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!

   (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.

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

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.

   (b) How many columns does `hills` have?

   (c) One of the races is called `Two Breweries`; change this to `Three Breweries`.

   (d) Using the function `with()`, find the mean time for races with a climb greater than 1000 feet.

   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`?

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

   (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
```

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`.]

(b) Evaluate the 20th Fibonacci number with it.

(c) How many times does the function have to evaluate itself to calculate this? Can you think of a faster way to do this with a loop?
Calculate F_{1000} with your new function.

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 `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 ...
```

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

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

(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

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

(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

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

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

(d) Check your answer with `model.matrix()`.

6. * **plyr**

   Look at the paper by Wickham (2011) on the `plyr` library: `www.jstatsoft.org/v40/i01/paper`

```
> library(plyr)
```

   Load the housing data from the MASS package:

```
> 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`.

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

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

7. * **Design Matrices**

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

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

   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 $\boldsymbol{X}^{(i)} = (X_{i1}, \ldots, X_{ik})$, 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.