

# 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 asks 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:

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

while the second expert says:

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

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](#)).

```
v <- cbind(c(0.2, 0.4, 0.5, 0.6, 0.8),
            c(0.5, 0.6, 0.7, 0.8, 0.9))
)
p <- c(0.1, 0.25, 0.5, 0.75, 0.9)

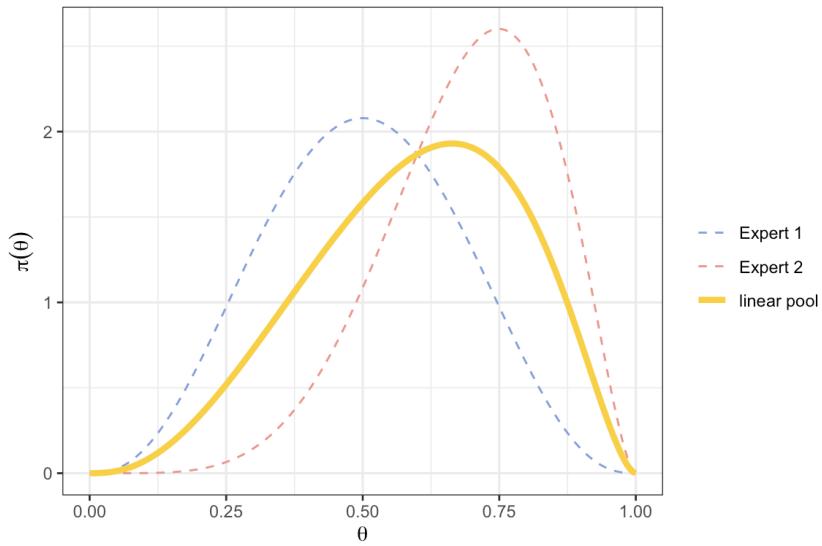
expertPriors <- fitdist(vals = v, probs = p, lower = 0, upper = 1,
                        expertnames=c("Expert 1", "Expert 2"))
```

```
expertPriors$Beta
```

```
##           shape1   shape2
## Expert 1 3.634819 3.634835
## Expert 2 6.047483 2.692690
```

2. Plot those two Beta prior distributions, along with the “linear pooling” of their 2 curves using the `plotfit()` function from the `SHELF`  package (**Protip:** use the `lp = TRUE` argument).

```
expertPlot <- plotfit(expertPriors, d = "beta", lp = TRUE, xl = 0, xu = 1,
                      xlab = expression(theta), ylab = expression(pi(theta)),
                      returnPlot = TRUE)
```



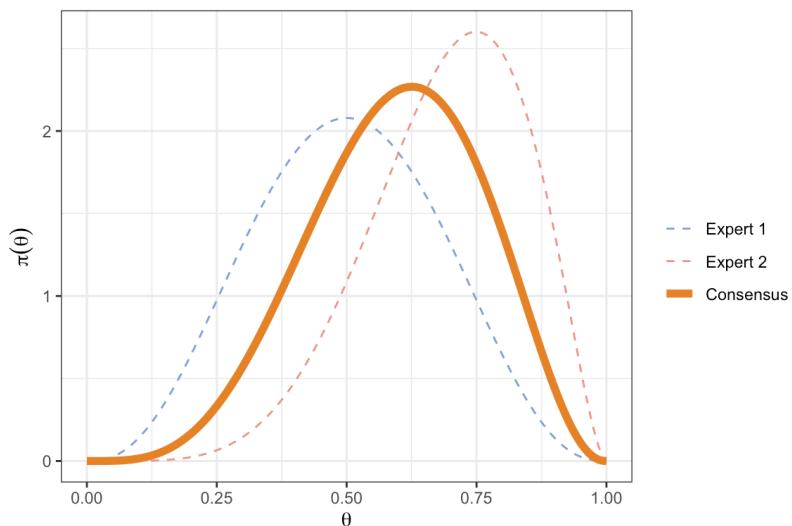
3. Derive a *consensus* Beta prior, by averaging each of both expert quantiles, and plot it.

```
v <- cbind(c(0.2, 0.4, 0.5, 0.6, 0.8),
            c(0.5, 0.6, 0.7, 0.8, 0.9),
            c(0.35, 0.5, 0.6, 0.7, 0.85))
p <- c(0.1, 0.25, 0.5, 0.75, 0.9)
consensusPrior <- fitdist(vals = v, probs = p, lower = 0, upper = 1,
                           expertnames=c("Expert 1", "Expert 2", "Consensus"))
```

```
consensusPrior$Beta
```

```
##           shape1   shape2
## Expert 1 3.634819 3.634835
## Expert 2 6.047483 2.692690
## Consensus 4.822237 3.288318
```

```
consensusPlot <- plotfit(consensusPrior, d = "beta", xl = 0, xu = 1,
                           xlab = expression(theta), ylab = expression(pi(theta)),
                           returnPlot = TRUE)
```



# 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  $s = 3$ , i.e. a variance of 9) and that returns the variance estimate over this sample (using the `var()` function).

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

- b. Use the the Monte-Carlo method to estimate the variance of the 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 # Monte Carlo sample size  
varMC <- numeric(nMC)  
for (i in 1:nMC) {  
  varMC[i] <- var_est(n = 10)  
}  
mean(varMC) # LLN estimate
```

```
## [1] 8.634555
```

```
nMC <- 1000 # Monte Carlo sample size  
for (i in 1:nMC) {  
  varMC[i] <- var_est(n = 10)  
}  
mean(varMC) # LLN estimate
```

```
## [1] 8.941702
```

## Correction:

*When the sample size increases, the estimator becomes more precise. This illustrates the Law of Large Numbers.*

## 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  function `sample()` (whose 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 between 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.

```

piMC <- function(in_disk){
  prop_in_disk <- mean(in_disk)
  pi_est <- 4 * prop_in_disk
  return(pi_est)
}

```

- 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)

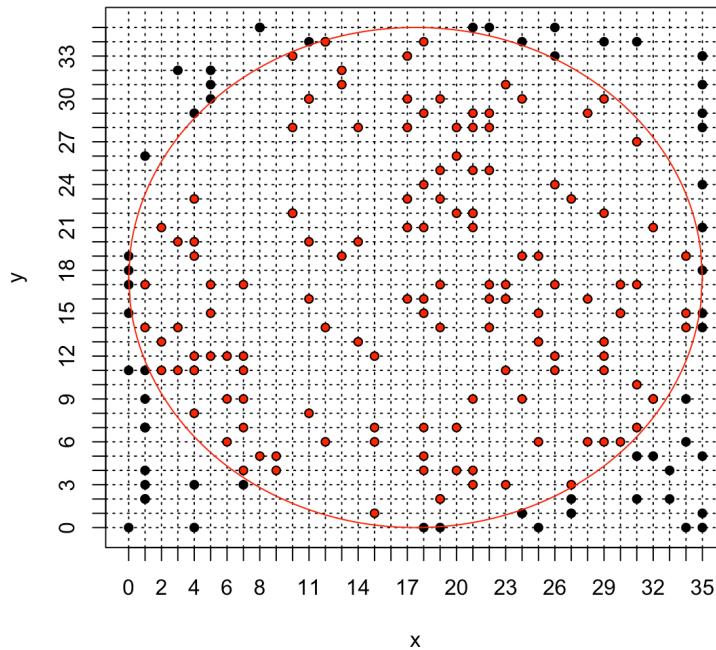
# Plot
## first we initialise an empty plot with the right size
## using argument
plot(x = pp[, 1], y = pp[, 2],
      xlim = c(0, ngrid), ylim = c(0, ngrid),
      axes = 0, xlab = "x", ylab = "y",
      type="n")
## we tick the x and then y axes from 1 to ngrid
axis(1, at=c(0:ngrid))
axis(2, at=c(0:ngrid))
## 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.circle <- seq(0, ngrid, by = 0.1)
y.circle <- sqrt((ngrid/2)^2 - (x.circle-ngrid/2)^2)
lines(x.circle, y = y.circle + ngrid/2, col = "red")
lines(x.circle, y = - y.circle + 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)

```



## Correction:

When the sample size increase, the Monte Carlo estimator becomes more precise (LLN). However, if the grid is too coarse,  $\hat{\pi}$  is underestimating the disk surface by missing the bits near the edge. Therefore, increasing the number of points on the grid also improves the precision of the Monte Carlo estimation.

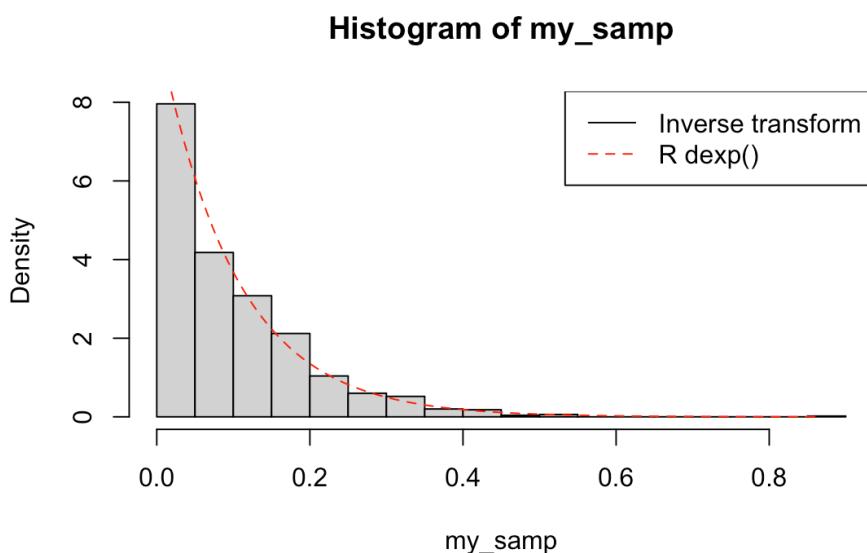
## Exercise 3

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

Compare the distribution of your sample to the theoretical target distribution (thanks to the built-in  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 <- runif(n)  
  x <- -1/lambda * log(1 - u)  
  return(x)  
}  
  
n_samp <- 1000  
my_samp <- generate_exp(n = n_samp, 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()"), lty = c(1, 2), col = c("black", "red"))
```



## 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  $\text{dbeta}$  function

$\text{dbeta}(x, 241945 + 1, 251527 + 1)$  and represented with the  $\text{curve}$  function

$\text{curve}(..., \text{from} = 0, \text{to} = 1, \text{n} = 10000)$ ). 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  $\theta$ .

```

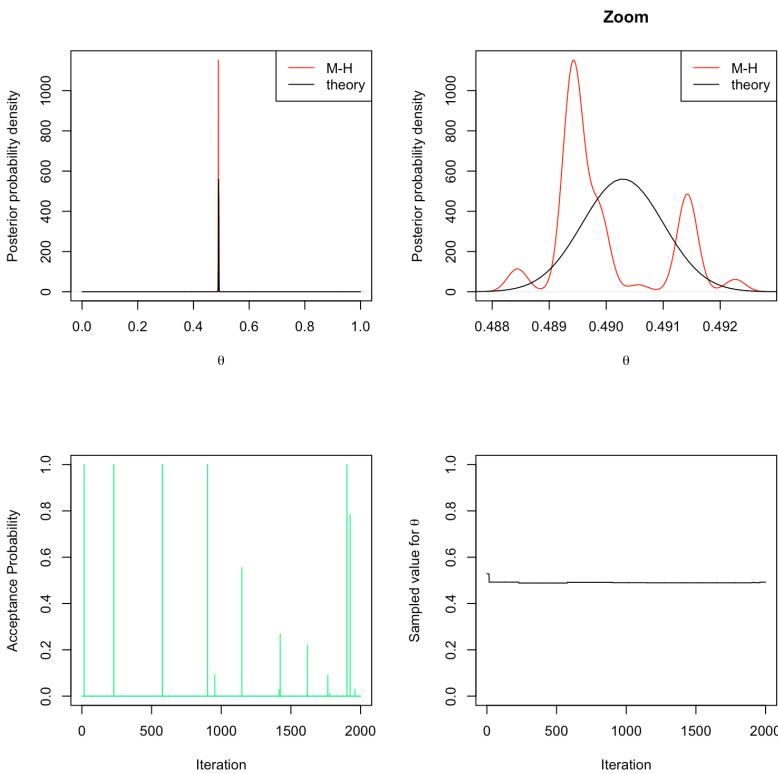
sampleMH <- myMH(2000, post_num = post_num_hist)

par(mfrow=c(2,2))
plot(density(sampleMH$theta[-c(1:500)]), col = "red", xlim = c(0,1),
      ylab = "Posterior probability density",
      xlab = expression(theta), main = "")
curve(dbeta(x, 241945 + 1, 251527 + 1), from = 0, to = 1, n = 10000, add = TRUE)
legend("topright", c("M-H", "theory"), col = c("red", "black"), lty=1)

plot(density(sampleMH$theta[-c(1:500)]), col = "red",
      ylab = "Posterior probability density", xlab = expression(theta),
      main = "Zoom")
curve(dbeta(x, 241945 + 1, 251527 + 1), from = 0, to = 1, n = 10000, add = TRUE)
legend("topright", c("M-H", "theory"), col=c("red", "black"), lty = 1)

plot(sampleMH$alpha, type = "h", xlab = "Iteration",
      ylab = "Acceptance Probability", ylim = c(0,1), col = "springgreen")
plot(sampleMH$theta, type = "l", xlab = "Iteration",
      ylab = expression(paste("Sampled value for ", theta))), ylim = c(0,1))

```



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).

```
post_num_beta <- function(theta, a = 3, b = 3, log = TRUE) {
  n <- 100 #number of trials (births)
  S <- 49 #number of success (feminine births)

  if (log) {
    num <- (a + S - 1) * (log(theta)) + (b + n - S - 1) * log(1 - theta)
  } else {
    num <- theta^(a + S - 1) * (1 - theta)^(b + n - S - 1)
  }
  return(num)
}

myMH_betaprior <- function(niter, post_num, a = 3, b = 3) {
  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 x
  x <- runif(n = 1, min = 0, max = 1)

  # acceptance-rejection loop
  for (t in 1:niter) {

    # sample a value from the proposal (beta prior)
    y <- rbeta(n = 1, shape1 = a, shape2 = b)

    # compute acceptance-rejection probability
    alpha[t] <- min(1, exp(post_num(y, a = a, b = b, log = TRUE) -
                           post_num(x, a = a, b = b, log = TRUE) +
                           dbeta(x, a, b, log = TRUE) - dbeta(y, a, b, log = TRUE)))

    # acceptance-rejection step
    u <- runif(1)
```

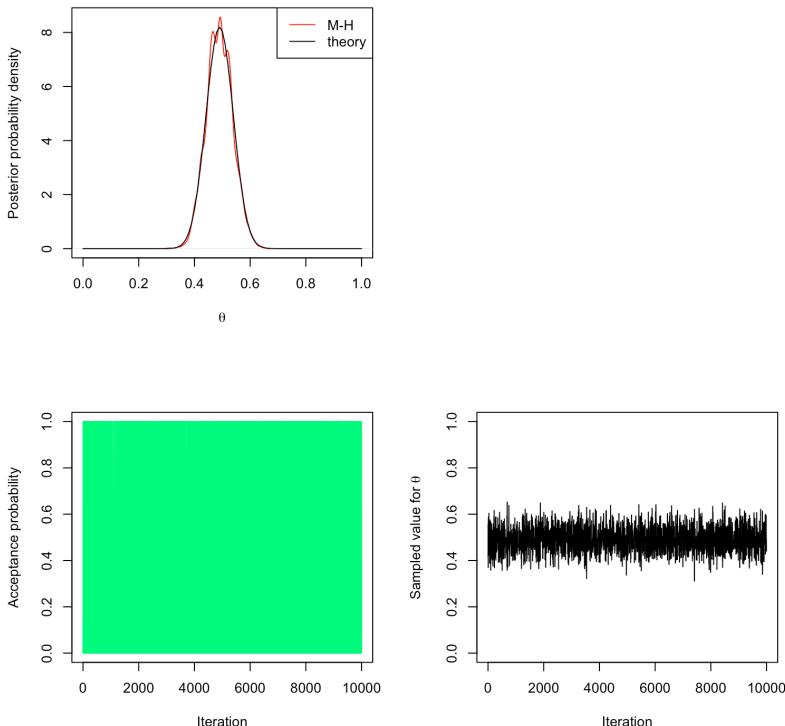
```

  if (u <= alpha[t]) {
    x <- y # acceptance of y as new current value
  }
  # saving the current value of x
  x_save[t] <- x
}
return(list("theta" = x_save, "alpha" = alpha))
}

sampleMH <- myMH_betaprior(10000, post_num = post_num_beta)

par(mfrow=c(2,2))
plot(density(sampleMH$theta[-c(1:500)]), col = "red", xlim = c(0,1),
      ylab = "Posterior probability density",
      xlab = expression(theta), main = "")
curve(dbeta(x, 3 + 49, 3 + 51), from = 0, to = 1, add = TRUE)
legend("topright", c("M-H", "theory"), col = c("red", "black"), lty = 1)
plot.new()
plot(sampleMH$alpha, type = "h", xlab = "Iteration",
      ylab = "Acceptance probability", ylim = c(0,1), col = "springgreen")
plot(sampleMH$theta, type = "l", xlab = "Iteration",
      ylab = expression(paste("Sampled value for ", theta)), ylim = c(0,1))

```



5. Using the full data from the historical example and with a  $\text{Beta}(\alpha = 3, \beta = 3)$  prior, program a random-walk Metropolis-Hastings algorithm (with a Gaussian random step of `sd=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 by reducing or augmenting its `sd` ).

```

post_num_beta_hist <- function(theta, a = 3, b = 3, log = TRUE){

  n <- 493472 #number of trials (births)
  S <- 241945 #number of success (feminine births)

  if(log){
    num <- (a + S - 1) * log(theta) + (b + n - S - 1) * log(1-theta)
  }
}

```

```

}else{
  num <- theta^(a + S - 1) * (1-theta)^(b + n - S - 1)
}
return(num)
}

myMH_betaprior_randomwalk <- function(niter, post_num, a=3, b=3){

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

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

  # acceptance-rejection loop
  for(t in 1:niter){
    # sample a value from the proposal (random walk)
    y <- rnorm(1, mean = x, sd=0.02)

    # compute acceptance-rejection probability
    alpha[t] <- min(1, exp(post_num(y, a=a, b=b, log=TRUE) -
                           post_num(x, a=a, b=b, log=TRUE)))

    # acceptance-rejection step
    u <- runif(1)
    if(u <= alpha[t]){
      x <- y # accept y and update current value
    }

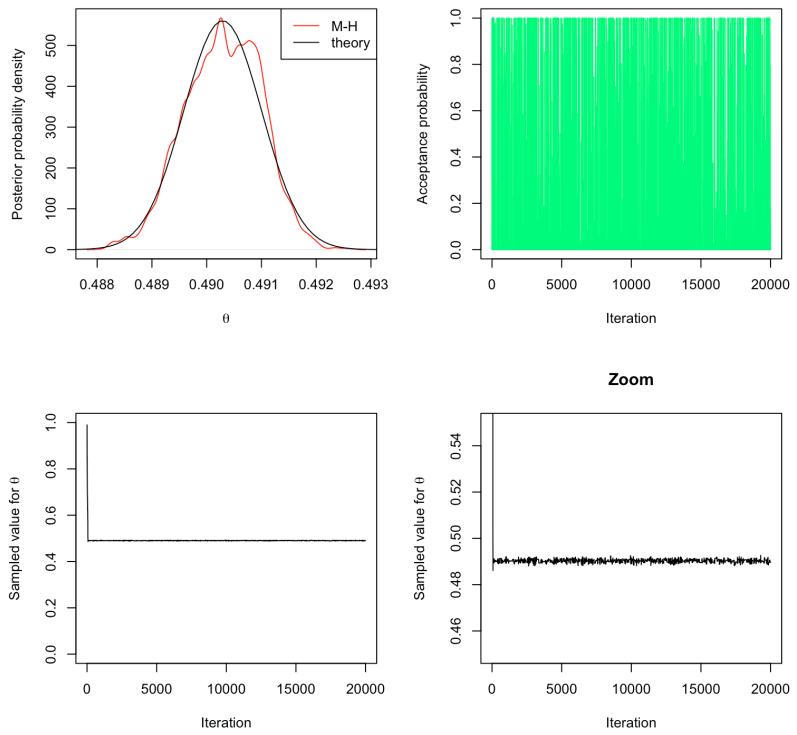
    # save current value
    x_save[t] <- x
  }

  return(list("theta" = x_save, "alpha" = alpha))
}

sampleMH <- myMH_betaprior_randomwalk(20000, post_num = post_num_beta_hist)

par(mfrow=c(2,2))
plot(density(sampleMH$theta[-c(1:1000)]), col = "red",
     ylab = "Posterior probability density",
     xlab=expression(theta), main = "")
curve(dbeta(x, 3 + 241945, 3 + 251527), from = 0, to = 1, n = 10000, add = TRUE)
legend("topright", c("M-H", "theory"), col = c("red", "black"), lty = 1)
plot(sampleMH$alpha, type = "h", xlab = "Iteration",
     ylab = "Acceptance probability", ylim = c(0,1), col = "springgreen")
plot(sampleMH$theta, type="l", xlab = "Iteration",
     ylab = expression(paste("Sampled value for ", theta)), ylim = c(0,1))
plot(sampleMH$theta, type="l", xlab = "Iteration", main = "Zoom",
     ylab = expression(paste("Sampled value for ", theta)), ylim = c(0.45, 0.55))

```



# Exercise 5

The [BUGS project](#) (*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](#) (*Just another Gibbs Sampler*) is a different and newer implementation that also relies on the `BUGS` language. Finally, the [STAN](#) software must also be mentioned, recently developed et the Columbia University, ressemble `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](#).

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 in `R`.

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

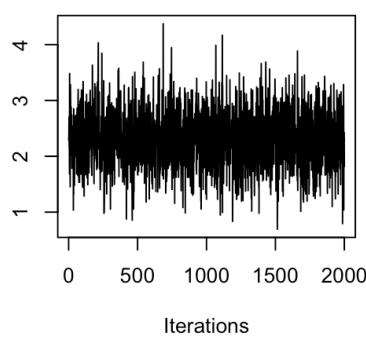
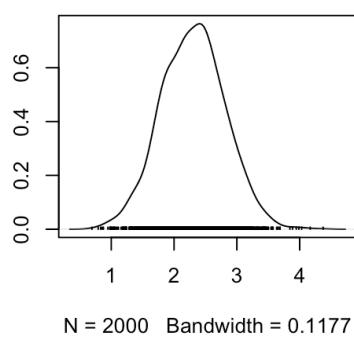
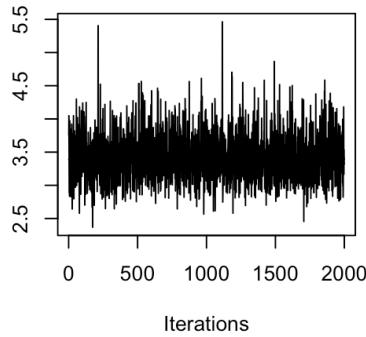
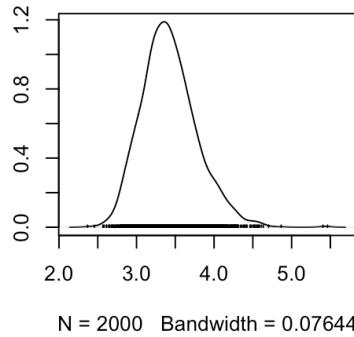
```
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 50
##   Unobserved stochastic nodes: 2
##   Total graph size: 58
##
## Initializing model
```

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.

```
plot(res)
```

**Trace of mu****Density of mu****Trace of sigma****Density of sigma**

```
res_sum <- summary(res)
res_sum
```

```
## 
## Iterations = 1:2000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 2000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD Naive SE Time-series SE
## mu     2.305  0.508  0.011358      0.011358
## sigma  3.425  0.358  0.008005      0.008005
##
## 2. Quantiles for each variable:
##
##        2.5%   25%   50%   75% 97.5%
## mu     1.317  1.945  2.309  2.639  3.30
## sigma  2.820  3.186  3.392  3.628  4.22
```

```
res_sum$statistics["mu", "Mean"]
```

```
## [1] 2.304586
```

```
res_sum$statistics["sigma", "Mean"]
```

```
## [1] 3.424725
```

```
res_sum$quantiles["mu", "50%"]
```

```
## [1] 2.309279
```

```
res_sum$quantiles["sigma", "50%"]
```

```
## [1] 3.392066
```

```
res_sum$quantiles["mu", c(1,5)]
```

```
##      2.5%    97.5%
## 1.316796 3.300451
```

```
res_sum$quantiles["sigma", c(1,5)]
```

```
##      2.5%    97.5%
## 2.819924 4.219521
```

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

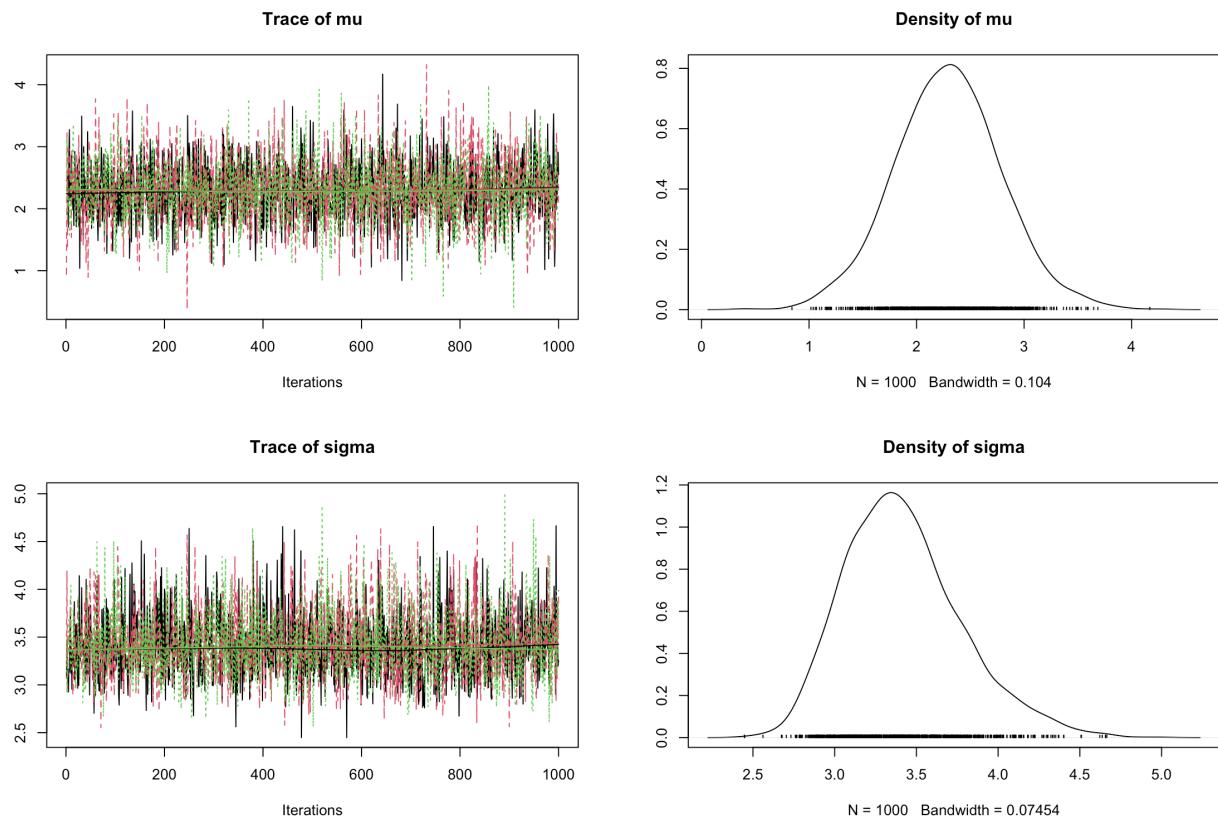
```
library(coda)
```

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  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).

```
myjags2 <- jags.model("normalBUGSmodel.txt", data = list('obs' = obs, 'N' = N),
                      n.chains = 3, inits = list(list("mu" = 0, "tau" = 1),
                      list("mu" = -10, "tau" = 1/100),
                      list("mu" = 100, "tau" = 1/10)))
```

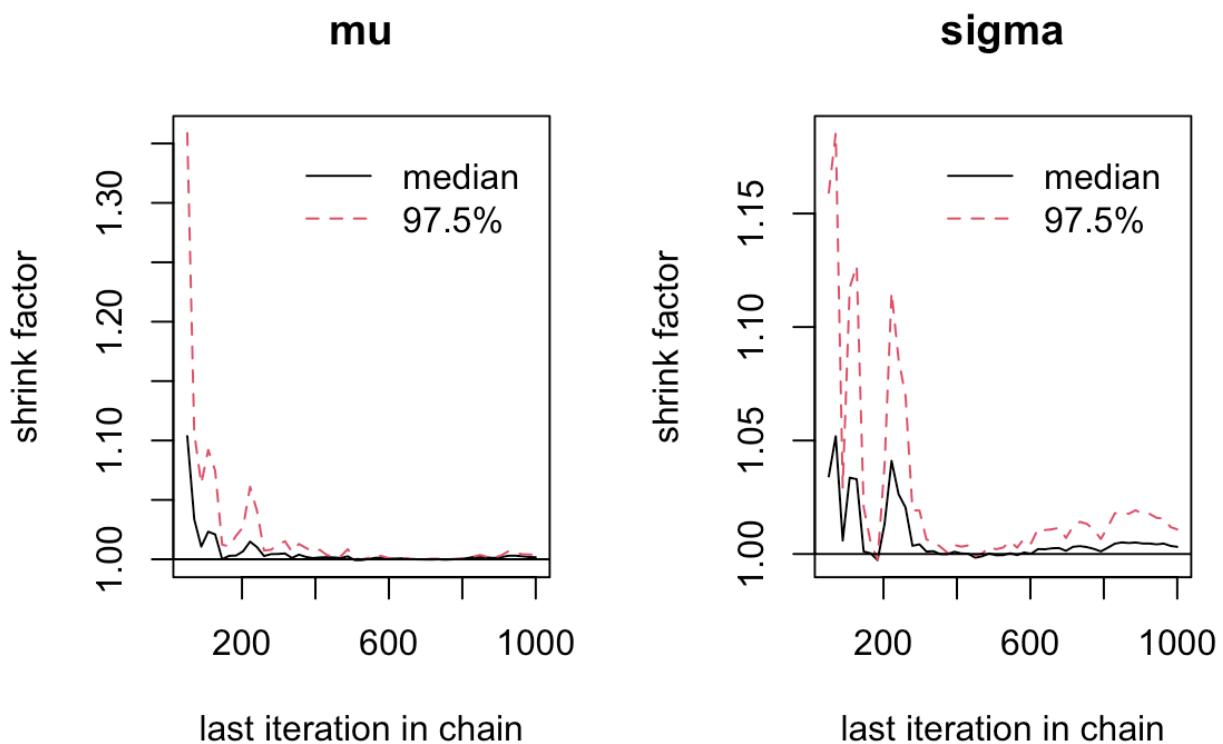
```
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 50
##   Unobserved stochastic nodes: 2
##   Total graph size: 58
##
## Initializing model
```

```
res2 <- coda.samples(model = myjags2, variable.names = c('mu', 'sigma'),
                      n.iter = 1000)
plot(res2)
```



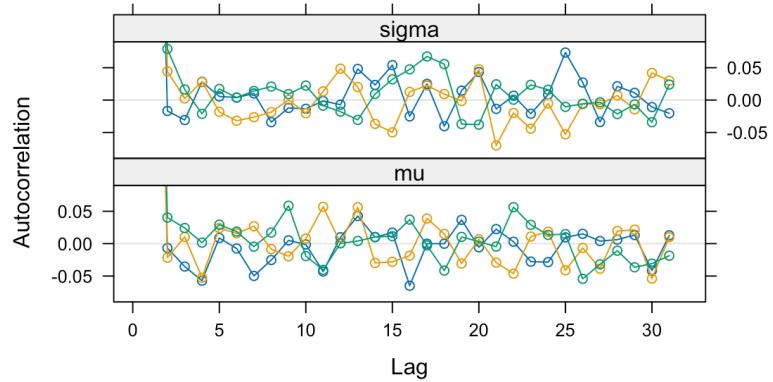
8. With the function `gelman.plot()`, plot the Gelman-Rubin statistic.

```
gelman.plot(res2)
```

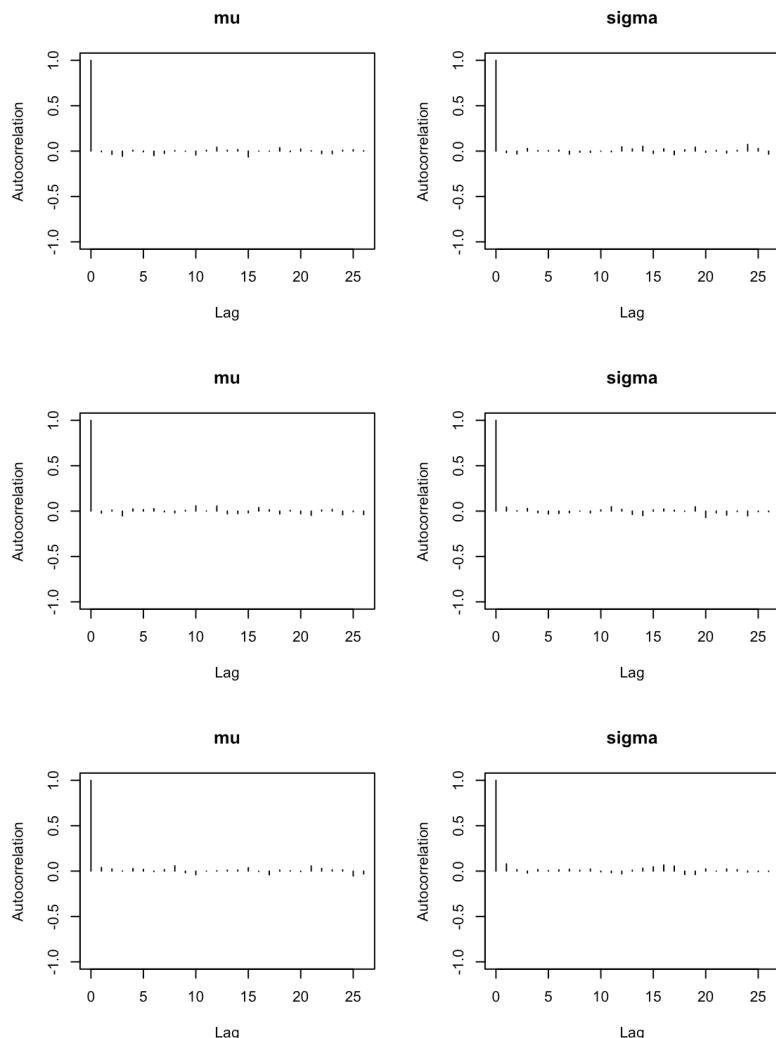


9. With the functions `autocorr.plot()` and `acfplot()` evaluate the autocorrelation of the studied parameters.

```
acfplot(res2)
```

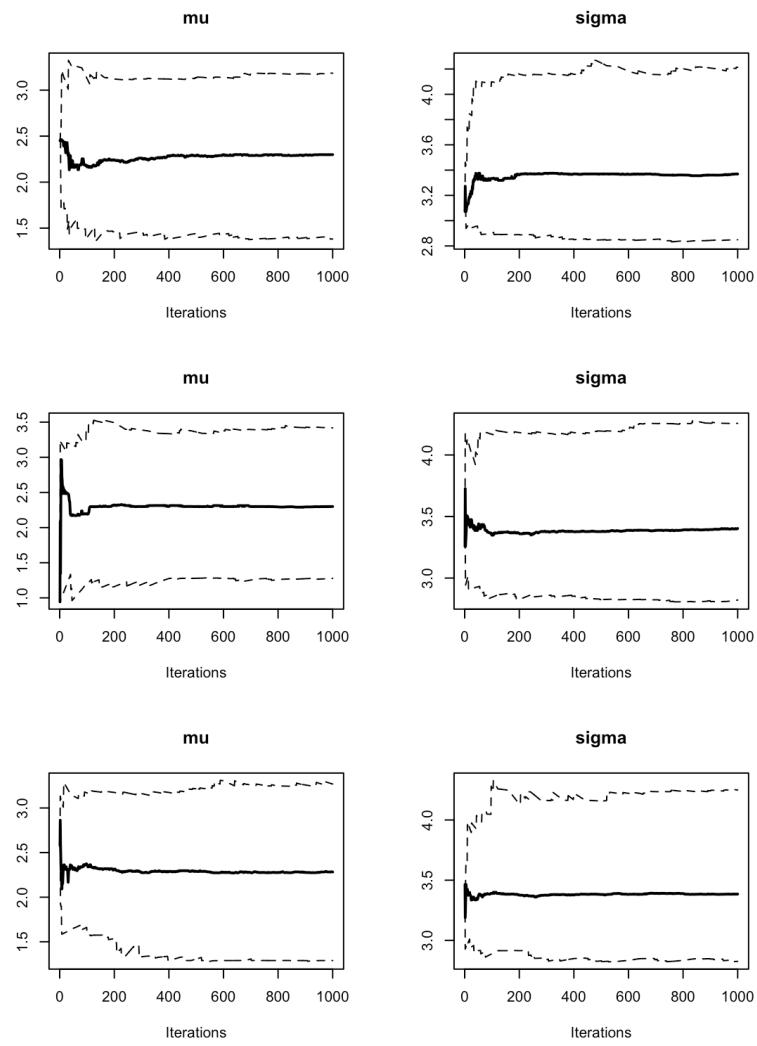


```
par(mfrow=c(3,2))
autocorr.plot(res2, ask = FALSE, auto.layout = FALSE)
```



10. With the function `cumuplot()` evaluate the running quantiles of the studied parameters. How can you interpret them ?

```
par(mfrow=c(3,2))
cumuplot(res2, ask=FALSE, auto.layout = FALSE)
```



Each row of the above graph is a different chain. The cumulative quantiles are indeed stable after the first few iterations in all chains.

- With the function `hdi()` from the package `HDIInterval`, provide highest density posterior credibility intervals at 95%, and compare them to those obtained with the 2.5% and 97.5% quantiles.

```
hdCI <- HDInterval::hdi(res2)
hdCI
```

```
##           mu      sigma
## lower  1.276974 2.790113
## upper  3.267396 4.164756
## attr(,"credMass")
## [1] 0.95
```

```
symCI <- summary(res2)$quantiles[,c(1,5)]
symCI
```

```
##           2.5%     97.5%
## mu      1.296159 3.305287
## sigma   2.825876 4.246684
```

```
symCI[, 2] - symCI[, 1]
```

```
##           mu      sigma
## 2.009128 1.420808
```

```
hdCI[2, ] - hdCI[1, ]
```

```
##      mu    sigma
## 1.990423 1.374643
```

## 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.

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

Observed data from the *EOLIA* trial

1. Write the Bayesian model used by Goligher *et al.* (2019).

### Correction:

#### I) Question of interest:

*Is the Relative Risk of death under ECMO compared to the conventional mechanical treatment less than one ?*

#### II) Sampling model:

Let  $Z_{control}$  be the number of death in the control group, and  $Z_{ecmo}$  the number of death in the ECMO group

$$Z_{control} \sim Binomial(p_c, 125)$$

$$Z_{ecmo} \sim Binomial(RR \times p_c, 124)$$

#### III) Priors:

$$p_c \sim U_{[0,1]}$$

$$\log(RR) \sim U_{[-17,17]}$$

**NB:**  $U_{[-17,17]}$  has approximately a standard deviation of 10 (cf Table 1 and Figure 2.A – orange lines – of Goligher *et al.*, 2019<sup>3</sup>)

**NB:** One can also define a sampling model at the individual level:

Let  $Y_{control_i}$  be a binary variable indicating whether the patient  $i$  from the control group died before 60 days, and  $Y_{ecmo_i}$  a similar variable for patient from the ecmo group.

$$Y_{control_i} \stackrel{iid}{\sim} Bernoulli(p_c)$$

$$Y_{ecmo_i} \stackrel{iid}{\sim} Bernoulli(RR \times p_c)$$

2. Write the corresponding BUGS model, and save it into a .txt file (for instance called `goligherBUGSmodel.txt` )

## Correction:

- As we have seen above, there are two equivalent ways of defining the sampling model:
- either at the population level with a **Binomial** likelihood,
  - or at the individual level with a **Bernoulli** likelihood

```
# Population model
model{

  # Sampling model
  zcontrol~dbin(pc, ncontrol)
  zecmo~dbin(RR*pc, necmo)

  # Prior
  logRR~dunif(-17, 17) # SD is approximately 10
  pc~dunif(0,1) #probability of death in the control group

  # Re-parameterizations
  RR <- exp(logRR)
  ARR <- pc - RR*pc
}
```

```
# Individual model
model{

  # Sampling model
  for (i in 1:ncontrol){
    ycontrol[i]~dbern(pc)
  }
  for (i in 1:necmo){
    yecmo[i]~dbern(RR*pc)
  }

  # Prior
  logRR~dunif(-17, 17) # SD is approximately 10
  pc~dunif(0,1) #probability of death in the control group

  # Re-parameterizations
  RR <- exp(logRR)
  ARR <- pc - RR*pc
}
```

3. First create two binary data vectors `ycontrol` and `yecmo` (or `ycontrol` and `yecmo` that are either `1` or `0`, using the `rep()` R function if you prefer the individual model), 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.)

```
#Individual data
ncontrol <- 125
zcontrol <- 57
ycontrol <- c(rep(1, zcontrol), rep(0, ncontrol-zcontrol))

necmo <- 124
zecmo <- 44
yecmo <- c(rep(1, zecmo), rep(0, necmo-zecmo))

#Population data model
#sampling
```

```

library(rjags)
goligher_jags_pop <- jags.model(file = "goligherBUGSmodel_pop.txt",
                                  data = list("zcontrol" = zcontrol,
                                             "ncontrol" = ncontrol,
                                             "zecmo" = zecmo,
                                             "necmo" = necmo
                                             ),
                                  n.chains = 3)

```

```

## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 2
##   Unobserved stochastic nodes: 2
##   Total graph size: 13
##
## Initializing model

```

```

res_goligher_pop <- coda.samples(model = goligher_jags_pop,
                                   variable.names = c('pc', 'RR', 'ARR'),
                                   n.iter = 40000)

#post-processing
res_goligher_burnt_pop <- window(res_goligher_pop, start=21001) # removes burn-in for Markov chain
#NB: by default coda.samples() performs 1000 first burnin iterations that are automatically removed

```

#Individual data model

```

goligher_jags_indiv <- jags.model(file = "goligherBUGSmodel_indiv.txt",
                                      data = list("ycontrol" = ycontrol,
                                                 "ncontrol" = ncontrol,
                                                 "yecmo" = yecmo,
                                                 "necmo" = necmo
                                                 ),
                                      n.chains = 3)

```

```

## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 249
##   Unobserved stochastic nodes: 2
##   Total graph size: 260
##
## Initializing model

```

```

res_goligher_indiv <- coda.samples(model = goligher_jags_indiv,
                                      variable.names = c('pc', 'RR', 'ARR'),
                                      n.iter = 40000)

#postprocessing
res_goligher_burnt_indiv <- window(res_goligher_indiv, start=21001) # removes burn-in for Markov chain

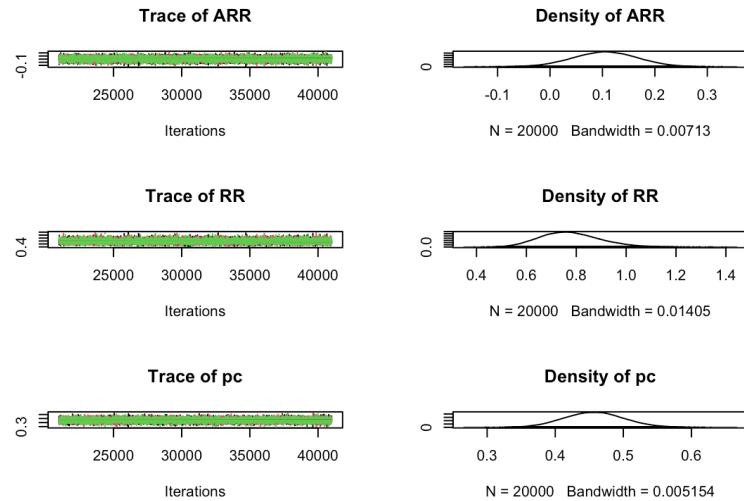
```

4. Check the convergence, and then comment the estimate results (**ProTip:** look at the effective sample size with the `effectiveSize()`  function).

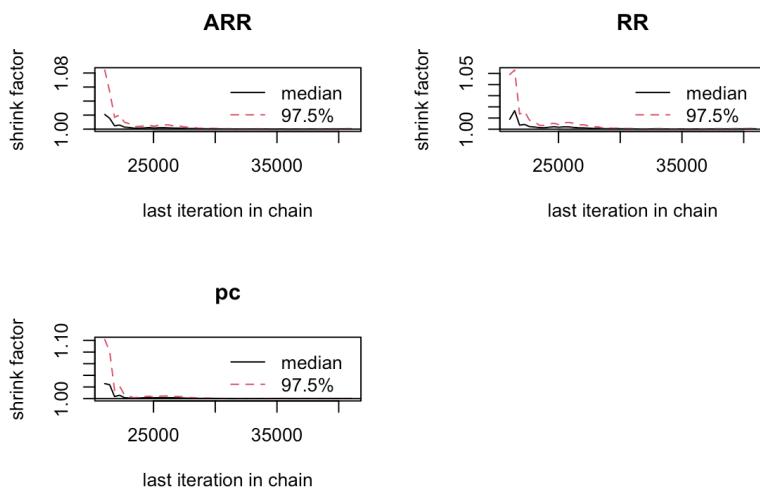
```
effectiveSize(res_goligher_burnt_pop)
```

```
##      ARR      RR      pc
## 15178.56 16481.17 16729.32
```

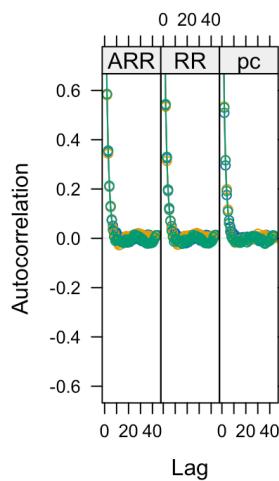
```
plot(res_goligher_burnt_pop)
```



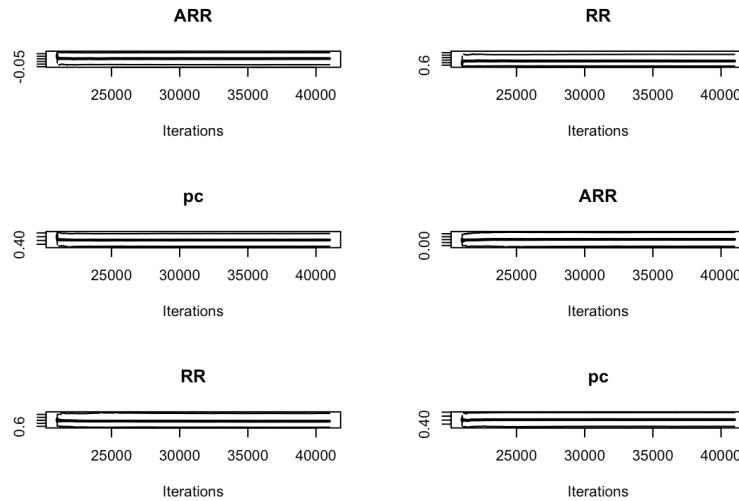
```
gelman.plot(res_goligher_burnt_pop)
```



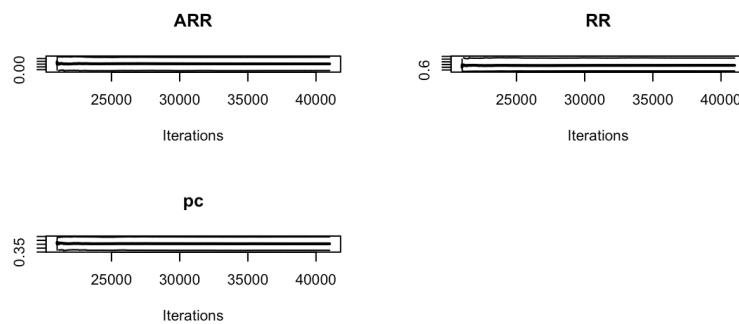
```
acfplot(res_goligher_burnt_pop)
```



```
par(mfrow=c(3, 2))
cumuplot(res_goligher_burnt_pop, ask=FALSE, auto.layout = FALSE)
```



```
par(mfrow=c(1, 1))
```



```
summary(res_goligher_burnt_pop)
```

```
##
## Iterations = 21001:41000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD  Naive SE Time-series SE
## ARR  0.1051  0.06073  0.0002479      0.0004930
## RR   0.7772  0.12068  0.0004927      0.0009399
## pc   0.4570  0.04390  0.0001792      0.0003394
##
## 2. Quantiles for each variable:
##
##        2.5%     25%     50%     75%   97.5%
## ARR -0.0143  0.06423  0.1054  0.1464  0.2230
```

```
## RR  0.5650 0.69219 0.7689 0.8526 1.0358  
## pc  0.3722 0.42723 0.4567 0.4862 0.5445
```

```
summary(res_goligher_burnt_indiv)
```

```
##  
## Iterations = 21001:41000  
## Thinning interval = 1  
## Number of chains = 3  
## Sample size per chain = 20000  
##  
## 1. Empirical mean and standard deviation for each variable,  
##    plus standard error of the mean:  
##  
##      Mean        SD  Naive SE Time-series SE  
## ARR 0.1050 0.06101 0.0002491      0.0004930  
## RR  0.7774 0.12115 0.0004946      0.0009486  
## pc   0.4567 0.04394 0.0001794      0.0003417  
##  
## 2. Quantiles for each variable:  
##  
##      2.5%     25%     50%     75%   97.5%  
## ARR -0.01505 0.06366 0.1051 0.1463 0.2245  
## RR   0.56303 0.69235 0.7698 0.8542 1.0375  
## pc   0.37067 0.42689 0.4565 0.4862 0.5437
```

```
# shortest 95% Credibility interval:  
HDInterval::hdi(res_goligher_burnt_pop)
```

```
##          ARR        RR        pc  
## lower -0.01506739 0.5519202 0.3703146  
## upper  0.22192604 1.0181222 0.5423551  
## attr(,"credMass")  
## [1] 0.95
```

```
# posterior porbability of RR <1: (answering our question !)  
head(res_goligher_burnt_pop[[1]]) # first chain
```

```
## Markov Chain Monte Carlo (MCMC) output:  
## Start = 21001  
## End = 21007  
## Thinning interval = 1  
##          ARR        RR        pc  
## [1,] 0.1545363 0.6775445 0.4792487  
## [2,] 0.1380911 0.7117841 0.4791237  
## [3,] 0.2208782 0.5657908 0.5086907  
## [4,] 0.1173809 0.7851584 0.5463600  
## [5,] 0.1132941 0.7831471 0.5224469  
## [6,] 0.2084089 0.5967045 0.5167648  
## [7,] 0.2130601 0.6043759 0.5385418
```

```
head(res_goligher_burnt_pop[[2]]) # second chain
```

```
## Markov Chain Monte Carlo (MCMC) output:  
## Start = 21001  
## End = 21007  
## Thinning interval = 1
```

```
##          ARR         RR         pc
## [1,] 0.09942179 0.7770101 0.4458579
## [2,] 0.10214823 0.7791052 0.4624292
## [3,] 0.12604212 0.7301342 0.4670548
## [4,] 0.07917832 0.8303366 0.4666788
## [5,] 0.07890730 0.8336404 0.4743177
## [6,] 0.06989778 0.8421022 0.4426773
## [7,] 0.05316942 0.8611119 0.3828221
```

```
mean(c(sapply(res_goligher_burnt_pop, "[", , "RR"))<1)
```

```
## [1] 0.95775
```

5. 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.

```
1/(0.15^2)
```

```
## [1] 44.44444
```

```
logRR~dnorm(log(0.78), 45)
```

## Exercise 7

In 2014, Crins *et al.*<sup>4</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.

0. Load the `R` package `bayesmeta`<sup>5</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/>.

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

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

2. Directly in the `R` console 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 `vi`)

```

library("metafor")
crins.es <- escalc(measure="OR",
                     ai=exp.AR.events, n1i=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>6</sup>.

3. Perform a random-effect meta-analysis of those data using the `bayesmeta()` function from the [R](#) 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.

```

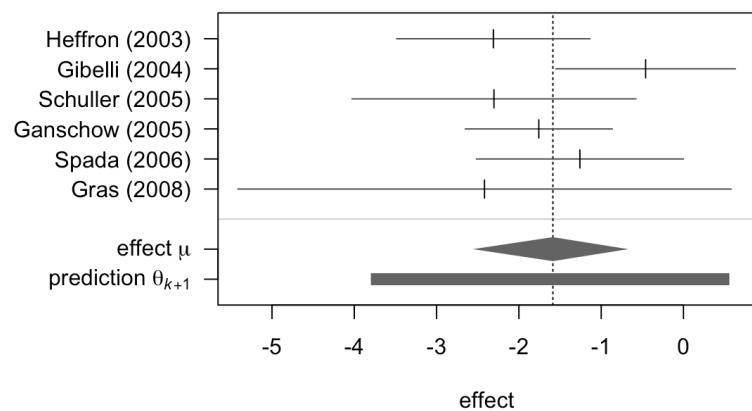
interval.type = "central")
summary(res_crins_bayesmeta)

## 'bayesmeta' object.
## data (6 estimates):
##          y      sigma
## Heffron (2003) -2.3097026 0.5994763
## Gibelli (2004) -0.4595323 0.5563956
## Schuller (2005) -2.3025851 0.8803408
## Ganschow (2005) -1.7578579 0.4558685
## Spada (2006)    -1.2584610 0.6419962
## Gras (2008)     -2.4178959 1.5288107
##
## tau prior (proper):
## function (t)
## {
##   dunif(t, max = 4)
## }
## <bytecode: 0x11ef66860>
##
## mu prior (proper):
## normal(mean=0, sd=4)
##
## ML and MAP estimates:
##          tau      mu
## ML joint 0.3258895 -1.578317
## ML marginal 0.4644136 -1.587347
## MAP joint 0.3244300 -1.569497
## MAP marginal 0.4644205 -1.576092
##
## marginal posterior summary:
##          tau      mu      theta
## mode 0.46442045 -1.5760916 -1.5656380
## median 0.61810119 -1.5866193 -1.5806176
## mean 0.73768542 -1.5935568 -1.5935568
## sd 0.56879971 0.4698241 1.0448965
## 95% lower 0.03724555 -2.5605641 -3.7989794
## 95% upper 2.22766946 -0.6704235 0.5580817
##
## (quoted intervals are central, equal-tailed credible intervals.)
##
## Bayes factors:
##          tau=0      mu=0
## actual 2.6800815 0.094852729
## minimum 0.7442665 0.008342187
##
## relative heterogeneity I^2 (posterior median): 0.4718317

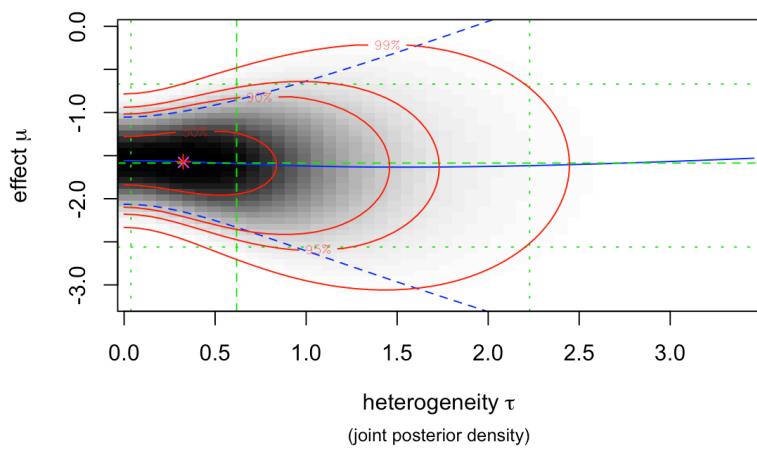
```

```
plot(res_crins_bayesmeta)
```

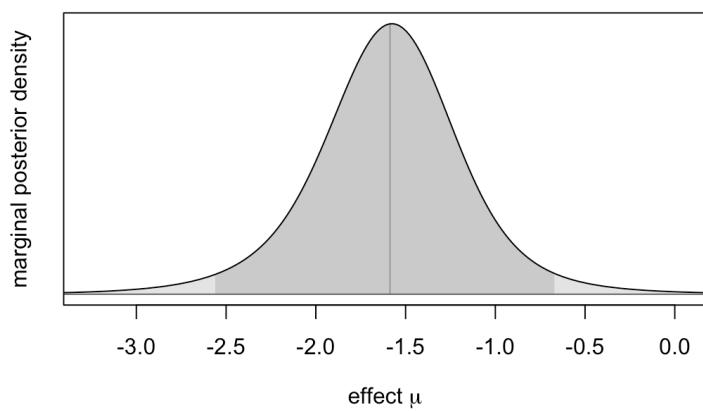
### res\_crins\_bayesmeta

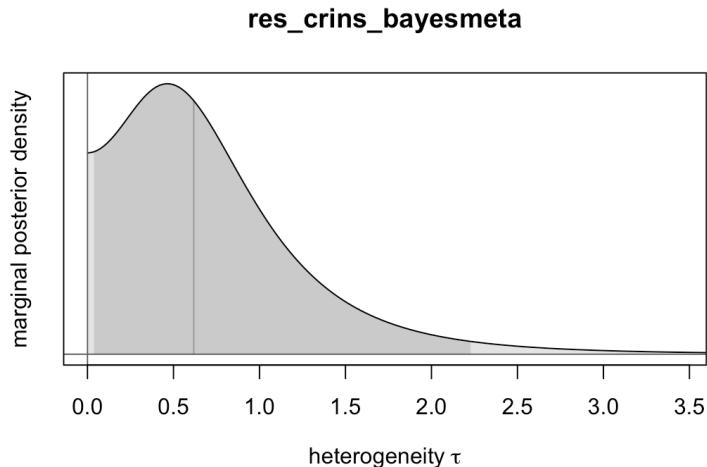


### res\_crins\_bayesmeta



### res\_crins\_bayesmeta





4. Write the corresponding random-effects Bayesian meta-analysis model (using math, not – yet).

## Correction:

### I) Question of interest:

*Is the treatment (IL2RA) odds ratio for Acute Rejection events inferior to 1 ?*

### II) Sampling model:

Let  $y_i$  be the log-odds-ratio reported by the study  $i$  and  $\sigma_i^2$  its sampling variance

$$y_i \stackrel{iid}{\sim} \mathcal{N}(\theta_i, \sigma_i^2)$$

$$\theta_i \stackrel{iid}{\sim} \mathcal{N}(\mu, \tau^2)$$

### III) Priors:

$$\theta_i \stackrel{iid}{\sim} \mathcal{N}(\mu, \tau^2)$$

$$\mu \sim \mathcal{N}(0, 4^2)$$

$$\tau \sim U_{[0,4]}$$

5. Use `rjags` to estimate the same model, saving the BUGS model in a `.txt` file (called `crinsBUGSmodel.txt` for instance).

```
# Random-effects model for Crins et al. 2014 Acute Rejection meta-analysis
model{

  # Sampling model/likelihood
  for (i in 1:N){
    logOR[i]~dnorm(theta[i], precision.logOR[i])
  }

  # Priors
  for (i in 1:N){
    theta[i]~dnorm(mu, precision.tau)
  }
  mu~dnorm(0, 0.0625) # 1/16 = 0.0625
  tau~dunif(0, 4)

  # Re-parameterization
  for(i in 1:N){
```

```

precision.logOR[i] <- pow(sigma[i], -2)
}
precision.tau <- pow(tau, -2)
OR <- exp(mu)
}

```

```

# Sampling
library(rjags)
crins_jags_res <- jags.model(file = "crinsBUGSmodel.txt",
                               data = list("logOR" = crins.es$yi,
                                          "sigma" = sqrt(crins.es$vi),
                                          "N" = length(crins.es$yi)
                                         ),
                               n.chains = 3)

```

```

## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 6
##   Unobserved stochastic nodes: 8
##   Total graph size: 34
##
## Initializing model

```

```

res_crins_jags_res <- coda.samples(model = crins_jags_res,
                                      variable.names = c("mu", "tau", "OR"),
                                      n.iter = 20000)

# Postprocessing
res_crins_jags_res <- window(res_crins_jags_res, start=5001) # remove burn-in for Markov chain convergence
summary(res_crins_jags_res)

```

```

##
## Iterations = 5001:21000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 16000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean      SD  Naive SE Time-series SE
## OR    0.2284 0.1625 0.0007418      0.001303
## mu   -1.5941 0.4703 0.0021464      0.004107
## tau   0.7291 0.5714 0.0026081      0.011199
##
## 2. Quantiles for each variable:
##
##           2.5%     25%     50%     75%   97.5%
## OR    0.07726 0.1570  0.2043  0.2658  0.5121
## mu   -2.56059 -1.8514 -1.5883 -1.3252 -0.6692
## tau   0.03180 0.3264  0.6090  0.9808  2.2206

```

```
HDIInterval::hdi(res_crins_jags_res)
```

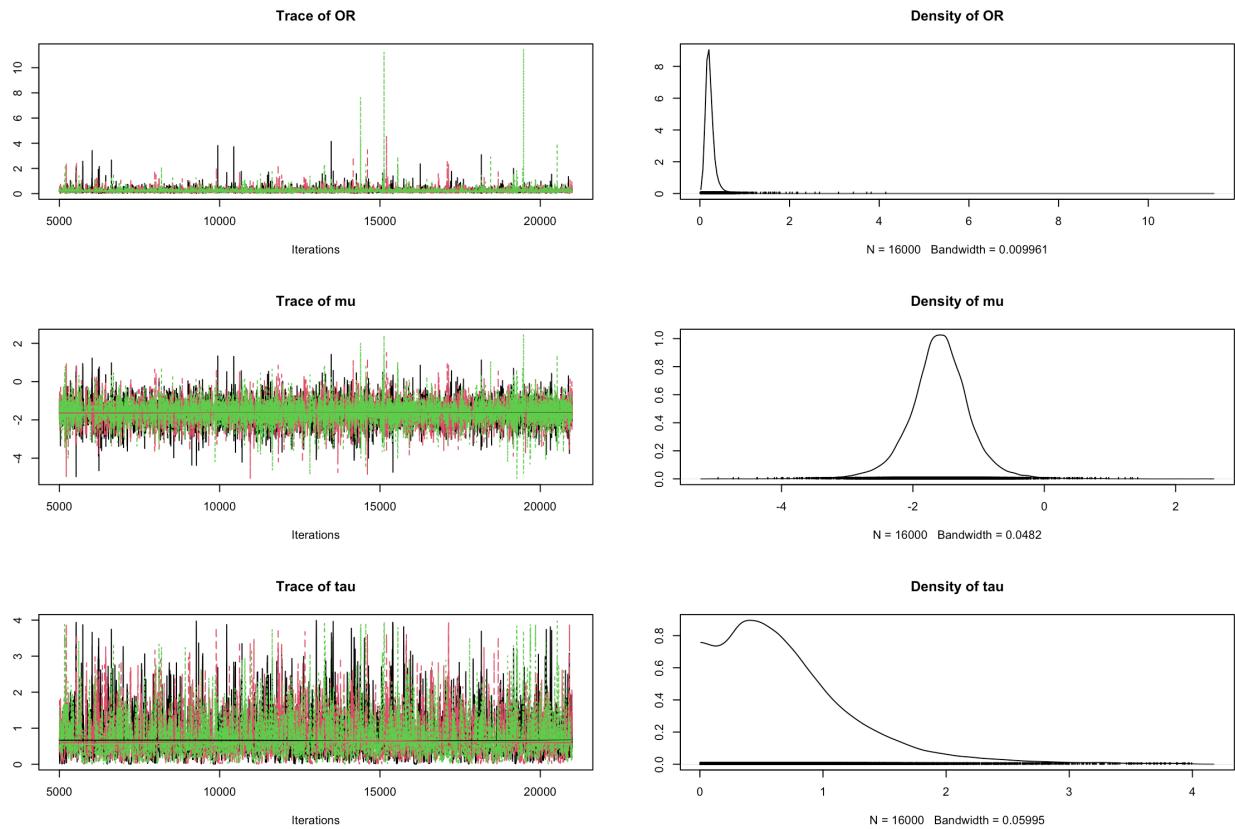
```

##          OR        mu        tau
## lower 0.04989464 -2.5635787 0.0005215018

```

```
## upper 0.43671281 -0.6728889 1.8120063364  
## attr(,"credMass")  
## [1] 0.95
```

```
plot(res_crins_jags_res)
```



# Exercise 8

We will now analyze data from the therapeutic clinical trial *ALBI-ANRS 070* which compared the efficacy and tolerance of 3 anti-retroviral strategies among HIV-1 positive patients naive of any anti-retroviral treatment [7](#).

1. Load the data from the `albianrs_data.csv` available [here](#) (ProTip: set `stringsAsFactors = TRUE` in `read.delim()`)

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

```
##      PatientID      ViralLoad        CD4       CD4t      CD4sup500
## ID112   : 7    Min.   :1.699   Min.   :149.0   Min.   :3.494   No  :558
## ID130   : 7  1st Qu.:2.042   1st Qu.:377.0   1st Qu.:4.406   Yes:392
## ID139   : 7 Median  :2.712   Median :465.0   Median :4.644
## ID15    : 7 Mean    :2.907   Mean   :491.8   Mean   :4.664
## ID156   : 7 3rd Qu.:3.692   3rd Qu.:585.0   3rd Qu.:4.918
## ID163   : 7 Max.    :5.604   Max.   :1318.0  Max.   :6.025
## (Other):908
##      Treatment      Day       Visit
## AZT+3TC:315  Min.   : 0.00   Min.   :0.000
## d4t+ddl:322  1st Qu.:29.00   1st Qu.:1.000
## switch :313  Median :84.00   Median :3.000
##                  Mean   :86.19   Mean   :2.946
##                  3rd Qu.:140.00  3rd Qu.:5.000
##                  Max.   :256.00  Max.   :6.000
##
```

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 = CD41/4`)
- `CD4sup500` : binary variable indicating whether 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.

Variable	N = 950 <sup>1</sup>
ViralLoad	2.71 (2.04, 3.70)
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%)

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

Variable	N = 950 <sup>1</sup>
Day	84 (29, 140)
Visit	
0	146 (15%)
1	140 (15%)
2	136 (14%)
3	130 (14%)
4	128 (13%)
5	135 (14%)
6	135 (14%)

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

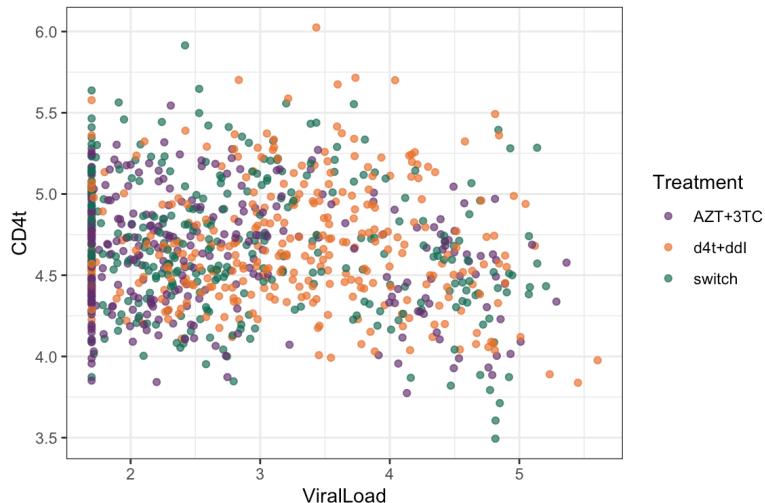
### Summary of the Albi-ANRS-070 trial data

**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`.

## Correction:

### I) Question of interest:

How does the viral load linearly explains the (transformed) CD4 T-cell rate ?

### II) Sampling model:

Let  $CD4t_i$  be the  $i^{\text{th}}$  (transformed) CD4 T-cell rate  $i$  and  $VL_i^2$  the corresponding observed viral load:

$$CD4t_i \stackrel{iid}{\sim} \mathcal{N}(\beta_0 + \beta_1 VL_i, \sigma^2)$$

**III) Priors:**

$$\begin{aligned}\beta_0 &\sim \mathcal{N}(0, 100^2) \\ \beta_1 &\sim \mathcal{N}(0, 100^2) \\ \sigma^2 &\sim \text{InvGamma}(0.001, 0.001)\end{aligned}$$

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

```
# Fixed effect linear regression Albi-ANRS
model{

  # Likelihood
  for (i in 1:N){
    CD4t[i]~dnorm(mu[i], tau)
    mu[i] <- beta0 + beta1*VL[i]
  }

  # Priors
  beta0~dnorm(0,0.0001)
  beta1~dnorm(0,0.0001) # proper but very flat (vague: weakly informative)
  tau~dgamma(0.0001,0.0001) # proper but very flat (vague: weakly informative)

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

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

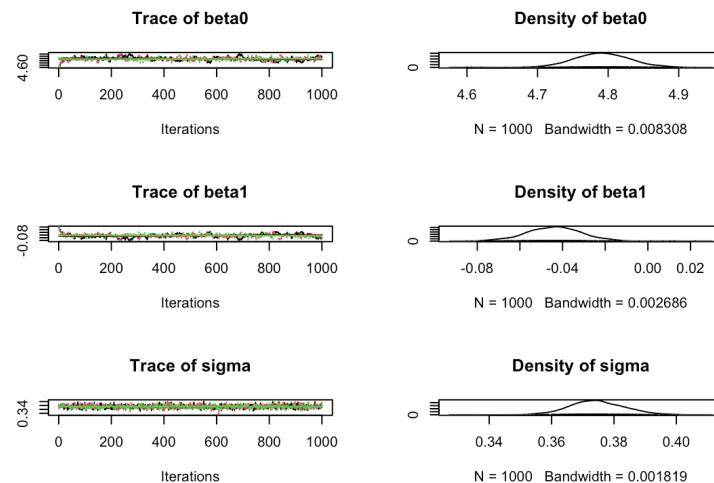
```
library(rjags)
albi_fixed_jags <- jags.model("albiBUGSmodel_fixed.txt",
                               data = list('CD4t' = albi$CD4t,
                                          'N' = nrow(albi),
                                          'VL' = albi$ViralLoad),
                               n.chains = 3)
```

```
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 950
##   Unobserved stochastic nodes: 3
##   Total graph size: 3195
##
## Initializing model
```

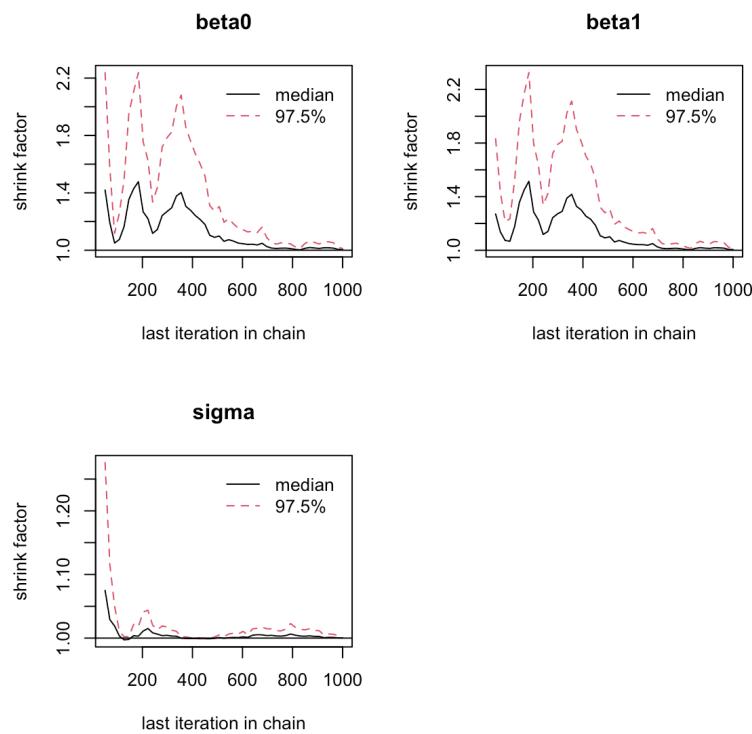
```
res_albi_fixed <- coda.samples(model = albi_fixed_jags,
                                 variable.names = c('beta0', 'beta1', 'sigma'),
                                 n.iter = 1000)
```

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

```
library(coda)
plot(res_albi_fixed)
```

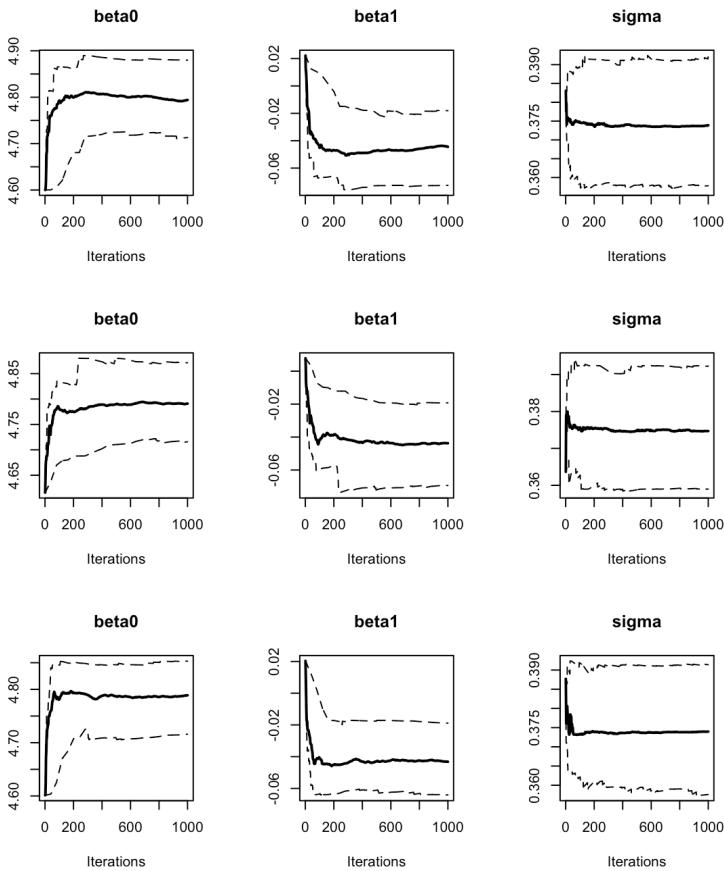


```
gelman.plot(res_albi_fixed)
```



```
acfplot(res_albi_fixed)
```

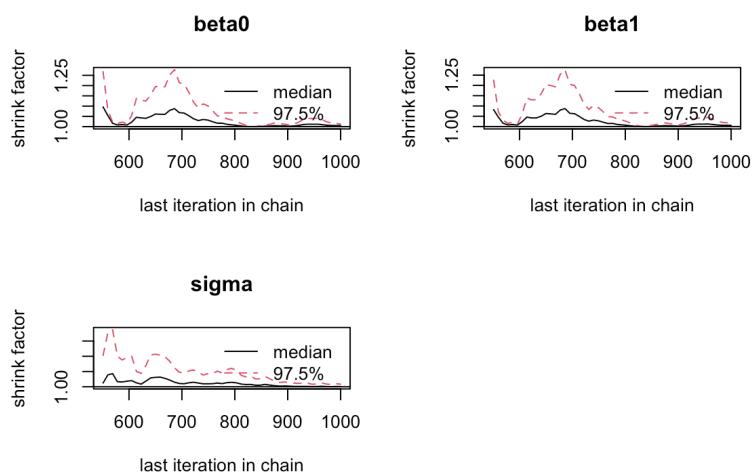
```
par(mfrow=c(3, 3))
cumuplot(res_albi_fixed, ask = FALSE, auto.layout = FALSE)
```



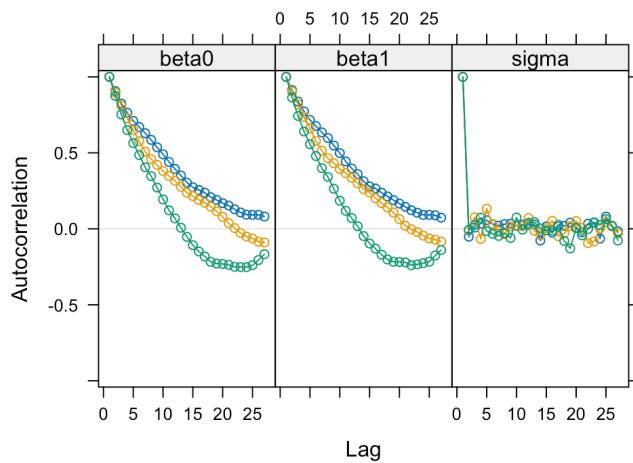
f. Using the `window` function, remove the 500 first iterations as burn-in to reach Markov Chain convergence to its stationary distribution (the posterior). Is it sufficient to solve convergence issues ?

```
res_albi_fixed2 <- window(res_albi_fixed, start=501)

gelman.plot(res_albi_fixed2)
```



```
acfplot(res_albi_fixed2)
```



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

```
?effectiveSize
effectiveSize(res_albi_fixed2)
```

```
##      beta0      beta1      sigma
##  84.78146  84.10904 1384.65863
```

```
albi_fixed_jags <- jags.model("albiBUGSmodel_fixed.txt",
                                data = list('CD4t' = albi$CD4t,
                                           'N' = nrow(albi),
                                           'VL' = albi$ViralLoad),
                                n.chains = 3)
```

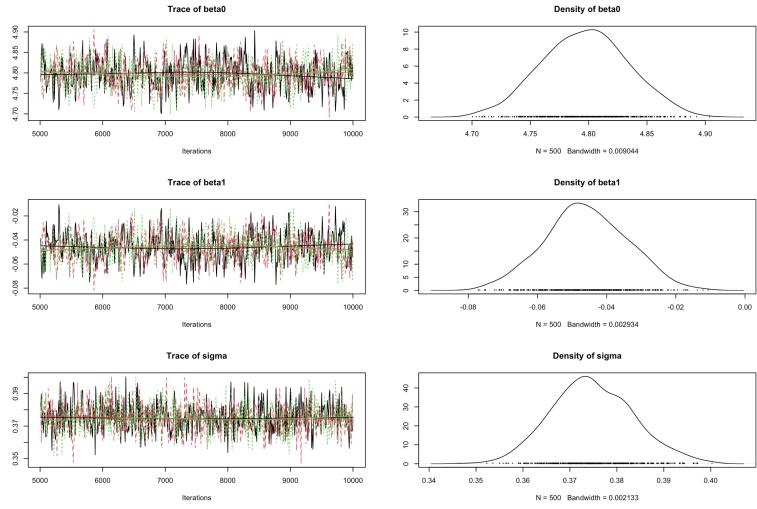
```
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 950
##   Unobserved stochastic nodes: 3
##   Total graph size: 3195
##
## Initializing model
```

```
res_albi_fixed_thin <- coda.samples(model = albi_fixed_jags,
                                       variable.names = c('beta0', 'beta1', 'sigma'),
                                       n.iter = 10000, thin = 10)
res_albi_fixed_thin <- window(res_albi_fixed_thin, start=5001)

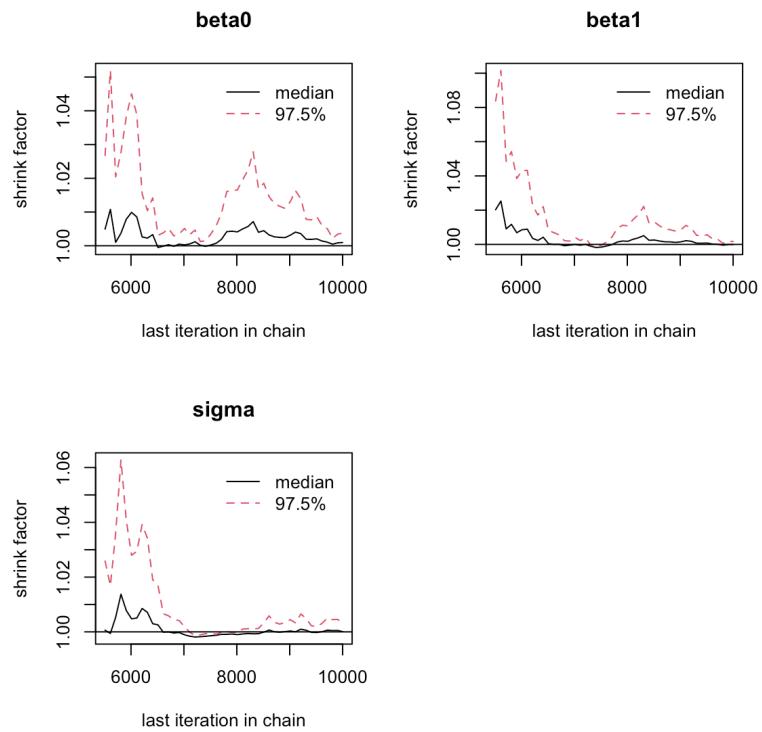
effectiveSize(res_albi_fixed_thin)
```

```
##      beta0      beta1      sigma
##  700.7145  680.0478 1500.00000
```

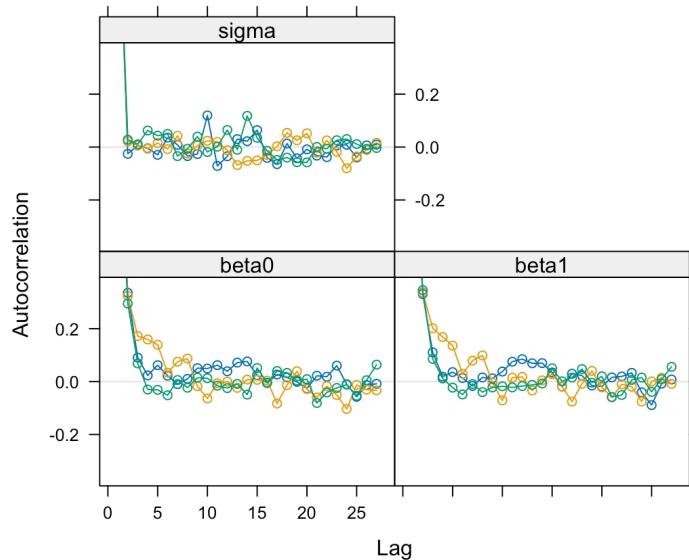
```
plot(res_albi_fixed_thin)
```



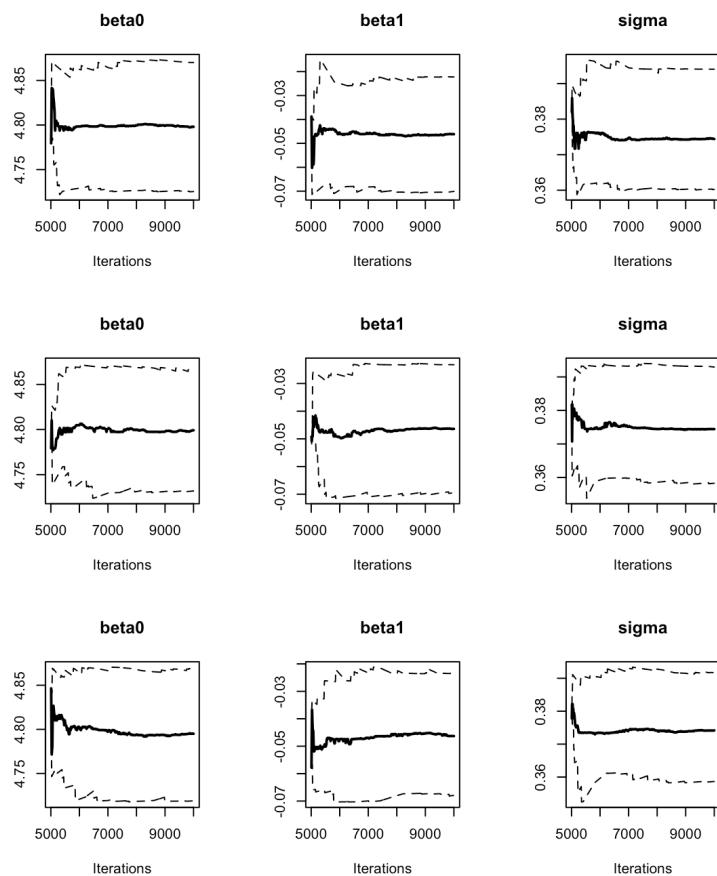
```
gelman.plot(res_albi_fixed_thin)
```



```
acfplot(res_albi_fixed_thin)
```



```
par(mfrow=c(3, 3))
cumuplot(res_albi_fixed_thin, ask=FALSE, auto.layout = FALSE)
```



```
summary(res_albi_fixed_thin)
```

```
## 
## Iterations = 5010:10000
## Thinning interval = 10
## Number of chains = 3
```

```

## Sample size per chain = 500
##
## 1. Empirical mean and standard deviation for each variable,
##     plus standard error of the mean:
##
##           Mean      SD  Naive SE Time-series SE
## beta0  4.7972 0.036837 0.0009511      0.0014051
## beta1 -0.0459 0.011949 0.0003085      0.0004615
## sigma  0.3748 0.008687 0.0002243      0.0002243
##
## 2. Quantiles for each variable:
##
##       2.5%    25%    50%    75%   97.5%
## beta0  4.72589 4.77192 4.79753 4.82202 4.86981
## beta1 -0.06956 -0.05397 -0.04631 -0.03763 -0.02282
## sigma  0.35886 0.36899 0.37435 0.38069 0.39328

```

3. Compare those results with a frequentist analysis.

```
summary(lm(CD4t~ViralLoad, data=albi))
```

```

##
## Call:
## lm(formula = CD4t ~ ViralLoad, data = albi)
##
## Residuals:
##   Min     1Q   Median     3Q    Max
## -1.0828 -0.2604 -0.0197  0.2449  1.3856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.79635   0.03696 129.8 < 2e-16 ***
## ViralLoad   -0.04564   0.01201   -3.8 0.000154 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3742 on 948 degrees of freedom
## Multiple R-squared:  0.01501,    Adjusted R-squared:  0.01397
## F-statistic: 14.44 on 1 and 948 DF,  p-value: 0.0001537

```

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.

```

# Fixed effect logistic regression Albi-ANRS
model{

  # likelihood
  for (i in 1:N){
    CD4sup500[i] ~ dbern(proba[i])
    logit(proba[i]) <- beta0 + beta1*VL[i] + beta2*Tt_switch[i] + beta3*d4t[i] + beta4*VL[i]*d4t[i]
  }

  #Priors
  beta0~dnorm(0,0.0001)
  beta1~dnorm(0,0.0001)
  beta2~dnorm(0,0.0001)
  beta3~dnorm(0,0.0001)
  beta4~dnorm(0,0.0001)
  beta5~dnorm(0,0.0001)
}
```

```

#ORs
OR_VL <- exp(beta1)
OR_switch <- exp(beta2)
OR_d4t <- exp(beta3)
OR_d4tVL <- exp(beta4)
OR_swVL <- exp(beta5)
}

```

```

library(rjags)
albi_logis_jags <- jags.model("albiBUGSmodel_logistic_fixed.txt",
                                data = list('CD4sup500' = 1*(albi$CD4sup500=="Yes"),
                                            'N' = nrow(albi),
                                            'VL' = albi$ViralLoad,
                                            'd4t' = 1*(albi$Treatment == "d4t+ddI"),
                                            'Tt_switch' = 1*(albi$Treatment == "switch")),
                                n.chains = 3)

```

```

## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 950
##   Unobserved stochastic nodes: 6
##   Total graph size: 7299
##
## Initializing model

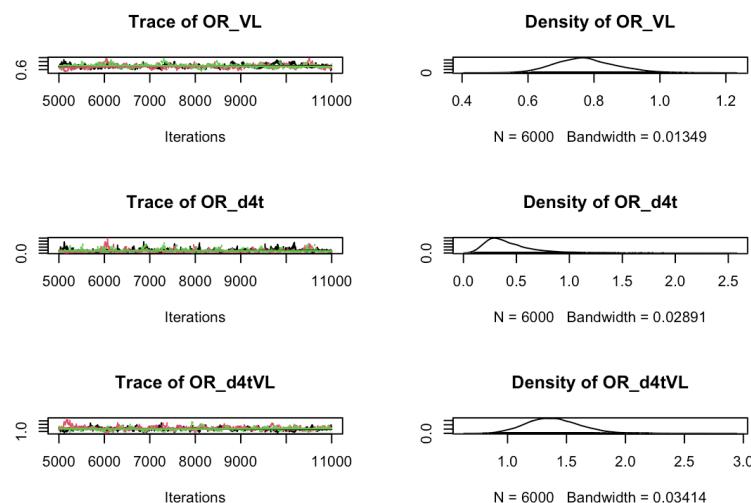
```

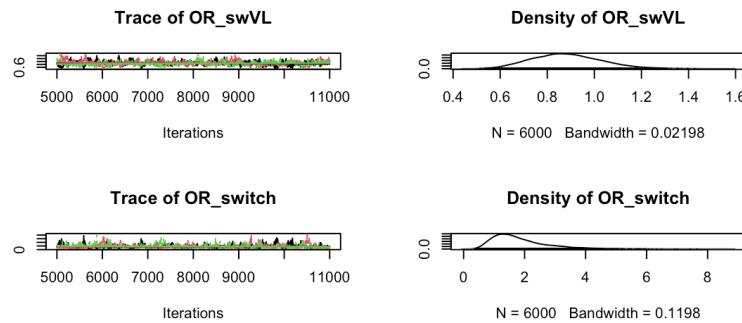
```

res_albi_logis <- coda.samples(model = albi_logis_jags,
                                 variable.names = c('OR_VL', 'OR_switch', 'OR_d4t', 'OR_d4tVL', 'OR_swVL'),
                                 n.iter = 10000)
res_albi_logis_burnt <- window(res_albi_logis, start = 5001)

plot(res_albi_logis_burnt)

```





```
summary(res_albi_logis_burnt)
```

```
## 
## Iterations = 5001:11000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 6000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean        SD  Naive SE Time-series SE
## OR_VL      0.7756  0.09302  0.0006933   0.006697
## OR_d4t     0.4311  0.24092  0.0017957   0.014445
## OR_d4tVL   1.3961  0.23990  0.0017881   0.016064
## OR_swVL    0.8843  0.14712  0.0010966   0.009168
## OR_switch  1.8490  0.94414  0.0070372   0.059436
##
## 2. Quantiles for each variable:
##
##           2.5%       25%       50%       75%    97.5%
## OR_VL      0.6118  0.7118  0.7695  0.8329  0.9728
## OR_d4t     0.1356  0.2689  0.3731  0.5283  1.0569
## OR_d4tVL   0.9847  1.2315  1.3794  1.5378  1.9073
## OR_swVL    0.6241  0.7802  0.8762  0.9790  1.1983
## OR_switch  0.6719  1.1887  1.6202  2.2639  4.2556
```

```
summary(glm(CD4sup500=="Yes"~ ViralLoad*Treatment, family="binomial", data=albi))
```

```
## 
## Call:
## glm(formula = CD4sup500 == "Yes" ~ ViralLoad * Treatment, family = "binomial",
##      data = albi)
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 0.3279    0.3241   1.012   0.3117
## ViralLoad                  -0.2556    0.1197  -2.135   0.0328 *
## Treatmentd4t+ddI            -0.9409    0.5338  -1.763   0.0779 .
## Treatmentswitch              0.4836    0.4807   1.006   0.3143
## ViralLoad:Treatmentd4t+ddI  0.3084    0.1737   1.776   0.0757 .
## ViralLoad:Treatmentswitch   -0.1303    0.1690  -0.771   0.4409
## ---
```

```

## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 1287.8 on 949 degrees of freedom
## Residual deviance: 1270.9 on 944 degrees of freedom
## AIC: 1282.9
##
## Number of Fisher Scoring iterations: 4

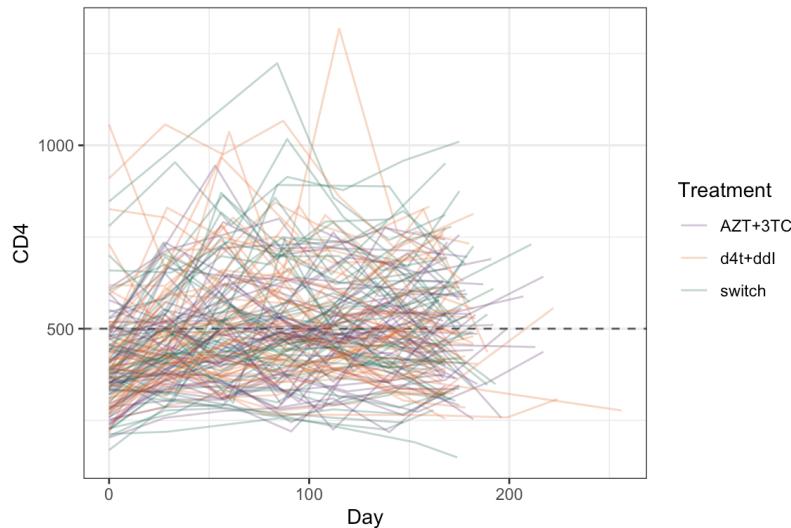
```

5. So far we have ignored the longitudinal nature of our measurements. Now, lets add a random intercept in our logistic regression to account for the intra-patient correlation across visits.

```

ggplot(albi, aes(y = CD4, x = Day, color = Treatment)) +
  geom_hline(aes(yintercept=500), color="grey35", linetype = 2) +
  geom_line(aes(group=PatientID), alpha=0.3) +
  MetBrewer::scale_color_met_d("Java") +
  theme_bw()

```



```

# Mixed effect logistic regression Albi-ANRS
model{

  # likelihood
  for (i in 1:N){
    CD4sup500[i] ~ dbern(proba[i])
    logit(proba[i]) <- beta0 + b0[PAT_ID[i]] + beta1*VL[i] + beta2*Tt_switch[i] + beta3*d4t[i] + l
  }

  for (j in 1:Npat){
    b0[j]~dnorm(0, tau_b0)
  }

  #Priors
  beta0~dnorm(0,0.0001)
  beta1~dnorm(0,0.0001)
  beta2~dnorm(0,0.0001)
  beta3~dnorm(0,0.0001)
  beta4~dnorm(0,0.0001)
  beta5~dnorm(0,0.0001)
  tau_b0~dgamma(0.001,0.001)

  #ORs
  OR_VL <- exp(beta1)
  OR_switch <- exp(beta2)
  OR_d4t <- exp(beta3)
}

```

```

OR_d4tVL <- exp(beta4)
OR_swVL <- exp(beta5)
sigma_RI <- pow(tau_b0, -0.5)
}

```

```

library(rjags)
albi_logis_RI_jags <- jags.model("albiBUGSmodel_logistic_RandomInt.txt",
                                    data = list('CD4sup500' = 1*(albi$CD4sup500=="Yes"),
                                                'N' = nrow(albi),
                                                'VL' = albi$ViralLoad,
                                                'd4t' = 1*(albi$Treatment == "d4t+ddI"),
                                                'Tt_switch' = 1*(albi$Treatment == "switch"),
                                                'Npat' = length(levels(albi$PatientID)),
                                                'PAT_ID' = as.numeric(albi$PatientID)),
                                    n.chains = 3)

```

```

## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 950
##   Unobserved stochastic nodes: 156
##   Total graph size: 8670
##
## Initializing model

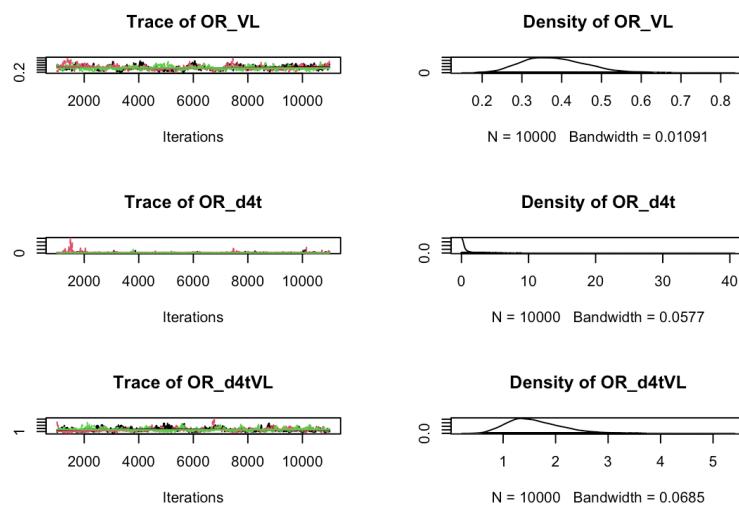
```

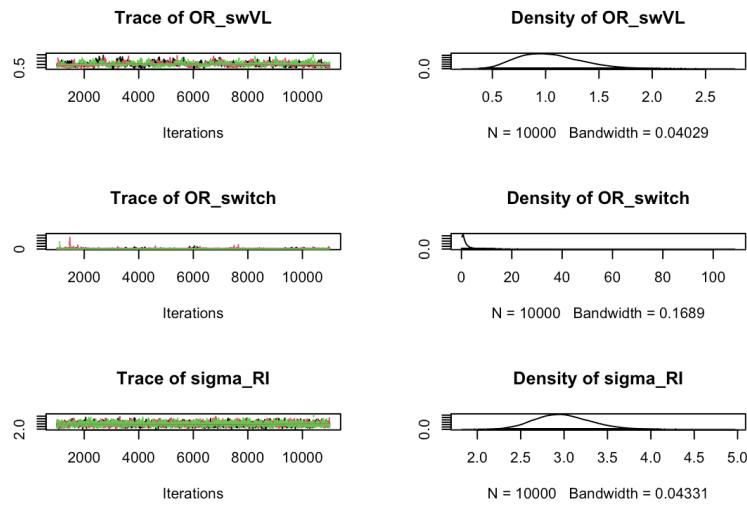
```

res_albi_logis_RI <- coda.samples(model = albi_logis_RI_jags,
                                     variable.names = c('OR_VL', 'OR_switch', 'OR_d4t', 'OR_d4tVL', 'OR_swVL'),
                                     n.iter = 10000)
res_albi_logis_RI_burnt <- window(res_albi_logis_RI, start = 1001)

plot(res_albi_logis_RI_burnt)

```





```
summary(res_albi_logis_RI_burnt)
```

```
## 
## Iterations = 1001:11000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean        SD  Naive SE Time-series SE
## OR_VL     0.3836  0.08091  0.0004671   0.005147
## OR_d4t    0.7059  1.56191  0.0090177   0.104433
## OR_d4tVL 1.6073  0.52567  0.0030349   0.033714
## OR_swVL   1.0433  0.29873  0.0017247   0.015018
## OR_switch 1.9231  2.90817  0.0167903   0.130917
## sigma_RI  2.9892  0.32354  0.0018679   0.006577
##
## 2. Quantiles for each variable:
##
##           2.5%      25%      50%      75%   97.5%
## OR_VL     0.24876  0.3249  0.3766  0.4344  0.5592
## OR_d4t    0.03199  0.1416  0.3136  0.7149  3.5807
## OR_d4tVL  0.78549  1.2328  1.5335  1.9134  2.8048
## OR_swVL   0.55936  0.8245  1.0123  1.2251  1.7011
## OR_switch 0.16207  0.5471  1.0965  2.2257  8.5558
## sigma_RI  2.40564  2.7636  2.9703  3.1940  3.6754
```

```
library(lme4)
summary(lme4::glmer(CD4sup500=="Yes"~ (1|PatientID) + ViralLoad*Treatment, family="binomial", data=albi))
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
## Approximation) [glmerMod]
## Family: binomial ( logit )
## Formula: CD4sup500 == "Yes" ~ (1 | PatientID) + ViralLoad * Treatment
## Data: albi
##
##           AIC         BIC       logLik -2*log(L)  df.resid
##         971.8      1005.8     -478.9      957.8      943
##
## Scaled residuals:
##       Min      1Q  Median      3Q      Max
## -1.5000 -0.8000 -0.2000  0.4000  1.5000
```

```

## -2.9273 -0.4021 -0.1947  0.4119  3.9688
##
## Random effects:
## Groups      Name        Variance Std.Dev.
## PatientID (Intercept) 7.317     2.705
## Number of obs: 950, groups: PatientID, 149
##
## Fixed effects:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                1.901075  0.689952  2.755  0.00586 **
## ViralLoad                 -0.955590  0.212725 -4.492 7.05e-06 ***
## Treatmentd4t+ddI          -1.177377  1.103323 -1.067  0.28592
## Treatmentswitch           0.101717  1.000232  0.102  0.91900
## ViralLoad:Treatmentd4t+ddI 0.439794  0.308829  1.424  0.15443
## ViralLoad:Treatmentswitch  0.008679  0.291172  0.030  0.97622
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) VirlLd Trt4+I Trtmnt VL:T4+
## ViralLoad   -0.780
## Trtmntd4t+I -0.616  0.473
## Trtmntswtch -0.679  0.521  0.423
## VrlLd:Tr4+I  0.518 -0.657 -0.821 -0.355
## VrlLd:Trtmn  0.549 -0.695 -0.341 -0.786  0.475

```

# BONUS Exercise 9

In this exercise, we will first do a critical reading of the article from Kaguelidou *et al.* (2016)<sup>8</sup>.

1. List the method elements that are missing from this article.

- *the prior distribution*
- *the sampling model/lielihood*
- *underlying PK/PD assumptions*
- *a sensitivity analysis with different priors (e.g. using simulations)*
- *BONUS: errors in table 3 (some of the bold numbers are not the right ones...)*

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(0.010, 0.015, 0.020, 0.025, 0.030, 0.040, 0.050)
bcrm(stop = list(nmax = 42), p.tox0 = p.tox0, dose = dose.label, ff = "power", prior.alpha = list(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. Goligher et al., ibid.
  4. 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).
  5. 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).
  6. 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.
  7. 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).
  8. 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).