Solutions to the Exercices in
Analysis of Phylogenetics and Evolution with R

Chapter 2

1. `getwd()`

   Say the three files have the same name `file.txt`, and are located in `~/data1/`, `~/data2/`, and `~/data3/`. First possibility:

   ```
   setwd("~/data1/")
   df1 <- read.table("file.txt")
   setwd("~/data2/")
   df2 <- read.table("file.txt")
   setwd("~/data3/")
   df3 <- read.table("file.txt")
   ```

   Second possibility:

   ```
   df1 <- read.table("~/data1/file.txt")
   df2 <- read.table("~/data2/file.txt")
   df3 <- read.table("~/data3/file.txt")
   ```

   Under Windows, this could be, respectively:

   ```
   setwd("D:/data1/")
   df1 <- read.table("file.txt")
   setwd("D:/data2/")
   df2 <- read.table("file.txt")
   setwd("D:/data3/")
   df3 <- read.table("file.txt")

   df1 <- read.table("D:/data1/file.txt")
   df2 <- read.table("D:/data2/file.txt")
   df3 <- read.table("D:/data3/file.txt")
   ```

2. ```
   X <- matrix(NA, 1000, 3)
   lamb <- c(1, 5, 10)
   for (i in 1:3) X[, i] <- rpois(1000, lamb[i])
   ```

   Another possibility is to call `rpois` only once benefitting of the recycling of the second argument that specifies the value of λ, but here we must take care to fill the matrix row-wise (with `byrow = TRUE`) since the values are repeated 1, 5, 10, 1, 5, 10, ...:

   ```
   X <- matrix(rpois(3000, c(1, 5, 10)), ncol = 3, byrow = TRUE)
   ```
   ```

The mean of each column may be calculated with:

```
apply(X, 2, mean)
```

or:

```
for (i in 1:3) print(mean(X[, i]))
```

The first solution is to be preferred because it returns a numeric vector (this can also be done with the second one but in a more complicated way).

3. (a)
```
x <- numeric(0)
for (i in 1:10) x <- c(x, rnorm(1))
```

   (b)
```
x <- numeric(10)
for (i in 1:10) x[i] <- rnorm(1)
```

   (c)
```
x <- rnorm(10)
```

   The timings of these three possibilities are so minute with 10 values that it makes no difference, but the difference is visible with 10,000 values (the braces are needed for `system.time` to work correctly):

```
> system.time({x <- numeric(0);
               for (i in 1:1e4) x <- c(x, rnorm(1))})
[1] 0.614 0.010 0.624 0.000 0.000
> system.time({x <- numeric(1e4);
               for (i in 1:1e4) x[i] <- rnorm(1)})
[1] 0.134 0.003 0.142 0.000 0.000
> system.time(x <- rnorm(1e4))
[1] 0.003 0.000 0.003 0.000 0.000
```

4. (a)
```
> df <- read.table("file")
> str(df)

'data.frame':        3 obs. of  2 variables:
 $ V1: Factor w/ 3 levels "Balaenoptera_musculus",..: 3 2 1
 $ V2: int  10 70000 120000000
```

   The character strings have been transformed as a factor which is the default behavior of `read.table`. The option `as.is = TRUE` should be used to avoid this.

   (b) First solution:

```
> df <- read.table("file", as.is = TRUE)
> x <- df$V2
> names(x) <- df$V1
> x

         Mus_musculus          Homo_sapiens Balaenoptera_musculus
                   10                 70000             120000000

> x["Mus_musculus"]
```

```
Mus_musculus
          10
```

Second solution:

```
> df <- read.table("file", as.is = TRUE, row.names = 1)
> df
                              V2
Mus_musculus                  10
Homo_sapiens               70000
Balaenoptera_musculus 120000000
> df["Mus_musculus", ]
[1] 10
```

We can look at the difference in these two solutions with `str`:

```
> str(x)
 Named int [1:3] 10 70000 120000000
 - attr(*, "names")= chr [1:3] "Mus_musculus" "Homo_sapiens" "Balaenoptera_musculus"
> str(df)
'data.frame':        3 obs. of  1 variable:
 $ V2: int  10 70000 120000000
```

5. (a) `TreeOfLife <- list(Archaea = Archaea, Bacteria = Bacteria)`
   (b) `TreeOfLife <- c(TreeOfLife, list(Eukaryotes = Eukaryotes))`
   (c) `TreeOfLife$Archaea <- c(TreeOfLife$Archaea, "Actinobacteria")`
   (d) `unlist(TreeOfLife, use.names = FALSE)`

Chapter 3

1. `tr <- rtree(10)`
   (a) `x <- tr$edge.length`
   (b) `tr$edge.length <- NULL`
   `plot(tr)`
   (c) `tr$edge.length <- runif(dim(tr$edge)[1], 0, 10)`
   (d) `tr$edge.length <- x`

2. `tr <- rtree(5)`
   `tr`
   `plot(tr)`

   The class of `tr` is deleted with:

   ```
   class(tr) <- NULL
   ```

   Printing `tr` gives nearly the same result than before, but plotting it[1] gives a message error:

[1] In the book, "Try to print it again" should be "Try to plot it again"; see the list of errata.

```
> plot(tr)
Error in plot.window(xlim, ylim, log, asp, ...) :
        need finite 'xlim' values
....
```

Because `tr` now has no class, the generic function `plot` uses the default function `plot.default` which is inappropriate for the present data structure. The generic function can be bypassed with:

```
plot.phylo(tr)
```

3. A straightforward solution follows:

```
t1 <- rtree(10)
t2 <- rtree(10)
t3 <- rtree(10)
write.tree(t1, "treefile.tre")
write.tree(t2, "treefile.tre", append = TRUE)
write.tree(t3, "treefile.tre", append = TRUE)
tr <- read.tree("treefile.tre")
lapply(tr, summary)
```

A program that does the same for any number of trees `N` and any number of tips `n` is:

```
N <- 3
n <- 10
res <- replicate(N, rtree(n), simplify = FALSE)
write.tree(res[[1]], "treefile.tre")
if (N > 1)
  for (i in 2:N)
    write.tree(res[[1]], "treefile.tre", append = TRUE)
tr <- read.tree("treefile.tre")
lapply(tr, summary)
```

4.
```
T1000 <- dbtrees("treebase", 1000)
T1000.copy1 <- T1000.copy2 <- T1000.copy3 <- T1000
T1000.copy1 <- compute.brlen(T1000.copy1, 1)
T1000.copy2 <- compute.brlen(T1000.copy2, "Grafen")
T1000.copy3 <- compute.brlen(T1000.copy3, runif, 0, 0.1)
```

5. `X <- read.GenBank(paste("U157", 17:24, sep = ""))`

   (a) `attr(X, "species")`

   (b) `sapply(X, length)` (`lapply` returns the same result as a list instead of a vector).

   (c) `Xmat <- matrix(unlist(X), length(X), byrow = TRUE)`

(d)
```
> X1 <- Xmat[, seq(1, 1045, 3)]
> X2 <- Xmat[, seq(2, 1045, 3)]
> X3 <- Xmat[, seq(3, 1045, 3)]
> base.freq(X1)

         a          c          g          t
0.37464183 0.51038682 0.03044413 0.08452722
> base.freq(X2)

        a         c         g         t
0.2331178 0.2956178 0.2413793 0.2298851
> base.freq(X3)

        a         c         g         t
0.1936063 0.2471264 0.1311063 0.4281609
```

Note that it is very easy to include these commands in a loop:

```
X.position <- list()
length(X.position) <- 3
for (i in 1:3) X.position[[i]] <- Xmat[, seq(i, 1045, 3)]
lapply(X.position, base.freq)
```

(e)
```
write.dna(X1, "X1.txt")
write.dna(X2, "X2.txt")
write.dna(X3, "X3.txt")
X1b <- read.dna("X1.txt")
X2b <- read.dna("X2.txt")
X3b <- read.dna("X3.txt")
Xconcat <- cbind(X1b, X2b, X3b)
```

Chapter 4

1.
```
co <- rev(rainbow(6))
lim <- 4:9 * 10
p <- character(length(bs.ml))
for (i in 1:6) p[bs.ml >= lim[i]] <- co[i]
plot(tr, no.margin = TRUE)
nodelabels(node = 2:13, pch = 22, bg = p[-1], cex = 3)
yc <- 0.75 # controls the height of the rectangles of the scale
yb <- seq(1, by = xc, length.out = 6)
rect(0, yb, 0.0075, yb + yc, col = co, border = FALSE)
text(.014, c(xb, yb[6] + yc), c(lim, "100"), adj = 1)
```

2. In this solution, the option `tip.color`, introduced since the issue of the book, is used.

```
data(bird.orders)
sel <- which.edge(bird.orders, 1:5)
coledge <- rep("black", dim(bird.orders$edge)[1])
coledge[sel] <- "blue"
coltip <-  rep("black", 23)
```

```
coltip[1:5] <- "blue"
plot(bird.orders, edge.color = coledge, font = 1,
     tip.color = coltip)
```

To have only the terminal branches of this clade colored:

```
co <- rep("black", dim(bird.orders$edge)[1])
term <- which(as.numeric(bird.orders$edge[, 2]) > 0)
co[term[term %in% sel]] <- "blue"
plot(bird.orders, edge.color = co, font = 1)
```

3. Suppose the tree is called `tr`, the observed character `x`, and the same values for the nodes of `tr` are in `y`. Both vectors are indexed with the numbers of the tips and nodes, respectively, so that `x[1]` is the character value for the first tip in `tr`, and `y[1]` is the value for node number -1. The idea is to build a character for the branches of the tree giving the states at the start and the end separated by a dash (e.g., `"A-A"`, `"A-B"`, ...). We first get these values at the start which is relatively easy because all nodes are internal (hence with a negative index):

```
z <- y[-tr$edge[, 1]]
```

For the end of the branch, we must take care that the node may be internal or terminal (i.e., a tip)

```
terms <- tr$edge[, 2] > 0
z[terms] <- paste(z[terms], x[tr$edge[terms, 2]], sep = "-")
z[!terms] <- paste(z[!terms], y[-tr$edge[!terms, 2]], sep = "-")
```

Automatic coloring can be done with the function `rainbow`; the options of `plot.phylo` give a representation close to what is usually used in character mapping:

```
z <- factor(z)
co <- rainbow(nlevels(z))
plot(tr, "c", FALSE, edge.color = co[unclass(z)], edge.width = 3)
```

Chapter 5

1. (a)
```
> Q <- matrix(3e-04, 4, 4)
> diag(Q) <- 0
> diag(Q) <- -rowSums(Q)
> Q
```
```
       [,1]   [,2]   [,3]   [,4]
[1,] -9e-04  3e-04  3e-04  3e-04
[2,]  3e-04 -9e-04  3e-04  3e-04
[3,]  3e-04  3e-04 -9e-04  3e-04
[4,]  3e-04  3e-04  3e-04 -9e-04
```

(b) The first approach is to compute the matrix exponential of Q as described page 102:

```
> library(rmutil)
> mexp(Q)
             [,1]         [,2]         [,3]         [,4]
[1,] 0.9991005398 0.0002998201 0.0002998201 0.0002998201
[2,] 0.0002998201 0.9991005398 0.0002998201 0.0002998201
[3,] 0.0002998201 0.0002998201 0.9991005398 0.0002998201
[4,] 0.0002998201 0.0002998201 0.0002998201 0.9991005398
> mexp(1000 * Q)
          [,1]      [,2]      [,3]      [,4]
[1,] 0.4758957 0.1747014 0.1747014 0.1747014
[2,] 0.1747014 0.4758957 0.1747014 0.1747014
[3,] 0.1747014 0.1747014 0.4758957 0.1747014
[4,] 0.1747014 0.1747014 0.1747014 0.4758957
> mexp(1e+06 * Q)
     [,1] [,2] [,3] [,4]
[1,] 0.25 0.25 0.25 0.25
[2,] 0.25 0.25 0.25 0.25
[3,] 0.25 0.25 0.25 0.25
[4,] 0.25 0.25 0.25 0.25
```

The second approach is to use equation 5.6 to calculate the probability that a site changes (i.e., the off-diagonal elements of the above matrices):

```
> (1 - exp(-4 * 3e-04))/4
[1] 0.0002998201
> (1 - exp(-4 * 3e-04 * 1000))/4
[1] 0.1747014
> (1 - exp(-4 * 3e-04 * 1e+06))/4
[1] 0.25
```

2. (a)
```
> Q <- matrix(0, 4, 4)
> Q[1, 2] <- Q[2, 1] <- 0.001
> Q[1, 3] <- Q[3, 1] <- 5e-04
> Q[1, 4] <- Q[4, 1] <- 2e-04
> Q[2, 3] <- Q[3, 2] <- 3e-04
> Q[2, 4] <- Q[4, 2] <- 1e-04
> Q[3, 4] <- Q[4, 3] <- 5e-05
> PI <- c(0.35, 0.17, 0.25, 0.23)
> Q <- Q * rep(PI, each = 4)
> diag(Q) <- -rowSums(Q)
> Q
           [,1]      [,2]       [,3]      [,4]
[1,] -0.000341  0.000170  0.0001250  4.60e-05
[2,]  0.000350 -0.000448  0.0000750  2.30e-05
[3,]  0.000175  0.000051 -0.0002375  1.15e-05
[4,]  0.000070  0.000017  0.0000125 -9.95e-05
```

(b) `> mexp(Q)`

```
             [,1]         [,2]         [,3]         [,4]
[1,] 9.996591e-01 1.699365e-04 1.249705e-04 4.599254e-05
[2,] 3.498693e-04 9.995521e-01 7.499631e-05 2.300218e-05
[3,] 1.749587e-04 5.099749e-05 9.997625e-01 1.150267e-05
[4,] 6.998865e-05 1.700161e-05 1.250291e-05 9.999005e-01
```

(c) The probability matrix (e.g., calculated with `P <- mexp(t*Q)`) gives, for each base on the rows, the probabilities to change to one of the bases on the columns. The function `sample` can be used for this simulation: this will give a random sample of the four bases for a given initial base. For instance, suppose the probabilities of change for adenine (i.e., the first row of `P`) are 0.9, 0.03, 0.06, and 0.01, and we want to simulate the evolution at 10 sites initially A:

```
> sample(c("A", "C", "G", "T"), 10, replace = TRUE, c(0.9, 0.03,
+     0.06, 0.01))
 [1] "G" "A" "A" "A" "A" "A" "A" "G" "A" "A"
```

This process can be repeated for each base by setting the appropriate vector of probabilities. A program follows for a given sequence stored as vector `S`.

```
> t <- 1
> P <- mexp(t * Q)
> base <- c("A", "C", "G", "T")
> S <- sample(base, 1000, replace = TRUE, PI)
> new.S <- character(length(S))
> for (i in 1:4) {
+     pos <- grep(base[i], S)
+     new.S[pos] <- sample(base, length(pos), TRUE, P[i, ])
+ }
```

Chapter 6

1. `X <- replicate(2, cumsum(c(0, rnorm(99))))`

(a)
```
X2 <- matrix(NA, 100, 4)
for (i in 1:4)
  X2[, i] <- cumsum(c(X[100, ceiling(i/2)], rnorm(99)))
matplot(101:200, X2, type = "l", col = 1, xlim = c(1, 200))
matlines(X, col = 1)
```

(b)
```
X2 <- matrix(NA, 101, 4)
X2[1, 1:2] <- X[100, 1]
X2[1, 3:4] <- X[100, 2]
a <- 0.2
t1 <- -1
t2 <- 1
for (j in 1:2) {
    e <- rnorm(100)
    for (i in 1:100)
      X2[i + 1, j] <- -a*(X2[i, j] - t1) + e[i]
```

```
    }
for (j in 3:4) {
    e <- rnorm(100)
    for (i in 1:100)
      X2[i + 1, j] <- -a*(X2[i, j] - t2) + e[i]
    }
matplot(101:201, X2, type = "l", col = 1, xlim = c(1, 201))
matlines(X, col = 1)
```

April 17, 2007