CDT R Review Sheet

Work through the sheet in any order you like. Skip the starred (*) bits in the first instance, unless you’re fairly confident.

1. Vectors

(a) Generate 100 standard normal random variables, and keep only the ones which are greater than 1. Don’t use a loop!

```
> x = rnorm(100)
> x = x[x > 1]
```

(b) Write a function which takes two arguments n and min, and returns n independent random variables from a standard normal distribution truncated below by min. Let min default to 0. Lots of ways to do this, but one way is...

```
> truncNorm = function(n, min = 0) {
+   out = c()
+   while (length(out) < n) {
+     tmp = rnorm(n)
+     tmp = tmp[tmp > min]
+     out = c(out, tmp)
+   }
+   return(out[seq_len(n)])
+ }
```

Note this wouldn’t work well if min was quite large (what could you do instead?).

(c) Generate 10,000 truncated normals with min set at -1, and plot as a histogram. Adjust the number of bins sensibly.

```
> x = truncNorm(10000, min = -1)
> hist(x, breaks = 50, col = 2)
```

2. Data

Load the hills data set.

```
> library(MASS)
> data(hills)
```

(a) What sort of object is hills? A list? A matrix? Use the is() and class() functions if you’re not sure. It’s a data frame, and therefore a list. It’s not a matrix.

(b) How many columns does hills have? It has three columns, which you can check with ncol() or dim(). The first ‘column’ which appears when you print is just the row names.
(c) One of the races is called Two Breweries; change this to Three Breweries. One possibility:

```r
> rownames(hills)[33] = "Three Breweries"
> hills[33, ]
## dist  climb   time
## Three Breweries 18 5200 170.2
```

(d) Using the function `with()`, find the mean time for races with a climb greater than 1000 feet. Don’t use `attach()`, it’s horrible.

```r
> with(hills, mean(time[climb > 1000]))
## [1] 85.57
```

Now, load the `Orthodont` data set from the `nlme` package (you may have to install `nlme` first).

```r
> library(nlme)
> data(Orthodont)
```

(e) What sort of object is `Orthodont`? Is it a data frame? What makes it different to `hills`?

Yes, it is a data frame, as seen with `class(Orthodont)`, but it’s also an `nfnGroupedData`, which inherits from `data.frame`. Using `attributes(Orthodont)` we see all the things which make it different to an ordinary data frame, especially the built-in formula.

(f) What is the name of the function used to print `Orthodont`? Try using `methods(print)`.

The generic `print()` first looks for `print.nfnGroupedData()`, then `print.nfGroupedData()`, and finally finds `print.groupedData()`.

(g) You should find that the function is ‘non-visible’, meaning it is not exported from the package. You can view it using

```r
> nlme:::print.groupedData
```

Inspection of the code reveals that it prints the `formula` attribute, and then just treats it as a data frame.

3. Recursion

The n<sup>th</sup> Fibonacci number is defined by the recursion $F_n = F_{n-1} + F_{n-2}$, with $F_0 = F_1 = 1$.

(a) Write a recursive function with argument `n` which returns the n<sup>th</sup> Fibonacci number. [Hint: you might want to look at the documentation `?Recall`.]
> fib = function(n) {
+   if (n < 2)
+     return(1)
+   Recall(n - 1) + Recall(n - 2)
+ }

(b) Evaluate the 20th Fibonacci number with it. \( F_n = 1.0946 \times 10^4 \).

(c) How many times does the function have to evaluate itself to calculate this? \( F_{20} \) times! Can you think of a faster way to do this with a loop? For example:

> fib2 = function(n) {
+   if (n < 2)
+     return(1)
+   tmp = numeric(n + 1)
+   tmp[1:2] = 1
+   for (i in seq(from = 3, to = n + 1)) tmp[i] = tmp[i - 1] + tmp[i - 2]
+   tmp[n + 1]
+ }

Calculate \( F_{1000} \) with your new function.

> fib2(1000)
## [1] 7.033e+208

4. Methods

We’re going to create a class for bivariate data, and a series of methods to print, summarise and plot that data.

(a) Create a list with entries \( x \) (consisting of 20 independent standard normal random variables) and \( y \) (consisting of 20 independent Poisson(5) random variables), and give it the class \( \text{biv} \).

> dat = list(x = rnorm(20), y = rpois(20, lambda = 5))
> class(dat) = "biv"
> dat

```
## $x
## [1] -0.89927 -0.40996 -0.69003 -0.33965 -0.60927 -0.21811 0.74651
## [8] 0.88852 0.60739 -0.80701 -0.02327 1.22291 -0.04895 1.07041
## [15] 0.31087 0.06358 -0.78591 -0.56062 0.09033 -0.10981
##
## $y
## [1] 5 4 4 3 3 5 6 7 7 3 2 8 3 3 0 4 2 7 4 10
##
## attr(,"class")
## [1] "biv"
```

(b) Write a print method for \( \text{biv} \) (i.e. a function called \text{print.biv}()) which shows (at most) the first 6 entries of your data in the following format this:
Bivariate data, 20 entries
x : -0.001616495 -0.07254921 -1.096251 -0.4702838 1.423081 -1.019105 ...
y : 7 5 2 4 29 3 ...

> print.biv = function(obj) {
+   n = length(obj$x)
+   cat("Bivariate data,", n, " entries\n")
+   len = min(n, 6)
+   dots = ifelse(n > 6, "...", "")
+   cat("x : ", obj$x[1:len], dots, "\n")
+   cat("y : ", obj$y[1:len], dots, "\n")
+   invisible(obj)
+ }
> print(dat)
## Bivariate data, 20 entries
## x : -0.8993 -0.41 -0.69 -0.3397 -0.6093 -0.2181 ...
## y : 5 4 4 3 3 5 ...

(c) * A print method should return the object itself invisibly: make sure your function does [Hint: type ?invisible]. [See above.]

(d) Construct a plot method for objects of class biv, which does a scatter plot and a pair of boxplots side-by-side.

> plot.biv = function(obj) {
+   par(mfrow = c(1, 2))
+   plot.default(obj$x, obj$y)
+   boxplot(obj$x, obj$y)
+   invisible()
+ }
> plot(dat)

(e) ** Do the above with S4 classes and methods.

5. Functions

(a) Write a function which, given two vectors x and z of the same length, returns the matrix

$$X = \begin{pmatrix}
1 & x_1 & z_1 & x_1z_1 \\
1 & x_2 & z_2 & x_2z_2 \\
\vdots & \vdots & \vdots & \vdots \\
1 & x_n & z_n & x_nz_n
\end{pmatrix}.$$  

> modelmat = function(x, z) {
+   cbind(1, x, z, x * z)
+ }

(b) What happens if you give arguments of different lengths? Cause your function to behave (or fail) in the way you think best.
(c) * Write a function which takes an arbitrary number of arguments, each being a covariate vector of the same length, and returns the model matrix consisting of all the main effects and interactions. In other words, if the vectors were \( x, y, z, w \) we’d get

\[
X = \begin{pmatrix}
1 & x_1 & y_1 & z_1 & w_1 & x_1 y_1 & x_1 z_1 & \cdots & z_1 w_1 \\
1 & x_2 & y_2 & z_2 & w_2 & x_2 y_2 & x_2 z_2 & \cdots & z_2 w_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_n & y_n & z_n & w_n & x_n y_n & x_n z_n & \cdots & z_n w_n
\end{pmatrix}.
\]

[You’re not allowed to use \texttt{model.matrix()} or similar!]

```r
> modelmat2 = function(...) {
+   vecs = list(...)  
+   k = length(vecs[[1]])  
+   n = length(vecs[[1]])  
+   if (!all(sapply(vecs, length) == n))  
+     stop("Lengths of vectors differ")  
+   out = matrix(1, n, 1 + k + k * (k - 1)/2)  
+   out[, 2:(k + 1)] = unlist(vecs)  
+   st = k + 1  
+   for (i in seq_len(k - 1)) {
+     out[, st + seq_len(k - i)] = vecs[[i]] * unlist(vecs[(i + 1):k])
+     st = st + (k - i)
+   }
+   out
+   }
> x = rnorm(5)
> y = rnorm(5)
> z = rnorm(5)
> all.equal(modelmat2(x, y, z), model.matrix(~(x + y + z)^2),
+   check.attributes = FALSE)

## [1] TRUE

(d) Check your answer with \texttt{model.matrix()}.

6. * plyr

Look at the paper by Wickham (2011) on the \texttt{plyr} library: \url{www.jstatsoft.org/v40/i01/paper}

```r
> library(plyr)

Load the housing data from the MASS package:

```r
> library(MASS)
> data(housing)
> head(housing)  # look at first few entries
Use `?housing` to see what the fields mean.

(a) Use `dlply` to transform housing data into three data frames, one for each level of `Infl`.

```r
> dlply(housing, "Infl")
```

(b) Use `plyr` to turn the data frame into a contingency table.

```r
> conTab = daply(housing, names(housing)[1:4], function(x) sum(x$Freq))
```

(c) Estimate the probability of having 'High' contact given different types of accommodation.

```r
> tab2 = daply(housing, c("Type", "Cont"), function(x) sum(x$Freq))
> aaply(tab2, 1, function(x) x/sum(x))
```

```
##
## Type Low High
## Tower 0.5475 0.4525
## Apartment 0.4144 0.5856
## Atrium 0.3431 0.6569
## Terrace 0.3430 0.6570
```

7. * Design Matrices*

Using `ddply`, create a data frame for the housing data in which each row represents one observation.

```r
> hous2 = ddply(housing, names(housing)[1:4], function(x) x[rep(1, + times = x$Freq), ])
```

Fit a binomial regression model for how contact (`Cont`) is determined by the other three variables. Choose the model you think most appropriate.

*Just looking at p-values, I chose:*

```r
> glm(Cont ~ Infl + Type, family = "binomial", data = hous2)
```

```
##
## Call: glm(formula = Cont ~ Infl + Type, family = "binomial", data = hous2)
##
## Coefficients:
```
Produce a design matrix for your chosen glm(). [Hint: Use model.matrix() to see what the answer should be if you’re not sure, then try to construct the matrix ‘by hand’.]

8. * Mixtures

Suppose we have i.i.d. observations $X^{(i)} = (X_{i1}, \ldots, X_{ik})$, where each $X_{ij}$ is binary (i.e. takes values in $\{0, 1\}$). A discrete mixture model assumes that each component of the vector $X^{(i)}$ is independent, conditional upon an unknown class label $U_i \in \{1, \ldots, l\}$.

(a) Write down the likelihood for one observation $X^{(1)}$, and then for $n$ observations. What are the parameters to be estimated?

(b) Write an R function to evaluate the likelihood.

(c) Write an R function to generate data from the model.

(d) Use nlm() to find the maximum likelihood estimator for your simulated data, and compare it to the parameters you chose.