

# Bayesian methods in medical research

## Practicals solutions

- Exercise 1
- Exercise 2
- Exercise 3
- Exercise 4
- Exercise 5
- Exercise 6
- Exercise 7
- Exercise 8
- BONUS Exercise 9

## Exercise 1

In the context of the historical example, we want to elicitate a prior distribution from expert knowledge.

Now, let's imagine we have 2 expert demographers, each giving their *expert opinion* about the value they think plausible for  $\theta$  (the probability of a birth being a girl rather than a boy).

We ask each of them to give values for which the probability of  $\theta$  being lower would be respectively 10%, 25%, 50%, 75%, and 90%. First expert says:



$$P(\theta < 0.2) = 10\%, \quad P(\theta < 0.4) = 25\%, \quad P(\theta < 0.5) = 50\%, \quad P(\theta < 0.6) = 75\%, \quad \text{and } P(\theta < 0.8) = 90\%$$

while the second expert says:

$$P(\theta < 0.5) = 10\%, \quad P(\theta < 0.6) = 25\%, \quad P(\theta < 0.7) = 50\%, \quad P(\theta < 0.8) = 75\%, \quad \text{and } P(\theta < 0.9) = 90\%$$

0. First, let's load the `SHELF` R package:

```
library(SHELF)
```

1. Using the `fitdist()` function from the `SHELF`  package, estimate the parameter form the Beta distribution that best fit each of those elicitations (**Protip:** have a look at the package intro vignette here (<https://cran.r-project.org/web/packages/SHELF/vignettes/SHELF-overview.pdf>)).
2. Plot those two Beta prior distributions, along with their "linear pooling" using the `plotfit()` function from the `SHELF`  package (**Protip:** use the `lp = TRUE` argument).
3. Create a consensus Beta prior by averaging both expert answers, and plot it.

## Exercise 2

1. The Law of Large Numbers and Monte-Carlo Estimation.
  - a. Write a function which generates a sample of 10 observations *iid* from a Gaussian distribution (with mean  $m = 2$  and standard deviation  $s = 3$ ) and that returns the standard-deviation estimate over this sample (using the `sd()` function).

```
sd_est <- function(n = 10) {
  s <- rnorm(n, mean = 2, sd = 3)
  return(sd(s))
}
```

- b. Use the the Monte-Carlo method to estimate the standard deviation of tthe distribution generating this sample, by using multiple realizations (e.g. 5) of the standard-deviation estimate implemented above. Do it again with 1,000 realizations, thus illustrating the law of large number convergence.

```
# Monte-Carlo method
nMC <- 5
sdMC <- numeric(nMC)
for (i in 1:nMC) {
  sdMC[i] <- sd_est(n = 10)
}
mean(sdMC)
```

## 2. Let's now program a Monte-Carlo estimate of $\pi \approx 3, 1416$

- a. Program a function `roulette_coord` which has only one argument `ngrid` (representing the number of different outcomes possible on the *roulette* used) whose default is `35`, generating the two coordinates of a point (between 0 and 35) as a vector. Use the R function `sample` (whhose help page is accessible through the command `?sample`). The function will return the vector of the 2 coordinates `x` and `y` generated this way.

```
roulette_coord <- function(ngrid = 35) {
  x <- sample(x = 0:ngrid, size = 1)
  y <- sample(x = 0:ngrid, size = 1)
  return(c(x, y))
}
```

- b. Thanks to the formula to compute the distance bewteen 2 points:  $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ , program a function computing the distance to the origin (here has coordinates  $(\frac{ngrid}{2}, \frac{ngrid}{2})$ ) that checks if the computed distance is less than the unit disk radius ( $R = \frac{ngrid}{2}$ ). This function, called for instance `inside_disk_fun()`, will have 2 arguments: the vector `p` containing the coordinates of the points on the one hand, and the integer `ngrid` on the other hand. It will return a boolean value (`TRUE` or `FALSE`) indicating the point is inside the disk.

```
inside_disk_fun <- function(p, ngrid = 35) {
  d <- sqrt((p[1] - ngrid/2)^2 + (p[2] - ngrid/2)^2)
  return(d <= ngrid/2)
}
```

- c. The surface ratio between the disk (radius  $\frac{ngrid}{2}$ ) and the square (side length `ngrid`) is equal to  $\frac{\pi}{4}$ . This means that the probability of sampling a point inside the disk rather than outside is  $\frac{\pi}{4}$ . Using this result, a Monte Carlo estimate of  $\pi$  can be implemented by computing the average number of time sampled points fall inside the disk multiplied by 4. Program such a function with their only input being a boolean vector of size `n` (the number of sampled points), which is `TRUE` if the point is indeed inside the disk and `FALSE` otherwise.

- d. Using the code below, generate 200 points and plot the data generated. What is the corresponding Monte Carlo estimate of  $\pi$ ? Change `npoints` and comment. How could the estimation be improved (*ProTip*: try `ngrid <- 1000` and `npoints <- 5000`)?

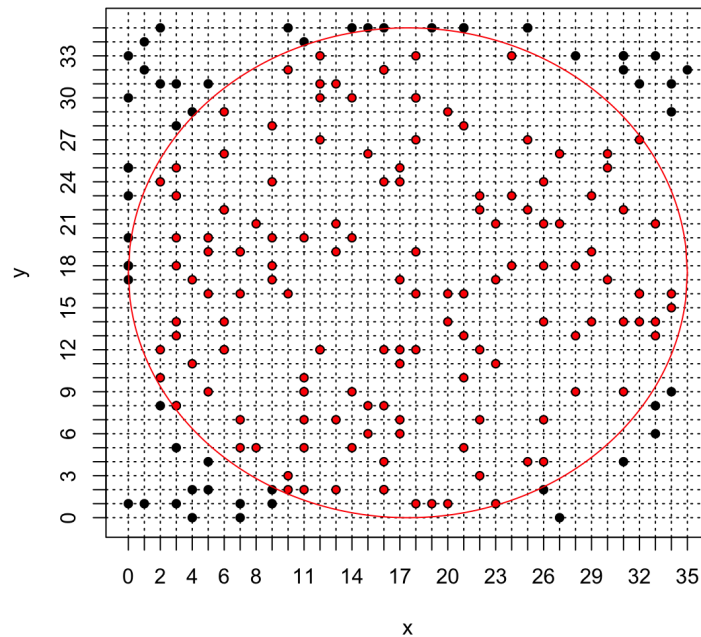
```
# Grid size (resolution)
ngrid <- 35

# Monte Carlo sample size
npoints <- 200

# Points generation
pp <- matrix(NA, ncol = 2, nrow = npoints)
for (i in 1:nrow(pp)) {
  pp[i, ] <- roulette_coord(ngrid)
}

# Estimate pi
in_disk <- apply(X = pp, MARGIN = 1, FUN = inside_disk_fun, ngrid = ngrid)
piMC(in_disk)

# Plot first we initialise an empty plot with the right size using argument
plot(x = pp[, 1], y = pp[, 2], xlim = c(0, ngrid), ylim = c(0, ngrid), axes = 0,
     xlab = "x", ylab = "y", type = "n")
## we tick the x and then y axes from 1 to ngrid
axis(1, at = c(0:ngrid))
axis(2, at = c(0:ngrid))
## we add a square around the plot
box()
## we plot the grid (using dotted lines thanks to the argument `lty = 3`) onto
## which the points are sample
for (i in 0:ngrid) {
  abline(h = i, lty = 3)
  abline(v = i, lty = 3)
}
## we add the sampled points
lines(x = pp[, 1], y = pp[, 2], xlim = c(0, ngrid), ylim = c(0, ngrid), xlab =
"x",
     ylab = "y", type = "p", pch = 16)
## we add the circle display
x.cercle <- seq(0, ngrid, by = 0.1)
y.cercle <- sqrt((ngrid/2)^2 - (x.cercle - ngrid/2)^2)
lines(x.cercle, y = y.cercle + ngrid/2, col = "red")
lines(x.cercle, y = -y.cercle + ngrid/2, col = "red")
## finally we color in red the points sampled inside the disk
lines(x = pp[in_disk, 1], y = pp[in_disk, 2], xlim = c(0, ngrid), ylim = c(0, ngrid),
     xlab = "x", ylab = "y", type = "p", pch = 16, col = "red", cex = 0.7)
```



## Exercise 3

Program a sampling algorithm to sample from the exponential distribution with parameter  $\lambda$  thanks to the inverse transform function (starting from the R function `runif`).

Compare the distribution of your sample to the theoretical target distribution (thanks to the built-in R function `dexp`).

Try out several values for the  $\lambda$  parameter of the exponential distribution (e.g. 1, 10, 0.78, ...) to check that the algorithm is indeed working.

```
generate_exp <- function(n, lambda) {
  u <- ...
  x <- ...
  return(x)
}
my_samp <- generate_exp(n = 100, lambda = 10)
hist(my_samp, probability = TRUE, n = 25)
curve(dexp(x, rate = 10), from = 0, to = max(my_samp), col = "red", lty = 2, add = TRUE)
legend("topright", c("Inverse transform", "R dexp()"), col = c("black", "red"), lty = c(1, 2))
```

## Exercise 4

Using the historical example, program an independent Metropolis-Hastings algorithm to estimate the *posterior* distribution of parameter  $\theta$  (i.e. the probability of having a girl for a birth). The *prior* distribution on  $\theta$  will be used as the instrumental proposal, and we will start by using a uniform *prior* on  $\theta$ . We will consider the 493,472 births observed in Paris between 1745 and 1770, of which 241,945 were girls.

1. Program a function that computes the numerator of the *posterior* density, which can be written  $p(\theta|n, S) \propto \theta^S(1 - \theta)^{n-S}$  with  $S = 241\,945$  and  $n = 493\,472$  (plan for a boolean argument that will allow to return — or not — the logarithm of the *posterior* instead).

```
post_num_hist <- function(theta, log = FALSE) {  
  
  n <- 493472 # the data  
  S <- 241945 # the data  
  
  if (log) {  
    num <- S * log(theta) + (n - S) * log(1 - theta) # the **log** numerator of  
the posterior  
  } else {  
    num <- theta^S * (1 - theta)^(n - S) # the numerator of the posterior  
  }  
  return(num) # the output of the function  
}  
  
post_num_hist(0.2, log = TRUE)
```

```
## [1] -445522.1
```

```
post_num_hist(0.6, log = TRUE)
```

```
## [1] -354063.6
```

2. Program the corresponding Metropolis-Hastings algorithm, returning a vector of size  $n$  sampled according to the *posterior* distribution. Also, have the algorithm return the vector of acceptance probabilities  $\alpha$ . What happens if this acceptance probability is **NOT** computed on the *log* scale ?

```

myMH <- function(niter, post_num) {

  x_save <- numeric(length = niter) #create a vector of 0s
  # of length niter to store the sampled values

  alpha <- numeric(length = niter) #create a vector of 0s
  # of length niter to store the acceptance probabilities

  # initialise x0
  x <- runif(n = 1, min = 0, max = 1)

  # acceptance-rejection loop
  for (t in 1:niter) {
    # sample y from the proposal (here uniform prior)
    y <- runif(n = 1, min = 0, max = 1)
    # compute the acceptance-rejection probability
    alpha[t] <- min(1, exp(post_num(y, log = TRUE) - post_num(x, log = TRUE)))
    # accept or reject
    u <- runif(1)
    if (u <= alpha[t]) {
      x_save[t] <- y
    } else {
      x_save[t] <- x
    }

    # update the current value
    x <- x_save[t]
  }
  return(list(theta = x_save, alpha = alpha))
}

```

3. Compare the *posterior* density obtained with this Metropolis-Hastings algorithm over 2000 iterations to the theoretical one (the theoretical density can be obtained with the R function `dbeta(x, 241945 + 1, 251527 + 1)` and represented with the R function `curve(..., from = 0, to = 1, n = 10000)`). Mindfully discard the first 500 iterations of your Metropolis-Hastings algorithm in order to reach the Markov chain convergence before constructing your Monte Carlo sample. Comment those results, especially in light of the acceptance probabilities computed throughout the algorithm, as well as the different sampled values for  $\theta$ .
4. Now imagine we only observe 100 births, among which 49 girls, and use a  $\text{Beta}(\alpha = 3, \beta = 3)$  distribution as *prior*. Program the corresponding M-H algorithm and study the new results (one can do 10,000 iterations of this new M-H algorithm for instance, again mindfully discarding the first 500 iterations).
5. Using the data from the historical example and with a  $\text{Beta}(\alpha = 3, \beta = 3)$  *prior*, program a random-walk Metropolis-Hastings algorithm (with a uniform random step of width 0.02 for instance). This means that the proposal is going to change, and is now going to depend on the previous value. Once again, study the results obtained this way (one can change the width of the random step).

## Exercise 5

The BUGS project (<https://www.mrc-bsu.cam.ac.uk/software/bugs/>) (*Bayesian inference Using Gibbs Sampling*) was initiated in 1989 by the MRC (*Medical Research Council*) Biostatistical Unit at the University of Cambridge (United-Kingdom) to develop a flexible and user-friendly software for Bayesian analysis of complex models through MCMC algorithms. Its most famous and original implementation is `WinBUGS`, a clicking software available under *Windows*.

`OpenBUGS` is an alternative implementation of `WinBUGS` running on either *Windows*, *Mac OS* ou *Linux*. `JAGS` (<http://mcmc-jags.sourceforge.net/>) (*Just another Gibbs Sampler*) is a different and newer implementation that also relies on the `BUGS` language. Finally, the `STAN` (<http://mc-stan.org/>) software must also be mentioned, recently developed et the Columbia Univeristy, resemble `BUGS` through its interface, but relies on innovative MCMC approaches, such as Hamiltonian Monte Carlo, or variational Bayes approaches. A very useful resource is the JAGS user manual ([http://sourceforge.net/projects/mcmc-jags/files/Manuals/3.x/jags\\_user\\_manual.pdf](http://sourceforge.net/projects/mcmc-jags/files/Manuals/3.x/jags_user_manual.pdf)).

To familiarise yourself with `JAGS` (and its `R` interface through the package `rjags`), we will look here at the *posterior* estimation of the mean and variance of observed data that we will model with a Gaussian distribution.

0. Start by loading the `R` package `rjags`.

```
library(rjags)
```

A `BUGS` model has 3 components:

- *the model*: specified in an external text file ( `.txt` ) according to a specific `BUGS` syntax
- *the data*: a list containing each observation under a name matching the one used in the model specification
- *the initial values*: (optional) a list containing the initial values for the various parameters to be estimated

1. Sample  $N = 50$  observations from a Gaussian distribution with mean  $m = 2$  and standard deviation  $s = 3$  using the `R` function `rnorm` and store it into an object called `obs`.

```
N <- 50 # the number of observations
obs <- rnorm(n = N, mean = 2, sd = 3) # the (fake) observed data
```

2. Read the help of the `rjags` package, then save a text file ( `.txt` ) the following code defining the `BUGS` model:

```
# Model
model{

  # Likelihood
  for (i in 1:N){
    obs[i]~dnorm(mu,tau)
  }

  # Prior
  mu~dnorm(0,0.0001) # proper but very flat (so weakly informative)
  tau~dgamma(0.0001,0.0001) # proper, and weakly informative (conjugate for Gaussian)

  # Variables of interest
  sigma <- pow(tau, -0.5)
}
```

Each model specification file must start with the instruction `model{` indicating JAGS is about to receive a model specification. Then the model must be set up, usually by cycling along the data with a `for` loop. Here, we want to declare `N` observations, and each of them `obs[i]` follows a Gaussian distribution (characterized with the command `dnorm`) of mean `mu` and precision `tau`.

⚠ In BUGS, the Gaussian distribution is parameterized by its **precision**, which is simply the inverse of the variance ( $\tau = 1/\sigma^2$ ). Then, one needs to define the *prior* distribution for each parameter -- here both `mu` and `tau`. For `mu`, we use a Gaussian *prior* with mean 0 and precision  $10^{-4}$  (thus variance 10,000: this corresponds to a weakly informative *prior* quite spread out given the scale of our data. For `tau` we use the conjugate *prior* for precision in a Gaussian model, namely the Gamma distribution (with very small parameters, here again to remain the least informative possible). Finally, we give a deterministic definition of the additional parameters of interest, here the standard deviation `sigma`.

**NB:** `~` indicates probabilistic distribution definition of a random variable, while `<-` indicates a deterministic calculus definition.

3. With the R function `jags.model()`, create a `jags` object R.

```
myfirstjags <- jags.model("normalBUGSmodel.txt", data = list(obs = obs, N = length(obs)))
```

4. With the R function `coda.samples()`, generate a sample of size 2,000 from the *posterior* distributions for the mean and standard deviation parameters.

```
res <- coda.samples(model = myfirstjags, variable.names = c("mu", "sigma"), n.iter = 2000)
```

5. Study the output of the `coda.samples()` R function, and compute both the *posterior* mean and median estimates for `mu` and `sigma`. Give a credibility interval at 95% for both.

6. Load the `coda` R package. This package functions for convergence diagnostic and analysis of MCMC algorithm outputs.

```
library(coda)
```

7. To diagnose the convergence of an MCMC algorithm, it is necessary to generate different Markov chains, with different initial values. Recreate a new `jags` object in R and specify the use of 3 Markov chains with the argument `n.chains`, and initialize `mu` at 0, -10, 100 and `tau` at 1, 0.01, 0.1 respectively with the argument `inits` (**ProTip:** use a `list` of `list`, one for each chain).

```
inits = list(list(mu = 0, tau = 1), list(mu = -10, tau = 1/100), list(mu = 100, tau = 1/10))
```

8. With the R function `gelman.plot()`, plot the Gelman-Rubin statistic.

9. With the R functions `autocorr.plot()` and `acfplot()` evaluate the autocorrelation of the studied parameters.



- With the R function `cumuplot()` evaluate the running quantiles of the studied parameters. How can you interpret them ?
- With the function `hdi()` from the R package `HDInterval`, provide highest density *posterior* credibility intervals at 95%, and compare them to those obtained with the 2.5% and 97.5% quantiles.

## Exercise 6

The randomized clinical trial *EOLIA*<sup>1</sup> evaluated a new treatment for severe acute respiratory distress syndrome (severe ARDS) by comparing the mortality rate after 60 days among 249 patients randomized between a control group (receiving conventional treatment, i.e. mechanical ventilation) and a treatment group receiving extracorporeal membrane oxygenation (ECMO) – the new treatment studied. A frequentist analysis of the data concluded to a Relative Risk of death of 0.76 in the ECMO group compared to controls (in Intention to Treat), with  $CI_{95\%} = [0.55, 1.04]$  and the associated p-value of 0.09.

Goligher *et al.* (2019)<sup>2</sup> performed a Bayesian re-analysis of these data, further exploring the evidence and how it can be quantified and summarized with a Bayesian approach.

Observed data from the *EOLIA* trial

	Control	ECMO
<b><i>n</i> observed</b>	125	124
<b>number of deceased at 60 days</b>	57	44

- Write the Bayesian model used by Goligher *et al.* (2019).
- Write the corresponding BUGS model, and save it into a `.txt` file (for instance called `goligherBUGSmodel.txt`)
- First create two binary data vectors `ycontrol` and `yecmo`, that are either 1 or 0, to encode the observations from the data table above. Then uses the `jags.model()` and `coda.samples()` to replicate the estimation from Goligher *et al.* (2019) (**ProTip:** use the function `window()` to remove the burn-in observation from the output of the `coda.samples` function.)
- Check the convergence, and then comment the estimate results (**ProTip:** look at the effective sample size with the `effectiveSize()` R function).
- Change to a more informative *prior* using a Gaussian distribution for the log(RR), centered on log(0.78) and with a standard deviation of 0.15 in the log(RR) scale (i.e. a precision of  $\approx 45$ ). Comment the results. Try out other *prior* distributions.


## Exercise 7

In 2014, Crins *et al.*<sup>3</sup> published a meta-analysis assessing the incidence of acute rejection (AR) with or without Interleukin-2 receptor antagonists. In this exercise we will recreate this analysis.

- Load the R package `bayesmeta`<sup>4</sup> and the data from Crins *et al.* (2014) with the R command `data("CrinsEtAl2014")`.

```
library(bayesmeta)
data(CrinsEtAl2014)
```

1. Play around with the companion shiny app at: <https://rshiny.gwdg.de/apps/bayesmeta/> (<https://rshiny.gwdg.de/apps/bayesmeta/>).

If the website is unavailable, you can launch the app locally by running the 2 following commands from :


```
library("shiny")
install.packages("rhandsontable")
runUrl("http://www.gwdg.de/~croever/bayesmeta/bayesmeta-app.zip")
```

2. Within R now, using the `escalc()` function from the package `metafor`, compute the estimated *log odds ratios* from the 6 considered studies alongside their sampling variances (**ProTip:** read the *Measures for Dichotomous Variables* section from the help of the `escalc()` function). Check that those are the same as the one on the online *shiny app* (**ProTip:** ‘sigma’ is the standard error, i.e. the square root of the sampling variance  $v_i$ )

```
library("metafor")
crins.es <- escalc(measure = "OR", ai = exp.AR.events, nli = exp.total, ci = cont.AR.events,
  n2i = cont.total, slab = publication, data = CrinsEtAl2014)
crins.es[, c("publication", "yi", "vi")]
```

publication	yi	vi
Heffron (2003)	-2.3097026	0.3593718
Gibelli (2004)	-0.4595323	0.3095760
Schuller (2005)	-2.3025851	0.7750000
Ganschow (2005)	-1.7578579	0.2078161
Spada (2006)	-1.2584610	0.4121591
Gras (2008)	-2.4178959	2.3372623

NB: Log-odds ratios are symmetric around zero and have a sampling distribution closer to the normal distribution than the natural OR scale. For this reason, they are usually preferred for meta-analyses. Their sample variance is then computed as the sum of the inverse of all the counts in the  $2 \times 2$  associated contingency table<sup>5</sup>.

3. Perform a random-effect meta-analysis of those data using the `bayesmeta()` function from the  package `bayesmeta`, within R. Use a uniform *prior* on  $[0, 4]$  for  $\tau$  and a Gaussian *prior* for  $\mu$  centered around 0 and with a standard deviation of 4.
4. Write the corresponding random-effects Bayesian meta-analysis model (using math, not R – yet).
5. Use `rjags` to estimate the same model, saving the BUGS model written below in a `.txt` file (called `crinsBUGSmodel.txt` for instance).

## Exercise 8

We will now analyze data from the therapeutic clinical trial *ALBI-ANRS 070* which compared the efficacy and tolerance of 3 anti-retrovirales strategies among HIV-1 positive patients naive of any anti-retroviral treatment <sup>6</sup>.

1. Load the data from the `albianrs_data.csv` available here [📄 \(/files/albianrs\\_data.csv\)](#) (**ProTip:** set `stringsAsFactors = TRUE` in `read.delim()`)

```
albi <- read.csv("albianrs_data.csv", stringsAsFactors = TRUE, )
summary(albi)
```

The following variables are available:

- `PatientID` : Patient ID
- `ViralLoad` : Plasma viral load
- `CD4` : CD4 T-cell lymphocytes rate (in cell/mm<sup>3</sup>)
- `CD4t` : transformed CD4 T-cell lymphocytes rate (in cell/mm<sup>3</sup>) ( $CD4t = CD4^{1/4}$ )
- `CD4sup500` : binary variable indicating wether CD4 cell counts is above 500
- `Treatment` : Treatment group 1 (*d4t + ddl*), 2 (alternance) ou 3 (*AZT + 3TC*)
- `Day` : Day of visit (since inclusion)
- `Visit` : Visit number

Below is a quantitative summary of the characteristics of those variables.

### Summary of the Albi-ANRS-070 trial data

Variable	N = 950 <sup>1</sup>
ViralLoad	2.71 (2.04, 3.69)
CD4	465 (377, 585)
CD4t	4.64 (4.41, 4.92)
CD4sup500	392 (41%)
Treatment	
AZT+3TC	315 (33%)
d4t+ddl	322 (34%)
switch	313 (33%)
Day	84 (29, 140)
Visit	
0	146 (15%)
1	140 (15%)
2	136 (14%)

<sup>1</sup> Median (IQR) or n (%)

Variable	N = 950 <sup>1</sup>
3	130 (14%)
4	128 (13%)
5	135 (14%)
6	135 (14%)

<sup>1</sup> Median (IQR) or n (%)

**NB:** for simplicity, NA values have been omitted.

2. First, we want to explain the CD4 rate (as a transformed variable) from the viral load

a. Display `CD4t` in scatterplot as a function of the `ViralLoad` and color the points according to `Treatment`.

```
library(ggplot2)
library(MetBrewer)
ggplot(albi, aes(y = CD4t, x = ViralLoad, color = Treatment)) + geom_point(alpha = 0.7) +
  MetBrewer::scale_color_met_d("Java") + theme_bw()
```

b. Write down the Bayesian mathematical model corresponding to a linear regression of `CD4t` against the `ViralLoad` stratified on the `Treatment`

c. Write the corresponding `BUGS` model and save it in an external `.txt` file.

d. Create the corresponding `jags` object in **R** and generate a Monte Carlo sample of size 1000 for the 3 parameters of interest

e. Before interpreting the results, check the convergence. What do you notice ?

f. Using the `window` function, remove the 500 first iterations as burn-in to reach Markov Chain convergence to its stationary distribution (the posterior). Is it sufficient to solve convergence issues ?

g. Check the effective sample size of the Monte Carlo sample with the `effectiveSize()` function. Reduce auto-correlation by increasing the `thin` parameter to 10 in `coda.samples`. Check the impact on the effective sample

3. Compare those results with a frequentist analysis.

4. Perform a Bayesian logistic regression to study the impact of the treatment on the binary outcome `CD4sup500` adjusted on the viral load. Once you have a first estimation, try adding an interaction between the treatment and the viral load.

## BONUS Exercise 9

In this exercise, we will first do a critical reading of the article from Kaguelidou *et al.* (2016)<sup>7</sup>.

1. List the method elements that are missing from this article.
2. Read and discuss Table 3.

3. Load the R package `bcrm` and conduct an imaginary CRM trial interactive with the following code lines:

```
library(bcrm)
sdose <- c(1, 1.5, 2, 2.5, 3)
dose.label <- c(5, 10, 15, 25, 40, 50, 60)
p.tox0 <- c(0.01, 0.015, 0.02, 0.025, 0.03, 0.04, 0.05)
bcrm(stop = list(nmax = 42), p.tox0 = p.tox0, dose = dose.label, ff = "power", prior.alpha = list(1, 1, 1), target.tox = 0.3, start = 1)
```

1. Alain Combes et al., “Extracorporeal Membrane Oxygenation for Severe Acute Respiratory Distress Syndrome,” *New England Journal of Medicine* 378, no. 21 (2018): 1965–75, doi:10.1056/NEJMoa1800385 (<https://doi.org/10.1056/NEJMoa1800385>).↵
2. Ewan C. Goligher et al., “Extracorporeal Membrane Oxygenation for Severe Acute Respiratory Distress Syndrome and Posterior Probability of Mortality Benefit in a Post Hoc Bayesian Analysis of a Randomized Clinical Trial,” *JAMA* 320, no. 21 (2018): 2251, doi:10.1001/jama.2018.14276 (<https://doi.org/10.1001/jama.2018.14276>).↵
3. Nicola D Crins et al., “Interleukin-2 Receptor Antagonists for Pediatric Liver Transplant Recipients: A Systematic Review and Meta-Analysis of Controlled Studies,” *Pediatric Transplantation* 18, no. 8 (2014): 839–50, doi:10.1111/petr.12362 (<https://doi.org/10.1111/petr.12362>).↵
4. Christian Röver, “Bayesian Random-Effects Meta-Analysis Using the Bayesmeta R Package,” *Journal of Statistical Software* 93 (2020): 1–51, doi:10.18637/jss.v093.i06 (<https://doi.org/10.18637/jss.v093.i06>).↵
5. Joseph L. Fleiss and Jesse A. Berlin, “Effect Sizes for Dichotomous Data,” in *The Handbook of Research Synthesis and Meta-Analysis, 2nd Ed* (New York, NY, US: Russell Sage Foundation, 2009), 237–53.↵
6. Jean-Michel Molina et al., “The ALBI Trial: A Randomized Controlled Trial Comparing Stavudine Plus Didanosine with Zidovudine Plus Lamivudine and a Regimen Alternating Both Combinations in Previously Untreated Patients Infected with Human Immunodeficiency Virus,” *The Journal of Infectious Diseases* 180, no. 2 (1999): 351–58, doi:10.1086/314891 (<https://doi.org/10.1086/314891>).↵
7. Florentia Kaguelidou et al., “Dose-Finding Study of Omeprazole on Gastric pH in Neonates with Gastro-Esophageal Acid Reflux Using a Bayesian Sequential Approach,” ed. Imti Choonara, *PLOS ONE* 11, no. 12 (2016): e0166207, doi:10.1371/journal.pone.0166207 (<https://doi.org/10.1371/journal.pone.0166207>).↵