UNIVERSITY OF TECHNOLOGY SYDNEY

School of Mathematical and Physical Sciences

37457 Advanced Bayesian Methods

LABORATORY 2

Due time and date: 9:55am, Wednesday 4th October, 2023.

Submission method: E-mail message to Professor Wand (`matt.wand@uts.edu.au`).

Goals: The goals of Laboratory 2 are:

- Introduce the R Bayesian inference package rstan and learn its basic infrastructure and syntax.
- Learn some of the more fundamental aspects of the R language, including lists, functions and looping mechanisms. As with any language (linguistic or computing) learning is gradual and expertise builds up over time. Note that R has thousands of functions, and is constantly being expanded on the Comprehensive R Archive Network.

1. **Using rstan for performing Markov Chain Monte Carlo for the Assignment 3, Question 5 setting**

   (a) This task assumes that the rstan package has been installed on the computer that you are using. If it is not yet installed then follow the instructions on this website:

   `http://matt-wand.utsacademics.info/webStatSem/rstanInstructs.html`

   (b) Start an R session and enter the following command:

   ```
   x2value <- 2.924
   ```

   Then hit `Enter`. This corresponds to setting the observed value of x_2 to be $\mathring{x}_2 = 2.924$.

   For the remainder of this laboratory when a command is given in `this font` then it assumed that you type this in an R session and then hit `Enter`.

   (c) **Aside: Recalling R commands.** Start an R session and enter the following command:

   ```
   789*314
   ```

   Then hit `Enter`. You should get `247746`. Now hit the up-arrow key. This would recall the previous command. Delete the `4` and replace it with `6` so that we now have the multiplication:

   ```
   789*316
   ```

   Then hit `Enter`. You should get `249324`.

   This exercise has nothing to do with the theme of Laboratory 2, but demonstrates a useful feature of R when using command mode when two or commands are similar to each other. Earlier commands can be recalled by hitting up-arrow multiple times.

   For the remainder of this laboratory when a command is given in `this font` then it assumed that you type this in an R session and then hit `Enter`.

(d) `library(rstan)`

This loads the rstan package to the current session.

(e) The next command is delicate. Firstly, it involves the *double* quote character `"`. It is important that you use the key for this character on your keyboard. Do not use the *single* quote characters `` ` `` or `´`. Also, note that the outer brackets are the curly variety:

$$\Big\{ \qquad\qquad \Big\}.$$

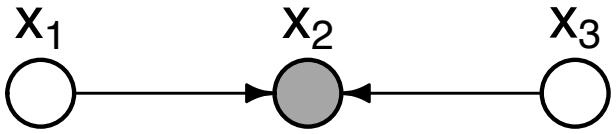
Lastly, make sure that you include all semi-colons (`;`).

```
as3qn5Model <- "data{real x2;}
   parameters{real x1;      real<lower=0> x3;}
   model{x2 ~ normal(3*x1+5,sqrt(1/(8*x3)));
         x1 ~ normal(13,0.25); x3 ~ gamma(10,1);}"
```

The command that you have just issued creates a character string named `as3qn5Model`. This character string will be used to tell rstan that you want to impose the following probabilistic DAG model on the data in `x2value`:

$$x_2|x_1, x_3 \sim N\big(3\,x_1 + 5, 1/(8x_3)\big), \quad x_1 \sim N\big(13, \tfrac{1}{16}\big), \quad x_3 \sim \text{Gamma}(10, 1). \qquad (1)$$

The DAG looks like:



In this model specification note the following:

- Since `x3` has to take on only positive values (since $1/(8x_3)$ is the conditional variance of $x_2|x_1, x_3$) the specification for `x3` includes `<lower=0>`. If we do not do this then negative variances may arise.
- The `normal` syntax is such that the second argument is the **standard deviation** rather than the variance. Hence, we need to take square-roots of what appears in the Normal distribution specifications in (1).

(f) This next command may take several seconds to run and, after typing it and hitting `Enter` there will be a pause before the result appears on the screen:

```
stanCompilObj <- stan(model_code = as3qn5Model,
                      data = list(x2 = x2value),
                      iter = 1,chains = 1)
```

Ignore any warning messages that appear. This command compiles the model in a form that the rstan Markov chain Monte Carlo (MCMC) engines can understand and stores it in `stanCompilObj`.

(g) This next command may take about 30–60 seconds to run depending on the capacity of your computer:

```
stanObj <- stan(model_code = as3qn5Model,
                data = list(x2 = x2value),warmup = 1000,
                iter = 1001000,chains = 1,
                refresh = 100000,
                fit = stanCompilObj)
```

(h)
```
x1MCMC <- extract(stanObj,"x1",permuted = FALSE)
x3MCMC <- extract(stanObj,"x3",permuted = FALSE)
```

These last three commands generate samples of sizes $1,001,000$ from the conditional distributions:

$$x_1|(x_2 = 2.924) \quad \text{and} \quad x_3|(x_2 = 2.924).$$

and then discard the first $1,000$ resulting in a kept sample of size 1 million.

(i)
```
par(mfrow=c(1,2))
hist(x1MCMC,col="lightblue",breaks=100,probability=TRUE)
hist(x3MCMC,col="lightblue",breaks=100,probability=TRUE)
```

These plots are histogram estimates of the conditional density functions

$$p(x_1|x_2 = 2.924) \quad \text{and} \quad p(x_3|x_2 = 2.924).$$

If you programmed the (correct) answer to Question 5 of Assignment 3 then you would obtain similar answers to those given here. The difference is that rstan does all of the work for you (i.e. it figures out the required full conditional distributions) and all you have to do is specify the DAG and feed in the data (i.e. an observed value of x_2).

(j) Before finishing this illustration of rstan we point out that the command in part (d) above is usually written as follows with a clearer layout of the data, parameters and model (but do not type this):

```
as3qn5Model <- "data
                {
                   real x2;
                }
                parameters
                {
                   real x1;
                   real<lower=0> x3;
                }
                model
                {
                   x2 ~ normal(3*x1+5,sqrt(1/(8*x3)));
                   x1 ~ normal(13,0.25);
                   x3 ~ gamma(10,1);
                }"
```

The more squished version was in part (d) was used to save time.

SUBMISSION ITEM: Save the graphic that is produced from this example in a file. These will form part of your submission for Laboratory 2.

2. **Learning about an R advanced data structure known as a list**
   Lists are a very useful data structure supported by R. Enter the following commands to produce a list of attributes about some Australian animals:

```
AusAnimals <- list(kangaroo=list(family="marsupial",legs=2,cover="fur"),
                   emu=list(family="bird",legs=2,cover="plumes"),
                   wombat=list(family="marsupial",legs=4,cover="fur"),
                   echidna=list(family="monotreme",legs=4,cover="spikes"))
```

   List components are accessed using the `$` sign. To get a feeling for this type

   (a) `AusAnimals$kangaroo`

   (b) `AusAnimals$wombat$legs`

SUBMISSION ITEM: Write R code to create a list named `myInfo` with the following components

   (a) `$name` - a character string with your full name.

   (b) `$universities` - an array of character strings containing each of the universities where you have either worked or studied.

   (c) `$otherinfo` - another list with the components: `$study` – a character string with your major area of study. `$birthmonth` – the month number corresponding to your birthday. `$postcode` – the post code of your home address.

3. **Learning about functions in R**
   Most of R involves calls to functions. Thousands of functions already exist in R. Examples are `sqrt()` for square-root, `plot()` for simple two-dimensional plots and `solve()` for matrix inversion. However, it is also useful to be able to create your own function.

   (a) Type

```
CtoF <- function(x)
   return(1.8*x + 32)
```

   You have just created a function called `CtoF()` for conversion from degrees Celsius to degrees Fahrenheit. To get a feeling for `CtoF()` type

   i. `CtoF(23)`

   ii. `CtoF(40)`

   iii. `CtoF(0)`

   iv. `CtoF(-40)`

   (b) Functions are not always mathematical. The following one takes the name of a common chemical element as a character string and returns its abbreviation.

   Type

```
elementAbbrev <- function(fullName)
{
   if (fullName=="copper") return("Cu")
   if (fullName=="iron") return("Fe")
   if (fullName=="lead") return("Pb")
   if (fullName=="gold") return("Au")
   if (fullName=="silver") return("Ag")
   if (!any(fullName==c("copper","iron","lead","gold","silver")))
      stop("argument not supported by this function")
}
```

To get a feeling for `elementAbbrev()` type

i. `elementAbbrev("lead")`

ii. `elementAbbrev("copper")`

iii. `elementAbbrev("chicken")`

SUBMISSION ITEMS:

(a) Write a function named `lenHypot()` that has two arguments, each of which is a positive number that represents the lengths of the sides of a right-angled triangle that are adjacent to the right angle. The function returns the length of the triangle's hypothenuse. For example, `lenHypot(5,12)` should return the value `13`. Include the code for `lenHypot()` in your submission.

(b) Write a function named `OlympicCity()` that has as its only argument a leap year between 2012 and 2032 inclusive. Ignore the fact that the 2020 summer Olympic Games were postponed for a year. The function returns a character string which is the name of the host city of the summer Olympic Games held in the specified year. For example, `OlympicCity(2024)` should return the value `"Paris"`. Include the code for `OlympicCity()` in your submission.

4. **Learning about `for` loops in R**

(a) A very simple example of a `for` loop is:

```
for (i in 1:5)
{
   print(i)
   print(i^2)
}
```

and prints the first 5 natural numbers and their squares.

(b) The *Fibonacci* numbers F_k, $k = 0, 1, 2, 3, \ldots$, are defined as according to:

$$F_0 = 0,\ F_1 = 1$$

and for $k > 1$

$$F_k = F_{k-1} + F_{k-2}.$$

Download the file `Fibonacci.r` and in an R session type:

`source("Fibonnacci.r")`

`Fibonacci(9)` which should give $F_9 = 34$.

`Fibonacci(22)` which should give $F_{22} = 17,711$.

Study the code in `Fibonacci.r` to see how a `for` loop is used to compute the kth Fibonacci number.

(c) Sometimes it is useful to have two or more `for` loops nested inside each other. Download the script `HockeyComp.R` and then type `source("HockeyComp.R")` to produce a data frame with all possible home and away matches for a hockey competition involving 6 teams, with randomisation of the order in which the matches are played. Study the code in `HockeyComp.R` to see how two nested `for` loops are used to construct the data frame.

(d) `library(MASS) ; x <- geyser$duration ; hist(x,col="gold",breaks=30)`

These commands store a univariate data set on durations of the eruption of a geyser in the array `x` and visualise the data via a histogram.

(e) In preparation for the next submission item we now give some simple commands involving arrays:

i. `x`
which prints the full data set.

ii. `x[88]`
which prints the 88th entry of the array `x`

iii. `length(x)`
which prints the length of the array `x` corresponding to the sample size.

SUBMISSION ITEM:
If $x_1, \ldots, x_n$ is a univariate data set then the *Gini mean difference* statistic is defined to be:

$$\text{GMD} \equiv \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|.$$

(a) Note that, in R, the function that computes the absolute value of a real number is `abs()`. Write an R function named `GiniMeanDiff()` that has as its input an array containing a univariate data set and returns the Gini mean difference of the data set.

Hint: Use two nested `for` loops in which the summation in the GMD formula is sequentially updated.

(b) Obtain `GiniMeanDiff(x)` where `x` is the geyser data set described above and report the result.

(c) Include the function `GiniMeanDiff()` in the submission.

5. **Learning about `while` loops in R**
A `while` loop differs from a `for` loop in that the number of times through the loop is not fixed in advance and instead looping continues until a specified condition is satisfied. We will illustrate the use of `while` loops to find roots of an equation via *Newton-Raphson* iteration (also known as *Newton's method*). If f is a smooth real-valued function on $\mathbb{R}$ then a solution to the equation

$$f(x) = 0$$

can be found via the following iterative scheme:

$$x_{\text{new}} = x_{\text{old}} - f(x_{\text{old}})/f'(x_{\text{old}}) \tag{2}$$

where f' is the first derivative of f. If the initial approximation is sufficiently close to the root then iterations of (2) lead to convergence to the root. As an illustration, consider the problem of finding the cube-root of $110,509$ (which is the lecturer's staff number). Whilst the R command `110509^(1/3)` gives the answer we will instead use Newton-Raphson iteration by noting that the answer is a the unique root of the equation

$$f(x) = 0 \quad \text{where} \quad f(x) = x^3 - 110,509. \tag{3}$$

(a) Download the file `NewtonRaphson.R` from the subject web-site and open it in an editor such as `WordPad` (but not `NotePad`).

(b) Study each of the commands in this script. In particular, the Newton-Raphson iterations are carried out using a `while` loop with the condition for continuing the loop being `!converged` where `converged` is a *Boolean* variable (i.e. a variable which takes the values `TRUE` or `FALSE`) and the exclamation mark (`!`) imposes negation on the logical condition.

(c) In R type `(1==2)` and hit `Enter`. Then type `!(1==2)` and hit `Enter`. These are simple illustrations of Boolean variables and negation via the exclamation mark.

(d) In R type `source("NewtonRaphson.R")`. This leads to Newton-Raphson iteration solution to (3).

(e) Enter the command `110509^(1/3)` as a check that the previous step produced the correct answer.

SUBMISSION ITEMS:

(a) Copy `NewtonRaphson.R` to a file named `myNewtonRaphson.R`. Edit the new file and replace `110509` by your student number. Run the script and check that it leads to the cube-root of your student number. Include `myNewtonRaphson.R` in your submission.

(b) Copy `myNewtonRaphson.R` to a new file named `myNewtonRaphsonHarder.R`. Consider the problem of obtaining the unique root of the equation

$$e^{x/38} + x^5/211 - 89 = 0.$$

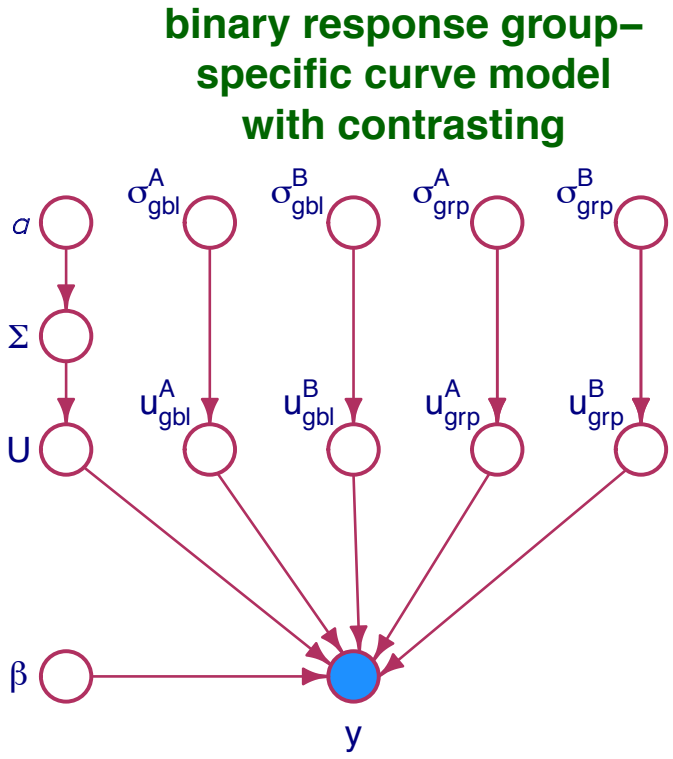
Modify `myNewtonRaphsonHarder.R` so that it finds this root and include the modified `myNewtonRaphsonHarder.R` in your submission.

OPTIONAL EXTRA: Running the Class 6 Coral Data Example

In Class 6 we presented a study involving comparison of different types of coral, including *Pocillopora* and *Porites*, in terms of their survival probabilities as a function of size. Fuller details are in the following 2015 paper involving the lecturer:

Kayal, M., Vercelloni, J., Wand, M.P. and Adjeroud, M. (2015). Searching for the best bet in life-strategy: a quantitative approach to individual performance and population dynamics in reef-building corals. *Ecological Complexity*, **23**, 73–84.

The DAG for the so-called group-specific curve model (to be detailed later in the subject) is as follows:



Fitting this model to the *Pocillopora*/*Porites* section of the coral data in Kayal *et al.* (2015) can be performed using just three commands in R assuming that the rstan package is installed. **However**, it is likely to take a few hours to run so it is better to start these commands before a long break from computer use (e.g. running overnight).

The commands are as follows. The first two commands are quick, but the last one requires a few hours depending on the capacity of your computer.

```
install.packages("HRW")

library(HRW)

demo(coralAna,package = "HRW")
```

♦ ♦ ♦