1 Block 1

Topics

Variables and constants

Basic instructions: variable/constant declaration, assignment, reading, printing

Math operations

Control structures: sequence

1.1

Write a program that reads two numbers a and b and computes the sum z between them.

0 SOLUTION

```
a = numeric(1)
b = numeric(1)
z = numeric(1)
a = as.numeric(readline(prompt = "Insert a number"))
b = as.numeric(readline(prompt = "Insert another number"))
z = a + b
print(z)
```

1.2

Consider $X \sim \mathcal{N}(x; \mu = 2.5, \sigma = 2.1)$. Write a program that computes $\mathbb{P}(X \leq x_0)$ with $x_0 \in \mathbb{R}$. Note that

$$\mathbb{P}(X \leq x_0) = \int_{-\infty}^{x_0} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \, dx$$

0 SOLUTION 1

```
x0 = px0 = numeric(1)
x0 = 2.3
px0 = pnorm(q = x0, mean = 2.5, sd = 2.1)
print(px0)
```

0 SOLUTION 2

```
x0 = px0 = mu = sd = numeric(1)
x0 = 2.3
mu = 2.5
sd = 2.1
px0 = integrate(f = function(x, mu, sd) {
    (1/(sd * sqrt(2 * pi))) * exp(-0.5 * ((x - mu)/sd)^2)
}, lower = -Inf, upper = x0, mu = mu, sd = sd)
print(px0)
```

1.3

Consider $X \sim \mathcal{N}(x; \mu = 2.5, \sigma = 2.1)$. Write a program that computes $\mathbb{P}(X \geq x_0)$ with $x_0 \in \mathbb{R}$. Note that

$$\mathbb{P}(X \geq x_0) = \int_{x_0}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{1}{2}\left(\frac{(x-\mu)}{\sigma}\right)^2\right) \, dx$$

0 SOLUTION 1

```r
x0 = px = numeric(1)
x0 = 1.1
px0 = 1 - pnorm(q = x0, mean = 2.5, sd = 2.1)
print(px0)
```

SOLUTION 2

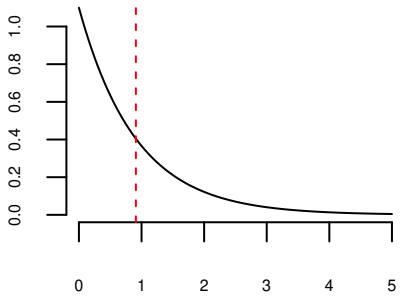
```r
x0 = px0 = mu = sd = numeric(1)
x0 = 1.1
mu = 2.5
sd = 2.1
px0 = integrate(f = function(x, mu, sd) {
    (1/(sd * sqrt(2 * pi))) * exp(-0.5 * ((x - mu)/sd)^2)
}, lower = x0, upper = Inf, mu = mu, sd = sd)
print(px0)
```

1.4

Consider $X \sim \mathcal{E}xp(x; \lambda = 1.1)$. Write a program that (i) plots the probability density function of X, (ii) computes $\mathbb{P}(X \geq x_0)$ with $x_0 \in \mathbb{R}^+$, (iii) represents graphically the quantile x_0 w.r.t. the density function.

SOLUTION

```r
x0 = px0 = numeric(1)
x0 = 0.91
curve(expr = dexp(x, rate = 1.1), from = 0, to = 5, bty = "n", xlab = "", ylab = "",
    cex.axis = 0.5)
px0 = 1 - pexp(q = x0, rate = 1.1)
abline(v = x0, lty = 2, col = "red")
```

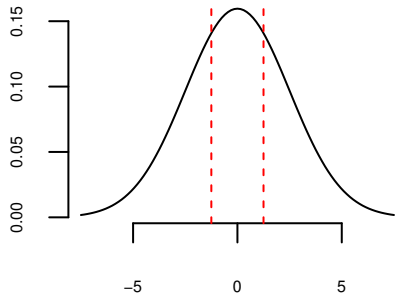


1.5

Consider $X \sim \mathcal{N}(x; \mu = 0, \sigma = 2.5)$. Write a program that (i) plots the probability density function of X, (ii) computes $\mathbb{P}(x_1 \leq X \leq x_2)$ with $(x_1, x_2) \in \mathbb{R}^2$, (iii) represents graphically the quantiles x_1 and x_2 w.r.t. the density function.

SOLUTION

```r
x1 = x2 = px0 = numeric(1)
x1 = -1.25
x2 = 1.25
curve(expr = dnorm(x, mean = 0, sd = 2.5), from = -7.5, to = 7.5, bty = "n", xlab = "",
    ylab = "", cex.axis = 0.5)
px0 = pnorm(q = x2, mean = 0, sd = 2.5) - pnorm(q = x1, mean = 0, sd = 2.5)
abline(v = c(x1, x2), lty = 2, col = "red")
```



1.6

Write an algorithm that reads two strings s_1 and s_2 as input and returns a new string s formed by concatenating s_1 and s_2.

SOLUTION

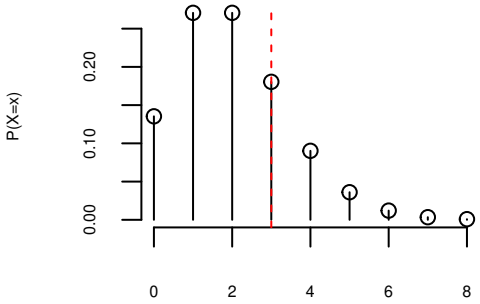
```r
s1 = s2 = s = character(1)
s1 = readline(prompt = "Write a first word")
s2 = readline(prompt = "Write a second word")
s = paste(s1, s2, sep = " ")
print(s)
```

1.7

Consider $X \sim \mathcal{P}oi(x; \lambda = 2)$. Write an algorithm that (i) plots the probability density function of X, (ii) computes $\mathbb{P}(X = x_0)$ with $x_0 \in \mathbb{N}$, (iii) represents graphically the quantile x_0 w.r.t. the density function.

SOLUTION

```r
x0 = px0 = numeric(1)
x0 = 3
plot(x = 0:8, y = dpois(x = 0:8, lambda = 2), type = "p", bty = "n", xlab = "", ylab = "P(X=x)",
    cex.axis = 0.5, cex.lab = 0.5)
segments(x0 = 0:8, y0 = 0, x1 = 0:8, y1 = dpois(x = 0:8, lambda = 2))
px0 = ppois(q = x0, lambda = 2)
abline(v = x0, lty = 2, col = "red")
```



1.8

Consider $X \sim \mathcal{N}(x; \mu = 0, \sigma = 1.5)$, $Y \sim \chi^2(y; k = 2)$ and $x_0 \in \mathbb{R}^+$. Write an algorithm that reads x_0, computes and prints out the following quantity:

$$r = \frac{\mathbb{P}(X \geq x_0)}{\mathbb{P}(Y \leq x_0)} \mathbb{P}(X \geq x_0)^2$$

SOLUTION

```
x0 = r = numeric(1)
p1 = p2 = numeric(1)
x0 = 1.26
p1 = 1 - pnorm(q = x0, mean = 0, sd = 1.5)
p2 = pchisq(q = x0, df = 2)
r = (p1/p2) * p1^2
print(r)
```

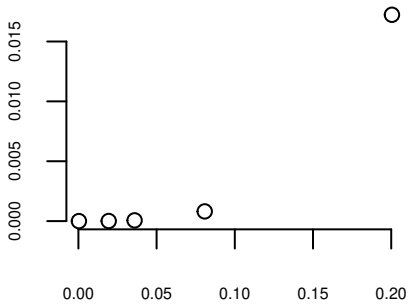
1.9

(*cont. Exercise 1.8*) Write an algorithm that (i) reads five positive real numbers $x_1, \ldots, x_5$ such that $x_1 < x_2 < \ldots < x_5$, (ii) computes for each input the quantity r, (iii) plots r as a function of $\min\left(\mathbb{P}(X \geq x_i), \mathbb{P}(Y \leq x_i)\right)$ $i = 1, \ldots, 5$.

SOLUTION

```
x1 = x2 = x3 = x4 = x5 = numeric(1)
p1 = p2 = p3 = p4 = p5 = numeric(1)
q1 = q2 = q3 = q4 = q5 = numeric(1)
r1 = r2 = r3 = r4 = r5 = numeric(1)
x1 = 1.26
x2 = 2.1
x3 = 3.1
x4 = 2.7
x5 = 5.1
p1 = 1 - pnorm(q = x1, mean = 0, sd = 1.5)
p2 = 1 - pnorm(q = x2, mean = 0, sd = 1.5)
p3 = 1 - pnorm(q = x3, mean = 0, sd = 1.5)
p4 = 1 - pnorm(q = x4, mean = 0, sd = 1.5)
p5 = 1 - pnorm(q = x5, mean = 0, sd = 1.5)
```

```r
q1 = pchisq(q = x1, df = 2)
q2 = pchisq(q = x2, df = 2)
q3 = pchisq(q = x3, df = 2)
q4 = pchisq(q = x4, df = 2)
q5 = pchisq(q = x5, df = 2)
r1 = (p1/q1) * p1^2
r2 = (p2/q2) * p2^2
r3 = (p3/q3) * p3^2
r4 = (p4/q4) * p4^2
r5 = (p5/q5) * p5^2
plot(x = c(min(p1, q1), min(p2, q2), min(p3, q3), min(p4, q4), min(p5, q5)), y = c(r1,
    r2, r3, r4, r5), type = "p", bty = "n", xlab = "", ylab = "", cex.axis = 0.5,
    cex.lab = 0.5)
```



1.10

Consider $Y \sim \mathcal{N}(y; \mu = 2.3, \sigma = 1)$ and $p_0 \in [0, 1]$. Write an algorithm that computes and prints out the quantile $y_0 = \inf\{x \in \mathbb{R} : \mathbb{P}(Y \leq y) = p_0\}$. Note that in this case the *quantile function* of a probability density function needs to be used.

SOLUTION

```r
y0 = numeric(1)
p0 = numeric(1)
p0 = 0.85
y0 = qnorm(p = p0, mean = 2.3, sd = 1)
print(y0)
# note that pnorm(q=y0,mean=2.3,sd=1) must be equal to p0
```

2 Block 2

Topics

Control structures: one-way selection, multi-way selection, for-loop

Arrays: vectors

Functions and subroutines

2.1

Write an algorithm that reads two real numbers x and y and computes the minimum between them.

SOLUTION 1

```
x = y = numeric(1)
x = readline(prompt = "Insert a number")
y = readline(prompt = "Insert another number")
if (x <= y) {
    z = x
} else {
    z = y
}
print(paste("The minimum is", z, sep = " "))
```

SOLUTION 2

```
x = y = numeric(1)
x = readline(prompt = "Insert a number")
y = readline(prompt = "Insert another number")
z = min(x, y)
print(paste("The minimum is", z, sep = " "))
```

2.2

Write an algorithm that reads two real numbers x and y and executes the following instructions:

1. if $\lfloor \min(x, y) \rfloor$ is an even number then evaluates the expression $z = x^{|y|}$
2. if $\lfloor \min(x, y) \rfloor$ is an odd number then evaluates the expression $z = x^{|y-x|}$
3. outputs the value of z

Note that $\lfloor x \rfloor$ can be computed using the primitive function `floor(..)`, the absolute value $|x|$ can be computed by means of the primitive `abs(..)`, whereas to verify whether a number is odd or even, one can use the command `x%%2` which gives as output `True` if the number is even.

SOLUTION

```
z = x = y = numeric(1)
cond = logical(1)
x = 2.1
y = 1.1
cond = floor(min(x, y))%%2
if (cond == 1) {
    # even number
    z = x^abs(y)
} else {
```

```
    z = x^abs(y - x)
}
print(z)
```

2.3

Write an algorithm that reads two real numbers x and y and computes the following expression:

$$z = \max(|x|, |y|) - \min(x, y) + (x - y)^2$$

If $z < 0$ then ask for a third number q and print z only if $z > q$.

SOLUTION

```
x = y = z = q = numeric(1)
x = readline(prompt = "Insert a first number")
y = readline(prompt = "Insert a second number")
z = max(abs(x), abs(y)) - min(x, y) + (x - y)^2
if (z < 0) {
    q = readline(prompt = "Insert a third number")
}
if (z > q) {
    print(z)
}
```

2.4

Write an algorithms that reads two intervals $i_1 = [a, b]$ and $i_2 = [c, d]$ and evaluates the following instructions:

- if $i_1 \subset i_2$ then computes $h = (\max(a, b) - \min(a, b))\,\frac{1}{j}$, where $j = \frac{a+b}{2}$
- if $i_1 \not\subset i_2$ then computes $h = (\max(a, b) - \min(a, b))\,\frac{1}{j}$, where $j = \frac{a}{|b|+\epsilon}$ and $\epsilon = \max(a, b, c, d)^{\frac{1}{3}}$
- if $i_1 = i_2$ then computes $h = (\max(a, b) - \min(a, b))$
- outputs h

SOLUTION 1

```
# Note: we are not declaring variables/constants
a = 1.3
b = 2
c = 1.6
d = 2.3
if (a > c & b < d) {
    # i1 is included into i2
    h = (max(a, b) - min(a, b)) * ((a + b)/2)
} else if (a < c & b > d) {
    # i1 is not included into i2
    h = (max(a, b) - min(a, b)) * (a/(abs(b) + max(a, b, c, d)^(1/3)))
} else if (a == c & b == d) {
    # i1 is equal to i2
    h = (max(a, b) - min(a, b))
}
print(h)
```

SOLUTION 2

```r
# Note: we are not declaring variables/constants
hfun = function(x = 0, y = 0, z = 0, w = 0, type = 1) {
    j = switch(type, `1` = ((x + y)/2), `2` = (x/(abs(y) + max(x, y, z, w)^(1/3))),
        `3` = 1)
    return(j)
}

a = 1.3
b = 2
c = 1.6
d = 2.3
if (a > c & b < d) {
    # i1 is included into i2
    h = (max(a, b) - min(a, b)) * hfun(a, b, c, d, type = 1)
} else if (a < c & b > d) {
    # i1 is not included into i2
    h = (max(a, b) - min(a, b)) * hfun(a, b, c, d, type = 2)
} else if (a == c & b == d) {
    # i1 is equal to i2
    h = (max(a, b) - min(a, b)) * hfun(a, b, c, d, type = 3)
}
print(h)
```

2.5

Rewrite Exercise 1.9 using vectors and for-loops. In this case, $x_i \sim \mathcal{U}(x; 0.5, 5),\ i = 1, \ldots, 5$.

SOLUTION 1

```r
n = numeric(1)
n = 5
p = q = r = x = numeric(n)
set.seed(1211)  #fix the seed of the random number generator
x = runif(n = n, min = 0.5, max = 5)
for (i in 1:n) {
    p[i] = 1 - pnorm(q = x[i], mean = 0, sd = 1.5)
    q[i] = pchisq(q = x[i], df = 2)
    r[i] = (p[i]/q[i]) * p[i]^2
}
plot(x = apply(cbind(p, q), 1, min), y = r, type = "p", xlab = "", ylab = "", bty = "n",
    cex.axis = 0.5, cex.lab = 0.5)
```

SOLUTION 2

```r
set.seed(1211)  #fix the seed of the random number generator
x = runif(n = 5, min = 0.5, max = 5)
p = 1 - pnorm(q = x, mean = 0, sd = 1.5)
q = pchisq(q = x, df = 2)
r = (p/q) * p^2
plot(x = apply(cbind(p, q), 1, min), y = r, type = "p", xlab = "", ylab = "", bty = "n",
    cex.axis = 0.5, cex.lab = 0.5)
```

2.6

Write an algorithm that reads a $I \times 1$ vector of real numbers $\mathbf{x} = (x_1, \ldots, x_i, \ldots, x_I)$ and gives as output the vector $\mathbf{y} = (x_I, \ldots, x_i, \ldots, x_1)$. Note $\mathbf{x}$ can be generated uniformly on $[-2, 2]$.

SOLUTION

```
set.seed(1212)
I = numeric(1)
I = 10
x = y = numeric(I)
x = runif(n = I, min = -2, max = 2)
for (i in 1:I) {
    y[i] = x[I - i + 1]
}
```

2.7

Write an algorithm that reads two natural numbers n_a and n_b and determines the vector $\mathbf{x}$ containing the sequence of numbers included between n_a and n_b.

SOLUTION 1

```
n_a = 2
n_b = 8
if (n_b > n_a) {
    n = n_b - n_a
    x = numeric(n)
    k = 0
    for (i in n_a:n_b) {
        k = k + 1
        x[k] = i
    }
    print(x)
} else {
    print("Note that the condition na<nb must hold")
}
```

SOLUTION 2

```
n_a = 2
n_b = 8
x = seq(from = n_a, to = n_b, by = 1)
print(x)
```

2.8

Write an algorithm that given the array $\mathbf{x}_{n\times 1}$ of real positive numbers, evaluates the following expressions:

1. $x^* = \sum_{i=1}^{n} x_i$
2. $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
3. $x^\dagger = x^*/(\max(\mathbf{x}) - \min(\mathbf{x}))$
4. $z = (x^* - 2x^\dagger)$
5. write as output z

Note that x_i can be generated uniformly in a given interval.

SOLUTION 1

```
set.seed(1232)
n = 100
x = runif(n = n, min = 0, max = 10)
# 1.
x1 = 0
for (i in 1:n) {
    x1 = x1 + x[i]
}
# 2.
x2 = 1/n * x1
# 3.
x3 = x1/(max(x) - min(x))
# 4.
z = x1 - 2 * x3

print(z)
```

SOLUTION 2

```
set.seed(1232)
n = 100
x = runif(n = n, min = 0, max = 10)
x1 = sum(x)  #1
x2 = mean(x)  #2
x3 = x1/(max(x) - min(x))  #3
z = x1 - 2 * x3  #4
print(z)
```

2.9

Write an algorithm that given a vector of integers $\mathbf{n}_{I\times 1}$, computes the vector $\mathbf{f}_{I\times 1}$ containing the frequencies of the elements of $\mathbf{n}_{I\times 1}$. Note: To compute frequencies, write a function that counts how many times the elements of $\mathbf{n}$ occur. The vector $\mathbf{x}$ can be generated via Poisson random numbers generator.

SOLUTION 1

```
count_integers = function(x) {
    z = sort(unique(x))
    n = length(z)
    f = numeric(n)
    for (i in 1:n) {
        f[i] = sum(x == z[i])
    }
    return(cbind(z, f))
}

set.seed(1232)
n = 100
x = rpois(n = n, lambda = 2.3)  #pay attention: x may contain replicates!
out = count_integers(x)
print(out)
```

SOLUTION 2

```
set.seed(1232)
n = 100
x = rpois(n = n, lambda = 2.3)  #pay attention: x may contain replicates!
out = table(x)
print(out)
```

2.10

(*cont. Exercise 2.9*) Write an algorithm that given as input the vector of frequencies $\mathbf{f}_{I\times 1}$, computes the vector $\mathbf{s}_{I\times 1}$ where the generic element s_i is defined as follows:

$$s_i = s_{i-1} + f_i$$

Note that $s_1 = f_1$ by definition.

SOLUTION 1

```
set.seed(1232)
n = 100
x = rpois(n = n, lambda = 2.3)
out = table(x)
f = as.numeric(out)
s = numeric(length(f))
s[1] = f[1]
for (i in 2:length(f)) {
    s[i] = s[i - 1] + f[i]
}
print(s)
```

SOLUTION 2

```
set.seed(1232)
n = 100
x = rpois(n = n, lambda = 2.3)
out = table(x)
f = as.numeric(out)
s = cumsum(f)
print(s)
```

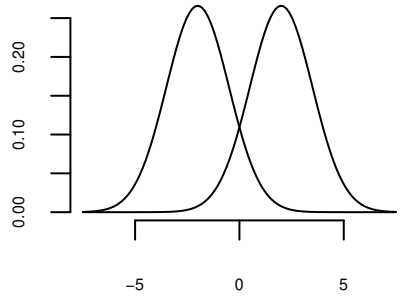
2.11

Consider $X \sim \mathcal{N}(x; \mu = -2, \sigma = 1.5)$ and $Y \sim \mathcal{N}(y; \mu = 2, \sigma = 1.5)$. Write an algorithm that plots the densities $f_X(x; \mu_x, \sigma_x)$ and $f_Y(x; \mu_y, \sigma_y)$ and computes the degree of overlapping between them:

$$p_{XY} = \int_{-\infty}^{+\infty} \min_z \left(f_X(z; \mu_x, \sigma_x), f_Y(z; \mu_y, \sigma_y)\right)$$

SOLUTION

```
p_xy = numeric(1)
curve(expr = dnorm(x, mean = -2, sd = 1.5), from = -7.5, to = 7.5, bty = "n", xlab = "",
    ylab = "", cex.axis = 0.5)
curve(expr = dnorm(x, mean = 2, sd = 1.5), from = -7.5, to = 7.5, bty = "n", xlab = "",
    ylab = "", cex.axis = 0.5, add = TRUE)
```




```r

p_xy = integrate(f = function(x) {
    apply(cbind(dnorm(x, mean = -2, sd = 1.5), dnorm(x, mean = 2, sd = 1.5)), 1,
        min)
}, lower = -Inf, upper = Inf)
```

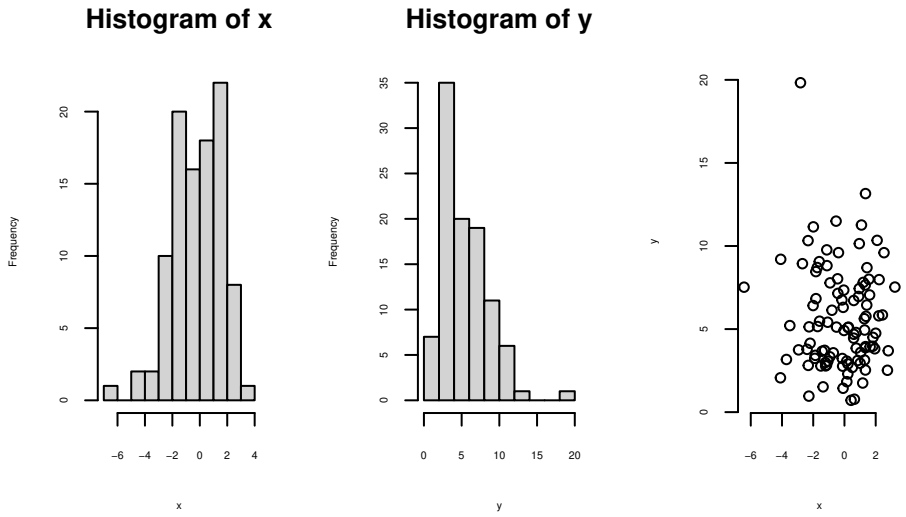
2.12

Consider two vectors $\mathbf{x}_{n\times 1}$ and $\mathbf{y}_{n\times 1}$ where $x_i \sim \mathcal{N}(x; \mu = 0, \sigma = 2.0)$ and $y_i \sim \chi^2(y; k = 6)$. Write an algorithm that (i) plots the distribution of the input vectors, (ii) produces the scatter plot of the two vectors, and (iii) computes the following quantity:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

SOLUTION 1

```r
set.seed(121)
n = 100
x = rnorm(n = n, mean = 0, sd = 2)
y = rchisq(n = n, df = 6)
par(mfrow = c(1, 3))
hist(x, bty = "n", cex.axis = 0.5, cex.lab = 0.5)
hist(y, bty = "n", cex.axis = 0.5, cex.lab = 0.5)
plot(x, y, bty = "n", cex.axis = 0.5, cex.lab = 0.5)
```




```
r = sum((x - mean(x)) * (y - mean(y)))/sqrt(sum((x - mean(x))^2) * sum((y - mean(y))^2))
```

SOLUTION 2

```
set.seed(121)
n = 100
x = rnorm(n = n, mean = 0, sd = 2)
y = rchisq(n = n, df = 6)
par(mfrow = c(1, 3))
hist(x, bty = "n", cex.axis = 0.5, cex.lab = 0.5)
hist(y, bty = "n", cex.axis = 0.5, cex.lab = 0.5)
plot(x, y, bty = "n", cex.axis = 0.5, cex.lab = 0.5)
r = cor(x, y)
```

2.13

Consider $X \sim \mathcal{U}(x; a, b)$ with $a = 0.5$ and $b = 3.4$. Write an algorithm that

1. Generates a vector $\mathbf{x}_{n\times 1}$ containing random samples from X with $n = 100$
2. Computes and prints out the mean and the variance of $\mathbf{x}$
3. Compares the empirical mean and variance with the true mean and variance[1] and indicates whether these empirical moments are greater or less then the true moments
4. Computes the probability $\mathbb{P}(X \leq 1.7)$ using $\mathbf{x}$ and compares it with the true probability calculated using the cumulative density function of the Uniform distribution
5. Computes $U = 1/(X + 1)$
6. Compares the histograms of X and U graphically

SOLUTION

```
set.seed(121)
n = 1000
# 1.
x = runif(n = n, min = 0.5, max = 3.4)
# 2.
m_x = mean(x)
v_x = var(x)
```

[1] https://en.wikipedia.org/wiki/Continuous_uniform_distribution

```r
print(c(m_x, v_x))
# 3.
m0 = (1/2) * (0.5 + 3.4)
v0 = (1/12) * (3.4 - 0.5)^2
m_out = ifelse(m_x > m0, "greater", "lesser")
v_out = ifelse(v_x > v0, "greater", "lesser")
out = cbind(m_out, v_out)
colnames(out) = c("true mean", "true variance")
print(out)
# 4.
px = sum(x <= 1.7)/n
p0 = punif(q = 1.7, min = 0.5, max = 3.4)
out = cbind(px, p0)
colnames(out) = c("obs prob", "true prob")
# 5.
u = 1/(x + 1)
# 6.
par(mfrow = c(1, 2))
hist(x, bty = "n", xlab = "", ylab = "", cex.axis = 0.5, cex.lab = 0.5)
hist(u, bty = "n", xlab = "", ylab = "", cex.axis = 0.5, cex.lab = 0.5)
```

2.14

Write an algorithm that (i) creates $n = 10000$ random m-grams of $m = 5$ letters from the English alphabet (26 letters) and (ii) computes the probability that the generated m-grams contain the word "nn". Note: Consider the case where letters are uniformly generated.

SOLUTION

```r
set.seed(121)
n = 10000
m = 5
x = character(n)
for (i in 1:n) {
    iid = floor(runif(n = m, min = 1, max = 26 + 1))
    # alternatively, you may use: sample(x=1:26,size=m,replace=TRUE)
    x[i] = paste(letters[iid], collapse = "")
}
iid = grep(x = x, pattern = "nn")
print(length(iid)/n)
print(x[iid])
```

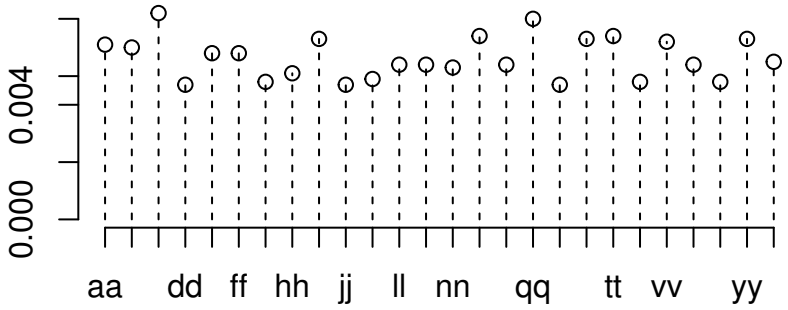
2.15

(Cont. Exercise 2.14) Compute the probability distribution that uniformly generated m-grams contain two consecutive letters (e.g., "aa","bb",...,"zz").

SOLUTION

```r
set.seed(121)
n = 10000
m = 5
x = character(n)
for (i in 1:n) {
    iid = floor(runif(n = m, min = 1, max = 26 + 1))
    x[i] = paste(letters[iid], collapse = "")
}
z = apply(X = cbind(letters, letters), MARGIN = 1, FUN = function(x) {
```

```
    paste(x, collapse = "")
})
k = length(z)
pz = numeric(k)
for (j in 1:k) {
    iid = grep(x = x, pattern = z[j])
    pz[j] = length(iid)/n
}
plot(x = 1:k, y = pz, type = "p", bty = "n", xlab = "", ylab = "", cex.axis = 0.5,
    cex.lab = 0.5, ylim = c(0, max(pz)), axes = FALSE)
segments(x0 = 1:k, y0 = 0, x1 = 1:k, y1 = pz, lty = 2)
axis(side = 1, at = 1:k, labels = z)
axis(side = 2, at = round(seq(from = 0, to = max(pz), length.out = 5), 3))
```



3 Block 3

Topics

Control structures: while-loop

Arrays: matrices

Lists, dataframes

3.1

Write an algorithm that computes the square root of a number $s > 0$ using the Heron's approximation:

$$\sqrt{s} = \lim_{n \to \infty} x_n$$

where the generic element x_n is defined as:

$$x_n = \frac{1}{2}\left(x_{n-1} + \frac{s}{x_{n-1}}\right)$$

with $x_1 = 1.5$. The iterative approximation must be executed until the condition $(x_n - x_{n-1}) < \epsilon$ is met, with ϵ being a user's defined small real number (e.g., $\epsilon = 0.001$).

0 SOLUTION

```
s = 17
eps = 1e-04
cond = 0
x = 1.5
n = 1
while (cond == 0) {
    n = n + 1
    x = c(x, 0.5 * (x[n - 1] + s/x[n - 1]))
    if (abs(x[n] - x[n - 1]) < eps) {
        cond = 1
    }
}
print(x[length(x)])  #it approximates sqrt(s)
```

3.2

Write an algorithm that given a $N \times M$ matrix $\mathbf{X}$ with $N = M$, computes the following quantities:

1. $q = \sum_{i=1}^{N} \text{diag}(\mathbf{X})_i + \max(\mathbf{X})$
2. $p = \prod_{i=1}^{N} \text{diag}(\mathbf{X})_i + \min(\mathbf{X})$
3. if $q > p$ outputs p, otherwise outputs q

Note that elements of X can be uniformly generated random numbers in a given interval.

0 SOLUTION 1

```
extract_diag = function(X) {
    n = dim(X)[1]
    m = dim(X)[2]
    d = numeric(n)
    for (i in 1:n) {
        for (j in 1:m) {
```

```
            if (i == j) {
                d[i] = X[i, j]
            }
        }
    }
    return(d)
}

set.seed(111)
n = m = 10
X = matrix(data = runif(n = n * m, min = -2, max = 2), nrow = n, ncol = m)
d = extract_diag(X)
q = sum(d) + max(X)
p = prod(d) + min(X)
if (q > p) {
    print(p)
} else {
    print(q)
}
```

SOLUTION 2

```
set.seed(111)
n = m = 10
X = matrix(data = runif(n = n * m, min = -2, max = 2), nrow = n, ncol = m)
d = diag(X)
q = sum(d) + max(X)
p = prod(d) + min(X)
if (q > p) {
    print(p)
} else {
    print(q)
}
```

3.3

Write an algorithm that reads two numbers $N > 0$ and K, then finds a square matrix $\mathbf{A}_{N \times N}$ such that the sum of elements in every row and column is equal to K. Note: (a) A can be populated using uniformly generated discrete numbers and (b) K should be large enough.

SOLUTION

```
set.seed(112)
N = 10
K = 100
found = 0
maxIter = 1000
j = 0
while (found == 0) {
    j = j + 1
    A = matrix(data = floor(runif(n = N * N, min = 0, max = 5)), nrow = N, ncol = N)
    if (sum(A) == K) {
        found = 1
    }
    if (j > maxIter) {
        print("maxIter exceeded!")
        break
    }
}
```

3.4

Write an algorithm that given a matrix of reals $\mathbf{X}_{I\times I}$, computes the following quantities:

1. $a = \sum_{i=1}^{I}\sum_{j=1}^{J}\mathbf{X}_{i,j}$ with $i \neq j$
2. $b = \sum_{l\in\mathcal{L}}\sum_{h\in\mathcal{H}}\mathbf{X}_{l,h}$ with $\mathcal{L}$ being the set of *even* indices and $\mathcal{H}$ the set of *odd* indices
3. computes $q = a + b$ if $a > b$, otherwise $q = \frac{1}{2}(a-b)^2$
4. outputs q

0 SOLUTION

```
compute_a = function(X) {
    n = NROW(X)
    m = NCOL(X)
    d = 0
    for (i in 1:n) {
        for (j in 1:m) {
            if (i != j) {
                d = d + X[i, j]
            }
        }
    }
    return(d)
}

compute_b = function(X) {
    n = NROW(X)
    m = NCOL(X)
    d = 0
    iid = seq(from = 1, to = n, by = 1)
    L = iid[iid%%2 == 0]  #even
    H = iid[iid%%2 == 1]  #odd
    for (l in L) {
        for (h in H) {
            d = d + X[l, h]
        }
    }
    return(d)
}

set.seed(112)
I = 10
X = matrix(data = runif(n = I * I, min = 0, max = 8), nrow = N, ncol = I)
a = compute_a(X)
b = compute_b(X)
q = ifelse(a > b, a + b, 0.5 * (a - b)^2)
print(q)
```

3.5

Write an algorithm that given a matrix of reals $\mathbf{X}_{N\times N}$ and an integer N, sorts diag($\mathbf{X}$) in ascending order.

0 SOLUTION 1

```
sort_desc = function(x) {
    n = length(x)
    for (i in 1:(n - 1)) {
        for (j in (i + 1):n) {
```

```
            if (x[i] >= x[j]) {
                c = x[i]
                x[i] = x[j]
                x[j] = c
            }
        }
    }
    return(x)
}

set.seed(112)
N = 10
X = matrix(data = runif(n = N * N, min = 0, max = 5), nrow = N, ncol = N)
x = diag(X)
y = sort_desc(x)
print(y)
```

SOLUTION 2

```
set.seed(112)
N = 10
X = matrix(data = runif(n = N * N, min = 0, max = 5), nrow = N, ncol = N)
x = diag(X)
y = sort(x, decreasing = FALSE)
print(y)
```

3.6

Write an algorithm that (i) finds a matrix $\mathbf{Y}_{n\times n}$ such that $\sum_{i=1}^{n} \text{diag}(\mathbf{Y})_i > \frac{1}{2}(a+b)$ with $Y_{ij} \sim \mathcal{U}(y; a, b)$, $a < b$, (ii) computes the sum of the elements of $\mathbf{Y}$ which are less then the quantity $b - a$.

SOLUTION 1

```
set.seed(112)
n = 5
a = 1
b = 8
Y = matrix(data = NA, nrow = n, ncol = n)
found = 0
while (found == 0) {
    for (i in 1:n) {
        for (j in 1:n) {
            Y[i, j] = runif(n = 1, min = a, max = b)
        }
    }
    q = 0.5 * (a + b)
    if (sum(diag(Y)) > q) {
        found = 1
    }
}

s = 0
for (i in 1:n) {
    for (j in 1:n) {
        if (Y[i, j] < (b - a))
            s = s + Y[i, j]
    }
}
```

SOLUTION 2

```r
set.seed(112)
n = 5
a = 1
b = 8
found = 0
while (found == 0) {
    Y = matrix(data = runif(n = n * n, min = a, max = b), nrow = n, ncol = n)
    q = 0.5 * (a + b)
    if (sum(diag(Y)) > q) {
        found = 1
    }
}

s = sum(Y[Y < (b - a)])
```

3.7

Write an algorithm that

1. finds $K > 0$ matrices $\mathbf{Y}_{n\times n}$ such that $\sum_{i<j} Y_{ij} > \frac{1}{2}(a+b)$ with $Y_{ij} \sim \mathcal{U}(y; a, b)$, $a < b$
2. saves the K matrices into a list of K elements
3. computes $\mathbb{P}(\sum_{i=1}^{n} \text{diag}(\mathbf{Y}^{(k)})_i > (\frac{a+b}{2})^2)$

SOLUTION 1

```r
found_matrix = function(a = 0, b = 1, nrow = NULL, ncol = NULL, seed = NULL) {
    if (!is.null(seed)) {
        set.seed(seed)
    }
    found = 0
    while (found == 0) {
        Y = matrix(data = runif(n = nrow * ncol, min = a, max = b), nrow = nrow,
            ncol = ncol)
        q = 0.5 * (a + b)
        if (sum(Y[lower.tri(Y)]) > q) {
            found = 1
        }
    }
    return(Y)
}

set.seed(121)
n = 5
a = 0
b = 10
K = 100
# 1. & 2.
list_matrices = list()
for (k in 1:K) {
    list_matrices[[k]] = found_matrix(a, b, n, n)
}
# 3.
c = 0
for (k in 1:K) {
    c = c + as.numeric(sum(diag(list_matrices[[k]])) > ((a + b)/2)^2)
}
prob_c = c/K
print(prob_c)
```

SOLUTION 2

```r
found_matrix = function(a = 0, b = 1, nrow = NULL, ncol = NULL, seed = NULL) {
    if (!is.null(seed)) {
        set.seed(seed)
    }
    found = 0
    while (found == 0) {
        Y = matrix(data = runif(n = nrow * ncol, min = a, max = b), nrow = nrow,
            ncol = ncol)
        q = 0.5 * (a + b)
        if (sum(Y[lower.tri(Y)]) > q) {
            found = 1
        }
    }
    return(Y)
}

set.seed(121)
n = 5
a = 0
b = 10
K = 100
# 1. & 2.
list_matrices = list()
for (k in 1:K) {
    list_matrices[[k]] = found_matrix(a, b, n, n)
}
# 3.
c = sum(unlist(lapply(X = list_matrices, FUN = function(x) {
    as.numeric(sum(diag(x)) > ((a + b)/2)^2)
})))
prob_c = c/K
print(prob_c)
```

3.8

Write an algorithm that given a matrix $\mathbf{X}_{I\times J}$, (i) computes the global maximum c columnwise, (ii) computes the global minimum r rowwise, (iii) prints out 0 if $c > r$, 1 otherwise.

SOLUTION 1

```r
set.seed(122)
I = 25
J = 10
X = matrix(data = rpois(n = I * J, lambda = 5.6), nrow = I, ncol = J)
c = numeric(J)
for (j in 1:J) {
    c[j] = max(X[, j])
}
c = max(c)

r = numeric(I)
for (i in 1:I) {
    r[i] = max(X[i, ])
}
r = min(r)
if (c > r) {
    print(1)
} else {
    print(0)
}
```

SOLUTION 2

```
set.seed(122)
I = 25
J = 10
X = matrix(data = rpois(n = I * J, lambda = 5.6), nrow = I, ncol = J)
c = max(apply(X = X, MARGIN = 2, FUN = max))
r = min(apply(X = X, MARGIN = 1, FUN = min))
if (c > r) {
    print(1)
} else {
    print(0)
}
```

3.9

Write an algorithm that (i) finds a matrix $\mathbf{X}_{I\times J}$ $(I > J)$ such that $s = \left(\sum_{j=1}^{J} s_j\right) < \epsilon$, $(\epsilon = 1e^{-03})$ with $s_j = \frac{1}{I}\sum_{i=1}^{I} x_{ij}$ for each $j = 1, \ldots, J$ (i.e., column-wise mean), (ii) counts how many times q such constrain is not satisfied.

SOLUTION 2

```
set.seed(122)
I = 25
J = 10
found = 0
maxIter = 1000
X = matrix(data = NA, nrow = I, ncol = J)
k = 0
while (found == 0) {
    k = k + 1
    for (j in 1:J) {
        X[, j] = rnorm(n = I, mean = 0, sd = 1)
    }
    s = sum(apply(X = X, MARGIN = 2, FUN = mean))
    if (s < 0.001 | k > maxIter) {
        found = 1
    }
}
if (k > maxIter) {
    print("Iterations exceeded maxIter!")
}
q = k
print(q)
```

3.10

Write an algorithm that

1. generates a vector $\mathbf{x}_{n\times 1}$ with $x_i \sim \mathcal{P}oi(x; \lambda = 1.25)$
2. generates a vector $\mathbf{s}_{n\times 1}$ of uniformly distributed letters (consider the 26 letters of the English alphabet)
3. generates a vector $\mathbf{b}_{n\times 1}$ with $b_i \sim \mathcal{B}ern(b; \pi = 0.5)$
4. saves $\mathbf{x}$, $\mathbf{s}$, and $\mathbf{b}$ into a new dataframe $\mathcal{D}$
5. computes $\frac{1}{n}\sum_{i=1}^{n} x_i$ conditioned on the value $b = 0$
6. computes $\frac{1}{n}\sum_{i=1}^{n} x_i^2$ conditioned on the value $b = 0$ and $s = \text{a}$

7. computes $\frac{1}{2n}\sum_{i=1}^{n} x_i$ conditioned on the value $b = 1$ and $s \neq$ c
8. compares **x** w.r.t. the levels of **b** graphically (i.e., boxplot)
9. compares **x** w.r.t. $s \neq$ c and $s \neq$ d graphically (i.e., boxplot)

SOLUTION

```r
set.seed(124)
n = 100
x = rpois(n = n, lambda = 1.25)
s = sample(x = letters, size = n, replace = TRUE)
b = rbinom(n = n, size = 1, prob = 0.5)
D = data.frame(x, s, b)
# 5.
x1 = mean(D$x[D$b == 0])
# 6.
x2 = 1/n * sum((D$x[D$b == 0 & D$s == "a"])^2)
# 7.
x2 = 1/(2 * n) * sum(D$x[D$b == 0 & D$s != "c"])
# 8.
boxplot(D$x[D$s != "c"], D$x[D$s != "d"])
```

4 Block 4

4.1

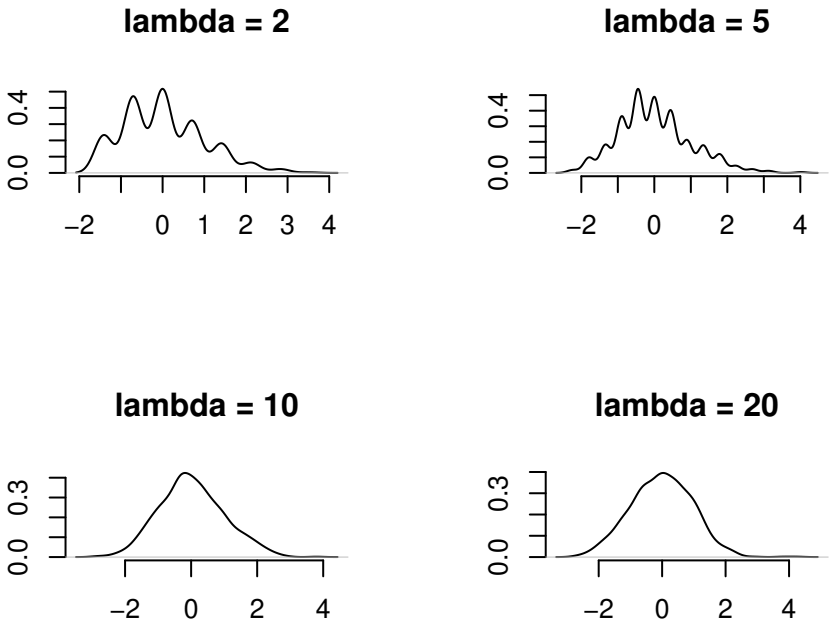
Write an algorithm that approximates and plots the distribution of the following random variable:

$$Z = \frac{Y - \lambda}{\sqrt{\lambda}}$$

where $Y \sim \mathcal{P}oi(y; \lambda)$ and $\lambda > 0$. Note that (i) a Monte Carlo based solution should be used with B large enough and (ii) several values of λ should be considered.

SOLUTION

```r
set.seed(122)
lambda0 = c(2, 5, 10, 20)
B = 1000
Z = matrix(data = NA, nrow = B, ncol = length(lambda0))
for (j in 1:length(lambda0)) {
    x = rpois(n = B, lambda = lambda0[j])
    Z[, j] = (x - lambda0[j])/sqrt(lambda0[j])
}

par(mfrow = c(2, 2))
for (j in 1:length(lambda0)) {
    plot(density(Z[, j]), bty = "n", xlab = "", ylab = "", main = paste0("lambda = ",
        lambda0[j]))
}
```



4.2

Write a Monte Carlo algorithm that computes the following integral: $\int_a^b x^4 \sin x \, dx$ with $a = 0$ and $b = 3$.

SOLUTION

```r
set.seed(121)
B = 2500
a = 0
b = 3
x = runif(n = B, min = a, max = b)
z = x^4 * sin(x)
out = mean(z) * (b - a)
print(out)
```

4.3

Write a Monte Carlo algorithm that computes the following integral: $\int_1^3 \int_1^4 \cos(x^2 \sin y)\ dx\ dy$.

SOLUTION

```r
set.seed(121)
B = 5000
x = runif(n = B, min = 1, max = 3)
y = runif(n = B, min = 1, max = 4)
z = cos(x^2 * sin(y))
k = (3 - 1) * (4 - 1)
out = mean(z) * k
print(out)

# The same integral can be computed using numerical bivariate quadrature as
# follows:
cubature::adaptIntegrate(f = function(x) {
    cos(x[1]^2 * sin(x[2]))
}, lowerLimit = c(1, 1), upperLimit = c(3, 4))
```

4.4

Write a Monte Carlo algorithm that computes the following integral: $\int_0^3 \int_0^1 \cos(x - y)\ dx\ dy$.

SOLUTION

```r
set.seed(121)
B = 5000
x = runif(n = B, min = 0, max = 3)
y = runif(n = B, min = 0, max = 1)
z = cos(x - y)
k = (3 - 0) * (1 - 0)
out = mean(z) * k
print(out)
```

4.5

Write a Monte Carlo algorithm that (i) estimates the mean $\mu \in \mathbb{R}^3$ of the model $\mathcal{N}_3(\mathbf{x}; \boldsymbol{\mu}_{3\times 1}, \boldsymbol{\Sigma}_{3\times 3})$ given a sample of iid data $\mathbf{Y}_{n\times 3}$ and (ii) plots the results of the Monte Carlo procedure. Note that data should be generated in advance.

SOLUTION

```r
set.seed(121)

## Generate data Y
```

```r
n = 80
mu0 = c(-2.1, 0.7, 1.3)  #mean of the true model
Sigma0 = diag(c(1, 1.7, 0.6))  #covariance matrix of the true model
Y = mvtnorm::rmvnorm(n = n, mean = mu0, sigma = Sigma0)

## Metropolis algorithm
B = 5000
Theta = matrix(data = NA, nrow = B, ncol = length(mu0))
Theta[1, ] = c(1, 1, 1)  #starting point
S0 = diag(rep(0.025, length(mu0)))  #covariance matrix of the proposal distribution
for (b in 2:B) {
    # proposal distribution: Multivariate Normal
    theta_temp = mvtnorm::rmvnorm(n = 1, mean = Theta[b - 1, ], sigma = S0)

    log_r = sum(log(mvtnorm::dmvnorm(x = Y, mean = theta_temp, sigma = Sigma0))) -
        sum(log(mvtnorm::dmvnorm(x = Y, mean = Theta[b - 1, ], sigma = Sigma0)))  #acceptance ratio

    log_c = log(runif(n = 1, min = 0, max = 1))

    if (log_c < log_r) {
        Theta[b, ] = theta_temp
    } else {
        Theta[b, ] = Theta[b - 1, ]
    }
}

## Get posterior means via marginal distributions
mu_est = apply(X = Theta, MARGIN = 2, FUN = mean)
print(mu_est)

## Plot results
par(mfrow = c(2, 3))
plot(1:B, Theta[, 1], type = "l", bty = "n", xlab = "", ylab = "mu1")
plot(1:B, Theta[, 2], type = "l", bty = "n", xlab = "", ylab = "mu2")
plot(1:B, Theta[, 3], type = "l", bty = "n", xlab = "", ylab = "mu3")
hist(x = Theta[, 1], xlab = "", ylab = "", main = "mu1")
abline(v = mean(Theta[, 1]), col = 2, lty = 2, lwd = 2)
hist(x = Theta[, 2], xlab = "", ylab = "", main = "mu2")
abline(v = mean(Theta[, 2]), col = 2, lty = 2, lwd = 2)
hist(x = Theta[, 3], xlab = "", ylab = "", main = "mu3")
abline(v = mean(Theta[, 3]), col = 2, lty = 2, lwd = 2)
```

4.6

Write a Monte Carlo algorithm that (i) estimates the parameter $\pi \in [0, 1]$ of the model $\mathcal{B}ern(y; \pi)$ given a sample of iid data $\mathbf{y}_{n\times 1}$ and (ii) plots the results of the Monte Carlo procedure. To simplify the procedure, the parameter π can be transformed via $\pi^* = \pi/(1 - \pi)$ so that $\pi^* \in \mathbb{R}$ and $\pi = \text{plogis}(\pi^*)$. Note that data should be generated in advance.

SOLUTION

```r
set.seed(112)

logit = function(x) {
    z = log(x/(1 - x))
    return(z)
}

## Generate data Y
n = 120
pi0 = 0.78
```

```r
y = rbinom(n = n, size = 1, prob = pi0)

## Metropolis algorithm
B = 5000
theta = matrix(data = NA, nrow = B, ncol = 1)
theta[1, ] = logit(0.5)  #starting point
s0 = 0.2  #variance of the proposal distribution
for (b in 2:B) {
    # proposal distribution: Multivariate Normal
    theta_temp = rnorm(n = 1, mean = theta[b - 1, ], sd = s0)

    log_r = sum(log(dbinom(x = y, size = 1, prob = plogis(theta_temp)))) - sum(log(dbinom(x = y,
        size = 1, prob = plogis(theta[b - 1, ]))))  #acceptance ratio

    log_c = log(runif(n = 1, min = 0, max = 1))

    if (log_c < log_r) {
        theta[b, ] = theta_temp
    } else {
        theta[b, ] = theta[b - 1, ]
    }
}

## Get posterior means via marginal distributions
pi_est = plogis(mean(theta))

## Plot results
par(mfrow = c(1, 3))
plot(1:B, theta, type = "l", bty = "n", xlab = "", ylab = "pi*")
hist(x = theta, xlab = "", ylab = "", main = "pi*")
abline(v = mean(theta), col = 2, lty = 2, lwd = 2)
hist(x = plogis(theta), xlab = "", ylab = "", main = "pi")
abline(v = plogis(mean(theta)), col = 2, lty = 2, lwd = 2)
```

4.7

Write a Monte Carlo algorithm that (i) estimates the parameter $\lambda \in [0, +\infty)$ of the model $\mathcal{E}xp(y; \lambda)$ given a sample of iid data $\mathbf{y}_{n\times 1}$ and (ii) plots the results of the Monte Carlo procedure. Note that (a) data should be generated in advance and (b) a suitable transformation $g : [0, +\infty) \to \mathbb{R}$ should be used for λ.

SOLUTION

```r
## Note that if g() := log(), then g()^-1 := exp()
set.seed(114)

## Generate data Y
n = 80
lambda0 = 0.28
y = rexp(n = n, rate = lambda0)

## Metropolis algorithm
B = 5000
theta = matrix(data = NA, nrow = B, ncol = 1)
theta[1, ] = exp(0)  #starting point
s0 = 0.2  #variance of the proposal distribution
for (b in 2:B) {
    # proposal distribution: Multivariate Normal
    theta_temp = rnorm(n = 1, mean = theta[b - 1, ], sd = s0)

    log_r = sum(log(dexp(x = y, rate = exp(theta_temp)))) - sum(log(dexp(x = y, rate = exp(theta[b -
```

```
        1, ])))))  #acceptance ratio

    log_c = log(runif(n = 1, min = 0, max = 1))

    if (log_c < log_r) {
        theta[b, ] = theta_temp
    } else {
        theta[b, ] = theta[b - 1, ]
    }
}

## Get posterior means via marginal distributions
pi_est = exp(mean(theta))

## Plot results
par(mfrow = c(1, 3))
plot(1:B, theta, type = "l", bty = "n", xlab = "", ylab = "pi*")
hist(x = theta, xlab = "", ylab = "", main = "pi*")
abline(v = mean(theta), col = 2, lty = 2, lwd = 2)
hist(x = plogis(theta), xlab = "", ylab = "", main = "pi")
abline(v = exp(mean(theta)), col = 2, lty = 2, lwd = 2)
```

4.8

Consider the statistical model $\mathbf{y}_{n\times 1} \overset{\text{iid}}{\sim} \mathcal{N}(y; \mu, \sigma)$ with $\boldsymbol{\theta} = \{\mu, \sigma\} \in \mathbb{R} \times \mathbb{R}^+$. Write a Monte Carlo algorithm to maximize the log-likelihood function $\mathcal{L}(\boldsymbol{\theta}; \mathbf{y})$ w.r.t. $\boldsymbol{\theta}$. Note that the log-likelihood surface can be easily explored by rejecting all those candidates which do not improve the maximization path.

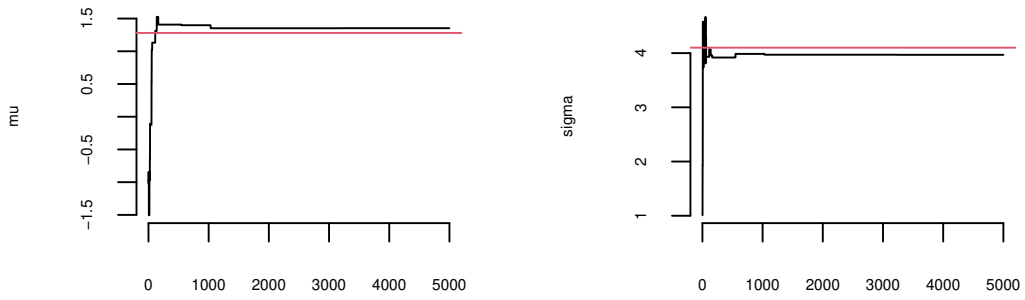
SOLUTION

```
set.seed(211)

## Generate data Y
n = 150
mu0 = 1.28
sigma0 = 4.1
y = rnorm(n = n, mean = mu0, sd = sigma0)

## Monte Carlo algorithm
B = 5000
Theta = matrix(data = NA, nrow = B, ncol = 2)
loglik = rep(NA, B)
Theta[1, ] = c(-1, 0.01)
loglik[1] = sum(log(dnorm(x = y, mean = Theta[1, 1], sd = exp(Theta[1, 2]))))
s0 = 0.5
for (b in 2:B) {

    theta_temp = c(rnorm(n = 1, mean = Theta[b - 1, 1], sd = s0), rnorm(n = 1, mean = Theta[b -
        1, 2], sd = s0))

    loglik[b] = sum(log(dnorm(x = y, mean = theta_temp[1], sd = exp(theta_temp[2]))))


    if (loglik[b] > loglik[b - 1]) {
        Theta[b, ] = theta_temp
    } else {
        Theta[b, ] = Theta[b - 1, ]
        loglik[b] = loglik[b - 1]  #overwrite with the previous loglikel
    }

```

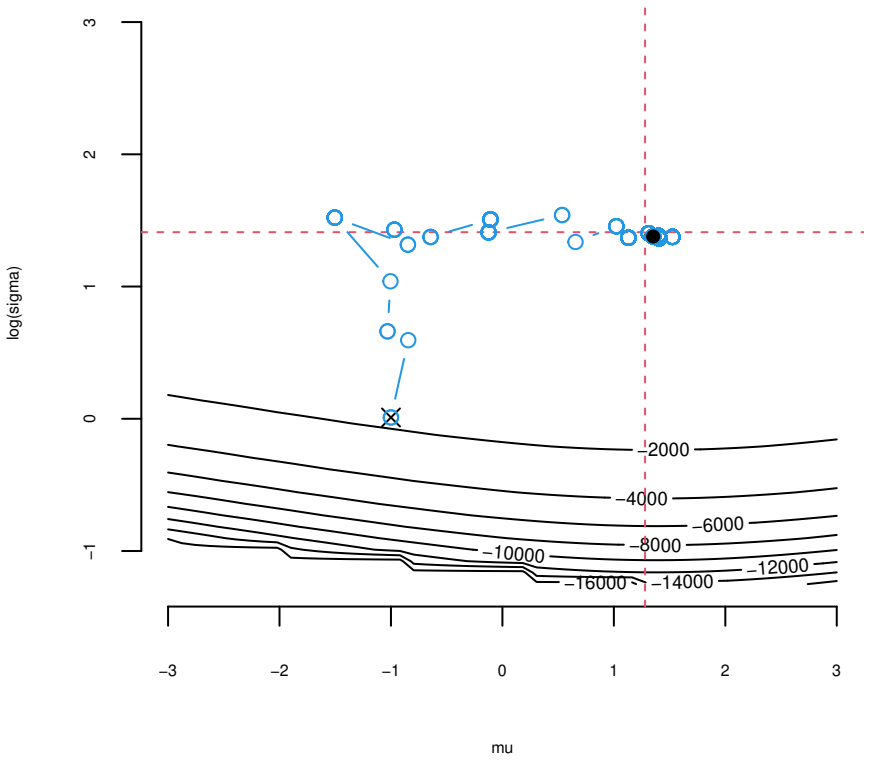
```r
}

## Get final estimates
iid = which.max(loglik)
mu_est = Theta[iid, 1]
sigma_est = exp(Theta[iid, 2])
print(c(mu_est, sigma_est))
```

```
[1] 1.353033 3.968113
```

```r
## Plot results
par(mfrow = c(1, 2))
plot(Theta[, 1], type = "l", xlab = "", ylab = "mu", bty = "n", cex.lab = 0.5, cex.axis = 0.5)
abline(h = mu0, col = 2)
plot(exp(Theta[, 2]), type = "l", xlab = "", ylab = "sigma", bty = "n", cex.lab = 0.5,
    cex.axis = 0.5)
abline(h = sigma0, col = 2)
```




```r

## See: Rizzo (2008), pp. 101-102
xax = seq(from = -3, to = 3, length.out = 50)  #x-axis
yax = seq(from = -1.25, to = 3, length.out = 50)  #y-axis
Z = matrix(data = NA, nrow = 50, ncol = 50)  #z-axis (grid)
for (i in 1:50) {
    for (j in 1:50) {
        Z[i, j] = sum(log(dnorm(x = y, mean = xax[i], sd = exp(yax[j]))))
    }
}
Z[is.infinite(Z)] = min(Z[!is.infinite(Z)])

# persp(xax,yax,Z,theta=65,phi=25,expand=0.6,ticktype =
# 'detailed',xlab='mu',ylab='sigma',zlab='loglikel')
contour(x = xax, y = yax, z = Z, bty = "n", xlab = "mu", ylab = "log(sigma)", cex.lab = 0.5,
    cex.axis = 0.5)
abline(h = log(sigma0), v = mu0, col = 2, lty = 2)
points(Theta[1, 1], Theta[1, 2], pch = 4, col = 1, cex = 1.25)  #starting point
points(Theta[, 1], Theta[, 2], type = "b", col = 4)
points(Theta[B, 1], Theta[B, 2], pch = 20, col = 1, cex = 1.25)
```



66

4.9

Consider two vectors $\mathbf{x}_{n\times 1}$ and $\mathbf{y}_{n\times 1}$ of data and the linear correlation between them $r = \text{cor}(\mathbf{x}, \mathbf{y})$. Write an algorithm that computes the standard error and the $(1-\alpha)\%$ confidence interval for r via non-parametric bootstrap. Note that data should be generated in advance.

0 SOLUTION

```r
## generate data
set.seed(112)
n = 125
S = matrix(data = c(1, 0.67, 0.67, 1), nrow = 2, ncol = 2)
X = mvtnorm::rmvnorm(n = n, mean = c(0, 0), sigma = S)

## plot data and compute sample correlation
plot(X[, 1], X[, 2], bty = "n", xlab = "X1", ylab = "X2", cex.lab = 0.5, cex.axis = 0.5)
R_est = cor(X)  #alternatively: cor(X[,1],X[,2])
print(R_est)

## Boostrapping the distribution of cor(X1,X2)
B = 500
r_boot = rep(NA, B)
for (b in 1:B) {
    iid = sample(x = 1:n, size = n, replace = TRUE)
    r_boot[b] = cor(X[iid, 1], X[iid, 2])
}

## Compute bootstrap statistics
```

```r
hist(x = r_boot, bty = "n", main = "", xlab = "", cex.lab = 0.5, cex.axis = 0.5)
se_r = sd(r_boot)
print(se_r)
ci_r = quantile(x = r_boot, probs = c(0.025, 0.975))
print(ci_r)
```

4.10

Consider the following linear models:

$$z_i = \beta_0^{(z)} + x_i\beta_1^{(z)} + \epsilon_i^{(z)}$$
$$y_i = \beta_0^{(y)} + x_i\beta_1^{(y)} + z_i\beta_2^{(y)} + \epsilon_i^{(y)}$$

where $\epsilon_i^{(z)} \sim \mathcal{N}(\epsilon; 0, \sigma_{\epsilon_z})$, $\epsilon_i^{(y)} \sim \mathcal{N}(\epsilon; 0, \sigma_{\epsilon_y})$, and $\epsilon_i^{(z)} \perp\!\!\!\perp \epsilon_i^{(y)}$ (independence). Write an algorithm to compute standard error and $(1-\alpha)\%$ confidence interval for the parameter $\xi = \beta_1^{(z)}\beta_2^{(y)}$ (a.k.a. *mediation effect*) via non-parametric bootstrap. Note that (a) data $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ should be generated in advance and (b) parameters $\{\beta_0^{(z)}, \beta_1^{(z)}, \beta_0^{(y)}, \beta_1^{(y)}, \beta_2^{(y)}, \sigma_{\epsilon_z}, \sigma_{\epsilon_y}\}$ can be estimated using maximum likelihood or least squares estimators.

SOLUTION

```r
## generate data
set.seed(119)
n = 365
# generating first linear model
b0_z = 0.98
b1_z = 1.34
s0_z = 0.98
x = runif(n = n, min = -3, max = 3)
z = rnorm(n = n, mean = cbind(1, x) %*% c(b0_z, b1_z), sd = s0_z)
# generating second linear model
b0_y = -2.11
b1_y = -1.34
b2_z = 0.76
s0_y = 2.31
y = rnorm(n = n, mean = cbind(1, x, z) %*% c(b0_y, b1_y, b2_z), sd = s0_y)

## plot sample data
par(mfrow = c(1, 3))
plot(x, z, bty = "n", xlab = "x", ylab = "z", main = "z~x", cex.lab = 0.5, cex.axis = 0.5)
plot(x, y, bty = "n", xlab = "x", ylab = "y", main = "y~x", cex.lab = 0.5, cex.axis = 0.5)
plot(z, y, bty = "n", xlab = "z", ylab = "y", main = "y~z", cex.lab = 0.5, cex.axis = 0.5)


## Boostrapping the distribution of xi = b1_z*b2_z
B = 500
xi_boot = rep(NA, B)
for (b in 1:B) {
    iid = sample(x = 1:n, size = n, replace = TRUE)

    # estimating b1_z
    out = lm(formula = z[iid] ~ x[iid])
    b1_z_boot = coef(out)[2]

    # estimating b2_z
    out = lm(formula = y[iid] ~ x[iid] + z[iid])
    b2_z_boot = coef(out)[3]

```

```
    # computing xi
    xi_boot[b] = b1_z_boot * b2_z_boot

}

## Compute bootstrap statistics
par(mfrow = c(1, 1))
hist(x = xi_boot, bty = "n", main = "", xlab = "", cex.lab = 0.5, cex.axis = 0.5)
abline(v = mean(xi_boot), lty = 2, col = 2, lwd = 1.5)
print(mean(xi_boot))
se_xi = sd(xi_boot)
print(se_xi)
ci_xi = quantile(x = r_boot, probs = c(0.025, 0.975))
print(ci_xi)
```