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)
```

   (d) What happens if `min` is quite large (say `> 4`)? How could you improve your method of sampling?

   For large values of ***`min`*** *the algorithm starts to run slowly because almost all samples are rejected. There are a lot of better ways to sample from this distribution, the most efficient would be to use inversion (i.e. plug uniform variables into the inverse CDF) to get exact samples.*

2. **Data**

   Load the `hills` data set.

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

(a) What sort of object is `hills`? Is it 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:*

```
> rownames(hills)[33] <- "Three Breweries"
> hills[33, ]

##                 dist climb   time
## Three Breweries   18  5200 170.25
```

(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.*

```
> with(hills, mean(time[climb > 1000]))

## [1] 85.56965
```

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

```
> 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

```
> 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 nth 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 nth 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] 7.033037e+208
```

4. **MCMC**

Suppose that $X_1, \ldots, X_n \overset{\text{i.i.d.}}{\sim} \text{Gamma}(\alpha, \beta)$, and let α and β have independent Exponential(1) priors.

(a) Write a function to evaluate the log-posterior of α and β given (a vector of) data $\boldsymbol{x}$. The function should have arguments `x`, `alpha` and `beta`.

```
> #' Log-Posterior distribution for Gamma data
> #'
> #' Evaluates log-posterior assuming i.i.d. \eqn{Gamma(\alpha,\beta)}
> #' with independent \eqn{Exponential(1)} priors.
> #'
> #' @param x vector of data
> #' @param alpha,beta parameter values
> #'
> log_post <- function(x, alpha, beta) {
+     log_prior <- dexp(alpha, 1, log = TRUE) + dexp(beta,
+         1, log = TRUE)
+     log_lik <- sum(dgamma(x, alpha, beta, log = TRUE))
+     out <- log_prior + log_lik
+     return(out)
+ }
```

(b) Write a function to perform a single Metropolis-Hastings step to explore the posterior above. Use a proposal

$$\alpha' = \alpha + \sigma Z_1 \qquad \beta' = \beta + \sigma Z_2$$

for Z_1, Z_2 independent standard normals (i.e. $q(\alpha' \mid \alpha) \sim N(\alpha, \sigma^2)$.) It should take as arguments `x`, `alpha`, `beta` and `sigma`.

```
> #' Function to perform one step of M-H
> #'
> #' @param x vector of data
> #' @param alpha,beta starting parameter values
> #' @param sigma s.d. of proposal distribution
> mh_step <- function(x, alpha, beta, sigma) {
+     new_pt <- c(alpha, beta) + rnorm(2, sd = sigma)
+     if (any(new_pt <= 0))
+         return(c(alpha, beta))  # proposed point not valid

+     U <- runif(1)
+     logalpha <- log_post(x, new_pt[1], new_pt[2]) - log_post(x,
+         alpha, beta)

+     if (log(U) < logalpha)
+         return(new_pt) else return(c(alpha, beta))
+ }
```

(c) Write a function to run the Metropolis-Hastings algorithm for N steps and return an $N \times 2$ matrix of the parameter values. It should take as input the data `x`, number of steps `N`, starting values `alpha` and `beta`, and proposal standard deviation `sigma`.

```
> #' Function to perform Metropolis-Hastings
> #'
> #' @param x vector of data
> #' @param N number of samples from posterior
> #' @param alpha,beta starting parameter values
> #' @param sigma s.d. of proposal distribution
> #'
> run_mh <- function(x, N, alpha = 1, beta = 1, sigma = 2/sqrt(length(x))) {
+     params <- matrix(0, N, 2)
+     cur = c(alpha, beta)

+     # iterate through MH steps
+     for (i in 1:N) {
+         cur = mh_step(x, cur[1], cur[2], sigma = sigma)
+         params[i, ] = cur
+     }

+     class(params) <- "mh"

+     return(params)
```

```
+ }
```

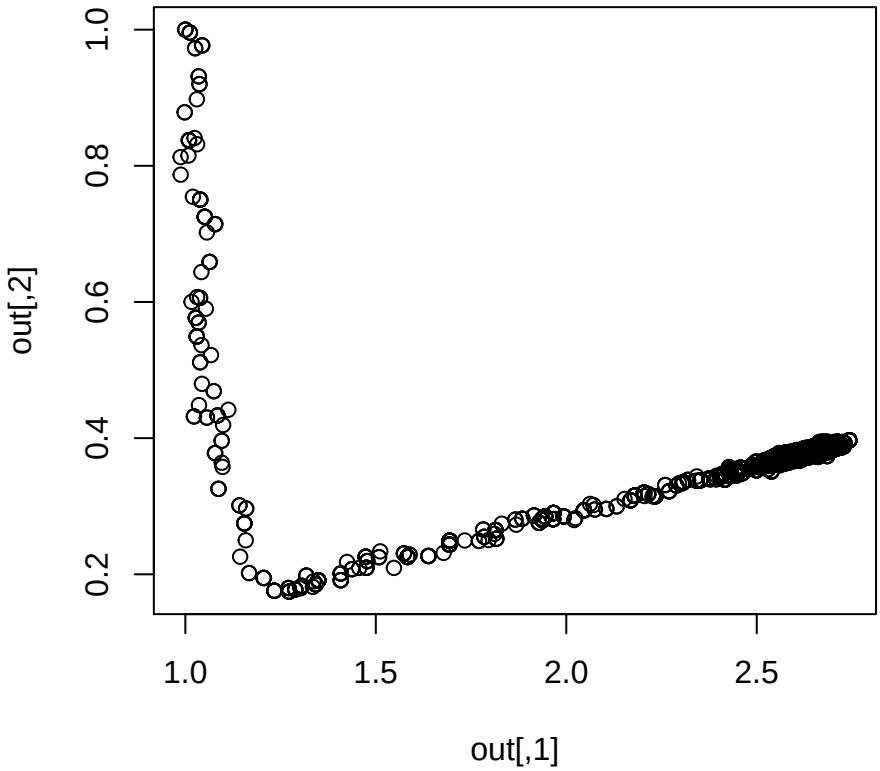
(d) The file `airpol.txt` contains daily PM2.5 readings taken from various measuring stations around Seattle during 2015. Read in the data as a vector and plot it in a histogram.

```
> x <- scan("airpol.txt")  # note use of scan(), not read.table()
> hist(x, breaks = 100, freq = FALSE)
```

Model the data as i.i.d. Gamma distributed observations using the priors above. Run your Metropolis-Hastings algorithm for 5,000 steps with starting point $\alpha = 1$, $\beta = 1$. Plot your output with `plot()` and investigate different values of $\sigma \in \{0.01, 0.02, 0.05\}$.

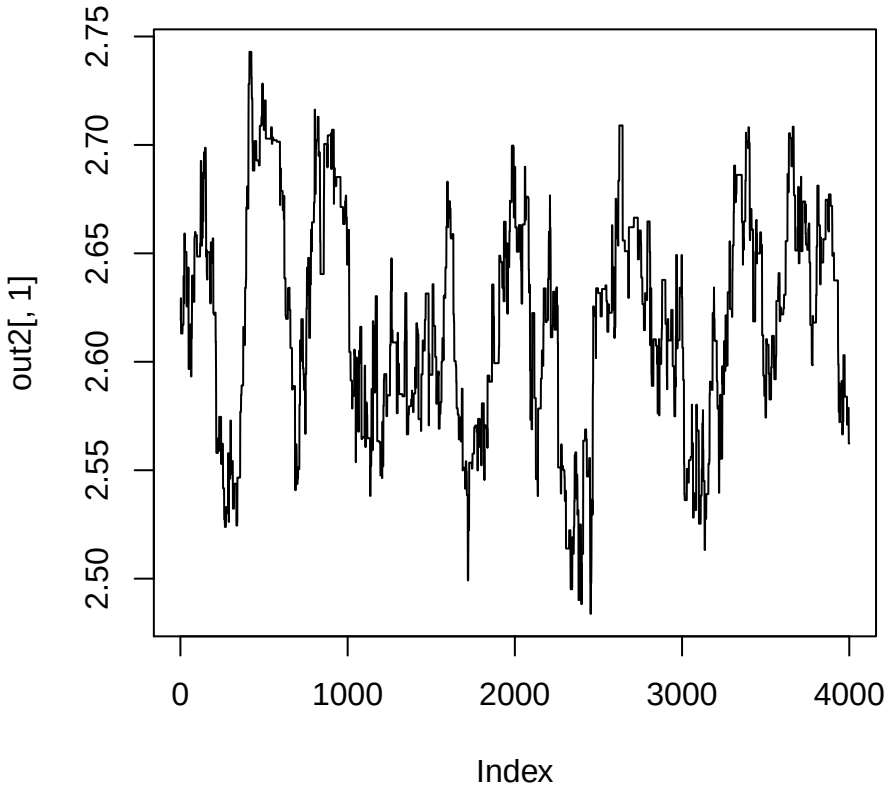
There is a compromise in how often we want the chain to move and how quickly it should be able to explore the space. 0.01 moves slowly, 0.05 rarely moves at all, but 0.02 works quite well.

```
> out <- run_mh(x, 5000, alpha = 1, beta = 1, sigma = 0.02)
> plot(out)
```

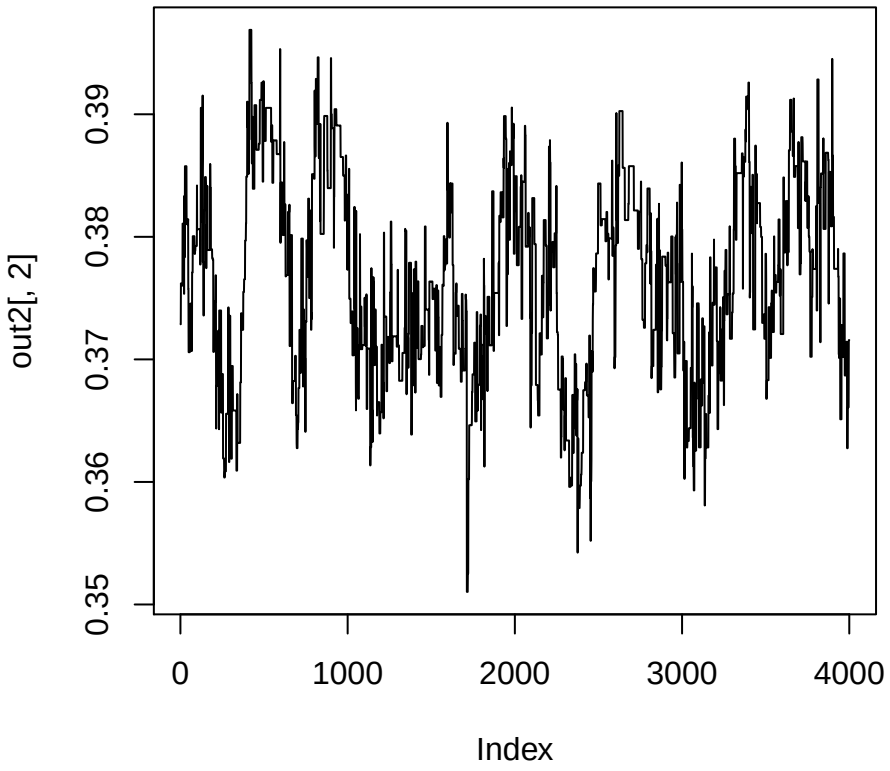


Of course, the chain takes some time to settle down (burn-in), so it is sensible to discard the first 1,000 or so observations for inference. This gives a clearer picture.

```
> out2 <- out[-c(1:1000), ]
> plot(out2[, 1], type = "l")
```

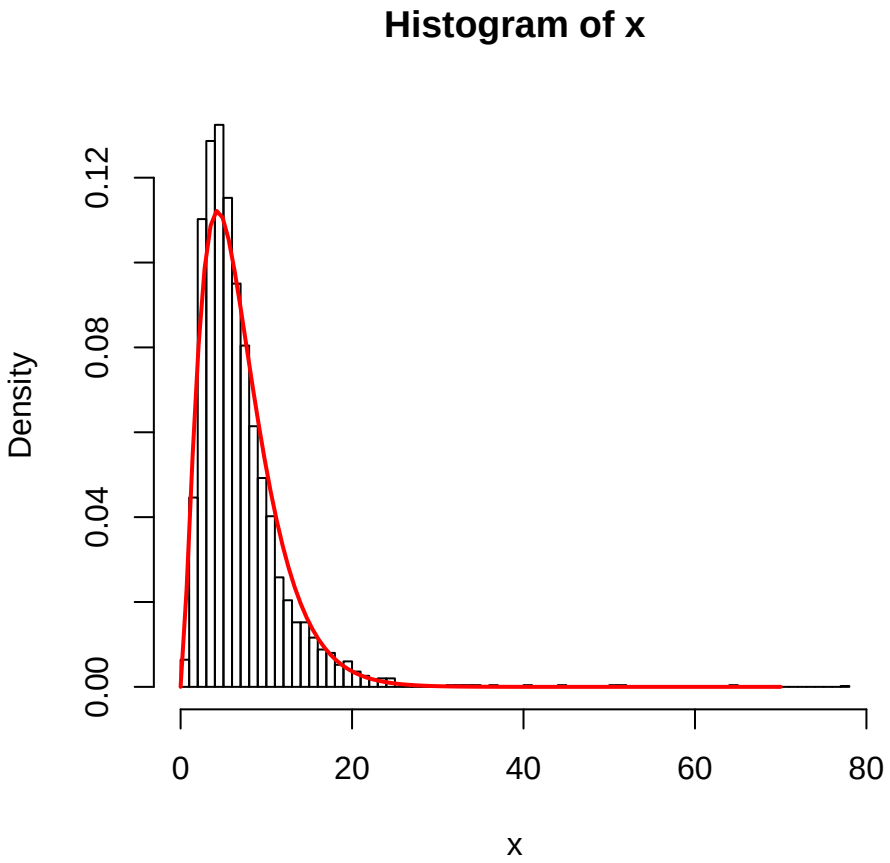



```
> plot(out2[, 2], type = "l")
```



(e) Find the posterior means for α and β. Plot the density of the corresponding Gamma distribution over the histogram of the data.

```
> hist(x, breaks = 100, freq = FALSE)
> alphahat <- mean(out2[, 1])
> betahat <- mean(out2[, 2])
> f <- function(y) dgamma(y, alphahat, betahat)
> plot(f, 0, 70, add = TRUE, col = 2, lwd = 2)
```



This looks like a reasonable fit, although actually the outliers are probably not modelled well by this distribution.

5. **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 `biv`.

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

## $x
##  [1]  1.46359532  0.01469893 -0.50192094 -0.52496882 -0.10071617
##  [6] -0.98791476 -0.66290314 -0.62183169  1.52414957  0.27134228
## [11] -1.20032589  0.34560332  0.31658560  0.61084736 -1.89824287
## [16]  0.53370213  0.70991664  1.28204780 -0.04829808  1.91677126
##
## $y
##  [1]  6  6  6  9  9 10  5  7  7  1  6  4  7  3  4  6  3  7  3  9
```

```
##
## attr(,"class")
## [1] "biv"
```

(b) Write a print method for `biv` (i.e. a function called `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 :  1.463595 0.01469893 -0.5019209 -0.5249688 -0.1007162 -0.9879148 ...
## y :  6 6 6 9 9 10 ...
```

(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.

6. **Functions**

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

the matrix

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

```
> 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 first order interactions. In other words, if the vectors were x, y, z, w we'd get

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

[You're not allowed to use `model.matrix()` or similar!]

```
> modelmat2 <- function(...) {
+     vecs <- list(...)
+     k <- length(vecs)
+     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 `model.matrix()`.

7. * **Mixtures**

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

(a) Write down the likelihood for one observation $\boldsymbol{X}^{(1)}$, 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.