Introduction to Bayesian analysis for medical studies

Practicals solutions

Contents

Exercise 1

1. Create a Monte Carlo sample of size 50 from a Gaussian distribution with mean $m = 2$ and standard deviation $s = 3$. Compute Monte Carlo estimates of both the mean and the standarddeviation on this 50-sample. Then create a sample of size 20 000 and compute again such estimates on this 20 000-sample. What do you notice ? Which famous theoretical property is illustrated here ?

```
sample_50 <- rnorm(50, mean = 2, sd = 3)
mean(sample_50)
```

```
## [1] 1.047278
```

```
sample_20000 <- rnorm(20000, mean = 2, sd = 3)
mean(sample_20000)
```

```
## [1] 2.004215
```

When the sample size increase, the Monte Carlo estimator becomes more precise. This illustrate 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 `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)
}
```

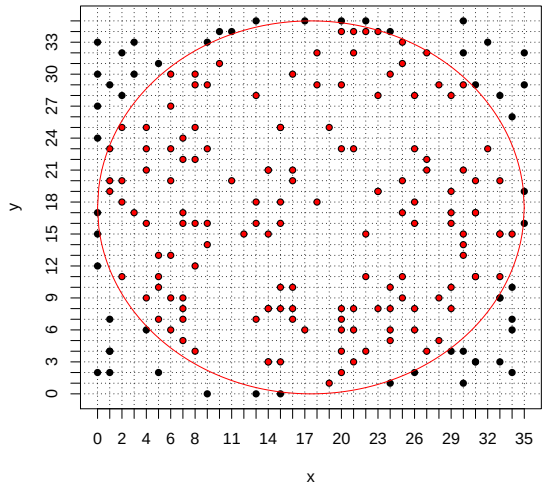
c. The surface ratio between the disk (radius $\frac{ngrid}{2}$) and the square (side length $ngrid$) is equal to $\frac{\pi}{4}$, i.e. the probability of sampling a point the disk rather than outside is $\frac{\pi}{4}$. Now, using this result, program a function to compute a Monte Carlo estimate of pi from 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) {
    return(mean(4 * in_disk))
}
```

d. Using the code below, generate 200 points and plot the data generated. What is the corresponding Monte Carlo estimate of π ? 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
```

```r
# 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)
```

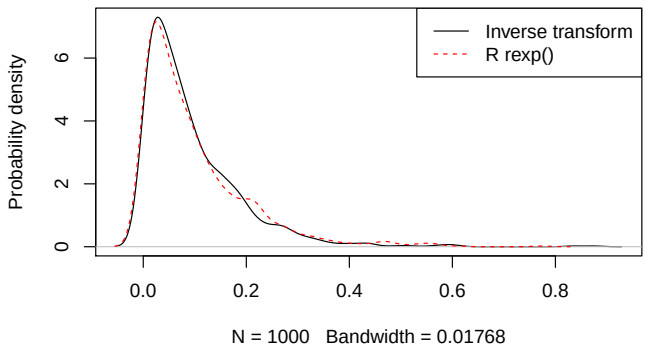


When the sample size increase, the Monte Carlo estimator becomes more precise (LLN). However, if the grid is too coarse, $\widehat{\pi}$ is underestimated (underestimating the disk surface by missing the bits near the edge). Therefore, increasing the number of points on the grid also improves the precision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

Exercise 2

1. Program a sampling algorithm to sample from the exponential distribution with parameter λ thanks to the inverse transform function (starting from the `R` function `runif`).
   Compare your results to the built-in `R` function `rexp` by comparing their histogram or density (e.g. by using the functions `density()`, `plot()` and `lines()`).
   Try out several values for the λ parameter of the exponential distribution (e.g. 1, 10, 0.78, ...).

```r
my_sampler_expodist <- function(n, lambda) {
    u <- runif(n)
    e <- -1/lambda * log(1 - u)
    return(e)
}

nsamp <- 1000
my_samp <- my_sampler_expodist(n = nsamp, lambda = 10)
r_samp <- rexp(n = nsamp, rate = 10)
plot(density(my_samp), main = "", ylab = "Probability density")
lines(density(r_samp), col = "red", lty = 2)
legend("topright", legend = c("Inverse transform",
    "R rexp()"), lty = c(1, 2), col = c("black",
    "red"))
```



Exercise 3

Using the historical example, program an independant Metropolis-Hastings algorithm to estimate the *posterior* distribution of parameterθ (i.e. the probability of having a girl for a birth). The *prior* distribution on θ will be used as the instrumental proposal, and we will start by using a uniform *prior* on θ. We will consider the $493,472$ births observed in Parisbetween 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 poster
    }
    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 returns the vector of acceptance probabilities α. What happens if this acceptance probability is ***NOT*** computed on the *log* scale ?

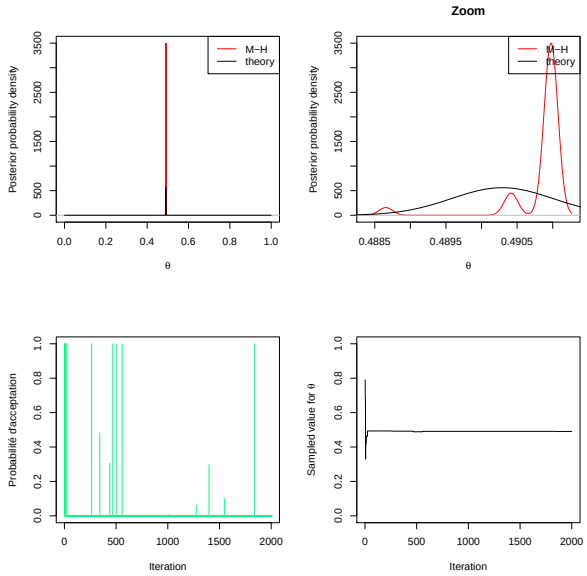
```
myMH <- function(niter, post_num) {
    x_save <- numeric(length = niter)  #create a vector of 0s of length ni
    alpha <- numeric(length = niter)  #create a vector of 0s of length nit
```

```
    # initialise x0
    x <- runif(n = 1, min = 0, max = 1)
    # accpetance-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 itérations to the theoretical one (the theoretical density can be obtained with the `R` fuction `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 θ.

```
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)
```

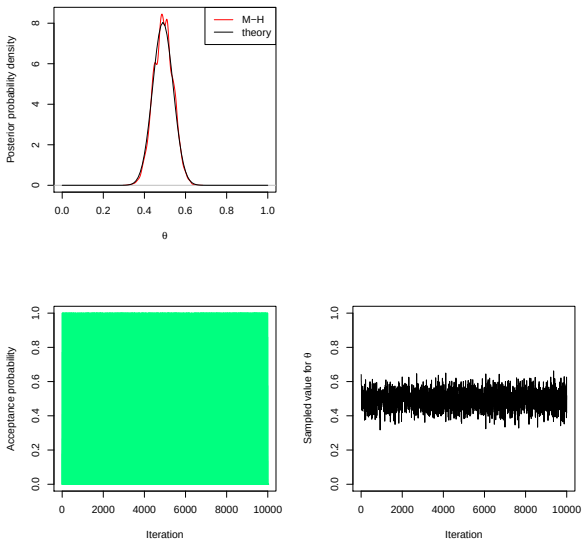
```
legend("topright", c("M-H", "theory"), col = c("red",
    "black"), lty = 1)

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$alpha, type = "h", xlab = "Iteration",
    ylab = "Probabilité d'acceptation", ylim = c(0,
        1), col = "springgreen")
plot(sampleMH$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 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).

```r
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) {

    theta_save <- numeric(length = niter)  # vector of theta values ready
    alpha <- numeric(length = niter)  # vector of alpha values ready to be

    # initialise theta
    theta <- runif(n = 1, min = 0, max = 1)
    for (t in 1:niter) {

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

        # compute acceptance-rejection probability
        alpha[t] <- min(1, exp(post_num(theta_prop,
            a = a, b = b, log = TRUE) - post_num(theta,
            a = a, b = b, log = TRUE)))
        # acceptance-rejection step
        u <- runif(1)
        if (u <= alpha[t]) {
            theta <- theta_prop  # acceptance of theta_prop as new current
        }
        # saving the current value of theta
```

```
        theta_save[t] <- theta
    }
    return(list(theta = theta_save, alpha = alpha))
}

sampleMH <- myMH_betaprior(10000, post_num = post_num_beta)


par(mfrow = c(2, 2))
plot(density(sampleMH$theta[-c(1:500)]), col = "red",
    xlim = c(0, 1), ylab = "Posterior probability density",
    xlab = expression(theta), main = "")
curve(dbeta(x, 49 + 1, 51 + 1), from = 0, to = 1,
    add = TRUE)
legend("topright", c("M-H", "theory"), col = c("red",
    "black"), lty = 1)
plot.new()
plot(sampleMH$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))
```

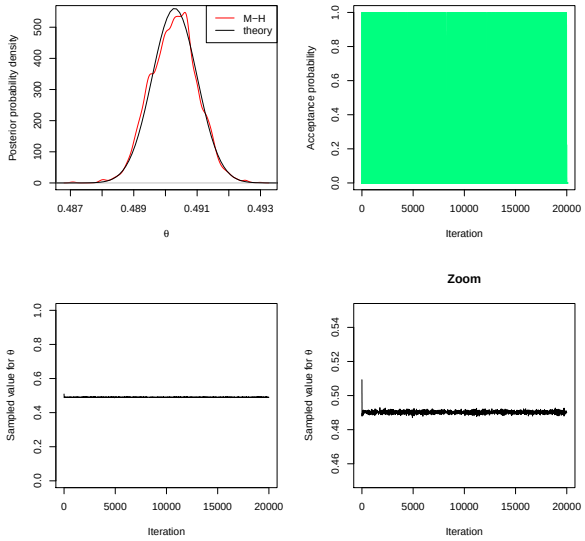


5. Using the data from the historical example and with a Beta($\alpha = 3, \beta = 3$) *prior*, program a random-walk Metropolis-Hastings algorithm (with

a Gaussian random step with a standard deviation of 0.02 for instance). Once again, study the results obtained this way (one can change the size of the random step).

```
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) {

    theta_save <- numeric(length = niter)
    alpha <- numeric(length = niter)

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

    for (t in 1:niter) {
        # sample a value from the proposal (random
        # walk)
        theta_prop <- theta + runif(1, -0.01,
            0.01)  #rnorm(1, mean = 0, sd = 0.02)

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

        # acceptance-rejection step
```

```
        u <- runif(1)
        if (u <= alpha[t]) {
            theta <- theta_prop  # accept theta_prop and update current va
        }

        # save current value
        theta_save[t] <- theta
    }

    return(list(theta = theta_save, alpha = alpha))
}

sampleMH <- myMH_betaprior_randomwalk(20000, post_num = post_num_beta_hist)


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


M−H
theory
Posterior probability density
θ
Acceptance probability
Iteration
Zoom
Sampled value for θ
Iteration
Sampled value for θ
Iteration

Exercise 4

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 mentionned, recently developed et the Columbia Univeristy, 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`.

```r
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`.

```r
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

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

Each model specification file must start with the instruction `model{` indicating `JAGS` it is about to recieve 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`. ***Warning***: 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)))
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 50
```

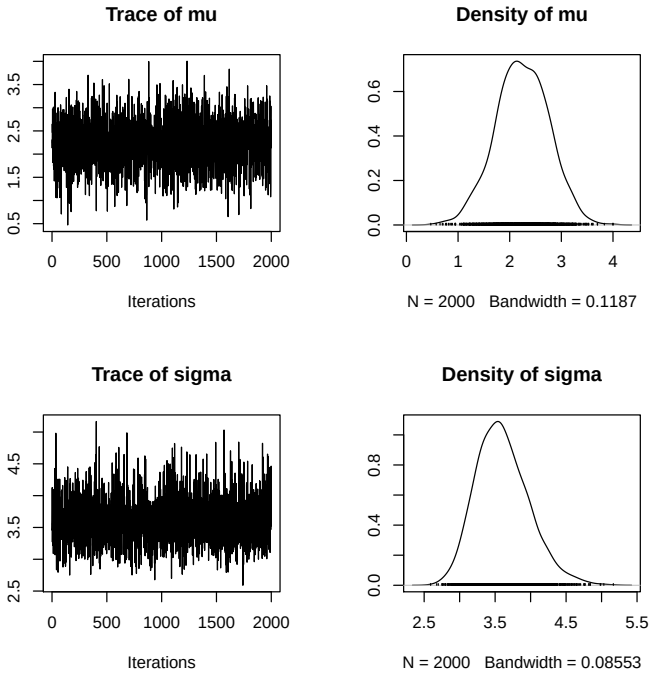
```
##    Unobserved stochastic nodes: 2
##    Total graph size: 58
##
## Initializing model
```

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)
```




```
resum <- summary(res)
resum
```

```
##
## 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    2.242 0.5122  0.01145        0.01145
## sigma 3.605 0.3721  0.00832        0.00832
##
## 2. Quantiles for each variable:
##
##         2.5%   25%   50%   75% 97.5%
## mu    1.222 1.894 2.238 2.593 3.218
## sigma 2.974 3.340 3.573 3.834 4.429
```

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

```
## [1] 2.242339
```

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

```
## [1] 3.605155
```

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

```
## [1] 2.238252
```

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

```
## [1] 3.572914
```

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

```
##     2.5%    97.5%
## 1.221985 3.217860
```

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

```
##     2.5%    97.5%
## 2.974311 4.428598
```

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

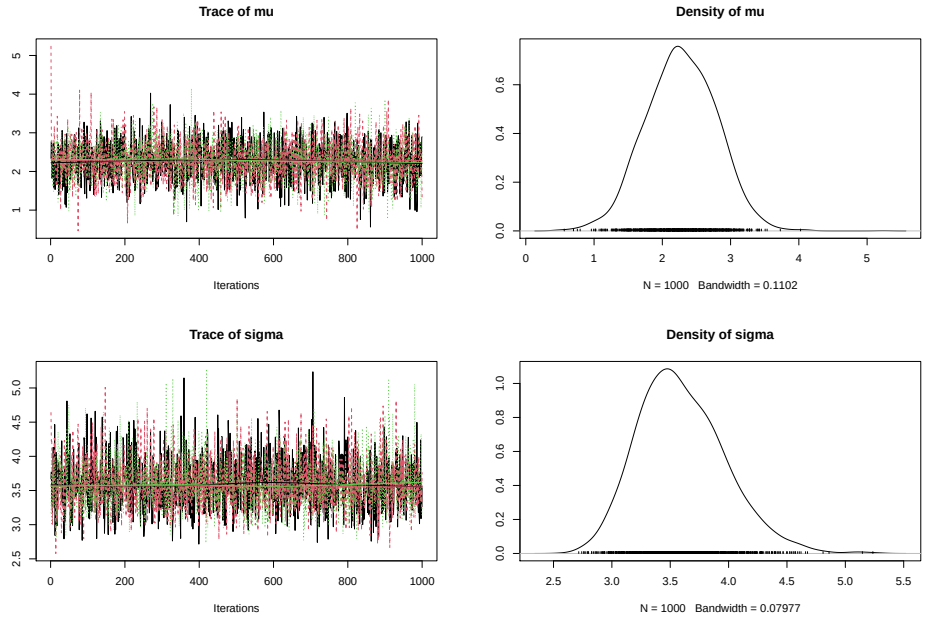
```
library(coda)
```

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 3 the use of 3 Markov chains with the argument `n.chains`, and initialize `mu` and `tau` at $0, -10, 100$ and 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)))
```

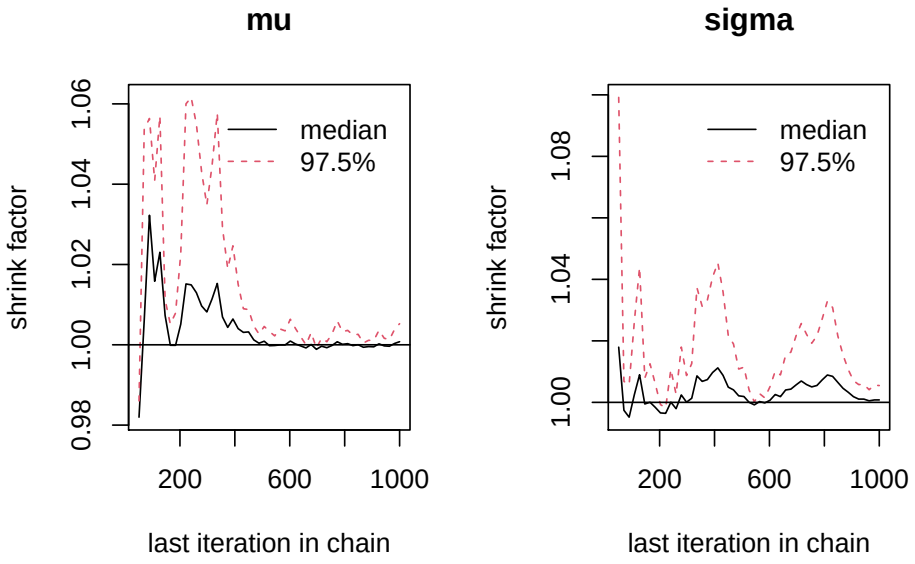
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 50
##    Unobserved stochastic nodes: 2
##    Total graph size: 58
##
## Initializing model
```

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



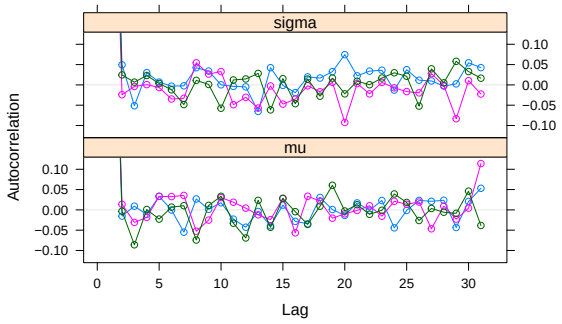
8. With the R function `gelman.plot()`, plot the Gelman-Rubin statistic.

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

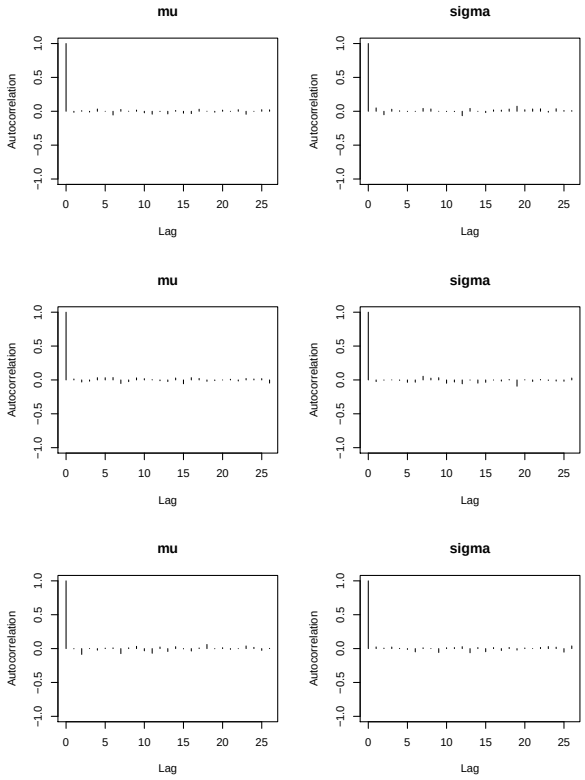


9. With the R functions `autocorr.plot()` and `acfplot()` evaluate the autocorrélation of the studied parameters.

```
acfplot(res2)
```

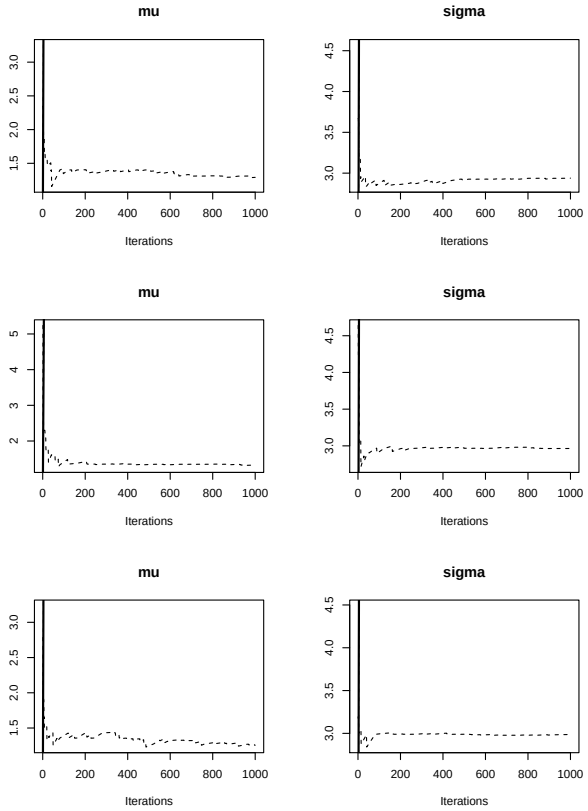



```
par(mfrow = c(3, 2))
autocorr.plot(res2, ask = FALSE, auto.layout = FALSE)
```



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

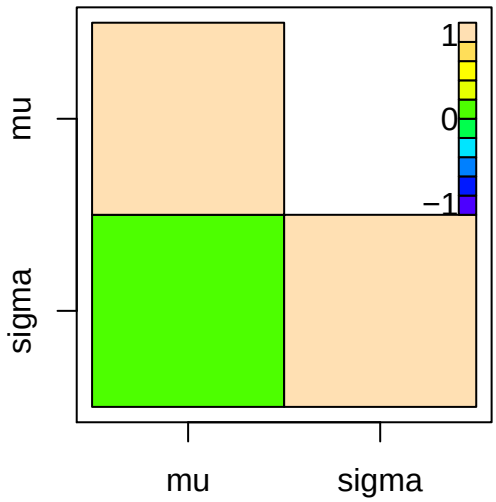
```
par(mfrow = c(3, 2))
cumuplot(res2, ask = FALSE, auto.layout = FALSE)
```



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

11. With the `R` function `crosscorr.plot()` evaluate the correlations between the studied parameters. How can you interpret them ?

```
crosscorr.plot(res2)
```



12. With the function `hdi()` from the `R` package `HDInterval`, provide highest densitity *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 1.250102 2.909750
## upper 3.211796 4.329276
## attr(,"credMass")
## [1] 0.95
```

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

```
##           2.5%    97.5%
## mu    1.283237 3.266868
## sigma 2.966852 4.411192
```

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

```
##       mu    sigma
## 1.983631 1.444340
```

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

```
##       mu    sigma
## 1.961694 1.419526
```

Exercise 5

The randomized clinical trial *EOLIA*[1] 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)[2] performed a Bayesian re-analysis of these data, further exploring the evidence and how it can be quantified and summarized with a Bayesian approach.

Table 1: Observed data from the *EOLIA* trial

	Control	ECMO
n **observed**	125	124
number of deceased at 60 days	57	44

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

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

[1] Alain Combes et al. "Extracorporeal Membrane Oxygenation for Severe Acute Respiratory Distress Syndrome," *New England Journal of Medicine* 378, no. 21 (2018): 1965–1975, doi: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.

$$log(RR) \sim U_{[-35,35]}$$

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.

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

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

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

   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(-35,35)
  pc~dunif(0,1) #probability of death in the control group

  # Reparametrizations
  RR <- exp(logRR)
}
```

```
# 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(-35,35)
  pc~dunif(0,1) #probability of death in the control group

  # Reparametrizations
  RR <- exp(logRR)
}
```

3. 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.)

```
# Individual data
ycontrol <- c(rep(0, 125 - 57), rep(1, 57))
yecmo <- c(rep(0, 124 - 44), rep(1, 44))

# sampling
library(rjags)
goligher_jags_indiv <- jags.model(file = "goligherBUGSmodel_indiv.txt",
    data = list(ycontrol = ycontrol, ncontrol = length(ycontrol),
        yecmo = yecmo, necmo = length(yecmo)),
    n.chains = 3)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 249
##    Unobserved stochastic nodes: 2
##    Total graph size: 259
##
## Initializing model
```

```
res_goligher_indiv <- coda.samples(model = goligher_jags_indiv,
    variable.names = c("pc", "RR"), n.iter = 20000)

# postprocessing
res_goligher_burnt_indiv <- window(res_goligher_indiv,
```

```
    start = 5001)  # remove burn-in for Markov chain convergence




# Population data
zcontrol <- 57
zecmo <- 44
# sampling
goligher_jags_pop <- jags.model(file = "goligherBUGSmodel_pop.txt",
    data = list(zcontrol = zcontrol, ncontrol = 125,
        zecmo = zecmo, necmo = 124), n.chains = 3)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2
##    Unobserved stochastic nodes: 2
##    Total graph size: 12
##
## Initializing model
```

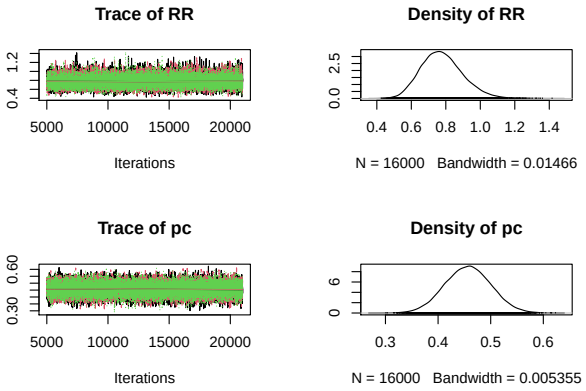
```
res_goligher_pop <- coda.samples(model = goligher_jags_pop,
    variable.names = c("pc", "RR"), n.iter = 20000)

# post-processing
res_goligher_burnt_pop <- window(res_goligher_pop,
    start = 5001)  # remove burn-in for Markov chain convergence
```

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

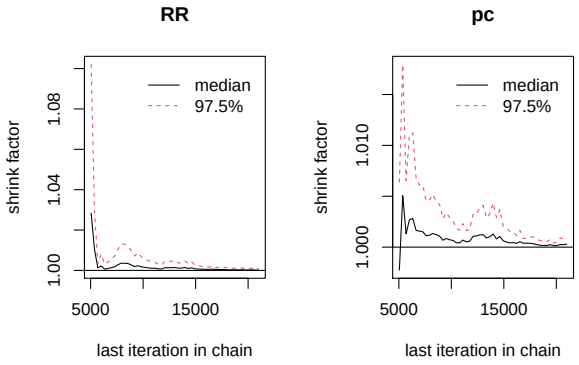
```
effectiveSize(res_goligher_burnt_pop)
```

```
##       RR       pc
## 13550.22 13549.92
```

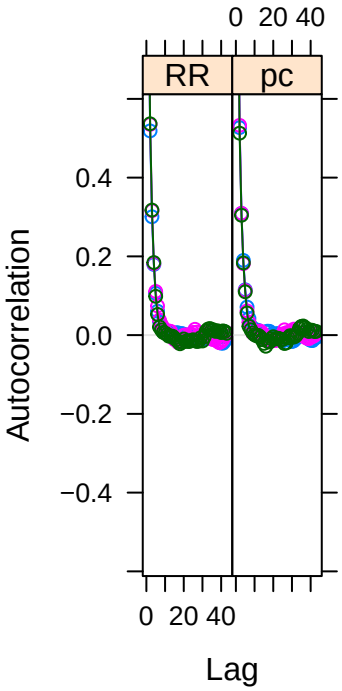
```
plot(res_goligher_burnt_pop)
```



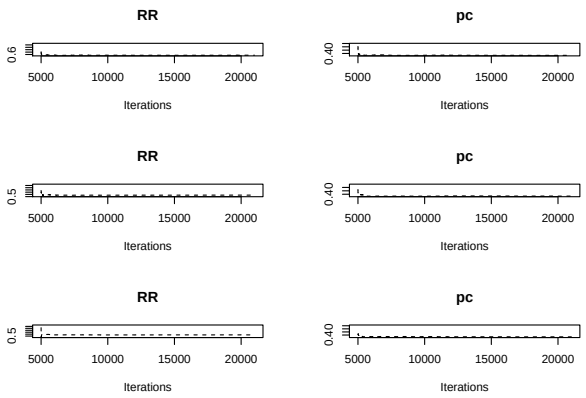

```
gelman.plot(res_goligher_burnt_pop)
```




```
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 = 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
## RR 0.7786 0.12110 0.0005527      0.0010403
## pc 0.4566 0.04362 0.0001991      0.0003747
##
## 2. Quantiles for each variable:
##
##      2.5%    25%    50%    75%  97.5%
## RR 0.5661 0.6939 0.7705 0.8539 1.0391
## pc 0.3714 0.4269 0.4568 0.4861 0.5421
```

```
summary(res_goligher_burnt_indiv)
```

```
##
## 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
## RR 0.7786 0.12259 0.0005595      0.0011051
## pc 0.4565 0.04429 0.0002022      0.0003963
##
## 2. Quantiles for each variable:
##
##      2.5%    25%    50%    75%  97.5%
## RR 0.5624 0.6928 0.7697 0.8548 1.0447
## pc 0.3702 0.4263 0.4562 0.4867 0.5442
```

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

```
##             RR        pc
## lower 0.551520 0.3693301
## upper 1.018657 0.5398412
```

```
## attr(,"credMass")
## [1] 0.95
```

```
# posterior porbability of RR <1:
mean(c(sapply(res_goligher_burnt_pop, "[", , 1)) <
    1)
```

```
## [1] 0.9560625
```

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 ≈ 45). Comment the results. Try out other *prior* distributions.

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

Exercise 6

In 2014, Crins *et al.*[3] 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`[4] 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: http://ams.med.uni-goettingen.de:3838/bayesmeta/app/. Explore and comment the results and the outputs, try out different options and *priors*, etc.

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 stantard 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

[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): 839–850, doi:10.1111/petr.12362.

[4] Christian Röver "Bayesian Random-Effects Meta-Analysis Using the Bayesmeta R Package," *arXiv Preprint 1711.08683* (2017), http://www.arxiv.org/abs/1711.08683.

Log-odds ratios are symmetric around zero and have a sampling distributions closer to the normal distibution than the natural OR scale. For this reason, they are usually prefered for meta-analyses. Their sample variance is then computed as the sum of the inverse of all the counts in the 2×2 associated contingency table[5].

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 τ and a Gaussian *prior* for μ centered around 0 and with a standard deviation of 4.

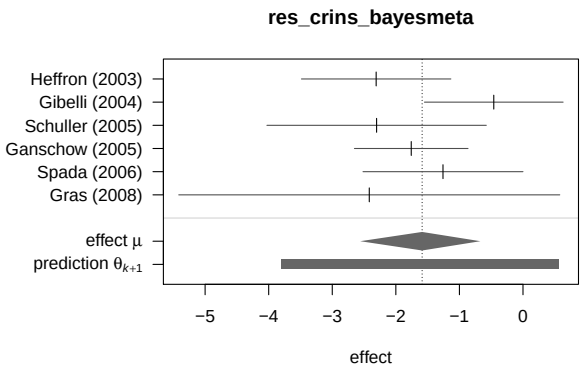
```
res_crins_bayesmeta <- bayesmeta(y = crins.es$yi,
    sigma = sqrt(crins.es$vi), labels = crins.es$publication,
    tau.prior = function(t) {
        dunif(t, max = 4)
    }, mu.prior = c(0, 4), interval.type = "central")
summary(res_crins_bayesmeta)
```

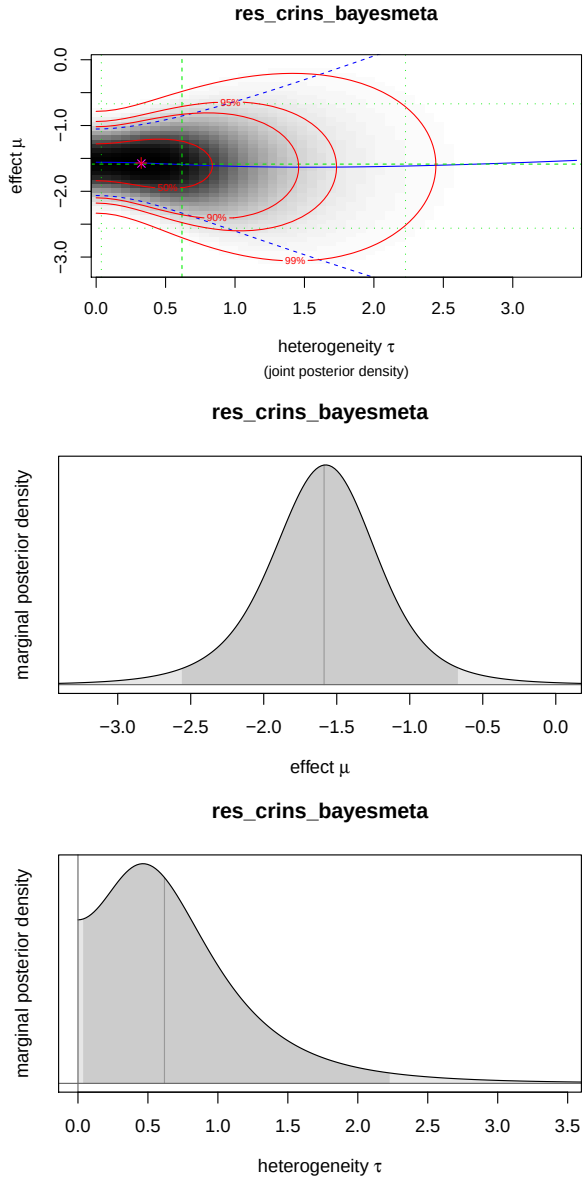
```
##  'bayesmeta' object.
## data (6 estimates):
##                            y     sigma
## Heffron (2003)  -2.3097026 0.5994763
## Gibelli (2004)  -0.4595323 0.5563956
## Schuller (2005) -2.3025851 0.8803408
## Ganschow (2005) -1.7578579 0.4558685
## Spada (2006)    -1.2584610 0.6419962
## Gras (2008)     -2.4178959 1.5288107
##
## tau prior (proper):
## function(t) {
##         dunif(t, max = 4)
##     }
## <bytecode: 0x7fd524bc1868>
##
## mu prior (proper):
## normal(mean=0, sd=4)
##
## ML and MAP estimates:
##                     tau        mu
```

[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), 237–253.

```
## 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)
```





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

   I) Question of interest:

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

   II) Sampling model:

   Let $logOR_i$ be the log-odds-ratio reported by the study i and σ_i^2 its sampling variance

$$logOR_i \overset{iid}{\sim} N(\theta_i, \sigma_i^2)$$

$$\theta_i \overset{iid}{\sim} N(\mu, \tau)$$

III) Priors:

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

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

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

```
# 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: 33
##
## Initializing model
```

```
res_crins_jags_res <- coda.samples(model = crins_jags_res,
    variable.names = c("mu", "tau"), 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)
```

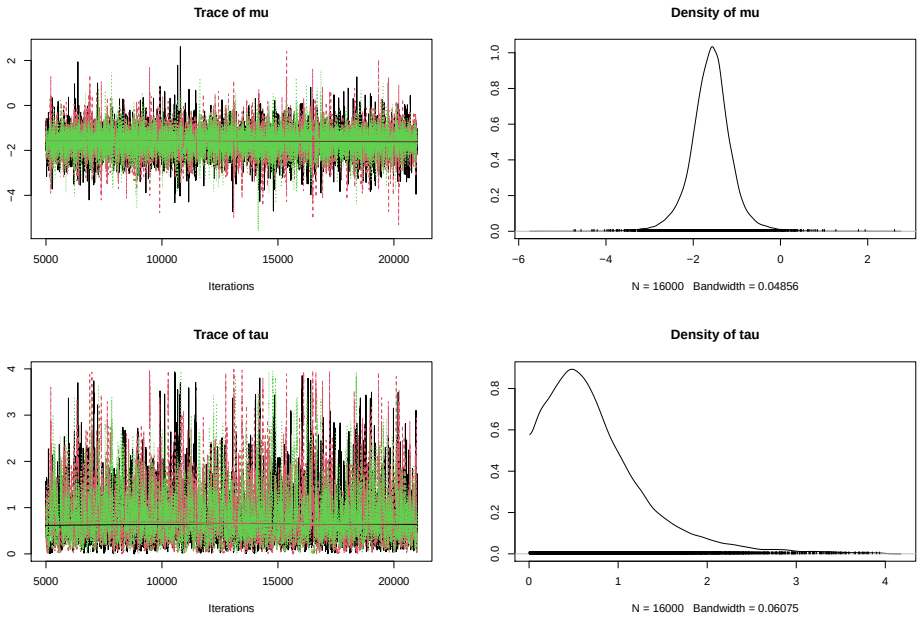
```
##
## Iterations = 5001:21000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 16000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
```

```
##         Mean     SD Naive SE Time-series SE
## mu  -1.5984 0.4803 0.002192       0.004151
## tau  0.7609 0.5841 0.002666       0.011202
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%    75%  97.5%
## mu  -2.59181 -1.8617 -1.5878 -1.332 -0.651
## tau  0.04861  0.3503  0.6349  1.014  2.306
```

```
HDInterval::hdi(res_crins_jags_res)
```

```
##                mu         tau
## lower -2.5890184 0.002642344
## upper -0.6494661 1.902794488
## attr(,"credMass")
## [1] 0.95
```

```
plot(res_crins_jags_res)
```



Exercise 7

In this exercise, we will first do a critical reading of the article from Kaguelidou *et al.* (2016).[6]

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

   - *the prior distribution*
   - *the sampling model/likelihood*
   - *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(0.01, 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)
```

[6] 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.