Worksheet 4

Foundations of Bayesian Methodology

Wenje Tu Lea Bührer Jerome Sepin Zhixuan Li
Elia-Leonid Mastropietro Jonas Raphael Füglistaler Nico Schümperli

Spring Semester 2022

Exercise 2 (Gibbs sampler)

2(a)

$$\begin{aligned}
p(x) &= \exp\left(-\frac{1}{2}(ax^2 - 2bx)\right)\\
&= \exp\left(-\frac{a}{2}\left\{x^2 - 2\frac{b}{a}x\right\}\right)\\
&= \exp\left(-\frac{a}{2}\left\{x^2 - 2\frac{b}{a}x + \left(\frac{b}{a}\right)^2 - \left(\frac{b}{a}\right)^2\right\}\right)\\
&= \exp\left(-\frac{a}{2}\left(x - \frac{b}{a}\right)^2 + \frac{b^2}{2a}\right)\\
&= \underbrace{\exp\left(\frac{b^2}{2a}\right)}_{\text{costant}} \exp\left(-\frac{a}{2}\left(x - \frac{b}{a}\right)^2\right)
\end{aligned}$$

Probability density function of $p(x)$:

$$\begin{aligned}
f(x) &= \frac{p(x)}{\int_{-\infty}^{\infty} p(x)dx}\\
&= \frac{\exp\left(\frac{b^2}{2a}\right)\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)}{\int_{-\infty}^{\infty}\exp\left(\frac{b^2}{2a}\right)\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)dx}\\
&= \frac{\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)}{\int_{-\infty}^{\infty}\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)dx}\\
&= \frac{\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)}{\sqrt{\frac{2\pi}{a}}\underbrace{\int_{-\infty}^{\infty}\sqrt{\frac{a}{2\pi}}\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)dx}_{\text{integrates to 1}}}\\
&= \sqrt{\frac{a}{2\pi}}\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)
\end{aligned}$$

$$X \sim \mathrm{N}\left(\frac{b}{a}, \frac{1}{a}\right)$$

2(b)

1. Generate data: a random normal sample by assuming `set.seed(44566)`, $n = 30$, $\mu = 4$ and $\sigma^2 = 16$.

```r
# Generate data
set.seed(44566)
mu <- 4
sigma2 <- 16
n <- 30
y <- rnorm(n=n, mean=mu, sd=sqrt(sigma2))
```

2. Set the parameters of the prior distributions to $\mu_0 = -3, \sigma_0^2 = 4, a_0 = 1.6$ (shape) and $b_0 = 0.4$ (rate), take a burn-in of 4000 simulations and run your code for a total of 10000 simulations when assuming your own initial values. Provide traceplots and summaries (mean, sd, 0.025, 0.5, 0.975 quantiles) of the marginal posteriors for μ, σ^2 and $1/\sigma^2$.

```r
# Define the parameters of the prior distributions
mu0 <- -3
sigma2_0 <- 4
a0 <- 1.6
b0 <- 0.4

####################
## Gibbs sampler (1 chain)
####################

# Initialization
set.seed(44566)

n.iter <- 10000
n.burnin <- 4000
n.thin <- 1
#n.thin <- floor((n.iter-n.burnin)/500)
n.chains <- 1
parameters <- c("mu", "sigma2", "inv_sigma2")
n.parameters <- length(parameters)

n.tot <- n.burnin + n.iter * n.thin

gibbs_samples <- matrix(NA, nrow = n.iter, ncol = n.parameters)
colnames(gibbs_samples) <- parameters

mu.sim <- rep(NA, length = n.tot)
sigma2.sim <- rep(NA, length = n.tot)
inv.sigma2.sim <- rep(NA, length = n.tot)

# Set the initial value
sigma2.sim[1] <- 1/runif(n.chains)

# Set the counter
k <- 1

# Run the for loop (only one chain)
for(i in 2:(n.burnin+n.iter*n.thin)){

  mu.sim[i] <- rnorm(1,
                     mean = (sum(y)/sigma2.sim[i-1] + mu0/sigma2_0) /
                       (n/sigma2.sim[i-1] + 1/sigma2_0),
                      sd = sqrt(1/(n/sigma2.sim[i-1] + 1/sigma2_0)))

  sigma2.sim[i] <- 1/rgamma(1, shape = n/2 + a0,
                             scale = 1 / (sum((y-mu.sim[i])^2)/2 + b0))

  inv.sigma2.sim[i] <- 1/sigma2.sim[i]

  # after the burnin save every n.thin'th sample
  if((i > n.burnin) && (i%%n.thin == 0)){
    gibbs_samples[k,] <- c(mu.sim[i], sigma2.sim[i], inv.sigma2.sim[i])
    k <- k + 1
  }

  if(i%%1000 == 0){
  # report on the fly in which iteration the chain is
    cat(i, "\n")
  }

}
```

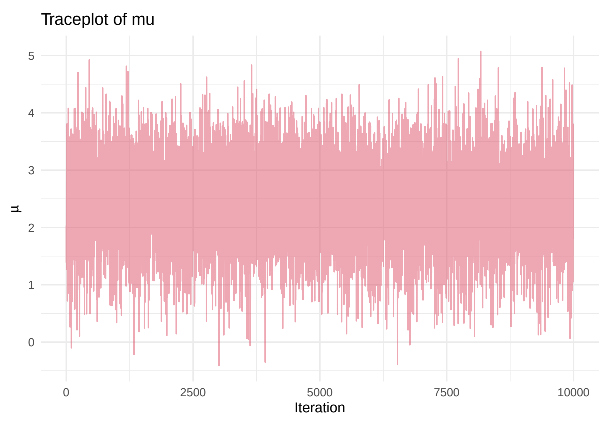
```
## 1000
## 2000
## 3000
## 4000
## 5000
## 6000
## 7000
## 8000
## 9000
## 10000
## 11000
## 12000
## 13000
## 14000
```

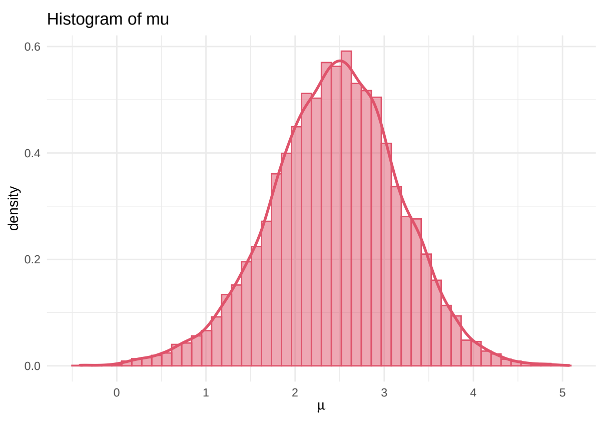
```
# n.iter samples after n.burnin taking every n.thin'th sample
dim(gibbs_samples)
```

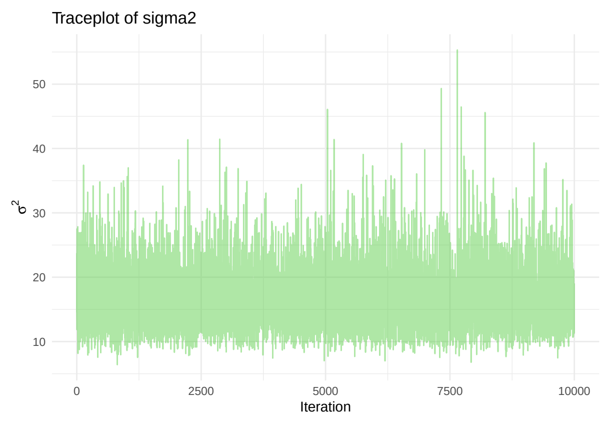
```
## [1] 10000     3
```

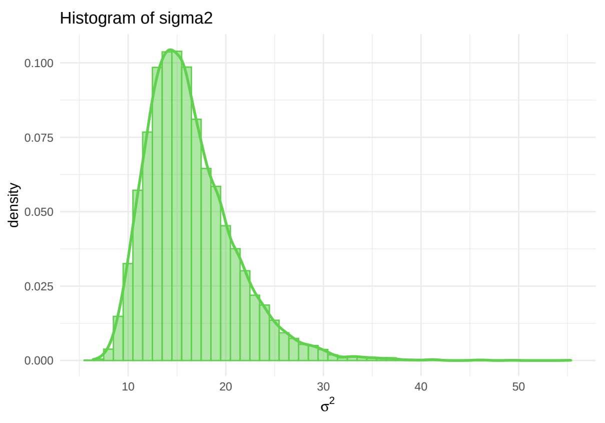
```
mu_gibbs_samples <- gibbs_samples[,"mu"]
sigma2_gibbs_samples <- gibbs_samples[,"sigma2"]
inv_sigma2_gibbs_samples <- gibbs_samples[,"inv_sigma2"]
```

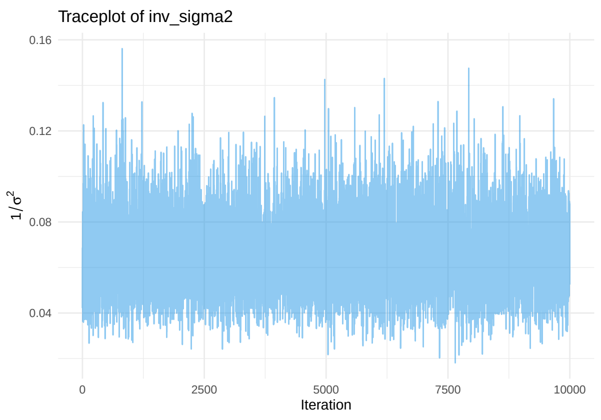
```
library(ggplot2)

d.gibbs <- data.frame(gibbs_samples)

## Traceplot of mu
ggplot(d.gibbs, aes(x=1:nrow(d.gibbs), y=mu)) +
  geom_line(color=2, alpha=0.5) +
  labs(title="Traceplot of mu", x="Iteration", y=expression(mu)) +
  theme_minimal()

## Histogram/Density of mu
ggplot(d.gibbs, aes(x=mu, y=..density..)) +
  geom_histogram(color=2, fill=2, alpha=0.5, bins=50) +
  geom_density(color=2, lwd=1) +
  labs(title="Histogram of mu", x=expression(mu)) +
  theme_minimal()

## Traceplot of sigma2
ggplot(d.gibbs, aes(x=1:nrow(d.gibbs), y=sigma2)) +
  geom_line(color=3, alpha=0.5) +
  labs(title="Traceplot of sigma2", x="Iteration", y=expression(sigma^2)) +
  theme_minimal()

## Histogram/Density of sigma2
ggplot(d.gibbs, aes(x=sigma2, y=..density..)) +
  geom_histogram(color=3, fill=3, alpha=0.5, bins=50) +
  geom_density(color=3, lwd=1) +
  labs(title="Histogram of sigma2", x=expression(sigma^2)) +
  theme_minimal()

## Traceplot of inv_sigma2
ggplot(d.gibbs, aes(x=1:nrow(d.gibbs), y=inv_sigma2)) +
  geom_line(color=4, alpha=0.5) +
  labs(title="Traceplot of inv_sigma2", x="Iteration", y=expression(1/sigma^2)) +
  theme_minimal()

## Histogram/Density of inv_sigma2
ggplot(d.gibbs, aes(x=inv_sigma2, y=..density..)) +
  geom_histogram(color=4, fill=4, alpha=0.5, bins=50) +
  geom_density(color=4, lwd=1) +
  labs(title="Histogram of inv_sigma2", x=expression(1/sigma^2)) +
  theme_minimal()
```

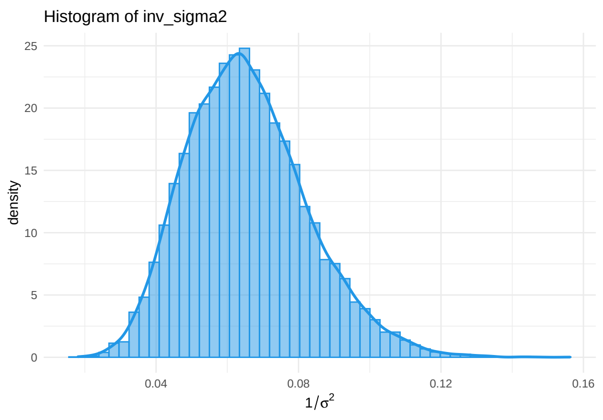












```r
d.summary <- t(
  rbind(
    colMeans(d.gibbs),
    apply(d.gibbs, 2, function(x) sd(x)),
    apply(d.gibbs, 2, function(x) quantile(x, probs=c(0.025, 0.5, 0.975)))
  )
)

d.summary <- data.frame(d.summary)
colnames(d.summary) <- c("Mean", "SD", "2.5%", "Median", "97.5%")

knitr::kable(d.summary, align="c", caption="Summary statistics of the marginal posteriors")
```

Table 1: Summary statistics of the marginal posteriors

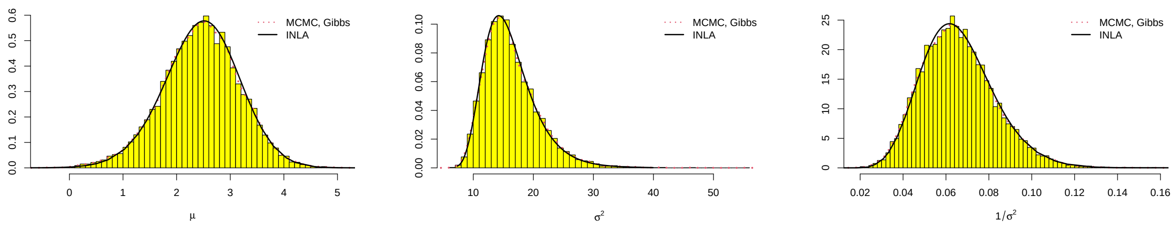
	Mean	SD	2.5%	Median	97.5%
mu	2.4479404	0.7206035	0.9436683	2.4692436	3.808022
sigma2	16.3331563	4.5483340	9.7333046	15.5669100	27.221752
inv_sigma2	0.0656163	0.0169525	0.0367353	0.0642388	0.102740

3. For this model, INLA (https://www.r-inla.org/) provides exact results. Compare results provided by the Gibbs sampler with those provided by INLA.

```r
# plots (compare with INLA)
# INLA is exact

# Question: How long should we run the MCMC Gibbs chain
#             to get close to the exact INLA approximation?
# Run for 10 min, for 30 min, for 1h...

library(INLA)
library(MASS)

formula <- y ~ 1
inla.output <- inla(formula,data=data.frame(y=y),
                    control.family=list(hyper=list(prec=list(prior="loggamma",
                                                             param=c(a0,b0)))),
                    control.fixed=list(mean.intercept=mu0, prec.intercept=1/sigma2_0))
```

```r
par(mfrow=c(1,1))
# plot for mean
```

```r
rg <- range(inla.output$marginals.fixed$"(Intercept)"[,2])
truehist(mu_gibbs_samples, prob=TRUE, col="yellow", xlab=expression(mu))
lines(density(mu_gibbs_samples),lty=3,lwd=3, col=2)
lines(inla.output$marginals.fixed$"(Intercept)",lwd=2)
legend("topright",c("MCMC, Gibbs","INLA"),lty=c(3,1),
       lwd=c(2,2),col=c(2,1),cex=1.0,bty="n")

# plot for variance
m_var <-inla.tmarginal(function(x) 1/x, inla.output$marginals.hyperpar[[1]])
truehist(sigma2_gibbs_samples, prob=TRUE, col="yellow", xlab=expression(sigma^2))
lines(density(sigma2_gibbs_samples),lty=3,lwd=3, col=2)
lines(m_var,lwd=2)
legend("topright",c("MCMC, Gibbs","INLA"),lty=c(3,1),
       lwd=c(2,2),col=c(2,1),cex=1.0,bty="n")

# plot for precision
truehist(inv_sigma2_gibbs_samples, prob=TRUE, col="yellow", xlab=expression(1/sigma^2))
lines(density(inv_sigma2_gibbs_samples),lty=3,lwd=3, col=2)
lines(inla.output$marginals.hyperpar[[1]],lwd=2)
legend("topright",c("MCMC, Gibbs","INLA"),lty=c(3,1),
       lwd=c(2,2),col=c(2,1),cex=1.0,bty="n")
```



We overlay the histogram of samples obtained from Gibbs sampling with the density plot of samples obtained from INLA and we can see that both results converge to the same target distribution (with trivial disagreement).

4. Explain step by step in your own words what the code in `04GibbsSampler.R` is doing.

$$\begin{aligned} y_1, \cdots, y_n &\overset{i.i.d.}{\sim} \mathrm{N}(\mu, \sigma^2) \\ \mu &\sim \mathrm{N}(\mu_0, \sigma_0^2) \\ 1/\sigma^2 &\sim \mathrm{G}(a_0, b_0) \end{aligned}$$

In the lecture script, we obtain posterior distribution:

$$\begin{aligned} f(\mu, \sigma^2 \mid y_1, \cdots, y_n) \propto &\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2\right) \\ &\times \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right) \\ &\times \frac{b_0^{a_0}}{\Gamma(a_0)} y^{-(a_0+1)} \exp\left(-\frac{b_0}{y}\right) \end{aligned}$$

- The Gibbs Sampler starts with selecting initial values for μ and σ.
- Then we sample a new value (denoted by $\mu^{(t)}$) from the full conditional probability distribution of μ given $\sigma^{(t-1)}$.
- Next we sample a new value (denoted by $\sigma^{(t)}$) conditional on $\mu^{(t)}$ obtained from the previous step.

- We repeat the procedure for an additional $$T iterations, alternating between drawing a new sample from the full conditional probability distribution of μ and the full conditional probability distribution of σ, given the current value of the other random variable.

Gibbs sampling algorithm proceeds as follows:

```
initialize $\mu^{(0)}$, $\sigma^{(0)}$
for $t = 1, 2, \cdots, T$
        $\mu^{(t)} \sim \mathrm{N}(\mathrm{mean}, \mathrm{var} \mid \sigma^{(\mathrm{t-1})})$
        $\sigma^{(t)} \sim \mathrm{InvG}(\mathrm{shape}, \mathrm{scale} \mid \mu^{(\mathrm{t})})$
end for
```

Exercise 3 (Metropolis-Hastings sampler for a logistic regression)

3.1

The aim of the Metropolis-Hastings algorithm is to generate a collection of states (here parameters $\theta = \{\alpha, \beta\}$) that describe our target distribution, which is the logistic regression $f(\theta \mid \mathbf{y}, \mathbf{n}, \mathbf{x})$. Via a Markov process, the state (the parameters) should converge to a stationary state which is the desired target distribution.

The algorithm starts with the so-called condition of detailed balance, which simply means that each transition, for example $\theta \to \theta'$, is reversible and happens with the same probability. More specifically, the probability of being in state θ and transitioning to state θ' is equal to the probability of being in state θ' and transitioning to state θ.

$$P(\theta' \mid \theta) \cdot f(\theta \mid \mathbf{y}, \mathbf{n}, \mathbf{x}) = P(\theta \mid \theta') \cdot f(\theta' \mid \mathbf{y}, \mathbf{n}, \mathbf{x})$$
$$\frac{P(\theta' \mid \theta)}{P(\theta \mid \theta')} = \frac{f(\theta' \mid \mathbf{y}, \mathbf{n}, \mathbf{x})}{f(\theta \mid \mathbf{y}, \mathbf{n}, \mathbf{x})}$$

- $P(\theta' \mid \theta)$: Transition probability from state θ to state θ'
- $f(\theta \mid \mathbf{y}, \mathbf{n}, \mathbf{x})$: the target distribution

For simplicity we focus here on one transition, namely $P(\theta' \mid \theta)$ but the same procedures also hold for the back-transition $P(\theta' \mid \theta)$. Now, the transition $P(\theta' \mid \theta)$ has to be split into two steps. The first one is to "propose" the next value (θ') and the second one is to accept this proposal or not. The proposal distribution $q(\theta' \mid \theta)$ is in our case a normal distribution, so $\theta' \sim \mathcal{N}(\theta, \sigma_\theta^2)$ and this means that the proposed value θ' is depending on the current value θ but the spread is defined by a tuning parameter σ_θ^2 which kind of defines the updating step. The acceptance distribution $A(\theta', \theta)$ does not have to be defined here as we will see. We can now rewrite:

$$\frac{A(\theta', \theta) \cdot q(\theta' \mid \theta)}{A(\theta, \theta') \cdot q(\theta \mid \theta')} = \frac{f(\theta' \mid \mathbf{y}, \mathbf{n}, \mathbf{x})}{f(\theta \mid \mathbf{y}, \mathbf{n}, \mathbf{x})}$$
$$\frac{A(\theta', \theta)}{A(\theta, \theta')} = \frac{f(\theta' \mid \mathbf{y}, \mathbf{n}, \mathbf{x}) \cdot q(\theta \mid \theta')}{f(\theta \mid \mathbf{y}, \mathbf{n}, \mathbf{x}) \cdot q(\theta' \mid \theta)}$$

- $A(\theta', \theta)$ is the probability to accept the proposed state θ' given the current state θ

In case of the Metropolis-Hastings algorithm the acceptance ratio $\frac{A(\theta', \theta)}{A(\theta, \theta')}$ can be written as A and it is forced to be lower or equal to 1 by definition. If we achieve an acceptance ratio larger than 1, which is theoretically possible, we will assume an acceptance ratio of 1, meaning that we will move from the current state θ to the proposed state θ' with a probability of 100%.

$$\begin{aligned}
A &= \min\left(1, \frac{f(\theta' \mid \mathbf{y}, \mathbf{n}, \mathbf{x}) \cdot q(\theta \mid \theta')}{f(\theta \mid \mathbf{y}, \mathbf{n}, \mathbf{x}) \cdot q(\theta' \mid \theta)}\right) \\
&= \min\left(1, \frac{f(\mathbf{y}, \mathbf{n}, \mathbf{x} \mid \theta') \cdot f(\theta') \cdot q(\theta \mid \theta')}{f(\mathbf{y}, \mathbf{n}, \mathbf{x} \mid \theta) \cdot f(\theta) \cdot q(\theta' \mid \theta)}\right) \\
&= \min\left(1, \frac{f(\mathbf{y}, \mathbf{n}, \mathbf{x} \mid \theta') \cdot f(\theta')}{f(\mathbf{y}, \mathbf{n}, \mathbf{x} \mid \theta) \cdot f(\theta)}\right)
\end{aligned}$$

- From line 1 to line 2, using Bayes rule for $f(\theta' \mid \mathbf{y}, \mathbf{n}, \mathbf{x})$ and $f(\theta \mid \mathbf{y}, \mathbf{n}, \mathbf{x})$
- From line 2 to line 3, using the fact that the proposal distribution (i.e. univariate normal distribution) is symmetric hence $q(\theta \mid \theta') = q(\theta' \mid \theta)$

Now, if we get a proposed value θ' that is more probable then the current value θ this means that the acceptance ratio A is higher than 1 and we should always accept this value. On the other hand, if we get a proposed value with an acceptance rate lower than 1 we have to add a stochastic component in the form of a random sample from the uniform distribution in order to determine if we accept the proposed value. If the acceptance ratio is higher than the random sample we still accept this point. If it is lower we reject and stick with the current value θ.

The parameters of interest in this task are the intercept (α) and the slope (β) of the logistical regression:

$$\text{logit}(p_i) = \ln\left(\frac{p_i}{1-p_i}\right) = \alpha + \beta x_i$$

- p_i is the estimated relative frequency of deaths, which can be obtained from the data ($p_i = \dfrac{y_i}{n_i}$)

Given two independent normal priors for α and β with mean = 0 and variance = 10000, we have:

$$\begin{aligned}
A(\alpha) &= \min\left(1, \frac{f(\mathbf{y}, \mathbf{n}, \mathbf{x} \mid \alpha') \cdot f(\alpha')}{f(\mathbf{y}, \mathbf{n}, \mathbf{x} \mid \alpha) \cdot f(\alpha)}\right) \\
&= \min\left(1, \frac{\text{Bin}(n, \frac{y}{n} \mid \alpha') \cdot \text{N}(\alpha' \mid 0, 100^2)}{\text{Bin}(n, \frac{y}{n} \mid \alpha) \cdot \text{N}(\alpha \mid 0, 100^2)}\right)
\end{aligned}$$

$$\begin{aligned}
A(\beta) &= \min\left(1, \frac{f(\mathbf{y}, \mathbf{n}, \mathbf{x} \mid \beta') \cdot f(\beta')}{f(\mathbf{y}, \mathbf{n}, \mathbf{x} \mid \beta) \cdot f(\beta)}\right) \\
&= \min\left(1, \frac{\text{Bin}(n, \frac{y}{n} \mid \beta') \cdot \text{N}(\beta' \mid 0, 100^2)}{\text{Bin}(n, \frac{y}{n} \mid \beta) \cdot \text{N}(\beta \mid 0, 100^2)}\right)
\end{aligned}$$

In the `R` code, we use the sum of log probabilities instead of the product of probabilities as the product of probabilities can be very small, which is numerically unstable in programming.

```r
# inverse logit: logit^(-1)(alpha + beta*x)
mypi <- function(alpha, beta, x){
  tmp <- exp(alpha + beta*x)
  pi <- tmp/(1+tmp)
  return(pi)
}

MH.sampler <- function(x,                # covariate values
                       y,                # number of mice deaths
                       n,                # total number of mice
                       sigma2 = 10^(4), # variance of normal priors
                       n.iter = 10000,  # number of MCMC iterations
                       n.burnin = 4000, # burnin length
                       n.thin = 1,      # thinning parameter
```

```r
                     alpha = 0,        # starting point
                     beta = 0,         # starting point
                     s_alpha = 1,      # SD for normal proposal
                     s_beta = 60       # SD for normal proposal
                     ) {

####################
## Bayesian analysis
####################

#####################################
## Step 1: R: (univariate proposal) Metropolis MCMC settings
#####################################

alpha_samples <- c()
beta_samples <- c()
# number of accepted proposals
alpha_yes <- 0
beta_yes <- 0

# counter
count <- 0

alpha_yes_history <- rep(0, n.burnin+n.iter*n.thin+1)
beta_yes_history <- rep(0, n.burnin+n.iter*n.thin+1)

cat(sprintf("%5s", c("i", "acc_rate_alpha", "acc_rate_beta\n")))

# start the MCMC algorithm (the first iteration after the burn-in is 1)
for(i in -n.burnin:(n.iter*n.thin)){
  count <- count + 1

  ## update alpha
  # generate a new proposal for alpha
  alpha_star <- rnorm(1, alpha, sd=s_alpha)

  # NOTE: it is more stable to calculate everything on the log scale
  enum <- sum(dbinom(y, size=n, prob=mypi(alpha_star, beta, x), log=TRUE)) +
    dnorm(alpha_star, mean=0, sd=sqrt(sigma2), log=TRUE)
  denom <- sum(dbinom(y, size=n, prob=mypi(alpha, beta, x), log=TRUE))  +
    dnorm(alpha, mean=0, sd=sqrt(sigma2), log=TRUE)

  # log accetpance rate (since we use a random walk proposal there is no
  #   proposal ratio in the acceptance probability)
  logacc <- enum - denom
  if(log(runif(1)) <= logacc){
    # accept the proposed value
    alpha <- alpha_star
    alpha_yes <- alpha_yes + 1
    alpha_yes_history[count] <- 1
  }

  ## update beta
  # generate a new proposal for beta
  beta_star <- rnorm(1, beta, sd=s_beta)


  enum <- sum(dbinom(y, size=n, prob=mypi(alpha, beta_star, x), log=TRUE)) +
      dnorm(beta_star, mean=0, sd=sqrt(sigma2), log=TRUE)
    denom<- sum(dbinom(y, size=n, prob=mypi(alpha, beta, x), log=TRUE)) +
      dnorm(beta, mean=0, sd=sqrt(sigma2), log=TRUE)
    # log accetpance rate
    logacc <- enum - denom

    if(log(runif(1)) <= logacc){
      # accept the proposed value
      beta <- beta_star
      beta_yes <- beta_yes + 1
      beta_yes_history[count] <- 1
    }

    # after the burnin save every kth sample
    if((i > 0) && (i%%n.thin == 0)){
      alpha_samples <- c(alpha_samples, alpha)
      beta_samples <- c(beta_samples, beta)
    }


    if(i%%1000 ==0){
      # print the acceptance rates on the fly
      # cat(c(i, alpha_yes/count, beta_yes/count), "\n")
      cat(sprintf("%5.0f\t%1.6f\t%1.6f\n",
                  i, alpha_yes/count, beta_yes/count))
    }
  }
  # output:
  output <- list("alpha_samples"=alpha_samples, "beta_samples"=beta_samples,
                 "alpha_yes"=alpha_yes, "beta_yes"=beta_yes,
                 "alpha_yes_history"= alpha_yes_history,
                 "beta_yes_history"= beta_yes_history)
return(output)
}
```

```
x_original <- c(0.0028, 0.0028, 0.0056, 0.0112, 0.0225, 0.0450)
# the centered covariate values (centered dose) from the Mice data from Collett
x <- x_original - mean(x_original)
# number of mice deaths
# y <- c(35, 21, 9, 6, 1)
y <- c(26, 9, 21, 9, 6, 1)
# total number of mice
# n <- c(40, 40, 40, 40, 40)
n <- c(28, 12, 40, 40, 40, 40)
```

```
d.mice <- data.frame(
  x_original, y, n, x, y/n
)
colnames(d.mice) <- c("$x$", "$y$", "$n$", "centered $x$", "$p$")
knitr::kable(d.mice, align="c", caption="Mice data from Collett (2003)")
```

Table 2: Mice data from Collett (2003)

$x$	$y$	$n$	centered $x$	$p$
0.0028	26	28	-0.0121833	0.9285714
0.0028	9	12	-0.0121833	0.7500000

x	y	n	centered x	p
0.0056	21	40	-0.0093833	0.5250000
0.0112	9	40	-0.0037833	0.2250000
0.0225	6	40	0.0075167	0.1500000
0.0450	1	40	0.0300167	0.0250000

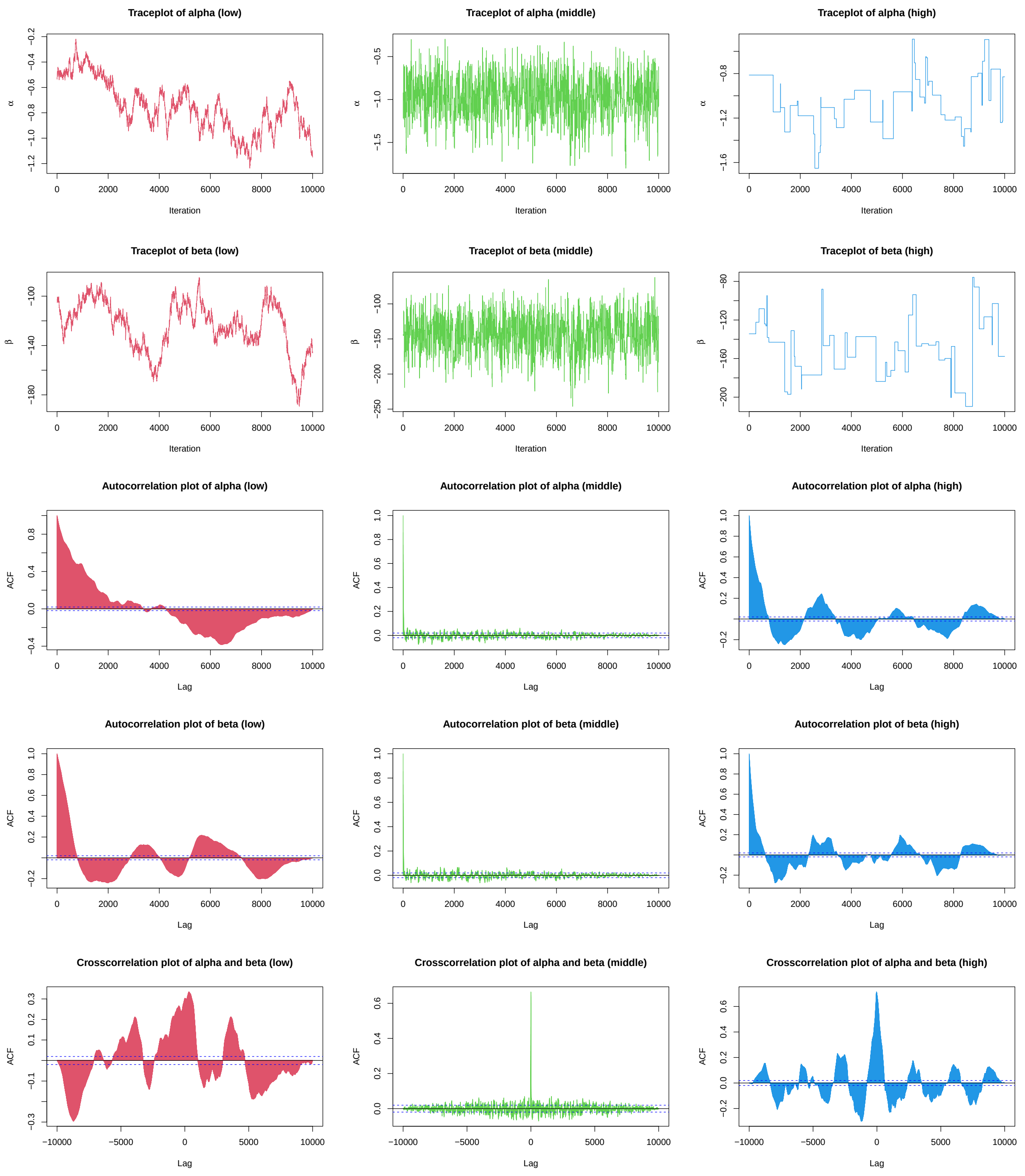
```r
low <- c(0.01, 1)
middle <- c(1, 100)
high <- c(50, 5000)

set.seed(44566)
cat("\n-------Low values (0.01, 1)-------\n")
low.sampler <- MH.sampler(x=x, y=y, n=n, s_alpha=low[1], s_beta=low[2])

cat("\n-------Middle values (1, 100)-------\n")
middle.sampler <- MH.sampler(x=x, y=y, n=n, s_alpha=middle[1], s_beta=middle[2])

cat("\n-------High values (50, 5000)-------\n")
high.sampler <- MH.sampler(x=x, y=y, n=n, s_alpha=high[1], s_beta=high[2])
```

```
##
## -------Low values (0.01, 1)-------
##      i acc_rate_alpha acc_rate_beta
## -4000    1.000000    0.000000
## -3000    0.944056    0.909091
## -2000    0.966017    0.935532
## -1000    0.972343    0.951016
##     0    0.974006    0.956261
##  1000    0.969606    0.963807
##  2000    0.969172    0.967172
##  3000    0.971433    0.970576
##  4000    0.971504    0.970379
##  5000    0.972670    0.970670
##  6000    0.974003    0.972003
##  7000    0.975275    0.974093
##  8000    0.975335    0.975002
##  9000    0.976002    0.974617
## 10000    0.976216    0.974359
##
## -------Middle values (1, 100)-------
##      i acc_rate_alpha acc_rate_beta
## -4000    0.000000    0.000000
## -3000    0.229770    0.232767
## -2000    0.227886    0.235882
## -1000    0.217261    0.237254
##     0    0.212447    0.238690
##  1000    0.216757    0.234353
##  2000    0.213298    0.228795
##  3000    0.214112    0.228682
##  4000    0.214473    0.229471
##  5000    0.213754    0.229419
##  6000    0.214179    0.230377
##  7000    0.216344    0.230524
##  8000    0.214732    0.230147
##  9000    0.213676    0.229367
## 10000    0.213413    0.230269
```

```
##
## -------High values (50, 5000)-------
##      i acc_rate_alpha acc_rate_beta
## -4000    0.000000    0.000000
## -3000    0.003996    0.004995
## -2000    0.004498    0.005497
## -1000    0.004332    0.005332
##     0    0.003999    0.004249
##  1000    0.003399    0.004799
##  2000    0.003833    0.004833
##  3000    0.004142    0.004714
##  4000    0.004000    0.004624
##  5000    0.003777    0.004333
##  6000    0.003700    0.004400
##  7000    0.004272    0.004454
##  8000    0.004250    0.004583
##  9000    0.004461    0.004538
## 10000    0.004714    0.004571
```

3.2

```
alpha_low <- low.sampler$alpha_samples
beta_low <- low.sampler$beta_samples

alpha_middle <- middle.sampler$alpha_samples
beta_middle <- middle.sampler$beta_samples

alpha_high <- high.sampler$alpha_samples
beta_high <- high.sampler$beta_samples

## Traceplots of alpha under different tuning parameters
plot(x=1:length(alpha_low), y=alpha_low, type="l", col=2,
     xlab="Iteration", ylab=expression(alpha), main="Traceplot of alpha (low)")
plot(x=1:length(alpha_middle), y=alpha_middle, type="l", col=3,
     xlab="Iteration", ylab=expression(alpha), main="Traceplot of alpha (middle)")
plot(x=1:length(alpha_high), y=alpha_high, type="l", col=4,
     xlab="Iteration", ylab=expression(alpha), main="Traceplot of alpha (high)")

## Traceplots of beta under different tuning parameters
plot(x=1:length(beta_low), y=beta_low, type="l", col=2,
     xlab="Iteration", ylab=expression(beta), main="Traceplot of beta (low)")
plot(x=1:length(beta_middle), y=beta_middle, type="l", col=3,
     xlab="Iteration", ylab=expression(beta), main="Traceplot of beta (middle)")
plot(x=1:length(beta_high), y=beta_high, type="l", col=4,
     xlab="Iteration", ylab=expression(beta), main="Traceplot of beta (high)")

## ACF plots of alpha under different tuning parameters
acf(alpha_low, col=2, main="Autocorrelation plot of alpha (low)", lag.max=10000)
acf(alpha_middle, col=3, main="Autocorrelation plot of alpha (middle)", lag.max=10000)
acf(alpha_high, col=4, main="Autocorrelation plot of alpha (high)", lag.max=10000)

## ACF plots of beta under different tuning parameters
acf(beta_low, col=2, main="Autocorrelation plot of beta (low)", lag.max=10000)
acf(beta_middle, col=3, main="Autocorrelation plot of beta (middle)", lag.max=10000)
acf(beta_high, col=4, main="Autocorrelation plot of beta (high)", lag.max=10000)

## CCF plots under different tuning parameters
ccf(alpha_low, beta_low, col=2, lag.max=10000,
    main="Crosscorrelation plot of alpha and beta (low)")
ccf(alpha_middle, beta_middle, col=3, lag.max=10000,
    main="Crosscorrelation plot of alpha and beta (middle)")
ccf(alpha_high, beta_high, col=4, lag.max=10000,
    main="Crosscorrelation plot of alpha and beta (high)")
```



With high tuning parameters s_alpha and s_beta which are in form of the standard deviations of the proposal distribution $\alpha' \sim \mathcal{N}(\alpha, \sigma_\alpha^2)$ and $\beta' \sim \mathcal{N}(\beta, \sigma_\beta^2)$ the proposed value gets easier or harder accepted.

- If the standard deviations of α and β are too low, the acceptance rate is high but each updating step only converges rather slowly which means the updated value is highly similar to the value before.

  - The traceplots show that the chain explores pretty much locally rather than globally so it takes so much time and many iterations for the chain to explore the full target distribution.
  - The ACF and CCF plots show that autocorrelation is very high at larger lags and it dies even more slowly. This is because for high values of parameters the samples accepted are still somewhat different while for low values of parameters the samples accepted are highly correlated.

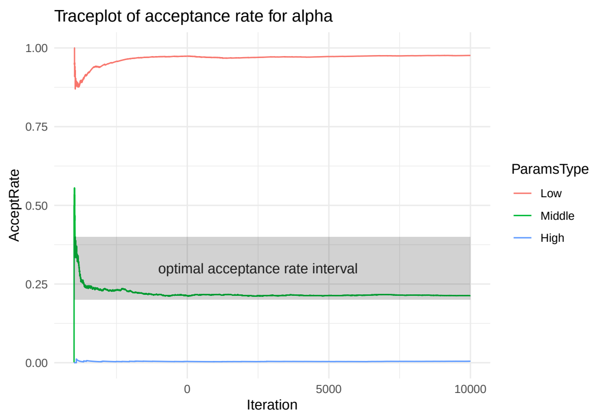
- If the standard deviations of α and β are somewhere in the middle, the acceptance rates for α and β are around 21% and 23% respectively, which is acceptable according to the "rule of thumb".
  - The traceplots show that the chain seems to explore the full target distribution as there is no apparent anomalies.
  - The ACF and CCF plots show that autocorrelation is large at short lags but it drops dramatically after several iterations.

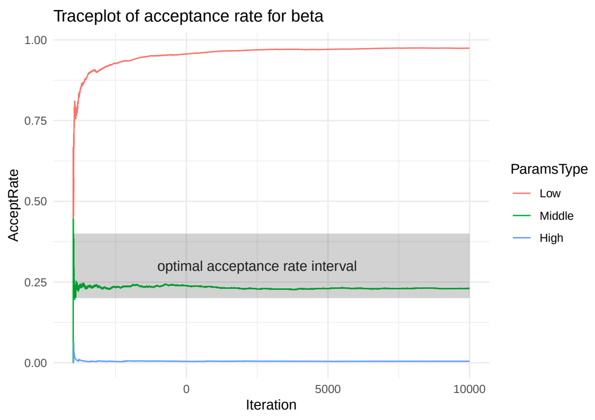
- If the standard deviations of α and β are too high, the proposed value is hardly accepted (low acceptance rate), which leads to inefficient sampling.
  - The traceplots show that many plateaus are reached or too many consecutive steps in one direction, which indicates the proposed sample is rejected for many iterations and the chain stays in the same state for too long.
  - The ACF and CCF plots show that autocorrelation is very high even at large lags and it also dies out very slowly, which also indicates a very inefficient sampling compared to the i.i.d sampling.

3.3

```r
d.alpha <- data.frame(
  Iteration = rep(-4000:10000, times=3),
  AcceptRate = c(
    cumsum(low.sampler$alpha_yes_history) / seq_along(low.sampler$alpha_yes_history),
    cumsum(middle.sampler$alpha_yes_history) / seq_along(middle.sampler$alpha_yes_history),
    cumsum(high.sampler$alpha_yes_history) / seq_along(high.sampler$alpha_yes_history)
  ),
  ParamsType = factor(rep(c("Low", "Middle", "High"), each=14001),
                      levels=c("Low", "Middle", "High"))
)


d.beta <- data.frame(
  Iteration = rep(-4000:10000, times=3),
  AcceptRate = c(
    cumsum(low.sampler$beta_yes_history) / seq_along(low.sampler$beta_yes_history),
    cumsum(middle.sampler$beta_yes_history) / seq_along(middle.sampler$beta_yes_history),
    cumsum(high.sampler$beta_yes_history) / seq_along(high.sampler$beta_yes_history)
  ),
  ParamsType = factor(rep(c("Low", "Middle", "High"), each=14001),
                      levels=c("Low", "Middle", "High"))
)
```

```r
ggplot(d.alpha, aes(x=Iteration, y=AcceptRate)) +
  geom_line(aes(color=ParamsType)) +
  geom_ribbon(aes(ymin=0.2, ymax=0.4), fill="grey12", alpha=0.2) +
  geom_text(aes(x=2500, y=0.3, label="optimal acceptance rate interval"),
            color="grey12", check_overlap=TRUE) +
  labs(title="Traceplot of acceptance rate for alpha", ylab="Acceptance Rate") +
  theme_minimal()

ggplot(d.beta, aes(x=Iteration, y=AcceptRate)) +
  geom_line(aes(color=ParamsType)) +
  geom_ribbon(aes(ymin=0.2, ymax=0.4), fill="grey12", alpha=0.2) +
  geom_text(aes(x=2500, y=0.3, label="optimal acceptance rate interval"),
            color="grey12", check_overlap=TRUE) +
  labs(title="Traceplot of acceptance rate for beta", ylab="Acceptance Rate") +
  theme_minimal()
```



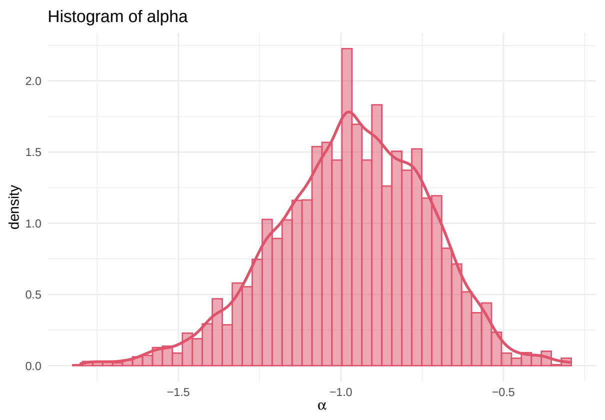


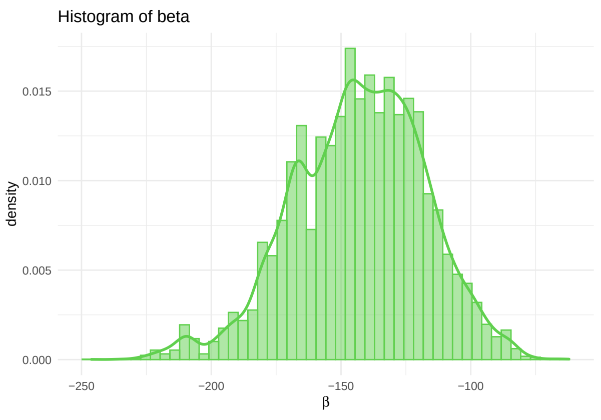
When the tuning parameters of the proposals are set to middle values ($s\_alpha = 1$, $s\_beta = 100$), the optimal acceptance rate is attained. From the traceplots of acceptance rates for α and β, we can also see that only acceptance rate under middle choice condition stabilizes within the “optimal acceptance rate interval” suggested by the “rule of thumb”.

3.4

We can visualize the marginal posteriors for α and β for the middle choice of tuning parameters by plotting the histograms and density curves:

```
d.MH <- data.frame(
  alpha = alpha_middle,
  beta  = beta_middle
)

## Posterior distribution for alpha for the middle choice of tuning parameters
ggplot(d.MH, aes(x=alpha, y=..density..)) +
  geom_histogram(color=2, fill=2, alpha=0.5, bins=50) +
  geom_density(color=2, lwd=1) +
  labs(title="Histogram of alpha", x=expression(alpha)) +
  theme_minimal()

## Posterior distribution for beta for the middle choice of tuning parameters
ggplot(d.MH, aes(x=beta, y=..density..)) +
  geom_histogram(color=3, fill=3, alpha=0.5, bins=50) +
  geom_density(color=3, lwd=1) +
  labs(title="Histogram of beta", x=expression(beta)) +
  theme_minimal()
```





The following table illustrates the summary statistics of the marginal posteriors for α and β for the middle choice of tuning parameters ($\alpha = 1$, $\beta = 100$).

```r
d.summary <- t(
  rbind(
    colMeans(d.MH),
    apply(d.MH, 2, function(x) sd(x)),
    apply(d.MH, 2, function(x) quantile(x, probs=c(0.025, 0.5, 0.975)))
  )
)

d.summary <- data.frame(d.summary)
colnames(d.summary) <- c("Mean", "SD", "2.5%", "Median", "97.5%")

knitr::kable(d.summary, align="c",
             caption="Summary statistics of marginal posteriors (middle choice)")
```

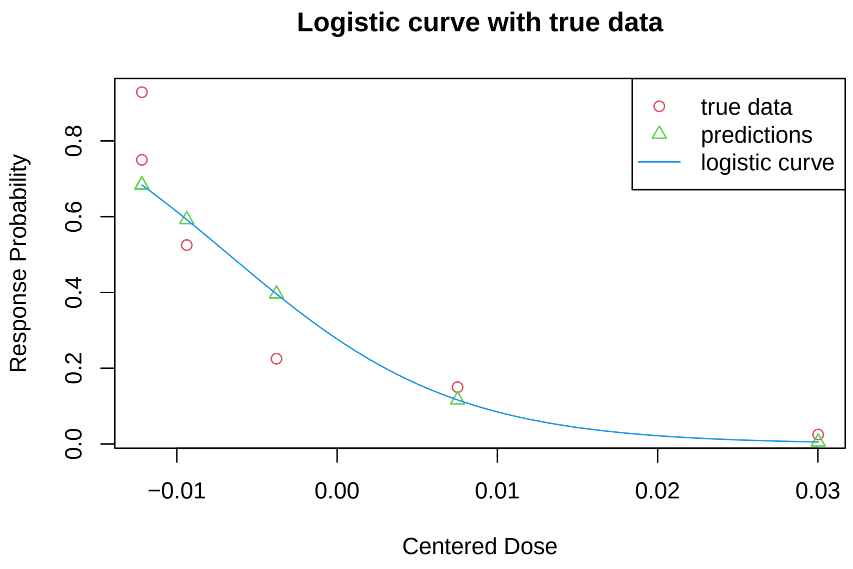
Table 3: Summary statistics of marginal posteriors (middle choice)

	Mean	SD	2.5%	Median	97.5%
alpha	-0.9673889	0.2352573	-1.461707	-0.9592253	-0.5537754
beta	-143.0714300	25.3517136	-196.708001	-142.0147341	-96.3710245

3.5

```r
med.alpha <- d.summary$Median[1]
med.beta <- d.summary$Median[2]

# inverse logit: logit^(-1)(alpha + beta*x)
mypi <- function(alpha, beta, x){
  tmp <- exp(alpha + beta*x)
  pi <- tmp/(1+tmp)
  return(pi)
}

x.grid <- seq(min(x), max(x), length.out=100)
y.pred <- mypi(alpha=med.alpha, beta=med.beta, x=x.grid)

plot(x=x, y=y/n, col=2, xlab="Centered Dose", ylab="Response Probability",
     main="Logistic curve with true data")
points(x=x, y=mypi(alpha=med.alpha, beta=med.beta, x=x), col=3, pch=2)
lines(y.pred ~ x.grid, col=4)
```

```
legend("topright", legend=c("true data", "predictions", "logistic curve"),
       col=2:4, lty=c(NA, NA, 1), pch=c(1, 2, NA))
```


Logistic curve with true data
true data
predictions
logistic curve
Response Probability
Centered Dose