 1))
> hist(g3, prob = TRUE, main = "Bernoulli(0.5)", xlim = c(0,
+     1))
> hist(g4, prob = TRUE, main = "Bernoulli(0.1)", xlim = c(0,
+     1))
> par(mfrow = c(1, 1))
```



6.10 *Construct an approximate 95% confidence interval for the Gini ratio $\gamma = E[G]$ if X is lognormal with unknown parameters. Assess the coverage rate of the estimation procedure with a Monte Carlo experiment.*

The confidence intervals below are computed for the standard lognormal distribution at 90% confidence for sample size $n = 100$, based on the asymptotic normal distribution of the MLE.

```
> n <- 100
> m <- 10000
> sigma <- 1
> alpha <- 0.1
> g0 <- 2 * pnorm(sigma/sqrt(2)) - 1
> g0
[1] 0.5204999
> ci <- matrix(0, m, 3)
> vhat <- replicate(m, expr = {
+     x <- rlnorm(n, 0, sigma)
+     mean((log(x) - mean(log(x)))^2)
```

```
+ })
> ci[, 1] <- 2 * pnorm(sqrt(vhat)/sqrt(2)) - 1
> s <- sd(ci[, 1])
> ci[, 2] <- ci[, 1] + qnorm(alpha/2) * s
> ci[, 3] <- ci[, 1] + qnorm(1 - alpha/2) * s
> j <- sum(ci[, 2] < g0 & ci[, 3] > g0)
> cat("confidence level ", 100 * (1 - alpha), " coverage rate ",
+     j/m, "\n")
confidence level  90  coverage rate  0.8971
> cat("misses low ", sum(ci[, 3] < g0)/m, "\n")
misses low  0.0626
> cat("misses high ", sum(ci[, 2] > g0)/m, "\n")
misses high  0.0403
```

The confidence intervals based on the normal approximation of the distribution of the MLE have approximately the correct coverage rate.

CHAPTER 7

Bootstrap and Jackknife

7.1 *Compute a jackknife estimate of the bias and the standard error of the correlation statistic for the* `law` *data.*

```
> library(bootstrap)
> attach(law)
> n <- nrow(law)
> theta.hat <- cor(LSAT, GPA)
> theta.jack <- numeric(n)
> for (i in 1:n) theta.jack[i] <- cor(LSAT[-i], GPA[-i])
> bias <- (n - 1) * (mean(theta.jack) - theta.hat)
> se <- sqrt((n - 1) * mean((theta.jack - mean(theta.jack))^2))
> detach(law)
> detach(package:bootstrap)
> print(list(est = theta.hat, bias = bias, se = se))
$est
[1] 0.7763745

$bias
[1] -0.006473623

$se
[1] 0.1425186
```

7.2 *Refer to the* `law` *data* (`bootstrap`). *Use the jackknife-after-bootstrap method to estimate the standard error of the bootstrap estimate of* $se(R)$.

```
> library(bootstrap)
> attach(law)
> n <- nrow(law)
> B <- 2000
> theta.b <- numeric(B)
> indices <- matrix(0, nrow = B, ncol = n)
> for (b in 1:B) {
+     i <- sample(1:n, size = n, replace = TRUE)
+     x <- LSAT[i]
+     y <- GPA[i]
+     theta.b[b] <- cor(x, y)
+     indices[b, ] <- i
+ }
> se.jack <- numeric(n)
> for (i in 1:n) {
```

```
+     keep <- (1:B)[apply(indices, MARGIN = 1, FUN = function(k) {
+         !any(k == i)
+     })]
+     se.jack[i] <- sd(theta.b[keep])
+ }
> detach(law)
> detach(package:bootstrap)
> print(list(sd = sd(theta.b), se = (sqrt((n - 1) * mean((se.jack -
+     mean(se.jack))^2)))))
$sd
[1] 0.1373016

$se
[1] 0.08788263
```

7.3 *Obtain a bootstrap t confidence interval estimate for the correlation statistic (`law` data in `bootstrap`).*

Two methods are shown below. The bootstrap t CI can be obtained using the `boot.t.ci` function given in Chapter 7, or by using `boot.ci` after `boot`.

To use `boot.t.ci`, provide a function that computes the statistic of interest. Although `cor` is available, it returns a correlation matrix when the argument is a data matrix. Here we need the correlation statistic.

```
> library(boot)
> library(bootstrap)
> attach(law)
> cor.stat <- function(x, i = 1:NROW(x)) {
+     cor(x[i, 1], x[i, 2])
+ }
> boot.t.ci <- function(x, B = 500, R = 100, level = 0.95,
+     statistic) {
+     x <- as.matrix(x)
+     n <- nrow(x)
+     stat <- numeric(B)
+     se <- numeric(B)
+     boot.se <- function(x, R, f) {
+         x <- as.matrix(x)
+         m <- nrow(x)
+         th <- replicate(R, expr = {
+             i <- sample(1:m, size = m, replace = TRUE)
+             f(x[i, ])
+         })
+         return(sd(th))
+     }
+     for (b in 1:B) {
+         j <- sample(1:n, size = n, replace = TRUE)
+         y <- x[j, ]
+         stat[b] <- statistic(y)
+         se[b] <- boot.se(y, R = R, f = statistic)
```

```
+     }
+     stat0 <- statistic(x)
+     t.stats <- (stat - stat0)/se
+     se0 <- sd(stat)
+     alpha <- 1 - level
+     Qt <- quantile(t.stats, c(alpha/2, 1 - alpha/2),
+         type = 1)
+     names(Qt) <- rev(names(Qt))
+     CI <- rev(stat0 - Qt * se0)
+ }
> print(boot.t.ci(law, B = 1000, R = 25, statistic = cor.stat))
      2.5%      97.5%
-0.1556726  0.9976787
```

To use `boot.ci` after `boot`, write a function that returns both the correlation statistic and an estimate of the variance for each bootstrap sample (see `cor.stat2` below). Then with `boot.ci` and `type="stud"` the variances for the studentized statistics are by default in the second position of the returned bootstrap object.

```
> cor.stat2 <- function(x, i = 1:NROW(x)) {
+     o <- boot(x[i, ], cor.stat, R = 25)
+     n <- length(i)
+     c(o$t0, var(o$t) * (n - 1)/n^2)
+ }
> b <- boot(law, statistic = cor.stat2, R = 1000)
> boot.ci(b, type = "stud")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = b, type = "stud")

Intervals :
Level    Studentized
95%   (-0.2466,  0.9469 )
Calculations and Intervals on Original Scale
```

The confidence interval estimation is repeated below to show that the bootstrap t intervals are unstable in this example. See Efron and Tibshirani 1993 for an explanation and a better approach (transform R to normal). The confidence limits are the last two numbers.

```
> b <- boot(law, statistic = cor.stat2, R = 1000)
> boot.ci(b, type = "stud")$stud
     conf
[1,] 0.95 975.98 25.03 -0.8707301 1.083004
> b <- boot(law, statistic = cor.stat2, R = 1000)
> boot.ci(b, type = "stud")$stud
     conf
[1,] 0.95 975.98 25.03 -0.4006036 1.002005
```

```
> b <- boot(law, statistic = cor.stat2, R = 1000)
> boot.ci(b, type = "stud")$stud
     conf
[1,] 0.95 975.98 25.03 0.003815318 0.948999
> print(boot.t.ci(law, B = 1000, R = 25, statistic = cor.stat))
      2.5%      97.5%
-0.2369926  0.9840712
> print(boot.t.ci(law, B = 1000, R = 25, statistic = cor.stat))
      2.5%      97.5%
-0.2942892  0.9798966
> print(boot.t.ci(law, B = 1000, R = 25, statistic = cor.stat))
      2.5%      97.5%
-0.5393726  0.9929993
> detach(law)
> detach(package:bootstrap)
```

7.4 *Refer to the air-conditioning data set* **aircondit** *provided in the* **boot** *package. Assume that the times between failures follow an exponential model Exp(λ). Obtain the MLE of the hazard rate λ and use bootstrap to estimate the bias and standard error of the estimate.*

The MLE of λ is $1/\bar{x}$. The estimates of bias and standard error are printed in the summary of `boot` below.

```
> library(boot)
> x <- aircondit[1]
> rate <- function(x, i) return(1/mean(as.matrix(x[i, ])))
> boot(x, statistic = rate, R = 2000)
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = x, statistic = rate, R = 2000)


Bootstrap Statistics :
      original      bias    std. error
t1* 0.00925212 0.001300935 0.004331923
> detach(package:boot)
```

7.5 *Refer to Exercise 7.4. Compute 95% bootstrap confidence intervals for the mean time between failures $1/\lambda$ by the standard normal, basic, percentile and BCa methods. Compare the intervals and explain why they may differ.*

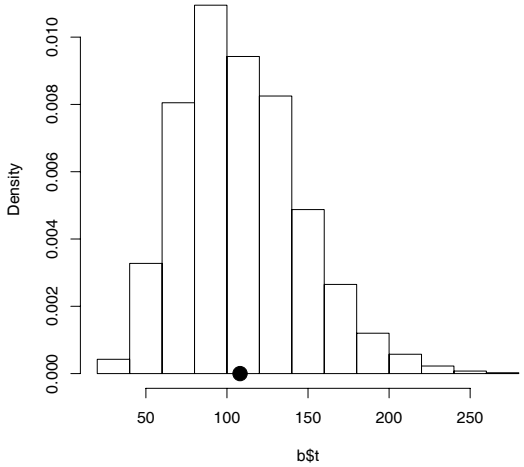
The `aircondit` data is a data frame with 1 variable, so `aircondit[1]` extracts this variable.

```
> library(boot)
> x <- aircondit[1]
> meant <- function(x, i) return(mean(as.matrix(x[i, ])))
```

```
> b <- boot(x, statistic = meant, R = 2000)
> b

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = x, statistic = meant, R = 2000)


Bootstrap Statistics :
    original    bias    std. error
t1* 108.0833 0.9393333    37.10921

> boot.ci(b, type = c("norm", "perc", "basic", "bca"))

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 2000 bootstrap replicates

CALL :
boot.ci(boot.out = b, type = c("norm", "perc", "basic", "bca"))

Intervals :
Level      Normal              Basic
95%   ( 34.4, 179.9 )   ( 25.8, 166.9 )

Level     Percentile            BCa
95%   ( 49.2, 190.3 )   ( 58.9, 230.4 )
Calculations and Intervals on Original Scale
Some BCa intervals may be unstable

> detach(package:boot)
```

The replicates are not approximately normal, so the normal and percentile intervals differ. From the histogram of replicates, it appears that the distribution of the replicates is skewed - although we are estimating a mean, the sample size is too small for CLT to give a good approximation here. The BCa interval is a percentile type interval, but it adjusts for both skewness and bias.

```
> hist(b$t, prob = TRUE, main = "")
> points(b$t0, 0, cex = 2, pch = 16)
```

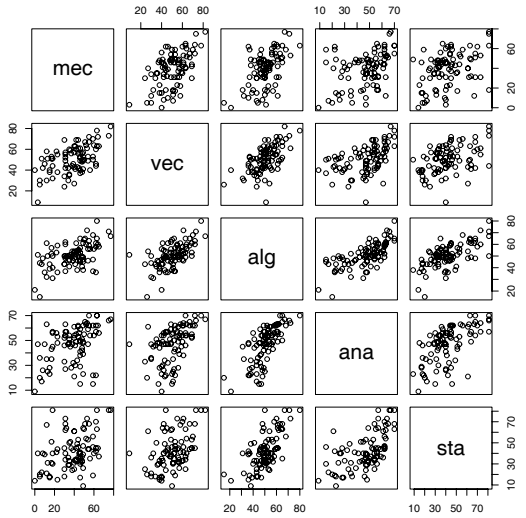


7.6 *Efron and Tibshirani (1993) discuss the* **scor (bootstrap)** *test score data on 88 students who took examinations in five subjects. The first two tests (mechanics, vectors) were closed book and the last three tests (algebra, analysis, statistics) were open book. Each row of the data frame is a set of scores* $(x_{i1}, \ldots, x_{i5})$ *for the* i^{th} *student. Use a panel display to display the scatter plots for each pair of test scores. Compare the plot with the sample correlation matrix. Obtain bootstrap estimates of the standard errors for each of the following estimates:* $\hat{\rho}_{12} = \hat{\rho}$*(mec, vec),* $\hat{\rho}_{34} = \hat{\rho}$*(alg, ana),* $\hat{\rho}_{35} = \hat{\rho}$*(alg, sta),* $\hat{\rho}_{45} = \hat{\rho}$*(ana, sta).*

```
> library(bootstrap)
> attach(scor)
> cor(scor)
```

```
          mec       vec       alg       ana       sta
mec 1.0000000 0.5534052 0.5467511 0.4093920 0.3890993
vec 0.5534052 1.0000000 0.6096447 0.4850813 0.4364487
alg 0.5467511 0.6096447 1.0000000 0.7108059 0.6647357
ana 0.4093920 0.4850813 0.7108059 1.0000000 0.6071743
sta 0.3890993 0.4364487 0.6647357 0.6071743 1.0000000
```

```
> pairs(scor)
```



From the plots and correlation matrix, it appears that open book scores have higher correlations than closed book. All test scores have positive sample correlations. The approximate standard errors of the estimates are given by the output from the `boot` function below.

```
> library(boot)
> cor.stat <- function(x, i = 1:NROW(x)) {
+     cor(x[i, 1], x[i, 2])
+ }
> x <- as.matrix(scor)
```

Bootstrap estimate of $se(\hat{\rho})_{12} = \hat{\rho}(mec, vec)$:

```
> boot(x[, 1:2], statistic = cor.stat, R = 2000)
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = x[, 1:2], statistic = cor.stat, R = 2000)


Bootstrap Statistics :
     original       bias    std. error
t1* 0.5534052 -0.007300376  0.07541025
```

Bootstrap estimate of $se(\hat{\rho})_{34} = \hat{\rho}(alg, ana)$:

```
> boot(x[, 3:4], statistic = cor.stat, R = 2000)
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = x[, 3:4], statistic = cor.stat, R = 2000)
```

```
Bootstrap Statistics :
     original       bias    std. error
t1* 0.7108059 -0.001189044  0.04869804
```

Bootstrap estimate of $se(\hat{\rho})_{35} = \hat{\rho}(alg, sta)$:

```
> boot(x[, c(3, 5)], statistic = cor.stat, R = 2000)
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = x[, c(3, 5)], statistic = cor.stat, R = 2000)


Bootstrap Statistics :
     original       bias    std. error
t1* 0.6647357 -0.001558886  0.06075636
```

Bootstrap estimate of $se(\hat{\rho})_{45} = \hat{\rho}(ana, sta)$:

```
> boot(x[, 4:5], statistic = cor.stat, R = 2000)
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = x[, 4:5], statistic = cor.stat, R = 2000)


Bootstrap Statistics :
     original       bias    std. error
t1* 0.6071743 -0.002175276  0.06731688
> detach(scor)
> detach(package:bootstrap)
> detach(package:boot)
```

7.7 *Refer to Exercise 7.6. Efron and Tibshirani (1993) discuss the following example. The five-dimensional scores data have a 5×5 covariance matrix Σ, with positive eigenvalues $\lambda_1 > \cdots > \lambda_5$. Let $\hat{\lambda}_1 > \cdots > \hat{\lambda}_5$ be the eigenvalues of $\hat{\Sigma}$, where $\hat{\Sigma}$ is the MLE of Σ. Compute the sample estimate $\hat{\theta} = \frac{\hat{\lambda}_1}{\sum_{j=1}^{5} \hat{\lambda}_j}$ of $\theta = \frac{\lambda_1}{\sum_{j=1}^{5} \lambda_j}$. Use bootstrap to estimate the bias and standard error of $\hat{\theta}$.*

```
> library(bootstrap)
> attach(scor)
> x <- cov(as.matrix(scor))
> e <- eigen(x)
> lambda <- e$values
> lambda
[1] 686.98981 202.11107 103.74731  84.63044  32.15329
> lambda/sum(lambda)
```

```
[1] 0.61911504 0.18214244 0.09349705 0.07626893 0.02897653
> th <- function(x, i) {
+     y <- as.matrix(x[i, ])
+     s <- cov(y)
+     e <- eigen(s)
+     lambda <- e$values
+     max(lambda/sum(lambda))
+ }
> library(boot)
> boot(scor, statistic = th, R = 2000)
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = scor, statistic = th, R = 2000)


Bootstrap Statistics :
    original       bias    std. error
t1* 0.619115 0.0007396812  0.04700042
> detach(scor)
> detach(package:boot)
> detach(package:bootstrap)
```

The estimates are $\hat{\lambda}_1 = 686.99$ and $\hat{\theta} \doteq 0.619$, with bias and std. error of $\hat{\theta}$ equal to 0.00074 and 0.047.

7.8 *Refer to Exercise 7.7. Obtain the jackknife estimates of bias and standard error of* $\hat{\theta}$.

```
> library(bootstrap)
> attach(scor)
> x <- as.matrix(scor)
> n <- nrow(x)
> theta.jack <- numeric(n)
> lambda <- eigen(cov(x))$values
> theta.hat <- max(lambda/sum(lambda))
> for (i in 1:n) {
+     y <- x[-i, ]
+     s <- cov(y)
+     lambda <- eigen(s)$values
+     theta.jack[i] <- max(lambda/sum(lambda))
+ }
> bias.jack <- (n - 1) * (mean(theta.jack) - theta.hat)
> se.jack <- sqrt((n - 1)/n * sum((theta.jack - mean(theta.jack))^2))
> c(theta.hat, bias.jack, se.jack)
[1] 0.619115038 0.001069139 0.049552307
> list(est = theta.hat, bias = bias.jack, se = se.jack)
```

```
$est
[1] 0.619115

$bias
[1] 0.001069139

$se
[1] 0.04955231
> detach(scor)
> detach(package:bootstrap)
```

The jackknife estimate of bias of $\hat{\theta}$ is approximately 0.001 and the jackknife estimate of se is approximately 0.05. These estimates are not very different from the bootstrap estimates above.

7.9 *Refer to Exercise 7.7. Compute 95% percentile and BCa confidence intervals for* $\hat{\theta}$.

```
> library(bootstrap)
> attach(scor)
> library(boot)
> b <- boot(scor, statistic = th, R = 2000)
> boot.ci(b, type = c("perc", "bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 2000 bootstrap replicates

CALL :
boot.ci(boot.out = b, type = c("perc", "bca"))

Intervals :
Level     Percentile            BCa
95%   ( 0.5205,  0.7121 )   ( 0.5256,  0.7161 )
Calculations and Intervals on Original Scale
> detach(scor)
> detach(package:boot)
> detach(package:bootstrap)
```

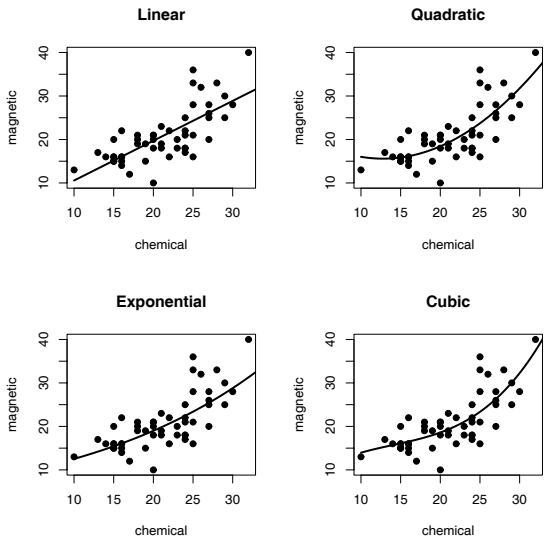
7.10 *Leave-one-out (n-fold) cross validation was used to select the best fitting model for the* ***ironslag*** *data. Repeat the analysis replacing the Log-Log model with a cubic polynomial model. Which of the four models is selected by the cross validation procedure? Which model is selected according to maximum adjusted* R^2*?*

```
> library(DAAG, warn.conflict = FALSE)
> attach(ironslag)
> a <- seq(10, 40, 0.1)
```

First we fit the four models on the complete data and plot the data with the fits from each estimated model. The values of R_a^2 can be extracted from `L1, L2, L3, L4`.

```
> par(mfrow = c(2, 2))
> L1 <- lm(magnetic ~ chemical)
> plot(chemical, magnetic, main = "Linear", pch = 16)
```

```
> yhat1 <- L1$coef[1] + L1$coef[2] * a
> lines(a, yhat1, lwd = 2)
> L2 <- lm(magnetic ~ chemical + I(chemical^2))
> plot(chemical, magnetic, main = "Quadratic", pch = 16)
> yhat2 <- L2$coef[1] + L2$coef[2] * a + L2$coef[3] * a^2
> lines(a, yhat2, lwd = 2)
> L3 <- lm(log(magnetic) ~ chemical)
> plot(chemical, magnetic, main = "Exponential", pch = 16)
> logyhat3 <- L3$coef[1] + L3$coef[2] * a
> yhat3 <- exp(logyhat3)
> lines(a, yhat3, lwd = 2)
> c2 <- chemical^2
> c3 <- chemical^3
> L4 <- lm(magnetic ~ chemical + c2 + c3)
> plot(chemical, magnetic, main = "Cubic", pch = 16)
> yhat4 <- L4$coef[1] + L4$coef[2] * a + L4$coef[3] * a^2 +
+     L4$coef[4] * a^3
> lines(a, yhat4, lwd = 2)
> par(mfrow = c(1, 1))
> Rsq <- numeric(4)
> Rsq[1] <- summary(L1)$adj.r.squared
> Rsq[2] <- summary(L2)$adj.r.squared
> Rsq[3] <- summary(L3)$adj.r.squared
> Rsq[4] <- summary(L4)$adj.r.squared
> Rsq
[1] 0.5281545 0.5768151 0.5280556 0.5740396
```



The quadratic model is selected by maximum adjusted R-squared.

Next we estimate the prediction error for each model by leave-one-out cross validation.

```
> n <- length(magnetic)
> e1 <- e2 <- e3 <- e4 <- numeric(n)
> for (k in 1:n) {
+     y <- magnetic[-k]
+     x <- chemical[-k]
+     J1 <- lm(y ~ x)
+     yhat1 <- J1$coef[1] + J1$coef[2] * chemical[k]
+     e1[k] <- magnetic[k] - yhat1
+     J2 <- lm(y ~ x + I(x^2))
+     yhat2 <- J2$coef[1] + J2$coef[2] * chemical[k] +
+         J2$coef[3] * chemical[k]^2
+     e2[k] <- magnetic[k] - yhat2
+     J3 <- lm(log(y) ~ x)
+     logyhat3 <- J3$coef[1] + J3$coef[2] * chemical[k]
+     yhat3 <- exp(logyhat3)
+     e3[k] <- magnetic[k] - yhat3
+     c2 <- x^2
+     c3 <- x^3
+     J4 <- lm(y ~ x + c2 + c3)
+     yhat4 <- J4$coef[1] + J4$coef[2] * chemical[k] +
+         J4$coef[3] * chemical[k]^2 + J4$coef[4] * chemical[k]^3
+     e4[k] <- magnetic[k] - yhat4
+ }
> c(mean(e1^2), mean(e2^2), mean(e3^2), mean(e4^2))
[1] 19.55644 17.85248 18.44188 18.17756
> detach(ironslag)
> detach(package:DAAG)
```

Model (2) has the smallest prediction error, so the quadratic model (2) is also selected by cross-validation.

7.11 *Leave-one-out (n-fold) cross validation was used to select the best fitting model for the* `ironslag` *data. Use leave-two-out cross validation to compare the models.*

The leave-two-out cross validation is accomplished below by a nested for loop. All five models are compared.

```
> library(DAAG, warn.conflict = FALSE)
> attach(ironslag)
> n <- length(magnetic)
> N <- choose(n, 2)
> e1 <- e2 <- e3 <- e4 <- e5 <- numeric(N)
> ij <- 1
> for (i in 1:(n - 1)) for (j in (i + 1):n) {
+     k <- c(i, j)
+     y <- magnetic[-k]
+     x <- chemical[-k]
+     J1 <- lm(y ~ x)
+     yhat1 <- J1$coef[1] + J1$coef[2] * chemical[k]
+     e1[ij] <- sum((magnetic[k] - yhat1)^2)
+     J2 <- lm(y ~ x + I(x^2))
```

```
+     yhat2 <- J2$coef[1] + J2$coef[2] * chemical[k] +
+         J2$coef[3] * chemical[k]^2
+     e2[ij] <- sum((magnetic[k] - yhat2)^2)
+     J3 <- lm(log(y) ~ x)
+     logyhat3 <- J3$coef[1] + J3$coef[2] * chemical[k]
+     yhat3 <- exp(logyhat3)
+     e3[ij] <- sum((magnetic[k] - yhat3)^2)
+     J4 <- lm(log(y) ~ log(x))
+     logyhat4 <- J4$coef[1] + J4$coef[2] * log(chemical[k])
+     yhat4 <- exp(logyhat4)
+     e4[ij] <- sum((magnetic[k] - yhat4)^2)
+     c2 <- x^2
+     c3 <- x^3
+     J5 <- lm(y ~ x + c2 + c3)
+     yhat5 <- J5$coef[1] + J5$coef[2] * chemical[k] +
+         J5$coef[3] * chemical[k]^2 + J5$coef[4] * chemical[k]^3
+     e5[ij] <- sum((magnetic[k] - yhat5)^2)
+     ij <- ij + 1
+ }
> c(sum(e1), sum(e2), sum(e3), sum(e4), sum(e5))/N
[1] 39.14455 35.74037 36.90983 40.93436 36.46996
> detach(ironslag)
> detach(package:DAAG)
```

The quadratic model (2) is again selected according to the minimum prediction error by leave-two-out cross-validation.

CHAPTER 8

Permutation Tests

8.1 *Implement the two-sample Cramér-von Mises test for equal distributions as a permutation test. Apply the test to the* `chickwts` *data.*

The Cramér-von Mises statistic is

$$W_2 = \frac{mn}{(m+n)^2}\left[\sum_{i=1}^{n}(F_n(x_i) - G_m(x_i))^2 + \sum_{j=1}^{m}(F_n(y_j) - G_m(y_j))^2\right],$$

where F_n is the ecdf of the sample $x_1, \ldots, x_n$ and G_m is the ecdf of the sample $y_1, \ldots, y_m$.

```
> cvm.test <- function(x, y, R = 199) {
+     n <- length(x)
+     m <- length(y)
+     z <- c(x, y)
+     N <- n + m
+     Fn <- numeric(N)
+     Gm <- numeric(N)
+     for (i in 1:N) {
+         Fn[i] <- mean(as.integer(z[i] <= x))
+         Gm[i] <- mean(as.integer(z[i] <= y))
+     }
+     cvm0 <- ((n * m)/N) * sum((Fn - Gm)^2)
+     cvm <- replicate(R, expr = {
+         k <- sample(1:N)
+         Z <- z[k]
+         X <- Z[1:n]
+         Y <- Z[(n + 1):N]
+         for (i in 1:N) {
+             Fn[i] <- mean(as.integer(Z[i] <= X))
+             Gm[i] <- mean(as.integer(Z[i] <= Y))
+         }
+         ((n * m)/N) * sum((Fn - Gm)^2)
+     })
+     cvm1 <- c(cvm, cvm0)
+     return(list(statistic = cvm0, p.value = mean(cvm1 >=
+         cvm0)))
+ }
> attach(chickwts)
> x1 <- as.vector(weight[feed == "soybean"])
> x2 <- as.vector(weight[feed == "sunflower"])
```

```
> x3 <- as.vector(weight[feed == "linseed"])
> detach(chickwts)
> cvm.test(x1, x3)
$statistic
[1] 4.228022

$p.value
[1] 0.405
```

The p-value for the CvM test comparing soybean and linseed supplements is not significant. There is not evidence of a difference between these distributions.

```
> cvm.test(x2, x3)
$statistic
[1] 36.5

$p.value
[1] 0.005
```

The p-value for the CvM test comparing sunflower and linseed supplements is significant at $\alpha = 0.01$, so there is strong evidence that the distributions of weights for these two groups are different.

8.2 *Implement the bivariate Spearman rank correlation test for independence as a permutation test. Compare the achieved significance level of the permutation test with the p-value reported by* `cor.test` *on the same samples.*

```
> spear.perm <- function(x, y) {
+     stest <- cor.test(x, y, method = "spearman")
+     n <- length(x)
+     rs <- replicate(R, expr = {
+         k <- sample(1:n)
+         cor.test(x, y[k], method = "spearman")$estimate
+     })
+     rs1 <- c(stest$estimate, rs)
+     pval <- mean(as.integer(stest$estimate <=
+         rs1))
+     return(list(rho.s = stest$estimate, p.value = pval))
+ }
```

In the following examples, the `mvrnorm` function is used to generate correlated samples. In the first example, the samples are bivariate normal. In the second example, the samples are lognormal. The p-values for `cor.test` and `spear.perm` should be approximately equal in both cases.

```
> library(MASS)
> mu <- c(0, 0)
> Sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
> n <- 30
> R <- 499
> x <- mvrnorm(n, mu, Sigma)
> cor.test(x[, 1], x[, 2], method = "spearman")
```

```
        Spearman's rank correlation rho

data:  x[, 1] and x[, 2]
S = 2782, p-value = 0.03841
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.3810901
> spear.perm(x[, 1], x[, 2])
$rho.s
      rho
0.3810901

$p.value
[1] 0.018
> x <- exp(mvrnorm(n, mu, Sigma))
> cor.test(x[, 1], x[, 2], method = "spearman")
        Spearman's rank correlation rho

data:  x[, 1] and x[, 2]
S = 1998, p-value = 0.001700
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.5555061
> spear.perm(x[, 1], x[, 2])
$rho.s
      rho
0.5555061

$p.value
[1] 0.002
```

The p-values for both tests are both significant and close in value.

```
> x <- exp(mvrnorm(n, mu, Sigma))
> cor.test(x[, 1], x[, 2], method = "spearman")
        Spearman's rank correlation rho

data:  x[, 1] and x[, 2]
S = 1708, p-value = 0.0003422
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.6200222
> spear.perm(x[, 1], x[, 2])
$rho.s
      rho
```

```
0.6200222

$p.value
[1] 0.002
> detach(package:MASS)
```

Again, the p-values are both significant and close in value.

8.3 *The Count 5 criterion is not applicable for unequal sample sizes. Implement a permutation test for equal variance that is based on the maximum outliers statistic that applies when sample sizes are not necessarily equal.*

```
> maxoutliers <- function(x, y) {
+     X <- x - mean(x)
+     Y <- y - mean(y)
+     outx <- sum(X > max(Y)) + sum(X < min(Y))
+     outy <- sum(Y > max(X)) + sum(Y < min(X))
+     return(max(c(outx, outy)))
+ }
> maxout <- function(x, y, R = 199) {
+     z <- c(x, y)
+     n <- length(x)
+     N <- length(z)
+     stats <- replicate(R, expr = {
+         k <- sample(1:N)
+         k1 <- k[1:n]
+         k2 <- k[(n + 1):N]
+         maxoutliers(z[k1], z[k2])
+     })
+     stat <- maxoutliers(x, y)
+     stats1 <- c(stats, stat)
+     tab <- table(stats1)/(R + 1)
+     return(list(estimate = stat, p = mean(stats1 >=
+         stat), freq = tab, cdf = cumsum(tab)))
+ }
```

In the first example, variances are equal. In the second example, variances are unequal. In both examples, sample sizes are unequal. Rather than return only a p-value, here the permutation test procedure returns the distribution of the `maxoutliers` statistic.

```
> set.seed(100)
> n1 <- 20
> n2 <- 40
> mu1 <- mu2 <- 0
> sigma1 <- sigma2 <- 1
> x <- rnorm(n1, mu1, sigma1)
> y <- rnorm(n2, mu2, sigma2)
> maxout(x, y)
$estimate
[1] 6
```

```
$p
[1] 0.195

$freq
stats1
    1     2     3     4     5     6     7     8     9    11
0.135 0.240 0.215 0.165 0.050 0.085 0.045 0.030 0.015 0.015
   16
0.005

$cdf
    1     2     3     4     5     6     7     8     9    11
0.135 0.375 0.590 0.755 0.805 0.890 0.935 0.965 0.980 0.995
   16
1.000
```

This is the equal variance example. The observed statistic is not significant. With the usual critical value of 5 for equal sample sizes, the significance level of the test is 0.055. Refer to the cdf of the replicates to determine if the statistic is significant at 0.055. In the next example the variances are unequal.

```
> set.seed(100)
> sigma1 <- 1
> sigma2 <- 2
> x <- rnorm(n1, mu1, sigma1)
> y <- rnorm(n2, mu2, sigma2)
> maxout(x, y)
$estimate
[1] 18

$p
[1] 0.005

$freq
stats1
    1     2     3     4     5     6     7     8     9    10
0.080 0.350 0.210 0.075 0.090 0.060 0.030 0.060 0.010 0.010
   11    12    14    18
0.005 0.010 0.005 0.005

$cdf
    1     2     3     4     5     6     7     8     9    10
0.080 0.430 0.640 0.715 0.805 0.865 0.895 0.955 0.965 0.975
   11    12    14    18
0.980 0.990 0.995 1.000
```

The observed statistic here, in the case of unequal variances, is significant.

8.4 *Complete the steps to implement a r^{th}-nearest neighbors test for equal distributions. Write a function to compute the test statistic. The function should take the data*

matrix as its first argument, and an index vector as the second argument. The number of nearest neighbors r should follow the index argument.

The `Tn3` statistic for third nearest neighbors can be generalized to r^{th} NN. We apply it to simulated multivariate normal data.

```
> library(knnFinder)
> library(boot)
> Tn.r <- function(z, ix, nbrs, sizes) {
+     n1 <- sizes[1]
+     n2 <- sizes[2]
+     n <- n1 + n2
+     r <- min(nbrs, n - 1)
+     z <- z[ix, ]
+     o <- rep(0, NROW(z))
+     z <- as.data.frame(cbind(z, o))
+     NN <- nn(z, p = r)
+     block1 <- NN$nn.idx[1:n1, ]
+     block2 <- NN$nn.idx[(n1 + 1):n, ]
+     i1 <- sum(block1 < n1 + 0.5)
+     i2 <- sum(block2 > n1 + 0.5)
+     return((i1 + i2)/(r * n))
+ }
```

The simulated data are distributed as $X \sim N_4(0, I)$ and $Y \sim N_4((1,1,1,1)^T, I)$ with unequal sample sizes.

```
> x <- matrix(rnorm(100), 25, 4)
> y <- matrix(rnorm(200, 1), 50, 4)
> N <- c(nrow(x), nrow(y))
> z <- rbind(x, y)
> boot.obj <- boot(data = z, statistic = Tn.r, sim = "permutation",
+     R = 199, sizes = N, nbrs = 10)
> boot.obj
DATA PERMUTATION


Call:
boot(data = z, statistic = Tn.r, R = 199, sim = "permutation",
    sizes = N, nbrs = 10)


Bootstrap Statistics :
    original     bias    std. error
t1*    0.724 -0.1751625  0.02089114
> b <- c(boot.obj$t, boot.obj$t0)
> b0 <- boot.obj$t0
> mean(b >= b0)
[1] 0.005
> detach(package:boot)
> detach(package:knnFinder)
```

The 10^{th} NN test statistic is significant at $\alpha = 0.05$, so the hypothesis $H_0 : X \stackrel{D}{=} Y$ is rejected.

Project 8.B. The following approach follows Example 4.12 in Davison and Hinkley (1997) p. 160.

Pool the two samples and obtain the ordered failure times $y_1 < \ldots y_m$. Let f_{ij} be the number of failures in group i at time y_j, and r_{ij} be the number at risk in group i at time y_j, $i = 1, 2$. Then

$$m_{1j} = \frac{f_{\cdot j} r_{1j}}{r_{\cdot j}}, \qquad v_{1j} = \frac{f_{\cdot j} r_{1j} r_{2j} (r_{\cdot j} - f_{\cdot j})}{r_{\cdot j}^2 (r_{\cdot j} - 1)},$$

are the conditional mean and conditional variance of the number in group 1 to fail at time t_j given $f_{\cdot j} = f_{1j} + f_{2j}$ and $r_{\cdot j} = r_{1j} + r_{2j}$.

The log-rank statistic is

$$T = \frac{\sum_{j=1}^{m} (f_{1j} - m_{1j})}{\left(\sum_{j=1}^{m} v_{1j} \right)^{1/2}}.$$

```
> library(boot)
> attach(aml)
> logrankstat <- function(dat, i) {
+     AML <- aml
+     j1 <- i[1:11]
+     j2 <- i[12:23]
+     x1 <- AML$time[j1]
+     x2 <- AML$time[j2]
+     c1 <- AML$cens[j1]
+     c2 <- AML$cens[j2]
+     t1 <- AML$time[j1][c1 == "1"]
+     t2 <- AML$time[j2][c2 == "1"]
+     y <- sort(c(t1, t2))
+     N <- length(c(t1, t2))
+     f1 <- numeric(N)
+     r1 <- numeric(N)
+     f2 <- numeric(N)
+     r2 <- numeric(N)
+     for (i in 1:N) {
+         f1[i] <- sum(t1 <= y[i])
+         f2[i] <- sum(t2 <= y[i])
+         r1[i] <- sum(x1 >= y[i])
+         r2[i] <- sum(x2 >= y[i])
+     }
+     f1 <- diff(c(0, f1))
+     f2 <- diff(c(0, f2))
+     f <- f1 + f2
+     r <- r1 + r2
+     m1 <- (f * r1)/r
+     v1 <- (f * r1 * r2) * (r - f)/(r^2 * (r -
+         1))
```

```
+     sum(f1 - m1)/sqrt(sum(v1))
+ }
> b <- boot(aml, statistic = logrankstat, R = 999,
+     sim = "permutation")
> b
DATA PERMUTATION


Call:
boot(data = aml, statistic = logrankstat, R = 999, sim = "permutation")


Bootstrap Statistics :
     original   bias    std. error
t1* -1.842929 1.819782    1.047254
> bt <- c(b$t, b$t0)
> mean(bt <= b$t0)
[1] 0.045
> detach(aml)
> detach(package:boot)
```

CHAPTER 9

Markov Chain Monte Carlo Methods

9.1 *Use the Metropolis-Hastings sampler with proposal distribution $\chi^2(1)$ to generate a sample from Rayleigh($\sigma = 4$) target distribution. Compare the performance of the Metropolis-Hastings sampler for Example 9.1 and this problem. In particular, what differences are obvious from the plot corresponding to Figure 9.1?*

The proposal distribution is $\chi^2(X_t)$. The code from Example 9.1 can be repeated, changing the parameter σ.

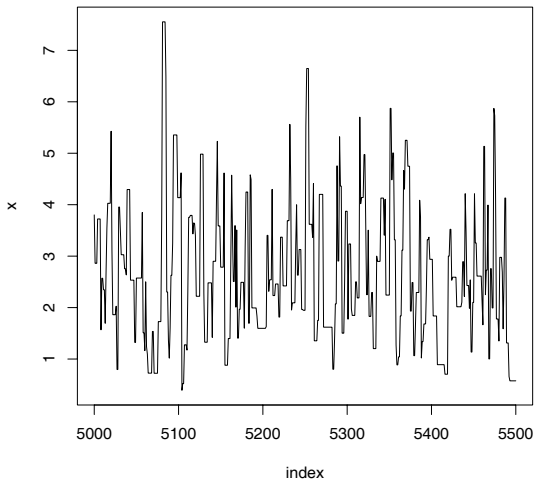
```
> f <- function(x, sigma) {
+     if (x < 0)
+         return(0)
+     stopifnot(sigma > 0)
+     return((x/sigma^2) * exp(-x^2/(2 * sigma^2)))
+ }
> m <- 10000
> sigma <- 2
> x <- numeric(m)
> x[1] <- rchisq(1, df = 1)
> k <- 0
> u <- runif(m)
> for (i in 2:m) {
+     xt <- x[i - 1]
+     y <- rchisq(1, df = xt)
+     num <- f(y, sigma) * dchisq(xt, df = y)
+     den <- f(xt, sigma) * dchisq(y, df = xt)
+     if (u[i] <= num/den)
+         x[i] <- y
+     else {
+         x[i] <- xt
+         k <- k + 1
+     }
+ }
> print(k)
[1] 5298
```

In this example more than half of the candidate points are rejected, so the chain is less efficient when $\sigma = 2$ than when $\sigma = 4$.

To display the plot corresponding to Figure 9.1:

```
> par(mfrow = c(1, 1))
> index <- 5000:5500
```

```
> y1 <- x[index]
> plot(index, y1, type = "l", main = "", ylab = "x")
```



The plot moves horizontally (indicating consecutive rejections) more often than when $\sigma = 4$.

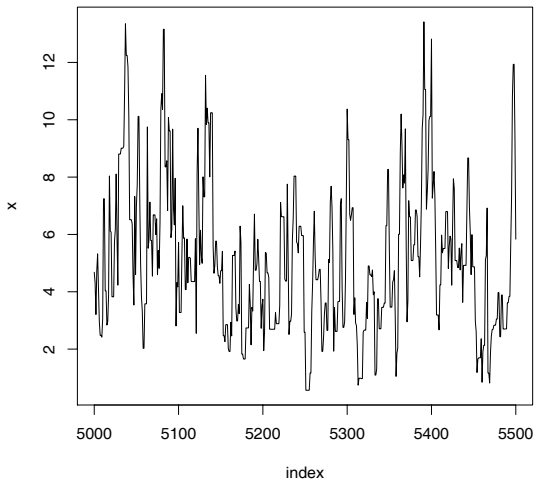
9.2 *Repeat Example 9.1 using the proposal distribution* $Y \sim Gamma(X_t, 1)$ *(shape parameter* X_t *and rate parameter 1).*

```
> f <- function(x, sigma) {
+     if (x < 0)
+         return(0)
+     stopifnot(sigma > 0)
+     return((x/sigma^2) * exp(-x^2/(2 * sigma^2)))
+ }
> m <- 10000
> sigma <- 4
> x <- numeric(m)
> x[1] <- 1
> k <- 0
> u <- runif(m)
> for (i in 2:m) {
+     xt <- x[i - 1]
+     y <- rgamma(1, xt, 1)
+     num <- f(y, sigma) * dgamma(xt, y, 1)
+     den <- f(xt, sigma) * dgamma(y, xt, 1)
+     if (u[i] <= num/den)
+         x[i] <- y
+     else {
+         x[i] <- xt
+         k <- k + 1
+     }
```

```
+ }
> print(k)
[1] 2956
```

In this example only about 30% of the candidate points are rejected, so this chain is somewhat more efficient for $\sigma = 4$ than the chain generated using the χ^2 proposal distribution.

```
> index <- 5000:5500
> y1 <- x[index]
> plot(index, y1, type = "l", main = "", ylab = "x")
```



9.3 *Use the Metropolis-Hastings sampler to generate random variables from a standard Cauchy distribution. Discard the first 1000 of the chain, and compare the deciles of the generated observations with the deciles of the standard Cauchy distribution.*

The following chain uses the $N(\mu_t, \sigma^2)$ proposal distribution, where $\mu_t = X_t$ is the previous value in the chain. Then

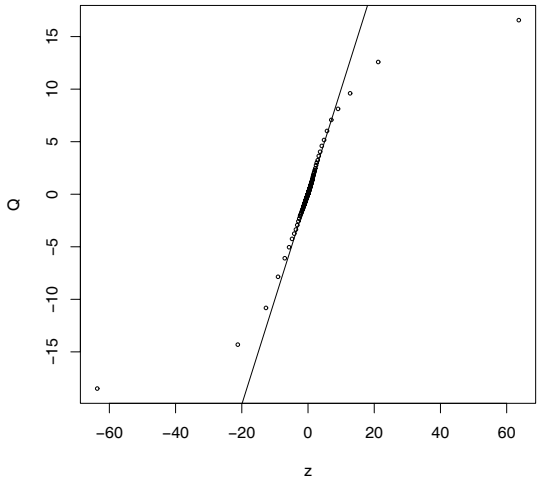
$$r(x_t, y) = \frac{f(y)g(x_t|y)}{f(x_t)g(y|x_t)} = \frac{(1+x_t^2)\pi}{(1+y^2)\pi} \frac{\sqrt{2\pi}\sigma e^{-(x_t-y)^2/(2\sigma^2)}}{\sqrt{2\pi}\sigma e^{-(y-x_t)^2/(2\sigma^2)}} = \frac{1+x_t^2}{1+y^2}.$$

```
> m <- 10000
> sigma <- 3
> x <- numeric(m)
> x[1] <- rnorm(1, 0, sigma)
> k <- 0
> u <- runif(m)
> for (i in 2:m) {
+     xt <- x[i - 1]
+     y <- rnorm(1, xt, sigma)
+     num <- 1 + xt^2
+     den <- 1 + y^2
```

```
+     num <- num * dnorm(xt, y)
+     den <- den * dnorm(y, xt)
+     if (u[i] <= num/den)
+         x[i] <- y
+     else {
+         x[i] <- xt
+         k <- k + 1
+     }
+ }
> print(k)

[1] 4800

> p <- seq(0.1, 0.9, 0.1)
> burn <- 1000
> xb <- x[(burn + 1):m]
> Q <- quantile(xb, p)
> round(rbind(Q, qcauchy(p)), 3)

     10%    20%    30%    40%   50%   60%   70%   80%   90%
Q -2.776 -1.206 -0.627 -0.266 0.026 0.357 0.739 1.506 3.414
  -3.078 -1.376 -0.727 -0.325 0.000 0.325 0.727 1.376 3.078

> p <- seq(0.95, 1, 0.01)
> Q <- quantile(xb, p)
> round(rbind(Q, qcauchy(p)), 3)

    95%   96%    97%    98%    99%  100%
Q 6.445 7.510  8.764 11.188 13.948 26.47
  6.314 7.916 10.579 15.895 31.821   Inf
```

We also computed the upper tail quantiles. The deciles of the generated chain are roughly in agreement with the deciles of standard Cauchy. In the upper tail the difference between sample and Cauchy quantiles is even greater. A QQ plot is below.

```
> p <- ppoints(100)
> Q <- quantile(xb, p)
> z <- qcauchy(p)
> qqplot(z, Q, cex = 0.5)
> abline(0, 1)
```



9.4 *Implement a random walk Metropolis sampler for generating the standard Laplace distribution. For the increment, simulate from a normal distribution. Compare the chains generated when different variances are used for the proposal distribution. Also, compute the acceptance rates of each chain.*

The standard Laplace density is

$$f(x) = \frac{1}{2}e^{-|x|}$$

and

$$r(x_t, y) = \frac{f(y)}{f(x_t)} = \frac{e^{-|y|}}{e^{-|x_t|}} = e^{|x_t|-|y|}.$$

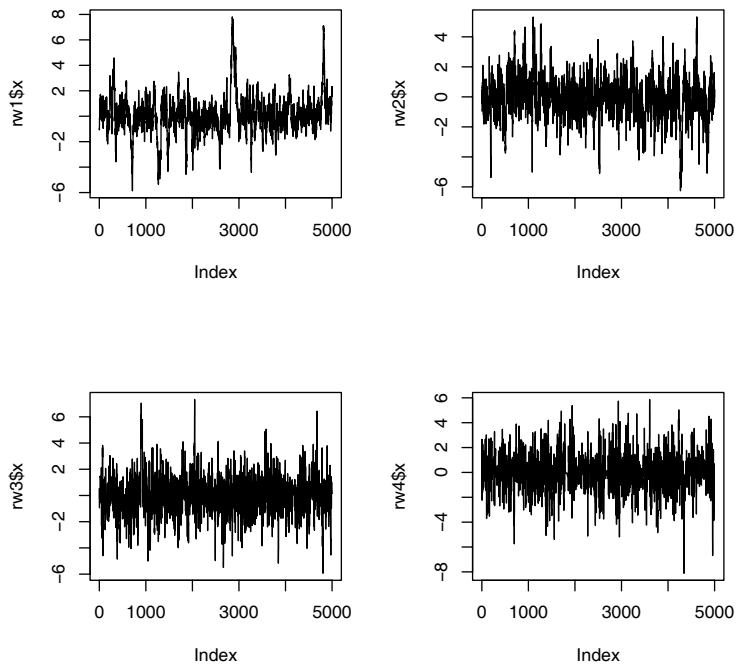
In the generator `rw.Laplace` below, `N` is the length of the chain to generate, `x0=x[1]` is the initial value and `sigma` is the standard deviation of the normal proposal distribution. At each step, the candidate point is generated from $N(\mu_t, \sigma^2)$, where $\mu_t = X_t$ is the previous value in the chain. The return value is a list containing the generated chain `$x` and the number of rejected points `$k`

```
> rw.Laplace <- function(N, x0, sigma) {
+     x <- numeric(N)
+     x[1] <- x0
+     u <- runif(N)
+     k <- 0
+     for (i in 2:N) {
+         xt <- x[i - 1]
+         y <- rnorm(1, xt, sigma)
+         if (u[i] <= exp(abs(xt) - abs(y)))
+             x[i] <- y
+         else {
+             x[i] <- x[i - 1]
+             k <- k + 1
+         }
```

```
+     }
+     return(list(x = x, k = k))
+ }
> N <- 5000
> sigma <- c(0.5, 1, 2, 4)
> x0 <- rnorm(1)
> rw1 <- rw.Laplace(N, x0, sigma[1])
> rw2 <- rw.Laplace(N, x0, sigma[2])
> rw3 <- rw.Laplace(N, x0, sigma[3])
> rw4 <- rw.Laplace(N, x0, sigma[4])
> print(c(rw1$k, rw2$k, rw3$k, rw4$k))


[1]  892 1501 2487 3342


> cat("rejection rates ", (c(rw1$k, rw2$k, rw3$k, rw4$k)/N),
+     "\n")


rejection rates  0.1784 0.3002 0.4974 0.6684


> b <- 100
> y1 <- rw1$x[(b + 1):N]
> y2 <- rw2$x[(b + 1):N]
> y3 <- rw3$x[(b + 1):N]
> y4 <- rw4$x[(b + 1):N]
```

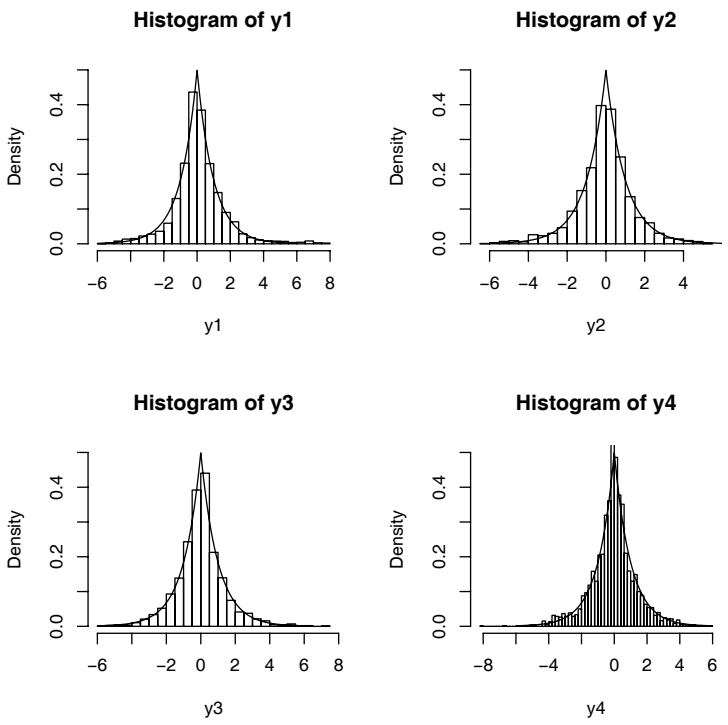
The generated chains are compared in the following plots.

```
> par(mfrow = c(2, 2))
> plot(rw1$x, type = "l")
> plot(rw2$x, type = "l")
> plot(rw3$x, type = "l")
> plot(rw4$x, type = "l")
> par(mfrow = c(1, 1))
```

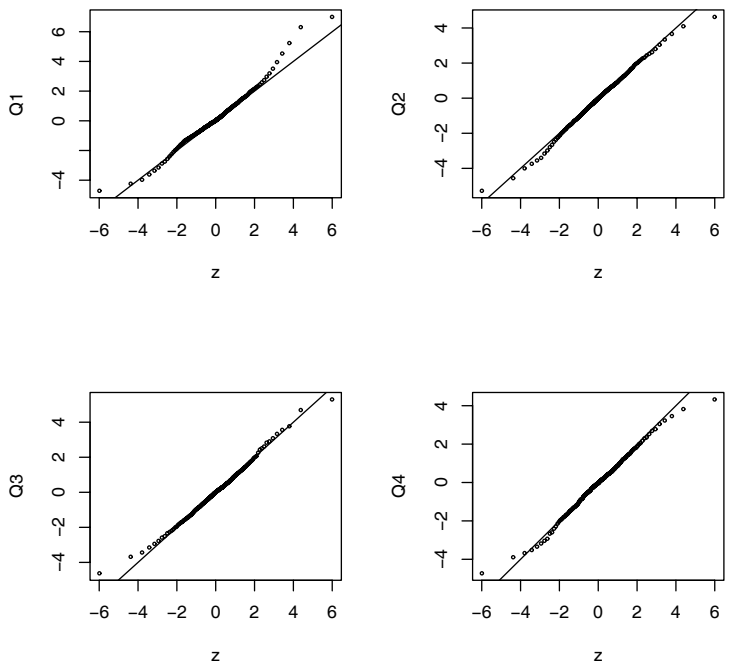


Based on the plots above, a short burn-in sample of size 100 is discarded from each chain. Each of the chains appear to have converged to the target Laplace distribution. Chains 2 and 3 corresponding to $\sigma = 1, 2$ have the best fits based on the QQ plots. The second chain is the more efficient of these two.

```
> par(mfrow = c(2, 2))
> p <- ppoints(200)
> y <- qexp(p, 1)
> z <- c(-rev(y), y)
> fx <- 0.5 * exp(-abs(z))
> hist(y1, breaks = "Scott", freq = FALSE, ylim = c(0,
+     0.5))
> lines(z, fx)
> hist(y2, breaks = "Scott", freq = FALSE, ylim = c(0,
+     0.5))
> lines(z, fx)
> hist(y3, breaks = "Scott", freq = FALSE, ylim = c(0,
+     0.5))
> lines(z, fx)
> hist(y4, breaks = "Scott", freq = FALSE, ylim = c(0,
+     0.5))
> lines(z, fx)
> par(mfrow = c(1, 1))
```




```
> par(mfrow = c(2, 2))
> Q1 <- quantile(y1, p)
> qqplot(z, Q1, cex = 0.4)
> abline(0, 1)
> Q2 <- quantile(y2, p)
> qqplot(z, Q2, cex = 0.4)
> abline(0, 1)
> Q3 <- quantile(y3, p)
> qqplot(z, Q3, cex = 0.4)
> abline(0, 1)
> Q4 <- quantile(y4, p)
> qqplot(z, Q4, cex = 0.4)
> abline(0, 1)
> par(mfrow = c(1, 1))
```



9.5 *What effect, if any, does the width w have on the mixing of the chain in the investment model? Repeat the simulation keeping the random number seed fixed, trying different proposal distributions based on the random increments from Uniform$(-w, w)$, varying w.*

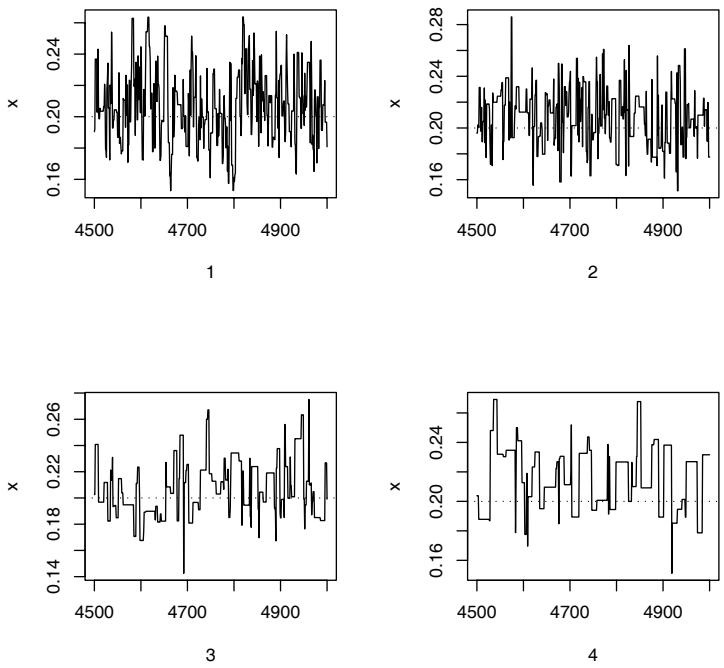
Only a minor modification to the original code is needed. The chains are combined in matrix `X`. A counter was added to compute a rejection rate for each chain.

```
> b <- 0.2
> m <- 5000
> burn <- 1000
> days <- 250
> win <- c(82, 72, 45, 34, 17)
> prob <- function(y, win) {
+     if (y < 0 || y >= 0.5)
+         return(0)
+     return((1/3)^win[1] * ((1 - y)/3)^win[2] * ((1 -
+         2 * y)/3)^win[3] * ((2 * y)/3)^win[4] * (y/3)^win[5])
+ }
> W <- c(0.05, 0.1, 0.2, 0.4)
> X <- matrix(0, m, 4)
> x <- numeric(m)
> k <- rep(0, 4)
> for (j in 1:4) {
```

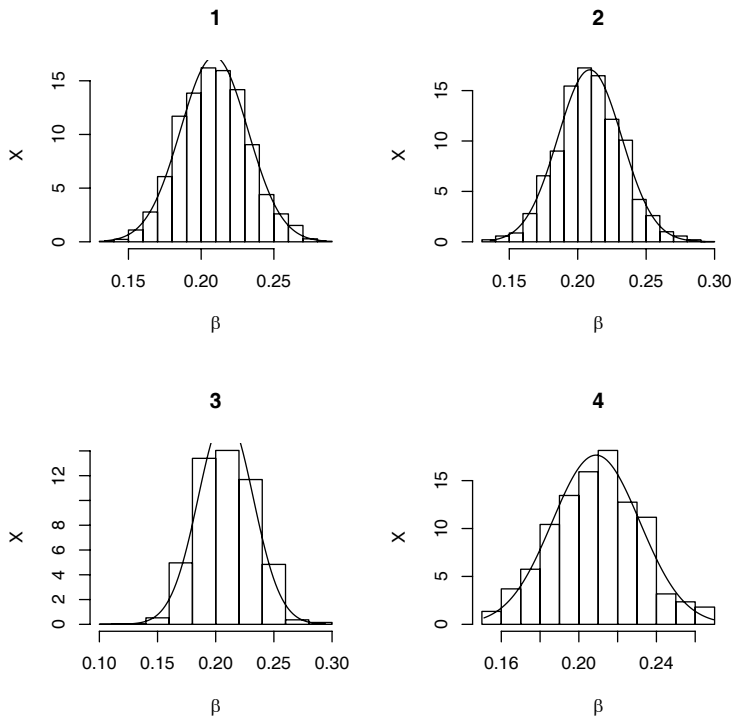
```
+     w <- W[j]
+     u <- runif(m)
+     v <- runif(m, -w, w)
+     x[1] <- 0.25
+     for (i in 2:m) {
+         y <- x[i - 1] + v[i]
+         if (u[i] <= prob(y, win)/prob(x[i - 1], win))
+             x[i] <- y
+         else {
+             x[i] <- x[i - 1]
+             k[j] <- k[j] + 1
+         }
+     }
+     X[, j] <- x
+ }
> win

[1] 82 72 45 34 17

> win/days

[1] 0.328 0.288 0.180 0.136 0.068

> round(c(1, 1 - b, 1 - 2 * b, 2 * b, b)/3, 3)

[1] 0.333 0.267 0.200 0.133 0.067

> colMeans(X[(burn + 1):m, ])

[1] 0.2089636 0.2088640 0.2086093 0.2088591

> k/m

[1] 0.3908 0.6238 0.8102 0.9030
```

Smaller w corresponds to lower rejection rates. All estimates are close to the actual value of the parameter 0.2. In the plots below it appears that the chains with lower w are mixing well and more efficient.

```
> par(mfrow = c(2, 2))
> for (j in 1:4) {
+     x <- X[(m - 500):m, j]
+     plot((m - 500):m, x, type = "l", xlab = j)
+     abline(h = b, v = burn, lty = 3)
+ }
> par(mfrow = c(1, 1))
```




```
> par(mfrow = c(2, 2))
> for (j in 1:4) {
+     xb <- X[(burn + 1):m, j]
+     hist(xb, prob = TRUE, xlab = bquote(beta), ylab = "X",
+         main = j)
+     z <- seq(min(xb), max(xb), length = 100)
+     lines(z, dnorm(z, mean(xb), sd(xb)))
+ }
> par(mfrow = c(1, 1))
```



9.6 *Rao (1973) presented an example on genetic linkage of 197 animals in four categories. The group sizes are* $(125, 18, 20, 34)$*. Assume that the probabilities of the corresponding multinomial distribution are*

$$\left(\frac{1}{2}+\frac{\theta}{4},\ \frac{1-\theta}{4},\ \frac{1-\theta}{4},\ \frac{\theta}{4}\right).$$

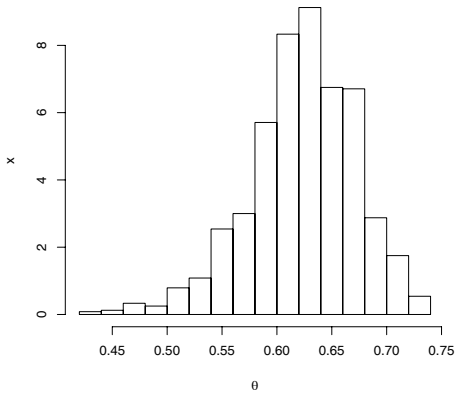
Estimate the posterior distribution of θ *given the observed sample, using one of the methods in this chapter.*

Use the M-H random walk sampler with a uniform proposal distribution. The posterior distribution of θ given the observed frequencies $k = (k_1, k_2, k_3, k_4)$ is

$$f_{\theta|K}(\theta) \propto (2+\theta)^{k_1}(1-\theta)^{k_2+k_3}\theta^{k_4}.$$

```
> k <- c(125, 18, 20, 34)
> w <- 0.1
> m <- 1500
> x <- numeric(m)
> burn <- 300
> pr <- function(th) {
+     if (th > 0 && th < 1)
+         (2 + th)^k[1] * (1 - th)^(k[2] + k[3]) * th^k[4]
+     else 0
+ }
> x[1] <- runif(1, 0, 1)
> u <- runif(m)
```

```
> v <- runif(m, -w, w)
> for (i in 2:m) {
+     y <- x[i - 1] + v[i]
+     if (u[i] <= pr(y)/pr(x[i - 1]))
+         x[i] <- y
+     else {
+         x[i] <- x[i - 1]
+     }
+ }
> xb <- x[(burn + 1):m]
> theta.hat <- mean(xb)
> theta.hat
[1] 0.6233872
> p <- c(0.5 + theta.hat/4, (1 - theta.hat)/4, (1 - theta.hat)/4,
+     theta.hat/4)
> p
[1] 0.6558468 0.0941532 0.0941532 0.1558468
> p * sum(k)
[1] 129.20182  18.54818  18.54818  30.70182
> k
[1] 125  18  20  34

> hist(xb, prob = TRUE, breaks = "Scott", xlab = bquote(theta),
+     ylab = "x", main = "")
```



9.7 *Implement a Gibbs sampler to generate a bivariate normal chain* (X_t, Y_t) *with zero means, unit standard deviations, and correlation 0.9. Plot the generated sample after discarding a suitable burn-in sample. Fit a simple linear regression model* $Y = \beta_0 + \beta_1 X$ *to the sample and check the residuals of the model for normality and constant variance.*

This Gibbs sampler has been implemented in the examples, with different values for the parameters.

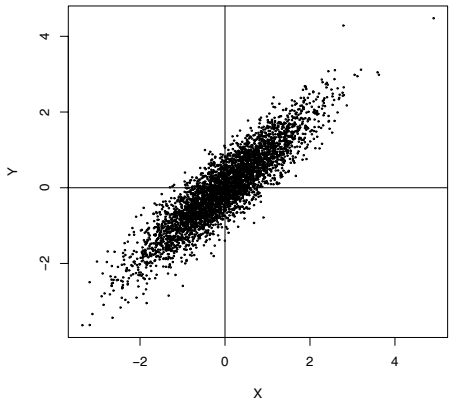
```
> N <- 5000
> burn <- 1000
> X <- matrix(0, N, 2)
> rho <- 0.9
> mu1 <- 0
> mu2 <- 0
> sigma1 <- 1
> sigma2 <- 1
> s1 <- sqrt(1 - rho^2) * sigma1
> s2 <- sqrt(1 - rho^2) * sigma2
> X[1, ] <- c(mu1, mu2)
> for (i in 2:N) {
+     x2 <- X[i - 1, 2]
+     m1 <- mu1 + rho * (x2 - mu2) * sigma1/sigma2
+     X[i, 1] <- rnorm(1, m1, s1)
+     x1 <- X[i, 1]
+     m2 <- mu2 + rho * (x1 - mu1) * sigma2/sigma1
+     X[i, 2] <- rnorm(1, m2, s2)
+ }
> b <- burn + 1
> x <- X[b:N, ]
> X <- x[, 1]
> Y <- x[, 2]
> L <- lm(Y ~ X)
> L

Call:
lm(formula = Y ~ X)

Coefficients:
(Intercept)            X
    0.00133      0.90466

> summary(L)

Call:
lm(formula = Y ~ X)

Residuals:
      Min        1Q    Median        3Q       Max
-1.695721 -0.291392 -0.003165  0.294272  1.761891

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.00133    0.00688   0.193    0.847
X            0.90466    0.00701 129.058   <2e-16 ***
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1

Residual standard error: 0.4351 on 3998 degrees of freedom
```

```
Multiple R-Squared: 0.8064,        Adjusted R-squared: 0.8064
F-statistic: 1.666e+04 on 1 and 3998 DF,  p-value: < 2.2e-16
```

The coefficients of the fitted model match the parameters of the target distribution well. If $Y = 0.9X$ and $Var(X) = Var(Y) = 1$, then $Cor(X, Y) = Cor(X, 0.9X) = 0.9$.

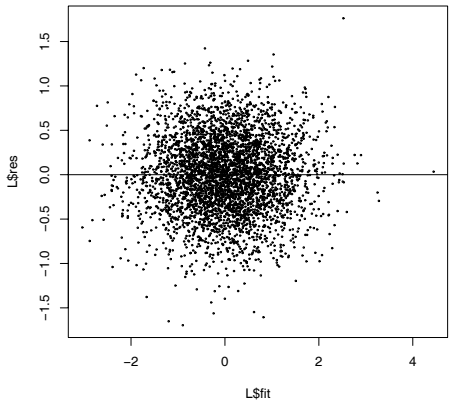
The scatterplot of the generated chain (after discarding the burn-in sample) is shown below. It has the elliptical symmetry and location at the origin of the target bivariate normal distribution. The strong positive correlation is also evident in the plot.

```
> plot(X, Y, cex = 0.25)
> abline(h = 0, v = 0)
```

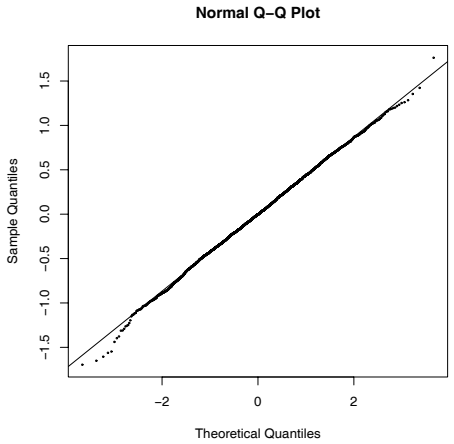


For residual plots, the easiest approach is `plot(L)`. Alternately, plots can be generated as shown below.

```
> plot(L$fit, L$res, cex = 0.25)
> abline(h = 0)
```



```
> qqnorm(L$res, cex = 0.25)
> qqline(L$res)
```



The plot of residuals vs fits suggests that the error variance is constant with respect to the response variable. The QQ plot of residuals is consistent with the normal error assumption of the linear model.

9.8 (Beta-binomial) Consider the bivariate density

$$f(x,y) \propto \binom{n}{x} y^{x+a-1}(1-y)^{n-x+b-1}, \quad x = 0, 1, \ldots, n,\ 0 \le y \le 1.$$

It can be shown that for fixed a, b, n, the conditional distributions are Binomial(n, y) and Beta$(x + a, n - x + b)$. Use the Gibbs sampler to generate a chain with target joint density $f(x, y)$.

In the example below, the parameters are $a = 2, b = 3, n = 10$.

For a bivariate distribution (X, Y), at each iteration the Gibbs sampler

(1) Generate $X^*(t)$ from Binomial$(n, p = Y(t-1))$.
(2) Update $x(t) = X^*(t)$;
(3) Generate $Y^*(t)$ from Beta$(x(t) + a, n - x(t) + b)$.
(4) Set $(X(t), (Y(t)) = (X^*(t), Y^*(t))$.

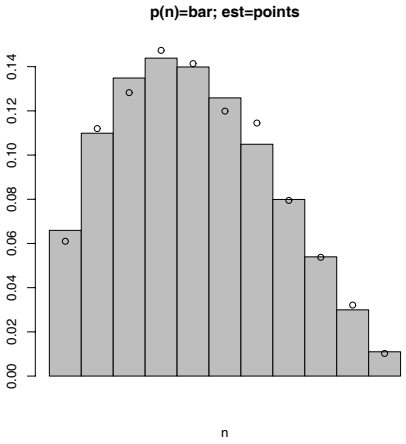
```
> N <- 10000
> burn <- 2000
> a <- 2
> b <- 3
> n <- 10
> x <- y <- rep(0, N)
> x[1] <- rbinom(1, prob = 0.5, size = n)
> y[1] <- rbeta(1, x[1] + a, n - x[1] + b)
> for (i in 2:N) {
+     x[i] <- rbinom(1, prob = y[i - 1], size = n)
+     y[i] <- rbeta(1, x[i] + a, n - x[i] + b)
+ }
> xb <- x[(burn + 1):N]
> f1 <- table(xb)/length(xb)
```

Above the estimated marginal distribution of $f(x|y)$ is shown. The true marginal probability mass function is

$$f(x) = \binom{n}{x} \frac{1}{B(a,b)B(x+a, n-x+b)}.$$

The estimate and target are compared below in a barplot.

```
> i <- 0:n
> fx <- choose(n, i) * beta(i + a, n - i + b)/beta(a, b)
> round(rbind(f1, fx), 3)
       0     1     2     3     4     5     6    7     8     9    10
f1 0.061 0.112 0.128 0.147 0.141 0.120 0.115 0.08 0.054 0.032 0.010
fx 0.066 0.110 0.135 0.144 0.140 0.126 0.105 0.08 0.054 0.030 0.011
> barplot(fx, space = 0, ylim = c(0, 0.15), xlab = "n",
+     main = "p(n)=bar; est=points")
> points(0:n + 0.5, f1)
```



9.9 Modify the Gelman-Rubin convergence monitoring so that only the final value of $\hat{R}$ is computed, and repeat the example, omitting the graphs.

```
> Gelman.Rubin <- function(psi) {
+     psi <- as.matrix(psi)
+     n <- ncol(psi)
+     k <- nrow(psi)
+     psi.means <- rowMeans(psi)
+     B <- n * var(psi.means)
+     psi.w <- apply(psi, 1, "var")
+     W <- mean(psi.w)
+     v.hat <- W * (n - 1)/n + (B/n)
+     r.hat <- v.hat/W
+     return(r.hat)
+ }
> normal.chain <- function(sigma, N, X1) {
+     x <- rep(0, N)
+     x[1] <- X1
```

```
+     u <- runif(N)
+     for (i in 2:N) {
+         xt <- x[i - 1]
+         y <- rnorm(1, xt, sigma)
+         r1 <- dnorm(y, 0, 1) * dnorm(xt, y, sigma)
+         r2 <- dnorm(xt, 0, 1) * dnorm(y, xt, sigma)
+         r <- r1/r2
+         if (u[i] <= r)
+             x[i] <- y
+         else x[i] <- xt
+     }
+     return(x)
+ }
> sigma <- 0.2
> k <- 4
> n <- 15000
> b <- 1000
> x0 <- c(-10, -5, 5, 10)
> X <- matrix(0, nrow = k, ncol = n)
> for (i in 1:k) X[i, ] <- normal.chain(sigma, n, x0[i])
> psi <- t(apply(X, 1, cumsum))
> for (i in 1:nrow(psi)) psi[i, ] <- psi[i, ]/(1:ncol(psi))
> rhat <- Gelman.Rubin(psi)
> rhat
[1] 1.201070
```

9.10 Use the Gelman-Rubin method to monitor convergence of the Rayleigh M-H chain, and run the chain until the chain has converged approximately to the target distribution according to $\hat{R} < 1.2$. (See Exercise 9.9.) Also use the `coda` package to check for convergence of the chain by the Gelman-Rubin method.

```
> f <- function(x, sigma) {
+     if (x < 0)
+         return(0)
+     stopifnot(sigma > 0)
+     return((x/sigma^2) * exp(-x^2/(2 * sigma^2)))
+ }
> Rayleigh.MH.chain1 <- function(sigma, m, x0) {
+     x <- numeric(m)
+     x[1] <- x0
+     u <- runif(m)
+     for (i in 2:m) {
+         xt <- x[i - 1]
+         y <- rchisq(1, df = xt)
+         num <- f(y, sigma) * dchisq(xt, df = y)
+         den <- f(xt, sigma) * dchisq(y, df = xt)
+         if (u[i] <= num/den)
+             x[i] <- y
+         else x[i] <- xt
```

```
+     }
+     return(x)
+ }
> sigma <- 4
> x0 <- c(1/sigma^2, 1/sigma, sigma^2, sigma^3)
> k <- 4
> m <- 2000
> X <- matrix(0, nrow = k, ncol = m)
> for (i in 1:k) X[i, ] <- Rayleigh.MH.chain1(sigma, m,
+     x0[i])
> psi <- t(apply(X, 1, cumsum))
> for (i in 1:nrow(psi)) psi[i, ] <- psi[i, ]/(1:ncol(psi))
> rhat <- Gelman.Rubin(psi)
> rhat

[1] 1.168618
```

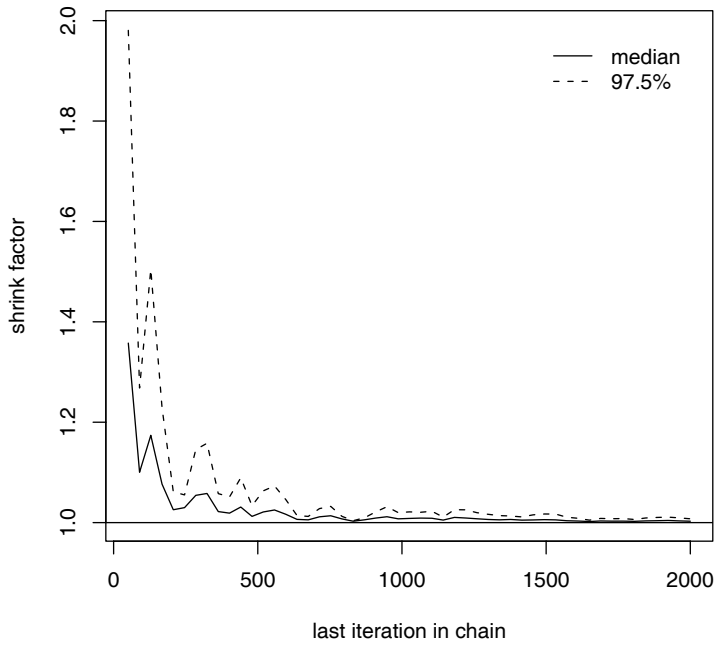
To use the Gelman-Rubin diagnostic functions in `coda`, convert the chains into `mcmc` objects. Then create a list of the `mcmc` objects. The `mcmc.list` is then the argument to `gelman.diag` and `gelman.plot`.

```
> library(coda)
> X1 <- as.mcmc(X[1, ])
> X2 <- as.mcmc(X[2, ])
> X3 <- as.mcmc(X[3, ])
> X4 <- as.mcmc(X[4, ])
> Y <- mcmc.list(X1, X2, X3, X4)
> print(gelman.diag(Y))

Potential scale reduction factors:

     Point est. 97.5% quantile
[1,]       1.00           1.01
```

Remark: help page for `gelman.plot` usage states that `x` is an `mcmc` object. Actually the function is expecting a list of `mcmc` objects or an object that can be converted into a list of this type.

By default the plots print in two colors. Here black and white is specified. The `lattice` library is loaded.

```
> gelman.plot(Y, col = c(1, 1))
```




```
> detach(package:coda)
> detach(package:lattice)
```

9.11 Use the Gelman-Rubin method to monitor convergence of the random walk Metropolis chain for β in the investment model. Also use the `coda` package to check for convergence of the chain by the Gelman-Rubin method.

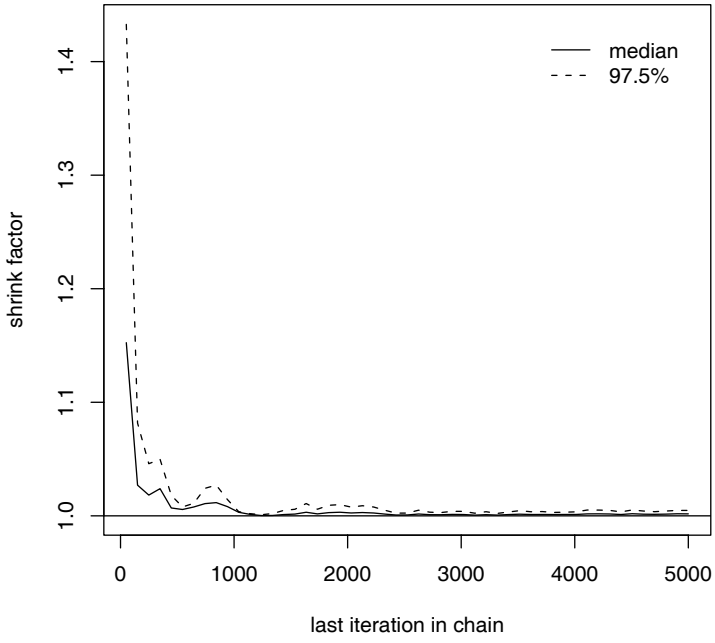
```
> b <- 0.2
> m <- 5000
> burn <- 1000
> days <- 250
> win <- c(82, 72, 45, 34, 17)
> prob <- function(y, win) {
+     if (y < 0 || y >= 0.5)
+         return(0)
+     return((1/3)^win[1] * ((1 - y)/3)^win[2] * ((1 -
+         2 * y)/3)^win[3] * ((2 * y)/3)^win[4] * (y/3)^win[5])
+ }
> w <- 0.1
> X <- matrix(0, 4, m)
> x <- numeric(m)
> x0 <- c(0.01, 0.1, 0.4, 0.49)
> for (j in 1:4) {
+     u <- runif(m)
+     v <- runif(m, -w, w)
```

```
+     x[1] <- x0[j]
+     for (i in 2:m) {
+         y <- x[i - 1] + v[i]
+         if (u[i] <= prob(y, win)/prob(x[i - 1], win))
+             x[i] <- y
+         else {
+             x[i] <- x[i - 1]
+             k[j] <- k[j] + 1
+         }
+     }
+     X[j, ] <- x
+ }
> rowMeans(X[, (burn + 1):m])

[1] 0.2099899 0.2086967 0.2105856 0.2096109

> psi <- t(apply(X, 1, cumsum))
> for (i in 1:nrow(psi)) psi[i, ] <- psi[i, ]/(1:ncol(psi))
> rhat <- Gelman.Rubin(psi)
> rhat

[1] 1.107664
```

If $\hat{R} < 1.2$ then the chain appears to have converged within the 5000 iterations. The analysis is repeated below using `gelman.diag` and `gelman.plot` in the `coda` package.

```
> library(coda)
> X1 <- as.mcmc(X[1, ])
> X2 <- as.mcmc(X[2, ])
> X3 <- as.mcmc(X[3, ])
> X4 <- as.mcmc(X[4, ])
> Y <- mcmc.list(X1, X2, X3, X4)
> print(gelman.diag(Y))

Potential scale reduction factors:

     Point est. 97.5% quantile
[1,]       1.00          1.00
```

```
> gelman.plot(Y, col = c(1, 1))
```




```
> detach(package:coda)
> detach(package:lattice)
```

9.12 Use the Gelman-Rubin method to monitor convergence of the independence sampler chain for the mixing probability p in the normal location mixture. Also use the `coda` package to check for convergence of the chain by the Gelman-Rubin method.

```
> m <- 5000
> a <- 1
> b <- 1
> p <- 0.2
> n <- 30
> mu <- c(0, 5)
> sigma <- c(1, 1)
> X <- matrix(0, 4, m)
> x0 <- c(0.01, 0.05, 0.95, 0.99)
> i <- sample(1:2, size = n, replace = TRUE, prob = c(p,
+     1 - p))
> x <- rnorm(n, mu[i], sigma[i])
> for (j in 1:4) {
+     xt <- numeric(m)
+     u <- runif(m)
+     y <- rbeta(m, a, b)
+     xt[1] <- x0[j]
+     for (i in 2:m) {
```

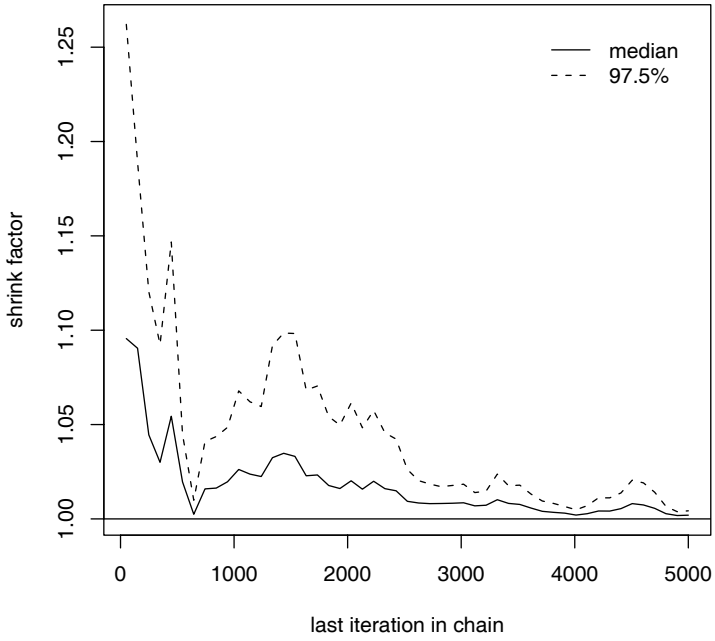
```
+         fy <- y[i] * dnorm(x, mu[1], sigma[1]) + (1 -
+             y[i]) * dnorm(x, mu[2], sigma[2])
+         fx <- xt[i - 1] * dnorm(x, mu[1], sigma[1]) +
+             (1 - xt[i - 1]) * dnorm(x, mu[2], sigma[2])
+         r <- prod(fy/fx) * (xt[i - 1]^(a - 1) * (1 -
+             xt[i - 1])^(b - 1))/(y[i]^(a - 1) * (1 -
+             y[i])^(b - 1))
+         if (u[i] <= r)
+             xt[i] <- y[i]
+         else xt[i] <- xt[i - 1]
+     }
+     X[j, ] <- xt
+ }
> rowMeans(X[, 1001:m])

[1] 0.1345625 0.1309324 0.1378163 0.1373828

> psi <- t(apply(X, 1, cumsum))
> for (i in 1:nrow(psi)) psi[i, ] <- psi[i, ]/(1:ncol(psi))
> rhat <- Gelman.Rubin(psi)
> rhat

[1] 1.211341
```

If $\hat{R} < 1.2$, the chain has converged to the target distribution within 5000 iterations. The analysis is repeated below using the `coda` package.

```
> library(coda)
> X1 <- as.mcmc(X[1, ])
> X2 <- as.mcmc(X[2, ])
> X3 <- as.mcmc(X[3, ])
> X4 <- as.mcmc(X[4, ])
> Y <- mcmc.list(X1, X2, X3, X4)
> print(gelman.diag(Y))

Potential scale reduction factors:

     Point est. 97.5% quantile
[1,]       1.00          1.00
```



```
> gelman.plot(Y, col = c(1, 1))
```




```
> detach(package:coda)
> detach(package:lattice)
```

CHAPTER 10

Density Estimation

10.1 *Construct a histogram estimate of density for a random sample of standard lognormal data using Sturges' Rule, for sample size* $n = 100$. *Repeat the estimate* for the same sample *using the correction for skewness proposed by Doane 1976. Compare the number of bins and break points using both methods. Compare the density estimates at the deciles of the lognormal distribution with the lognormal density at the same points. Does the suggested correction give better density estimates in this example?*

Doane's correction adds

$$K_e = \log_2\left(1 + \frac{|\sqrt{b_1}|}{\sigma(\sqrt{b_1})}\right),$$

classes, where

$$\sigma(\sqrt{b_1}) = \sqrt{\frac{6(n-2)}{(n+1)(n+3)}}.$$

```
> sk <- function(x) {
+     xbar <- mean(x)
+     m3 <- mean((x - xbar)^3)
+     m2 <- mean((x - xbar)^2)
+     return(m3/m2^1.5)
+ }
> n <- 100
> x <- rlnorm(n)
> b1 <- abs(sk(x))
> sig <- sqrt(6 * (n - 2)/((n + 1) * (n + 3)))
> Ke <- log2(1 + b1/sig)
> k <- nclass.Sturges(x)
> K <- round(k + Ke)
> h1 <- diff(range(x))/K
> br1 <- min(x) + h1 * 0:K
> h2 <- diff(range(x))/k
> br2 <- min(x) + h2 * 0:k
> c(k, Ke, K)
[1]  8.000000  3.824416 12.000000
> hist1 <- hist(x, breaks = br1, plot = FALSE)
> hist2 <- hist(x, breaks = br2, plot = FALSE)
> table(cut(x, br1))
```

```
(0.0619,1.05]   (1.05,2.03]   (2.03,3.01]      (3.01,4]      (4,4.98]
           41            29            15            11             1
  (4.98,5.96]   (5.96,6.95]   (6.95,7.93]   (7.93,8.91]   (8.91,9.9]
            1             0             0             0             0
   (9.9,10.9]   (10.9,11.9]
            0             1
> table(cut(x, br2))
(0.0619,1.54]   (1.54,3.01]   (3.01,4.49]   (4.49,5.96]   (5.96,7.44]
           62            23            11             2             0
  (7.44,8.91]   (8.91,10.4]   (10.4,11.9]
            0             0             1
> p <- seq(0.1, 0.9, 0.1)
> Q <- qlnorm(p)
> d <- d1 <- d2 <- dlnorm(Q)
> for (i in 1:9) {
+     Qb1 <- sum(Q[i] >= br1)
+     Qb2 <- sum(Q[i] >= br2)
+     d1[i] <- hist1$density[Qb1]
+     d2[i] <- hist2$density[Qb2]
+ }
> round(rbind(Q, d, d1, d2), 3)
    [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
Q  0.278 0.431 0.592 0.776 1.000 1.288 1.689 2.320 3.602
d  0.632 0.650 0.587 0.498 0.399 0.300 0.206 0.121 0.049
d1 0.427 0.427 0.427 0.427 0.427 0.295 0.295 0.153 0.112
d2 0.427 0.427 0.427 0.427 0.427 0.427 0.156 0.156 0.075
```

Doane's correction does not improve the density estimates except at one of the deciles. The tables show that when the data is binned, much more than 40% of the data falls into the first bin, and the are many zero-count bins, regardless of which method is used.

10.2 *Estimate the IMSE for three histogram density estimates of standard normal data, from a sample size $n = 500$. Use Sturges' Rule, Scott's Normal Reference Rule, and the FD Rule.*

An estimate of

$$IMSE = MISE = \int E[(\hat{f}(x) - f(x))^2]dx = \int MSE(\hat{f}(x))dx$$

is

$$\sum_{i=1}^{k} hMSE_i = \sum_{i=1}^{k} \frac{h}{\nu_i} \sum_{j=1}^{\nu_i} (\hat{f}(x_{ij}) - f(x_{ij}))^2$$

where k is the number of bins, h is the uniform bin width, ν_i is the frequency of observations in bin i, and x_{ij} is the j^{th} observation in bin i.

By sorting the sample data first, it is easy to find the bin containing each sample point as e.g.

```
bin1 <- rep(1:k[1], hg1$counts)
```

Then `bin1` will contain (in order) the bin number corresponding to each element of the ordered sample. We can then use the bin numbers to extract the density estimates and bin frequencies from the histogram objects.

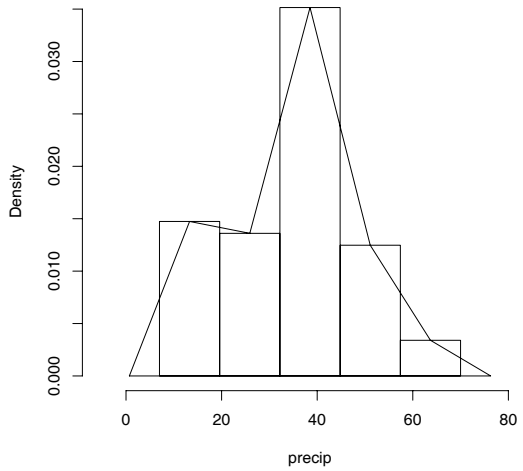
```
> n <- 500
> x <- sort(rnorm(n))
> f <- dnorm(x)
> k <- c(nclass.Sturges(x), nclass.scott(x), nclass.FD(x))
> R <- diff(range(x))
> h <- R/k
> br1 <- min(x) + h[1] * 0:k[1]
> br2 <- min(x) + h[2] * 0:k[2]
> br3 <- min(x) + h[3] * 0:k[3]
> hg1 <- hist(x, breaks = br1, plot = FALSE)
> hg2 <- hist(x, breaks = br2, plot = FALSE)
> hg3 <- hist(x, breaks = br3, plot = FALSE)
> bin1 <- rep(1:k[1], hg1$counts)
> bin2 <- rep(1:k[2], hg2$counts)
> bin3 <- rep(1:k[3], hg3$counts)
> imse1 <- sum((f - hg1$density[bin1])^2 * h[1]/hg1$counts[bin1])
> imse2 <- sum((f - hg2$density[bin2])^2 * h[2]/hg2$counts[bin2])
> imse3 <- sum((f - hg3$density[bin3])^2 * h[3]/hg3$counts[bin3])
> k
[1] 10 18 23
> h
[1] 0.7544384 0.4191324 0.3280167
> c(imse1, imse2, imse3)
[1] 0.007479438 0.003670251 0.004519484
```

Although the estimates vary, usually Scott's rule produces the lowest estimate of IMSE. The FD Rule is close to Scott's rule and Sturge's rule usually has a higher IMSE than either Scott or FD rules. Scott's rule is (asymptotically) optimal for normal distributions by the AMISE criterion.

10.3 *Construct a frequency polygon density estimate for the* `precip` *dataset in R. Verify that the estimate satisfies* $\int_{-\infty}^{\infty} \hat{f}(x)dx \doteq 1$ *by numerical integration of the density estimate.*

```
> n <- length(precip)
> h <- 2.15 * sqrt(var(precip)) * n^(-1/5)
> nbins <- ceiling(diff(range(precip))/h)
> br <- min(precip) + h * 0:nbins
> brplus <- c(min(br) - h, max(br + h))

> histg <- hist(precip, breaks = br, freq = FALSE, main = "",
+     xlim = brplus)
> vx <- histg$mids
> vy <- histg$density
> delta <- diff(vx)[1]
> k <- length(vx)
```

```
> vx <- vx + delta
> vx <- c(vx[1] - 2 * delta, vx[1] - delta, vx)
> vy <- c(0, vy, 0)
> polygon(vx, vy)
> c(h, delta)
[1] 12.59941 12.59941
> length(histg$counts)
[1] 5
```




```
> fpoly <- approxfun(vx, vy)
> integrate(fpoly, lower = min(vx), upper = max(vx))
1 with absolute error < 5.3e-05
```

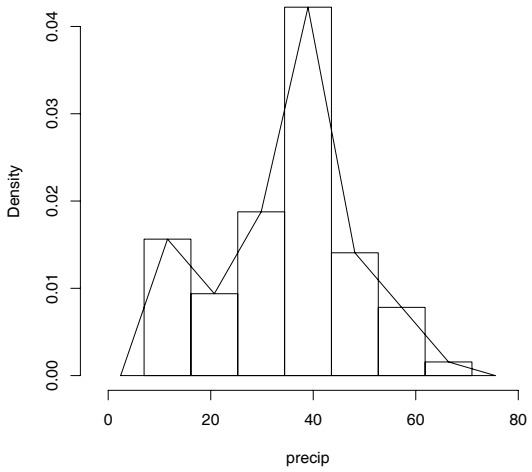
10.4 *Construct a frequency polygon density estimate for the* **precip** *dataset, using a bin width determined by substituting*

$$\hat{\sigma} = IQR/1.348$$

for standard deviation in the usual Normal Reference Rule for a frequency polygon.

```
> n <- length(precip)
> s <- IQR(precip)/1.348
> h <- 2.15 * s * n^(-1/5)
> nbins <- ceiling(diff(range(precip))/h)
> br <- min(precip) + h * 0:nbins
> brplus <- c(min(br) - h, max(br + h))

> histg <- hist(precip, breaks = br, freq = FALSE, main = "",
+     xlim = brplus)
> vx <- histg$mids
> vy <- histg$density
> delta <- diff(vx)[1]
```

```
> k <- length(vx)
> vx <- vx + delta
> vx <- c(vx[1] - 2 * delta, vx[1] - delta, vx)
> vy <- c(0, vy, 0)
> polygon(vx, vy)
> c(h, delta)
[1] 9.137634 9.137634
> length(histg$counts)
[1] 7
```




```
> fpoly <- approxfun(vx, vy)
> integrate(fpoly, lower = min(vx), upper = max(vx))
1 with absolute error < 1.1e-14
```

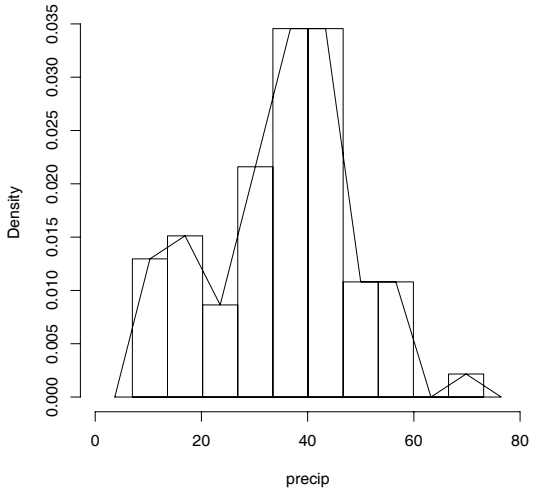
10.5 *Construct a frequency polygon density estimate for the* **precip** *dataset, using a bin width determined by the Normal Reference Rule for a frequency polygon adjusted for skewness. The skewness adjustment factor is computed with reference to a lognormal distribution:*

$$\frac{12^{1/5}\sigma}{e^{7\sigma^2/4}(e^{\sigma^2}-1)^{1/2}(9\sigma^4+20\sigma^2+12)^{1/5}}.$$

The adjustment factor should be multiplied times the bin width.

```
> n <- length(precip)
> s <- sd(precip)
> h1 <- 2.15 * s * n^(-1/5)
> s <- sd(log(precip))
> (exp(3 * s^2) - 3 * exp(s^2) + 2)/(exp(s^2) - 1)^1.5
[1] 1.885683
> a <- 12^0.2 * s/(exp(7 * s^2/4) * sqrt(exp(s^2) - 1))
> a <- a/(9 * s^4 + 20 * s^2 + 12)^0.2
```

```
> h <- h1 * a
> nbins <- ceiling(diff(range(precip))/h)
> br <- min(precip) + h * 0:nbins
> brplus <- c(min(br) - h, max(br + h))

> histg <- hist(precip, breaks = br, freq = FALSE, main = "",
+     xlim = brplus)
> vx <- histg$mids
> vy <- histg$density
> delta <- diff(vx)[1]
> k <- length(vx)
> vx <- vx + delta
> vx <- c(vx[1] - 2 * delta, vx[1] - delta, vx)
> vy <- c(0, vy, 0)
> polygon(vx, vy)
> c(h1, a, h, delta)
[1] 12.5994091  0.5249548  6.6141201  6.6141201
> length(histg$counts)
[1] 10
```

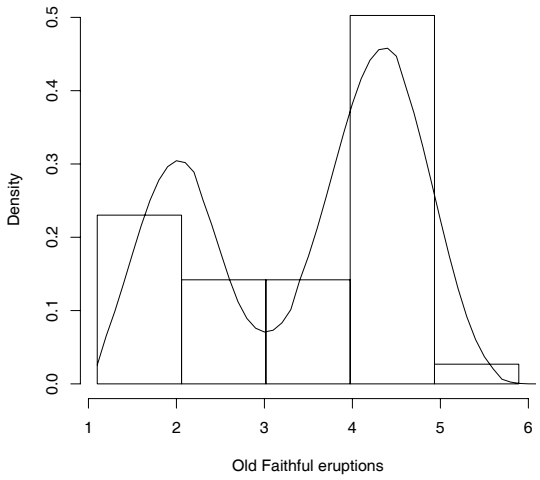



```
> fpoly <- approxfun(vx, vy)
> integrate(fpoly, lower = min(vx), upper = max(vx))
1 with absolute error < 0.00011
```

10.6 *Construct an ASH density estimate for the* `faithful$eruptions` dataset in R, using width h determined by the normal reference rule. Use a weight function corresponding to the biweight kernel,

$$K(t) = \frac{15}{16}(1-t^2)^2 \text{ if } |t| < 1, \qquad K(t) = 0 \text{ otherwise.}$$

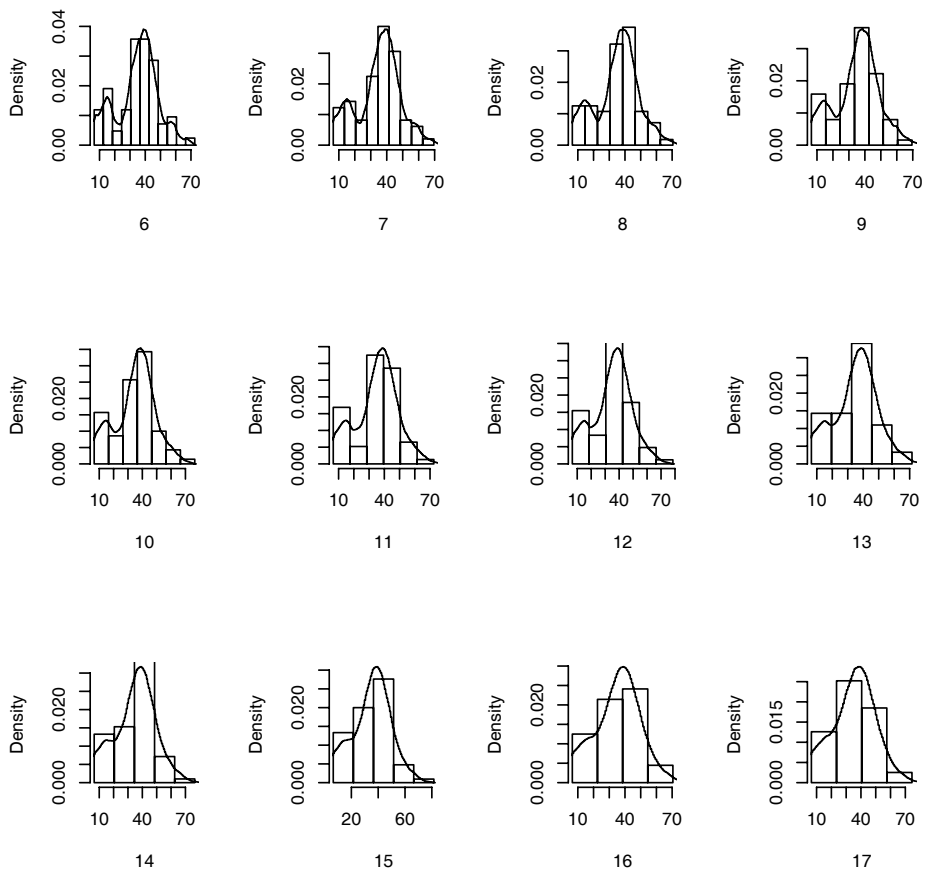
```
> x <- faithful$eruptions
> n <- length(x)
> m <- 20
> a <- min(x) - 0.5
> b <- max(x) + 0.5
> h <- 2.576 * sd(x) * n^(-1/5)
> delta <- h/m
> br <- seq(a - delta * m, b + 2 * delta * m, delta)
> histg <- hist(x, breaks = br, plot = FALSE)
> nk <- histg$counts
> j <- (1 - m):(m - 1)
> j <- j/m
> K <- (15/16) * (1 - j^2)^2
> fhat <- function(x) {
+     i <- max(which(x > br))
+     k <- (i - m + 1):(i + m - 1)
+     vk <- nk[k]
+     sum(K * vk)/(n * h)
+ }
> z <- as.matrix(seq(a, b + h, 0.1))
> f.ash <- apply(z, 1, fhat)


> br2 <- seq(a, b + h, h)
> hist(x, breaks = br2, freq = FALSE, main = "", xlab = "Old Faithful eruptions")
> lines(z, f.ash)
```



10.7 *Construct an ASH density estimate for the* `precip` *dataset in R. Choose the best value for width h^* empirically by computing the estimates over a range of possible values of h and comparing the plots of the densities. Does the optimal value h_n^{fp} correspond to the optimal value h^* suggested by comparing the density plots?*

The optimal value for a normal density is $h^* = 2.576\sigma n^{-1/5}$. Then depending on the method of estimating σ, $h^* \approxeq 10.95$ (robust estimate) or $h^* = 15.1$ (standard deviation). This data is non-normal.

```
> n <- length(precip)
> m <- 20
> a <- min(precip) - 0.5
> b <- max(precip) + 0.5
> par(mfrow = c(3, 4))
> for (h in seq(6, 17, )) {
+     delta <- h/m
+     br <- seq(a - delta * m, b + 2 * delta * m, delta)
+     histg <- hist(precip, breaks = br, plot = FALSE)
+     nk <- histg$counts
+     K <- abs((1 - m):(m - 1))
+     fhat <- function(x) {
+         i <- max(which(x > br))
+         k <- (i - m + 1):(i + m - 1)
+         vk <- nk[k]
+         sum((1 - K/m) * vk)/(n * h)
+     }
+     z <- as.matrix(seq(a, b + h, 0.1))
+     f.ash <- apply(z, 1, fhat)
+     br2 <- seq(a, b + h, h)
+     hist(precip, breaks = br2, freq = FALSE, main = "",
+         ylim = c(0, max(f.ash)), xlab = h)
+     lines(z, f.ash, xlab = "waiting")
+ }
> par(mfrow = c(1, 1))
```

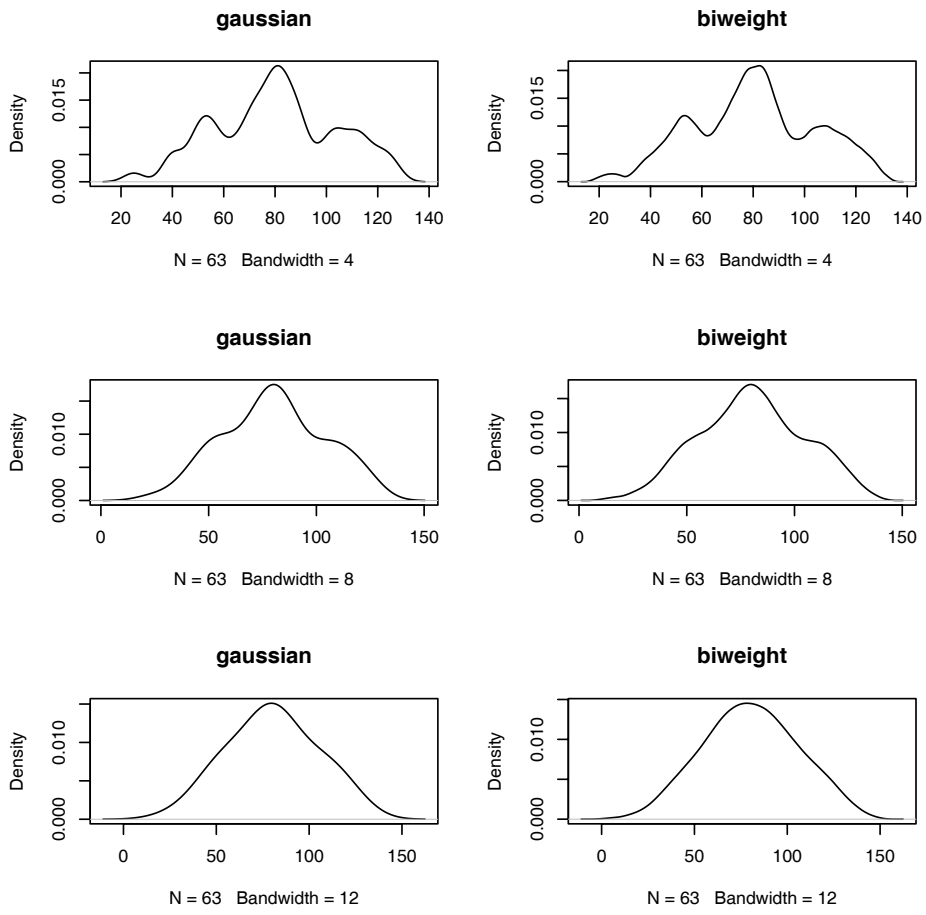


The "optimal" bin width appears to be close to 11. This value is is close to the h^* suggested by the normal reference rule using a robust estimator of σ.

10.8 *The* `buffalo` *dataset in the* `gss` *package contains annual snowfall accumulations in Buffalo, New York from 1910 to 1973. Construct kernel density estimates of the data using Gaussian and biweight kernels. Compare the estimates for different choices of bandwidth. Is the estimate more influenced by the type of kernel or the bandwidth?*

```
> library(gss)
> data(buffalo)
> par(mfrow = c(3, 2))
> for (h in seq(4, 12, 4)) {
+     plot(density(buffalo, bw = h, kernel = "gaussian"),
+         main = "gaussian")
+     plot(density(buffalo, bw = h, kernel = "biweight"),
+         main = "biweight")
+ }
```

```
> par(mfrow = c(1, 1))
> detach(package:gss)
```



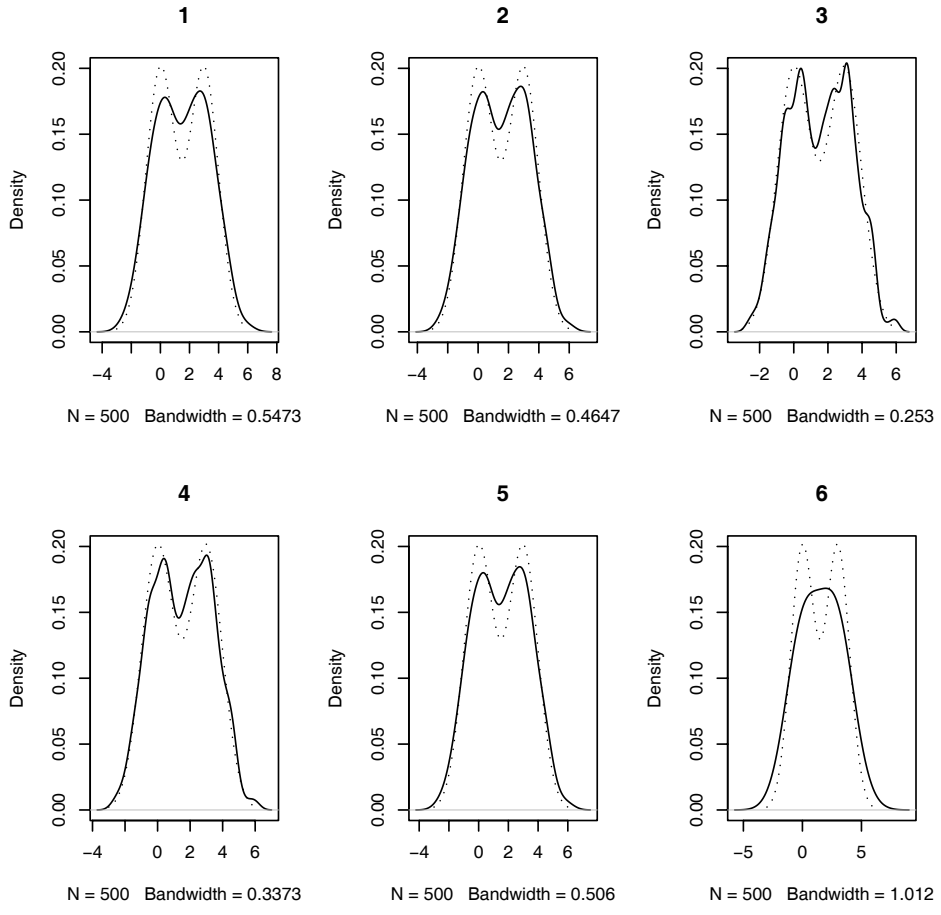
The density estimates are more influenced by the bandwidth.

10.9 *Construct a kernel density estimate of the normal location mixture* $p_1 N(1,1) + (1-p_1)N(3,1)$ *where* $p_1 = 0.5$. *Compare several choices of bandwidth, including* $h = 1.06\hat{\sigma}n^{-1/5}$ *and* $h = 0.9\min(S, IQR/1.34)n^{-1/5}$. *Plot the true density of the mixture over the density estimate, for comparison. Which choice of smoothing parameter appears to be best?*

Below the Gaussian kernel is applied with different choices for bandwidth.

```
> n <- 500
> p <- 0.5
> mu <- sample(c(0, 3), size = 1000, replace = TRUE, prob = c(p,
+     1 - p))
> x <- rnorm(n, mu, 1)
> S <- sd(x)
> h <- rep(1, 6)
```

```
> h[1] <- 1.06 * S * n^(-1/5)
> h[2] <- 0.9 * min(c(S, IQR(x)/1.34)) * n^(-1/5)
> h[3] <- (h[1] + h[2])/4
> h[4] <- (h[1] + h[2])/3
> h[5] <- (h[1] + h[2])/2
> h[6] <- h[1] + h[2]
> h
[1] 0.5361782 0.4552456 0.2478559 0.3304746 0.4957119 0.9914238
> z <- seq(-3, 6, 0.1)
> f <- 0.5 * dnorm(z, 0, 1) + 0.5 * dnorm(z, 3, 1)
> par(mfrow = c(2, 3))
> for (i in 1:6) {
+     plot(density(x, bw = h[i]), main = i, ylim = c(0,
+         0.2))
+     lines(z, f, lty = 3)
+ }
> par(mfrow = c(1, 1))
```

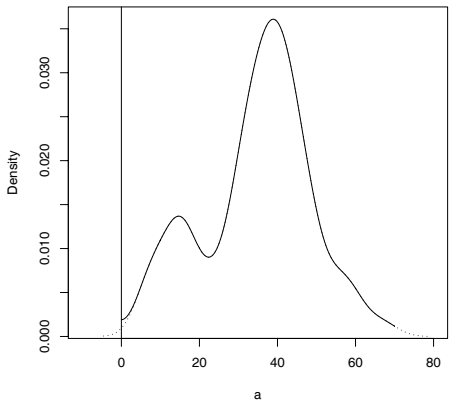


The best choices of smoothing parameter are the ones given by the formulas above and their average (1, 2, 5). Choice (2) corresponding to the default "Silverman's rule-of-thumb" appears to fit best.

10.10 *Apply the reflection boundary technique to obtain a better kernel density estimate for the precipitation data. Compare the estimates in a single graph. Also try setting* `from = 0` *or* `cut = 0` *in the* `density` *function.*

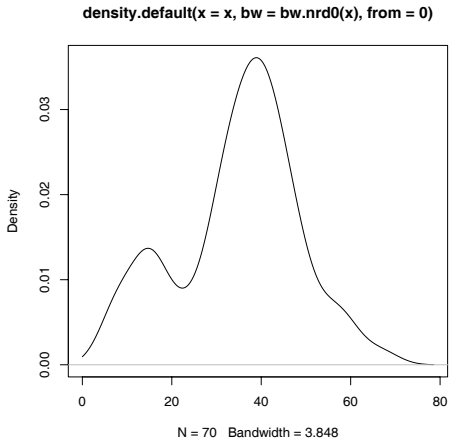
The first plot shows the reflection boundary method in the solid curve with the kernel density estimate as the dotted line.

```
> x <- precip
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   7.00   29.38   36.60   34.89   42.77   67.00
> xx <- c(x, -x)
> g <- density(xx, bw = bw.nrd0(x))
> a <- seq(0, 70, 0.01)
> ghat <- approx(g$x, g$y, xout = a)
> fhat <- 2 * ghat$y
> bw <- paste("Bandwidth = ", round(g$bw, 5))
> plot(a, fhat, type = "l", xlim = c(-10, 80), main = "",
+     ylab = "Density")
> abline(v = 0)
> lines(density(x), lty = 3)
```

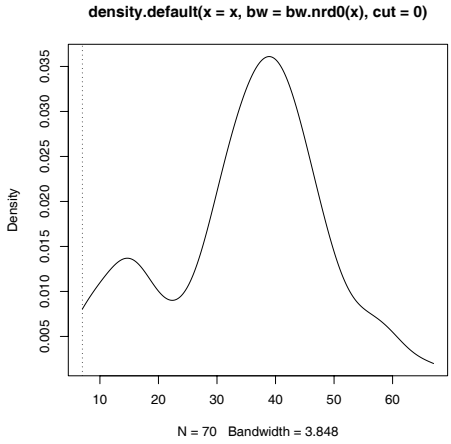


The second plot shows the effect of setting `from = 0` and the third plot shows the effect of setting `cut = 0`.

```
> plot(density(x, bw = bw.nrd0(x), from = 0))
```



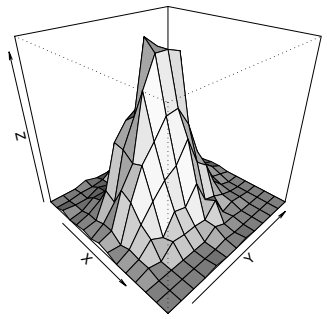

```
> plot(density(x, bw = bw.nrd0(x), cut = 0))
> abline(v = min(x), lty = 3)
```



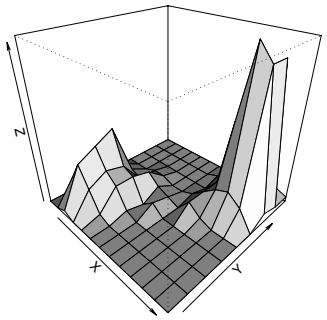
10.11 *Write a bivariate histogram plotting function. Apply your function to display the bivariate* `faithful` *data (Old Faithful geyser).*

```
> bin2d <- function(x, breaks1 = "Sturges", breaks2 = "Sturges") {
+     histg1 <- hist(x[, 1], breaks = breaks1, plot = FALSE)
+     histg2 <- hist(x[, 2], breaks = breaks2, plot = FALSE)
+     brx <- histg1$breaks
+     bry <- histg2$breaks
+     freq <- table(cut(x[, 1], brx), cut(x[, 2], bry))
+     return(list(call = match.call(), freq = freq, breaks1 = brx,
+         breaks2 = bry, mids1 = histg1$mids, mids2 = histg2$mids))
+ }
> fp2d <- function(x, breaks1 = "Sturges", breaks2 = "Sturges") {
+     b <- bin2d(x)
+     h1 <- diff(b$breaks1)
```

```
+     h2 <- diff(b$breaks2)
+     h <- outer(h1, h2, "*")
+     Z <- b$freq/(n * h)
+     persp(x = b$mids1, y = b$mids2, z = Z, shade = TRUE,
+         xlab = "X", ylab = "Y", main = "", theta = 45,
+         phi = 30, ltheta = 60)
+ }
> n <- 2000
> d <- 2
> x <- matrix(rnorm(n * d), n, d)

> fp2d(x)
```



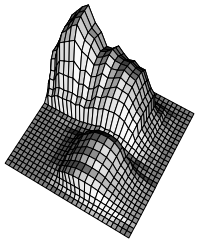

```
> fp2d(log(as.matrix(faithful)))
```



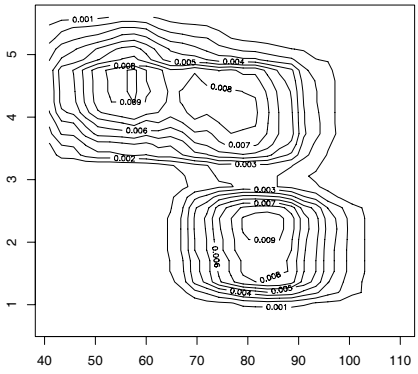
10.12 *Plot a bivariate ASH density estimate of the* `geyser(MASS)` *data.*

```
> library(ash)
> library(MASS)
```

```
> attach(geyser)
> x <- as.matrix(geyser)
> detach(geyser)
> detach(package:MASS)
> nbin <- c(30, 30)
> m <- c(5, 5)
> b <- bin2(x, nbin = nbin)
> est <- ash2(b, m = m, kopt = c(1, 0))

> persp(x = est$x, y = est$y, z = est$z, shade = TRUE,
+     xlab = "X", ylab = "Y", zlab = "", main = "", theta = 30,
+     phi = 75, ltheta = 30, box = FALSE)
```



```
> contour(x = est$x, y = est$y, z = est$z, main = "")
```



10.13 *Generalize the bivariate ASH algorithm to compute an ASH density estimate for a d-dimensional multivariate density, $d \geq 2$.*

Sort multivariate data $x = \{(x_1, \ldots, x_d)\}$ into an $nbin_1 \times \cdots \times nbin_d$ array of bins with frequencies $\nu = (\nu_{j_1,\ldots,j_d})$ and bin widths $h = (h_1, \ldots, h_d)$. (Use `table(cut())` or generalize the `bin2d` function.) The parameter $m = (m_1, \ldots, m_d)$ is the number of shifted histograms on each axis used in the estimate. The histograms are shifted in d directions, so that there are $m = \prod_{j=1}^{d} m_j$ histogram density estimates to be averaged.

The bivariate ASH estimate of the joint density $f(x) = f(x_1, \ldots, x_d)$ is

$$\hat{f}_{ASH}(x) = \frac{1}{m} \sum_{j_1=1}^{m_1} \cdots \sum_{j_d=1}^{m_d} \hat{f}_{j_1,\ldots,j_d}(x_1, \ldots, x_d).$$

To apply a similar algorithm as in the univariate ASH, the corresponding bin weights for the triangular kernel are

$$\begin{aligned} w_{j_1,\ldots,j_d} &= \left(1 - \frac{|j_1|}{m_1}\right) \times \cdots \times \left(1 - \frac{|j_d|}{m_d}\right) \\ &= \prod_{k=1}^{d} \left(1 - \frac{|j_k|}{m_k}\right), \qquad j_k = 1 - m_k, \ldots, m_k - 1, \quad k = 1, \ldots, d. \end{aligned}$$

10.14 *Write a function to bin three-dimensional data into a three-way contingency table, following the method in the* `bin2d` *function. Check the result on simulated* $N_3(0, I)$ *data. Compare the marginal frequencies returned by your function to the expected frequencies from a standard univariate normal distribution.*

The `bin2d` function can easily be converted to `bin3d`.

```
> bin3d <- function(x, breaks1 = "Sturges", breaks2 = "Sturges",
+     breaks3 = "Sturges") {
+     histg1 <- hist(x[, 1], breaks = breaks1, plot = FALSE)
+     histg2 <- hist(x[, 2], breaks = breaks2, plot = FALSE)
+     histg3 <- hist(x[, 3], breaks = breaks3, plot = FALSE)
+     brx <- histg1$breaks
+     bry <- histg2$breaks
+     brz <- histg3$breaks
+     freq <- table(cut(x[, 1], brx), cut(x[, 2], bry),
+         cut(x[, 3], brz))
+     return(list(call = match.call(), freq = freq, breaks1 = brx,
+         breaks2 = bry, breaks3 = brz, mids1 = histg1$mids,
+         mids2 = histg2$mids, mids3 = histg3$mids))
+ }
```

In the example below, a very large sample of standard 3-dimensional normal data is generated and binned. Then each of the marginal frequency distributions is compared with standard normal.

```
> n <- 2000
> x <- matrix(rnorm(n * 3), n, 3)
> b <- bin3d(x)
> h1 <- diff(b$breaks1)
> f1hat <- apply(b$freq, MARGIN = 1, FUN = "sum")
> f1 <- (pnorm(b$breaks1[-1]) - pnorm(b$breaks1[-length(b$breaks1)])) *
```

```
+     n
> round(cbind(f1, f1hat))
          f1 f1hat
(-3.5,-3]   2     5
(-3,-2.5]  10    12
(-2.5,-2]  33    31
(-2,-1.5]  88    83
(-1.5,-1] 184   177
(-1,-0.5] 300   287
(-0.5,0]  383   440
(0,0.5]   383   367
(0.5,1]   300   282
(1,1.5]   184   188
(1.5,2]    88    89
(2,2.5]    33    30
(2.5,3]    10     9
> h2 <- diff(b$breaks2)
> f2hat <- apply(b$freq, MARGIN = 2, FUN = "sum")
> f2 <- (pnorm(b$breaks2[-1]) - pnorm(b$breaks2[-length(b$breaks2)])) *
+     n
> round(cbind(f2, f2hat))
          f2 f2hat
(-3.5,-3]   2     1
(-3,-2.5]  10    11
(-2.5,-2]  33    41
(-2,-1.5]  88    83
(-1.5,-1] 184   178
(-1,-0.5] 300   271
(-0.5,0]  383   384
(0,0.5]   383   375
(0.5,1]   300   313
(1,1.5]   184   194
(1.5,2]    88   104
(2,2.5]    33    35
(2.5,3]    10    10
> h3 <- diff(b$breaks3)
> f3hat <- apply(b$freq, MARGIN = 3, FUN = "sum")
> f3 <- (pnorm(b$breaks3[-1]) - pnorm(b$breaks3[-length(b$breaks3)])) *
+     n
> round(cbind(f3, f3hat))
          f3 f3hat
(-4,-3.5]   0     1
(-3.5,-3]   2     4
(-3,-2.5]  10     5
(-2.5,-2]  33    34
(-2,-1.5]  88    70
(-1.5,-1] 184   190
(-1,-0.5] 300   309
```

```
(-0.5,0]  383   389
(0,0.5]   383   377
(0.5,1]   300   285
(1,1.5]   184   180
(1.5,2]    88   100
(2,2.5]    33    37
(2.5,3]    10    19
```

The numbers of bins are typically not equal across dimensions because the range of the data in each dimension varies.

CHAPTER 11

Numerical Methods in R

11.1 *The natural logarithm and exponential functions are inverses of each other, so that mathematically* $\log(\exp x) = \exp(\log x) = x$. *Show by example that this property does not hold exactly in computer arithmetic. Does the identity hold with near equality? (See* `all.equal`.*)*

```
> log(exp(3)) == exp(log(3))
[1] FALSE
> log(exp(3)) - exp(log(3))
[1] -4.440892e-16
> all.equal(log(exp(3)), exp(log(3)))
[1] TRUE
```

11.2 *Suppose that* X *and* Y *are independent random variables variables,* $X \sim$ *Beta(a, b) and* $Y \sim$ *Beta(r, s). Then it can be shown that*

$$P(X < Y) = \sum_{k=\max(r-b,0)}^{r-1} \frac{\binom{r+s-1}{k}\binom{a+b-1}{a+r-1-k}}{\binom{a+b+r+s-2}{a+r-1}}$$

Write a function to compute $P(X < Y)$ *for any* $a, b, r, s > 0$. *Compare your result with a Monte Carlo estimate of* $P(X < Y)$ *for* $(a, b) = (10, 20)$ *and* $(r, s) = (5, 5)$.

```
> comp.beta <- function(a, b, r, s) {
+     k <- max(c(r - b, 0)):(r - 1)
+     i1 <- lchoose(r + s - 1, k)
+     i2 <- lchoose(a + b - 1, a + r - 1 - k)
+     i3 <- lchoose(a + b + r + s - 2, a + r - 1)
+     return(sum(exp(i1 + i2 - i3)))
+ }
> a <- 10
> b <- 20
> r <- 5
> s <- 5
> comp.beta(a, b, r, s)
[1] 0.8259472
> m <- 10000
> x <- rbeta(m, a, b)
> y <- rbeta(m, r, s)
> mean(x < y)
[1] 0.8295
```

11.3 *(a) Write a function to compute the k^{th} term in*

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{k!\,2^k} \frac{\|a\|^{2k+2}}{(2k+1)(2k+2)} \frac{\Gamma\left(\frac{d+1}{2}\right)\Gamma\left(k+\frac{3}{2}\right)}{\Gamma\left(k+\frac{d}{2}+1\right)},$$

where $d \geq 1$ is an integer, a is a vector in $\mathbb{R}^d$, and $\|\cdot\|$ denotes the Euclidean norm. Perform the arithmetic so that the coefficients can be computed for (almost) arbitrarily large k and d. (This sum converges for all $a \in \mathbb{R}^d$).

It can be shown that the sum converges to $E\|a - Z\| - E\|Z\|$, where $Z \sim N_d(0, I_d)$.For large $\|a\|$ this sum is difficult to evaluate, but $E\|a - Z\| \approxeq \|a\|$ if $\|a\|$ is very large. Write the k^{th} term of the sum as

$$(-1)^k a_k c_k,$$

where

$$a_k = \|a\|^{2(k+1)}, \qquad c_k = \frac{1}{k!2^k(2k+1)(2k+2)} \frac{\Gamma\left(\frac{d+1}{2}\right)\Gamma\left(k+\frac{3}{2}\right)}{\Gamma\left(k+\frac{d}{2}+1\right)}.$$

Then use logarithms to avoid overflow.

```
> sk <- function(a, k) {
+     if (k < 0)
+         return(0)
+     d <- length(a)
+     aa <- sum(a * a)
+     log.ak <- (k + 1) * log(aa)
+     log.ck <- lgamma((d + 1)/2) + lgamma(k + 1.5) - lgamma(k +
+         1) - k * log(2) - log((2 * k + 1) * (2 * k +
+         2)) - lgamma(k + d/2 + 1)
+     y <- exp(log.ak + log.ck)
+     if (k%%2)
+         y <- -y
+     return(sqrt(2/pi) * y)
+ }
```

(b) Modify the function so that it computes and returns the sum.
(c) Evaluate the sum when $a = (1, 2)^T$.

```
> da <- function(a, K = 60) {
+     if (K < 0)
+         return(0)
+     k <- 0:K
+     d <- length(a)
+     aa <- sum(a * a)
+     log.ak <- (k + 1) * log(aa)
+     log.ck <- lgamma((d + 1)/2) + lgamma(k + 1.5) - lgamma(k +
+         1) - k * log(2) - log((2 * k + 1) * (2 * k +
+         2)) - lgamma(k + d/2 + 1)
+     y <- exp(log.ak + log.ck)
+     i <- rep(c(1, -1), length = K + 1)
+     z <- sqrt(2/pi) * sum(i * y)
```

```
+     return(min(c(z, sqrt(aa))))
+ }
> a <- c(1, 2)
> da(a = a)
[1] 1.22249
```

11.4 *Find the intersection points $A(k)$ of the curves*

$$S_{k-1}(a) = P\left(t(k-1) > \sqrt{\frac{a^2(k-1)}{k-a^2}}\right)$$

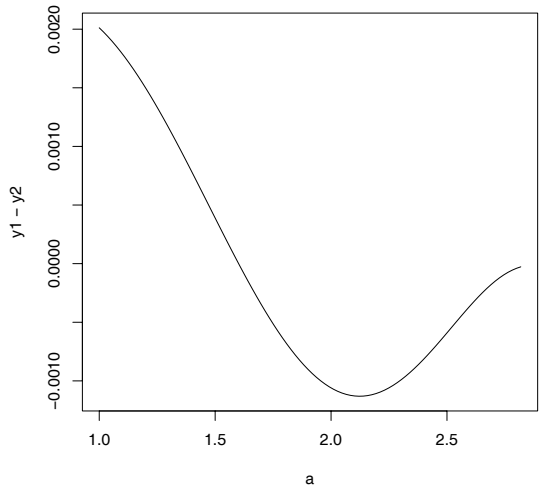
and

$$S_k(a) = P\left(t(k) > \sqrt{\frac{a^2 k}{k+1-a^2}}\right),$$

for $k = 4 : 25, 100, 500, 1000$, where $t(k)$ is a Student t random variable with k degrees of freedom.

First we plot one pair of functions, to find an interval that brackets the root. Actually, we plot the difference $S_{k-1}(a) - S_k(a)$. Note that the function can only be evaluated for $|a| < k$.

```
> k <- 8
> a <- seq(1, sqrt(k) - 0.01, length = 100)
> y1 <- 1 - pt(sqrt(a^2 * (k - 1)/(k - a^2)), df = k -
+     1)
> y2 <- 1 - pt(sqrt(a^2 * k/(k + 1 - a^2)), df = k)
> plot(a, y1 - y2, type = "l")
```



The plot for $k = 8$ suggests that there is one positive root in the interval $(1, 2)$. After further checking, the same interval can be used for all of the values of k needed. Write a function $f(a; k) = S_{k-1}(a) - S_k(a)$. Then use `uniroot` (Brent's method) to solve $f(a; k) = 0$.

```
> f <- function(a, k) {
+     c1 <- sqrt(a^2 * (k - 1)/(k - a^2))
+     c2 <- sqrt(a^2 * k/(k + 1 - a^2))
+     p1 <- pt(c1, df = k - 1, lower.tail = FALSE)
+     p2 <- pt(c2, df = k, lower.tail = FALSE)
+     p1 - p2
+ }
> K <- c(4:25, 100, 500, 1000)
> n <- length(K)
> r <- rep(0, n)
> pr <- rep(0, n)
> for (i in 1:n) {
+     k <- K[i]
+     u <- uniroot(f, lower = 1, upper = 2, k = k)
+     r[i] <- u$root
+     pr[i] <- pt(r[i], df = k - 1, lower.tail = FALSE)
+ }
> cbind(K, r, pr)
```

```
         K        r         pr
 [1,]    4 1.492103 0.11624401
 [2,]    5 1.533556 0.09995866
 [3,]    6 1.562744 0.08943713
 [4,]    7 1.584430 0.08209372
 [5,]    8 1.601185 0.07668485
 [6,]    9 1.614521 0.07253902
 [7,]   10 1.625390 0.06926236
 [8,]   11 1.634419 0.06660884
 [9,]   12 1.642038 0.06441697
[10,]   13 1.648554 0.06257645
[11,]   14 1.654190 0.06100942
[12,]   15 1.659114 0.05965937
[13,]   16 1.663452 0.05848431
[14,]   17 1.667303 0.05745240
[15,]   18 1.670745 0.05653908
[16,]   19 1.673840 0.05572507
[17,]   20 1.676637 0.05499504
[18,]   21 1.679178 0.05433669
[19,]   22 1.681496 0.05373996
[20,]   23 1.683620 0.05319661
[21,]   24 1.685572 0.05269980
[22,]   25 1.687373 0.05224382
[23,]  100 1.720608 0.04422306
[24,]  500 1.729755 0.04214617
[25,] 1000 1.730907 0.04188852
```

11.5 *Write a function to solve the equation*

$$\frac{2\Gamma(\frac{k}{2})}{\sqrt{\pi(k-1)}\Gamma(\frac{k-1}{2})}\int_0^{c_{k-1}}\left(1+\frac{u^2}{k-1}\right)^{-k/2}du = \frac{2\Gamma(\frac{k+1}{2})}{\sqrt{\pi k}\Gamma(\frac{k}{2})}\int_0^{c_k}\left(1+\frac{u^2}{k}\right)^{-(k+1)/2}du$$

for a, where

$$c_k = \sqrt{\frac{a^2k}{k+1-a^2}}.$$

Compare the solutions with the points $A(k)$ in Exercise 11.4.

The easiest way to solve the equation is to observe that the right-hand side of the equation is $F(c_k) - F(0)$, where $F(x)$ is the cdf of a Student t variable with k degrees of freedom. Similarly, the left-hand side of the equation can be expressed in terms of the cdf of a $t(k-1)$ variable. The solution c_k can be found from Exercise 11.4.

```
> a <- r
> ck <- sqrt(a^2 * K/(K + 1 - a^2))
> cbind(K, a, ck)
```

```
         K        a       ck
 [1,]    4 1.492103 1.791862
 [2,]    5 1.533556 1.795334
 [3,]    6 1.562744 1.793016
 [4,]    7 1.584430 1.789174
 [5,]    8 1.601185 1.785136
 [6,]    9 1.614521 1.781334
 [7,]   10 1.625390 1.777885
 [8,]   11 1.634419 1.774801
 [9,]   12 1.642038 1.772054
[10,]   13 1.648554 1.769606
[11,]   14 1.654190 1.767419
[12,]   15 1.659114 1.765459
[13,]   16 1.663452 1.763694
[14,]   17 1.667303 1.762099
[15,]   18 1.670745 1.760652
[16,]   19 1.673840 1.759335
[17,]   20 1.676637 1.758130
[18,]   21 1.679178 1.757025
[19,]   22 1.681496 1.756009
[20,]   23 1.683620 1.755070
[21,]   24 1.685572 1.754201
[22,]   25 1.687373 1.753395
[23,]  100 1.720608 1.737726
[24,]  500 1.729755 1.733212
[25,] 1000 1.730907 1.732638
```

11.6 *Write a function to compute the cdf of the Cauchy distribution, which has density*

$$\frac{1}{\theta\pi(1+[(x-\eta)/\theta]^2)}, \qquad -\infty < x < \infty,$$

where $\theta > 0$. *Compare your results to the results from the R function* **pcauchy**.

Let $y = (x-\eta)/\theta$, so y is standard Cauchy with density $f(y) = \frac{1}{\pi}\frac{1}{1+y^2}$. Hence $\int_0^y f(t)dt = \frac{1}{\pi}\arctan(y)$.

```
> prC <- function(x, eta = 0, theta = 1) {
+     y <- (x - eta)/theta
+     v <- atan(abs(y))/pi
+     if (y >= 0)
+         value <- v + 0.5
+     if (y < 0)
+         value <- 0.5 - v
+     value
+ }
> x <- matrix(seq(-3, 3, 1), ncol = 1)
> p1 <- apply(x, MARGIN = 1, FUN = prC)
> p2 <- pcauchy(x)
> cbind(x, p1, p2)
               p1
[1,] -3 0.1024164 0.1024164
[2,] -2 0.1475836 0.1475836
[3,] -1 0.2500000 0.2500000
[4,]  0 0.5000000 0.5000000
[5,]  1 0.7500000 0.7500000
[6,]  2 0.8524164 0.8524164
[7,]  3 0.8975836 0.8975836
```

The source code in pcauchy.c handles the integral as follows:

```
if (!lower_tail)
x = -x;
/* for large x, the standard formula suffers from cancellation.
 * This is from Morten Welinder thanks to  Ian Smith's  atan(1/x) : */
if (fabs(x) > 1) {
double y = atan(1/x) / M_PI;
return (x > 0) ? R_D_Clog(y) : R_D_val(-y);
} else
return R_D_val(0.5 + atan(x) / M_PI);
```

11.7 *Use the simplex algorithm to solve the following problem.*
Minimize $4x + 2y + 9z$ *subject to*

$$2x + y + z \le 2$$
$$x - y + 3z \le 3$$
$$x \ge 0,\ y \ge 0,\ z \ge 0.$$

See Example 11.16. The constraints can be written as $A_1 x \le b_1$ and $x \ge 0$. Enter the coefficients of the objective function in `a`.

```
> library(boot)
> A1 <- rbind(c(2, 1, 1), c(1, -1, 3))
> b1 <- c(2, 3)
> a <- c(4, 2, 9)
> simplex(a = a, A1 = A1, b1 = b1, maxi = TRUE)
Linear Programming Results

Call : simplex(a = a, A1 = A1, b1 = b1, maxi = TRUE)

Maximization Problem with Objective Function Coefficients
x1 x2 x3
 4  2  9


Optimal solution has the following values
  x1   x2   x3
0.00 0.75 1.25
The optimal value of the objective  function is 12.75.
```

11.8 *In the Morra game, the set of optimal strategies are not changed if a constant is subtracted from every entry of the payoff matrix, or a positive constant is multiplied times every entry of the payoff matrix. However, the simplex algorithm may terminate at a different basic feasible point (also optimal). Compute $B = A + 2$, find the solution of game B, and verify that it is one of the extreme points (11.12)–(11.15) of the original game A. Also find the value of game A and game B.*

```
> solve.game <- function(A) {
+     min.A <- min(A)
+     A <- A - min.A
+     max.A <- max(A)
+     A <- A/max(A)
+     m <- nrow(A)
+     n <- ncol(A)
+     it <- n^3
+     a <- c(rep(0, m), 1)
+     A1 <- -cbind(t(A), rep(-1, n))
+     b1 <- rep(0, n)
+     A3 <- t(as.matrix(c(rep(1, m), 0)))
+     b3 <- 1
+     sx <- simplex(a = a, A1 = A1, b1 = b1, A3 = A3, b3 = b3,
+         maxi = TRUE, n.iter = it)
+     a <- c(rep(0, n), 1)
+     A1 <- cbind(A, rep(-1, m))
+     b1 <- rep(0, m)
+     A3 <- t(as.matrix(c(rep(1, n), 0)))
+     b3 <- 1
+     sy <- simplex(a = a, A1 = A1, b1 = b1, A3 = A3, b3 = b3,
+         maxi = FALSE, n.iter = it)
+     soln <- list(A = A * max.A + min.A, x = sx$soln[1:m],
```

```
+         y = sy$soln[1:n], v = sx$soln[m + 1] * max.A +
+             min.A)
+     soln
+ }
> A <- matrix(c(0, -2, -2, 3, 0, 0, 4, 0, 0, 2, 0, 0, 0,
+     -3, -3, 4, 0, 0, 2, 0, 0, 3, 0, 0, 0, -4, -4, -3,
+     0, -3, 0, 4, 0, 0, 5, 0, 0, 3, 0, -4, 0, -4, 0, 5,
+     0, 0, 3, 0, 0, 4, 0, -5, 0, -5, -4, -4, 0, 0, 0,
+     5, 0, 0, 6, 0, 0, 4, -5, -5, 0, 0, 0, 6, 0, 0, 4,
+     0, 0, 5, -6, -6, 0), 9, 9)
> library(boot)
> B <- A + 2
> s <- solve.game(B)
> s$v
x10
  2
> round(cbind(s$x, s$y), 7)
        [,1]      [,2]
x1 0.0000000 0.0000000
x2 0.0000000 0.0000000
x3 0.4098361 0.4098361
x4 0.0000000 0.0000000
x5 0.3278689 0.3278689
x6 0.0000000 0.0000000
x7 0.2622951 0.2622951
x8 0.0000000 0.0000000
x9 0.0000000 0.0000000
> round(s$x * 61, 7)
x1 x2 x3 x4 x5 x6 x7 x8 x9
 0  0 25  0 20  0 16  0  0
```

The value of the game is $v = 2$ (the value of the original game was $v = 0$), and the simplex algorithm terminated at the extreme point given by (11.15).

C6965

ISBN 1-420-07696-5

Chapman & Hall/CRC
Taylor & Francis Group
an **informa** business

www.taylorandfrancisgroup.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
270 Madison Avenue
New York, NY 10016
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK