Markov chain Monte Carlo for Bayesian inference of Duchene Muscular Dystrophy disease using BUGS and NIMBLE

Van Thuan Romoli

Semester 2, 2024/2025

1 Introduction

Real statistical problems are typically characterised by a high number of unknown parameters to be estimated, resulting in high dimensional posterior distributions of complex and often of non-standard form. This was essentially the reason why the classical approach dominated statistics until the 1990s. Bayesian statistics were fine in theory, but intractable in practice. However, the increase in computational power, coupled with computational algorithms introduced to the statistical literature has made Bayesian analyses feasible (and often easier than alternative classical approaches!).

In this project we use the computer language BUGS (Bayesian inference Using Gibbs Sampling). BUGS is implemented in the free package `Nimble`, a `R` statistical package that allows users to fit complex statistical models using the BUGS language.

This project has three aims: (1) explore the theory that allows high dimensional, complex, and intractable posteriors to be calculated; (2) give an introduction to the BUGS language; and (3) carry out Bayesian inference of Duchene Muscular Dystrophy disease using BUGS.

This project has 5 sections and 1 appendix. In Section 2 we explore the background theory of Monte Carlo integration and Markov Chain Monte Carlo. In Section 3 we give a quick overview of the BUGS language and NIMBLE. Then, in Section 4 we show how to use the BUGS langauge to carry out Bayesian inference of Duchene Muscular Dystrophy disease. Finally, in Section 5 we give the conclusions and final remarks. At last, Appendix A is given to provide the details on the algorithms that BUGS uses to generate Markov chains.

2 Background theory

2.1 Monte Carlo integration

In many circumstances, we are faced with the problem of evaluating an integral that is too complex to calculate explicitly. With particular reference to Bayesian inference, posterior distributions are typically summarised using statistics such as the mean and/or variance. Such posterior summary statistics require integration of the posterior density (which is often analytically intractable, particularly for high-dimensional posterior distributions). For example, we may wish to estimate the posterior (marginal) expectation of a parameter θ, given observed data $\boldsymbol{x}$:

$$\mathbb{E}_\pi(\theta) = \int \theta \; \pi(\theta|\boldsymbol{x}) d\theta.$$

We can use the simulation technique of Monte Carlo integration to obtain an estimate of a given integral (and hence posterior expected value). The method is based upon drawing observations from the distribution of the variable of interest and simply calculating the empirical estimate of the expectation. For example, given a sample of observations, $\theta^1, \dots, \theta^n \sim \pi(\theta|\boldsymbol{x})$, we can estimate expectation by,

$$\frac{1}{n}\sum_{i=1}^{n} \theta^i.$$

For independent samples, the Law of Large Numbers ensures that

$$\frac{1}{n}\sum_{i=1}^{n}\theta^i \to \mathbb{E}_\pi(\theta) \quad \text{as } n \to \infty.$$

Similarly, we estimate the posterior variance, $\text{Var}_\pi(\theta)$, by the sample variance of observations taken from the posterior distribution, i.e.

$$\frac{1}{n-1}\left[\sum_{i=1}^{n}(\theta^i)^2 - \frac{1}{n}\left(\sum_{i=1}^{n}\theta^i\right)^2\right].$$

Independent sampling from $\pi(\theta|\boldsymbol{x})$ may be difficult, however this result still holds if we generate our samples, not independently, but via some other method (although this may be less effective than independently drawn samples in that larger sample sizes are needed to obtain the same level of accuracy).

In other words, we estimate the posterior mean by the sample mean of observations taken from the posterior distribution. This is *Monte Carlo integration.* The idea extends directly to any function of θ, denoted by $f(\theta)$. For example, suppose that we wish to calculate the posterior mean of $f(\theta)$. Given a sample of observations, $\theta^1, \dots, \theta^n \sim \pi(\theta|\boldsymbol{x})$, we can estimate the posterior mean of $f(\theta)$ by

$$\frac{1}{n}\sum_{i=1}^{n} f(\theta^i).$$

2.2 Markov chain Monte Carlo

A Markov chain is simply a stochastic sequence of numbers where each value in the sequence depends only upon the last. More formally, a Markov chain is a stochastic process $\{\theta^i : i = 0, 1, 2, ...\}$ with discrete state space S such that

$$\mathbb{P}(\theta^{i+1} = k_{i+1}|\theta^i = k_i, ..., \theta^0 = k_0) = \mathbb{P}(\theta^{i+1} = k_{i+1}|\theta^i = k_i)$$

for all $i = 0, 1, 2, ...$ and all $k_0, ..., k_{i+1} \in S$. Each element $s \in S$ is called a state of the Markov chain.

Given an arbitrary θ^0, we can simulate a Markov chain by generating a new state of the chain, say θ^{i+1}, from some distribution, dependent only on θ^i:

$$\theta^{i+1} \sim \mathcal{P}(\theta^i, \theta^{i+1}) \quad \big(\equiv \mathcal{P}(\theta^{i+1}|\theta^i)\big)$$

where $\mathcal{P}$ is the density of such distribution and is called the transition kernel for the chain. The transition kernel (or density) uniquely describes the dynamics of the chain.

Under certain conditions (that the chain is aperiodic and irreducible) the distribution over the states of the Markov chain will converge to a stationary distribution (in this project we shall always assume that these conditions are met). The stationary distribution is independent of the initial starting values specified for the chains. Our aim is to construct a Markov chain such that the stationary distribution is equal to the posterior distribution $\pi(\boldsymbol{\theta}|\boldsymbol{x})$. If we can do this (which we obviously can or this project wouldn't exist!), we can run the Markov chain until the stationary distribution is reached, so that realisations of the chain can be regarded as a dependent sample from the posterior distribution of interest (thus note that we need to discard the first portion of a Markov chain). We are then able to use this sample from the latter part of the chain, after it has converged, to obtain Monte Carlo estimates of the parameters of interest and/or plot their corresponding density function (see Section 2.2).

Since we are combining a Markov chain with Monte Carlo integration this method is called Markov chain Monte Carlo (MCMC). The beauty of MCMC is that the updating of the states in the Markov chain remains relatively simple, using standard techniques, irrespective of the complexity of the posterior distribution.

Clearly, several questions immediately arise for such a technique:

1. How do we construct such a Markov chain?

2. Even if we can construct such a Markov chain with the correct stationary distribution, how long do we need to run the chain until the stationary distribution of interest has been reached?
3. How many samples do we need from the posterior distribution so that we accurately estimate the quantities of interest?

The answer to the first question is beyond the scope of this project however, I am very interested in this topic and I discuss the algorithms necessary to construct the Markov chain in Appendix A. In practice, we let computers aid us, as shown in Section 3. Section 2.3 is dedicated to answering the last two questions.

2.3 Run lengths and Monte Carlo errors

There are two issues to be considered when determining the number of iterations of the Markov chain (i.e. how many values to simulate): (1) the time required for the Markov chain to reach the stationary distribution (i.e. for the chain to converge), and (2) the post-convergence sample size required for suitably small Monte Carlo errors.

2.3.1 Burn-in and Brooks-Gelman-Rubin method

We need to discard the realisations of the Markov chain before the chain has converged to the stationary distribution. The initial observations that we discard are referred to as the burn-in. The simplest method to determine the length of the burn-in period is to look at trace plots. These are simply the value of the parameter at each iteration of the Markov chain. It is often possible to see the individual parameters converging from their starting position to values based around a constant mean (i.e. the mean of the posterior distribution). For example, consider Figure 1, clearly the earliest values of the Markov chain do not look like the later values, these early values are dependent on the starting value, and hence would be discarded as burn-in. By eye we might suggest a suitable burn-in of around 200 iterations. Note it is always best to be conservative with regard to the burn-in and err on the side of overestimation to ensure that convergence has been achieved when obtaining the sample to be used to form Monte Carlo estimates of the parameters of interest.

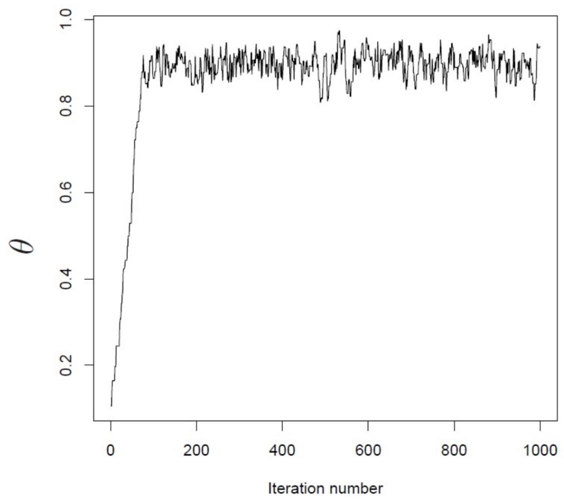

Figure 1: A single MCMC trace plot.

This use of a trace plot is often a fairly efficient method, but it is not robust. For example, an ad hoc interpretation of the first trace plot in Figure 2 might suggest that the chain had converged after around 500 iterations. However, when the chain is run for longer, it is clear that the chain has not converged within these first 500 iterations.

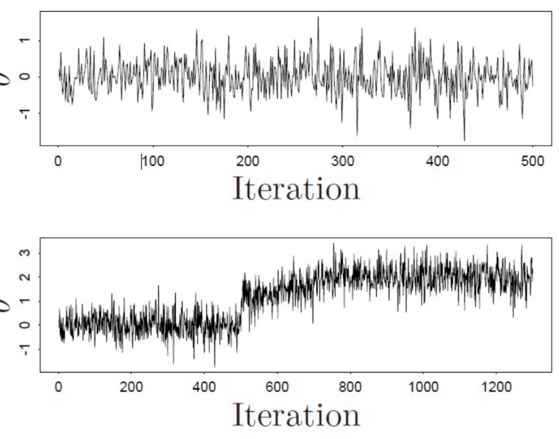

Figure 2: MCMC sample paths.

To solve this, we use a technique referred to as the "thick-pen" technique, which involves running two (or more chains) started at very different starting values and plotting the output on a single graph. A "thick pen" is then taken and run over either of the trace plots from one of the simulated Markov chains. When the pen touches both lines of the plot it could be concluded that the chains had converged. For example, consider Figure 3 where we run two chains and plot the values from each Markov chain (i.e. trace plot) on the same axes. Using this technique we might once again suggest a burn-in of at least 200.

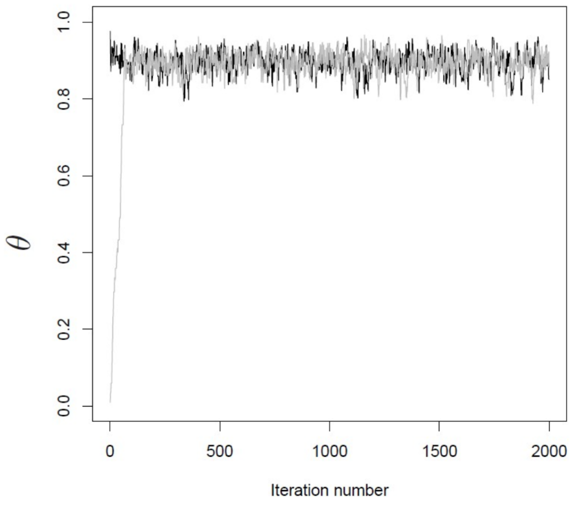

Figure 3: Two MCMC trace plots from independent chains starting at different values.

This idea motivated many of the more formal (and mathematical) techniques for assessing convergence to the stationary distribution via the assessment of multiple replications starting from overdispersed starting points. Essentially, this means running the Markov chains several times (from different starting points) and checking that given a suitable burn-in period the posterior estimates of all of the chains are essentially the same, providing evidence that no major nodes have been missed.

The most common approach is the Brooks-Gelman-Rubin (BGR) method. There are various implementations of this diagnostic procedure, all based upon the idea of using an analysis of variance technique to check whether there are any differences in the posterior estimates obtained from the different replications. In order to implement this procedure at least two chains need to be simulated. The simplest implementation for a chain containing $2n$ iterations is to discard the first n iterations and take the ratio of the width of the empirical 80% credible interval obtained from all chains combined after the burn-in, with the corresponding mean within-chain 80% interval width, i.e., set

$$\hat{R} = \frac{\text{width of 80\% credible interval of pooled chains}}{\text{mean of width of 80\% credible interval of individual chains}}.$$

Convergence is assumed when these are roughly equal, implying that all chains have roughly equal variability, so that $\hat{R} \approx 1$. The R value is plotted in Figure 4 for the two chains presented in Figure 3 for increasing value of n. Looking at this plot (and being conservative) we might suggest that convergence is achieved by around iteration 1000.

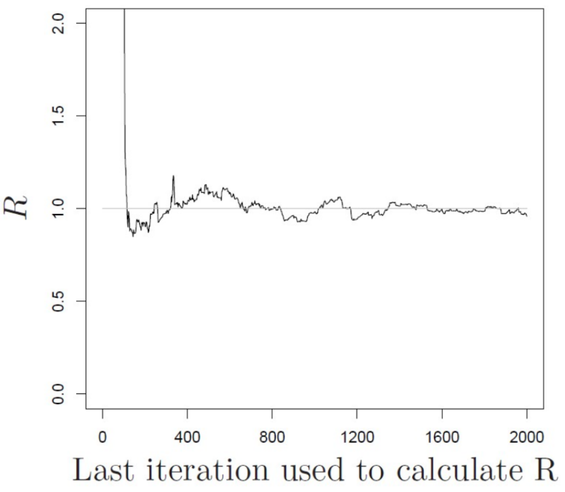

Figure 4: BGR statistic for the trace plots provided in Figure 3.

Note however that no convergence diagnostic can prove that the chain has converged, they can only identify when the chain has not converged. A common issue would be, for example, identifying the bi-modality in high dimensional spaces.

2.3.2 Monte Carlo error

To consider the issue of the number of iterations we need following the burn-in to obtain an accurate estimate of the summary statistics, we leverage a concept called the Monte Carlo error. Consider the sample mean $\bar{\theta} = \frac{1}{n}\sum_{i=1}^{n}\theta^i$ of a parameter θ. Then it satisfies

$$\bar{\theta} \sim N\left(\mathbb{E}_\pi(\theta), \frac{\sigma^2}{n}\right)$$

where $\sigma/\sqrt{n}$ is the Monte Carlo error.

It turns out that an estimate for the Monte Carlo error is

$$\sqrt{\frac{\hat{\sigma}^2}{n}}.$$

where $\hat{\sigma}^2$ is the sample variance, and for independent samples

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\theta_i - \bar{\theta})^2. \tag{1}$$

Since, in general, we don't have independent samples, we assess the dependence using trace plots, the autocorrelation function (AFC), and the effective sample size (ESS). The details are beyond the scope, but Table 1 is provided to summarise this.

	Trace plot	**ACF**	**ESS**
High correlation	Visible trends	High for all values	Smaller
Low correlation	Overlapping lines, random behaviour	Zero for $h \neq 1$	Larger

Table 1: Summary of observed behaviour of high and low correlation in trace plots, ACF, and ESS.

Should we observe dependent samples, there are several methods to correct (1) to account for dependence, such as batching or using the effect sample size. The details of this are also beyond the purpose of this project and, in practice, computers will help us.

As a rule of thumb, the sample size should be such that

$$\frac{\text{MC error (adjusted)}}{\sqrt{\text{Var}_\pi(\theta)}} < 0.05$$

is satisfied. In other words, the Monte Carlo error should be small compared to the posterior distribution standard deviation.

3 The BUGS language and NIMBLE

The BUGS language (Bayesian inference Using Gibbs Sampling) is a specialized modelling language created for describing complex Bayesian statistical models. It provides a way for statisticians and data scientists to formally specify probabilistic models using a notation that looks similar to the `S` programming language but is rooted in the structure of graphical models. In BUGS, relationships between variables are defined either stochastically (using to represent probability distributions) or deterministically (using `<-` to represent fixed functional relationships). This approach allows users to write down the joint probability distribution of a model in a clear and modular way. The language itself is not tied to a single software package, different implementations such as WinBUGS, OpenBUGS, JAGS, and MultiBUGS can interpret and run BUGS code to perform Markov Chain Monte Carlo (MCMC) sampling for Bayesian inference. Because it emphasizes a declarative style, BUGS lets users focus on describing "what the model is" rather than the low-level details of "how the inference is carried out."

NIMBLE builds directly on the BUGS language but extends it with greater flexibility and computational efficiency. It is implemented as an `R` package and allows users to continue writing models in familiar BUGS syntax, while also providing a system for customizing and programming algorithms. With NIMBLE, one can write `nimbleFunctions`, pieces of code that define algorithms such as MCMC samplers, particle filters, or optimization routines-and then apply them to BUGS-style models. Behind the scenes, NIMBLE compiles both models and algorithms into optimized C++ code, ensuring that the computations run efficiently while the user still works in the `R` environment. This combination of declarative modelling and customizable algorithm programming makes NIMBLE a bridge between model specification and statistical computing research, enabling applied analysts to use standard methods easily while also allowing advanced users to experiment with new inference strategies.

4 Bayesian inference of Duchene Muscular Dystrophy disease

Duchene Muscular Dystrophy (DMD) is a sex-linked genetic disease. Boys with the disease usually die at a young age, while affected girls usually do not suffer symptoms and may unknowingly carry the disease and pass it to their offspring. It is important to identify which variables can help discern whether or not a woman is a carrier of the disease.

4.1 DMD dataset

The data used in this project comes from a toy dataset that records the number of carriers from 15 hospitals where genetic studies on the prevalence of the DMD-causing allele were conducted. In each study, genetic tests were administered to a number of young female patients enrolled in an HPV screening (`ni`) and the resulting number of carriers recorded (`ncarriers`). The following information is also available from each hospital: `X_hp`: proportion of females enrolled in the HPV screening with a diagnosed cardiovascular disease; and `X_hi`: proportion of Hispanic females enrolled in the HPV screening.

4.2 Bayesian model

We model the number of carriers from each hospital i using a Binomial distribution with parameters n_i (known) and π_i (unknown). The prior distribution for π_i depends on the following relation:

$$\mathrm{logit}(\pi_i) = \beta_0 + \beta_{hp} Z_{hp_i} + \beta_{hi} Z_{hi_i}$$

where Z_{hp_i} and Z_{hi_i} are the standardised covariates `X_hp` and `X_hi`, respectively. We choose a Normal prior with mean 0 and standard deviation 5 for all beta parameters.

The first step is to specify the model.

```
# Model specification
DMDCode1 <- nimbleCode({

  # Specify the likelihood:
  for (i in 1:N){
    Y[i] ~ dbin(pi[i],ni[i])

    pi[i] <- expit(beta_0 +
                     beta_hp * ( ( X_hp[i]-mean(X_hp[]) ) / sd(X_hp[]) ) +
                     beta_hi * ( ( X_hi[i]-mean(X_hi[]) ) / sd(X_hi[]) ))
  }
  # Prior specification: (N(0,tau) where tau = 1/sigma^2)
  beta_0 ~ dnorm(0,1/25)
  beta_hp ~ dnorm(0,1/25)
  beta_hi ~ dnorm(0,1/25)

})
```

Now we write code to construct four Markov chains with different starting values, listed in the variable `DMDInits` (this automates the process described in Appendix A).

```
# Constant values in the model
DMDConsts <- list(N=15)

# Data values
DMDData <- list(ni = DMDdata$ni,
                X_hp = DMDdata$X_hp,
                X_hi = DMDdata$X_hi,
                Y = DMDdata$ncarriers)

# Initial values before building the model
DMDInits <- list(beta_0 = 1, beta_hp = 1, beta_hi = 1)

# To build the model
DMDmod <- nimbleModel(code = DMDCode1, name = "DMDmod", constants = DMDConsts,
                      data = DMDData, inits = DMDInits)
# To compile the model
CDMDmod <- compileNimble(DMDmod)


# Set up the monitored quantities
DMDmodConfig <- configureMCMC(DMDmod,enableWAIC = TRUE, monitors = c('beta_0','beta_hp
    ','beta_hi'), print = TRUE)
# Build the MCMC algorithm
DMDmodMCMC <- buildMCMC(DMDmodConfig)
# Compile the MCMC chain
CDMDmodMCMC <- compileNimble(DMDmodMCMC, project = DMDmod)

set.seed(10) # for replicability

DMDInits <- list(list(beta_0 = 0, beta_hp = 0, beta_hi = 0),
                 list(beta_0 = 7, beta_hp = 7, beta_hi = 7),
                 list(beta_0 = 1, beta_hp = 1, beta_hi = 1),
                    list(beta_0 = -10, beta_hp = -10, beta_hi = -10))

posterior <- runMCMC(CDMDmodMCMC, niter = 10000, thin=1, nburnin=1,
                         summary = TRUE, samples = TRUE, nchains=4,
                         samplesAsCodaMCMC=TRUE, inits = DMDInits)

combinedchains <- mcmc.list(posterior$samples$chain1,
                                 posterior$samples$chain2,
                                 posterior$samples$chain3,
                                 posterior$samples$chain4)
```

4.3 Model diagnostic

As discussed in Section 2.3, we want to evaluate how long it takes for the Markov chain to converge to its stationary distribution and we want to ensure that we capture any multimodality. We decide to run four chains with diffused starting points (listed in the variable `DMDInits`) to minimise the risks of local convergence and to clearly identify when the chain converges. We plot the trace plots (Figure 5) and the Brooks-Gelman-Rubin (BGR) plots (Figure 6) to assess visually.

```
# Checks for convergence and burn-in
plot(combinedchains)
gelman.plot(combinedchains)
```

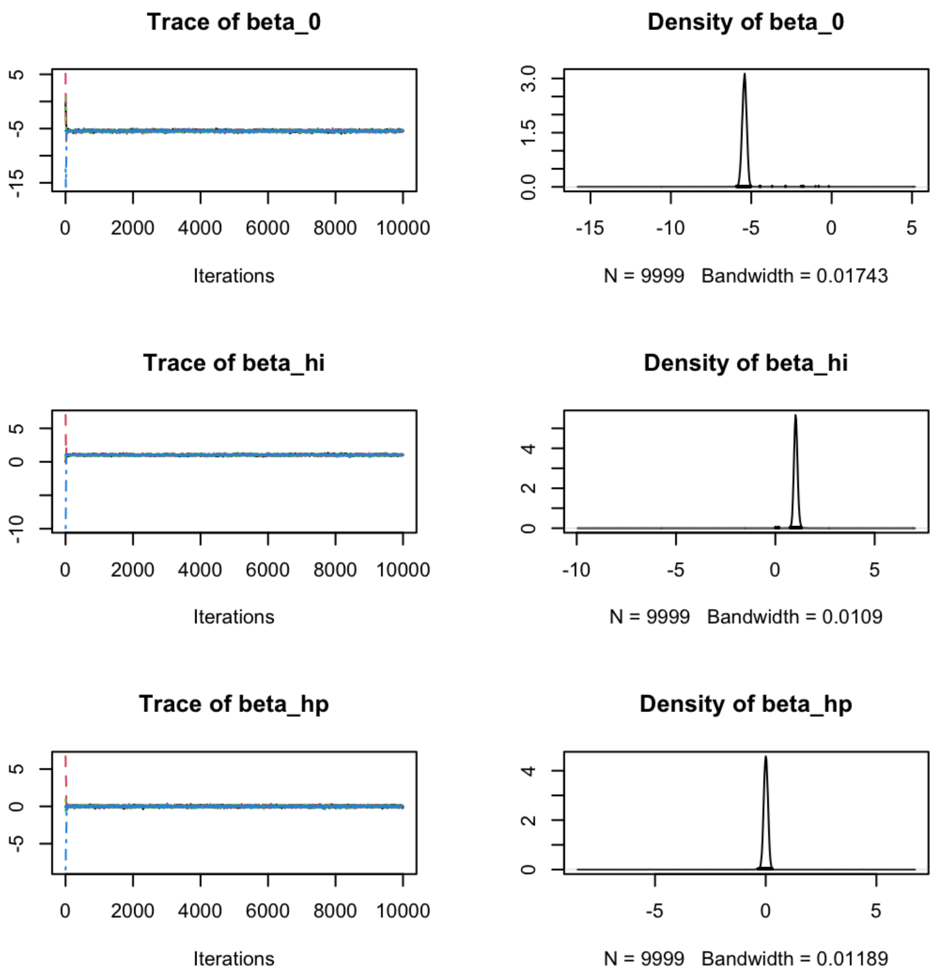

Figure 5: Trace plots for β_0, β_{hp}, β_{hi}, with `nburnin=1`.

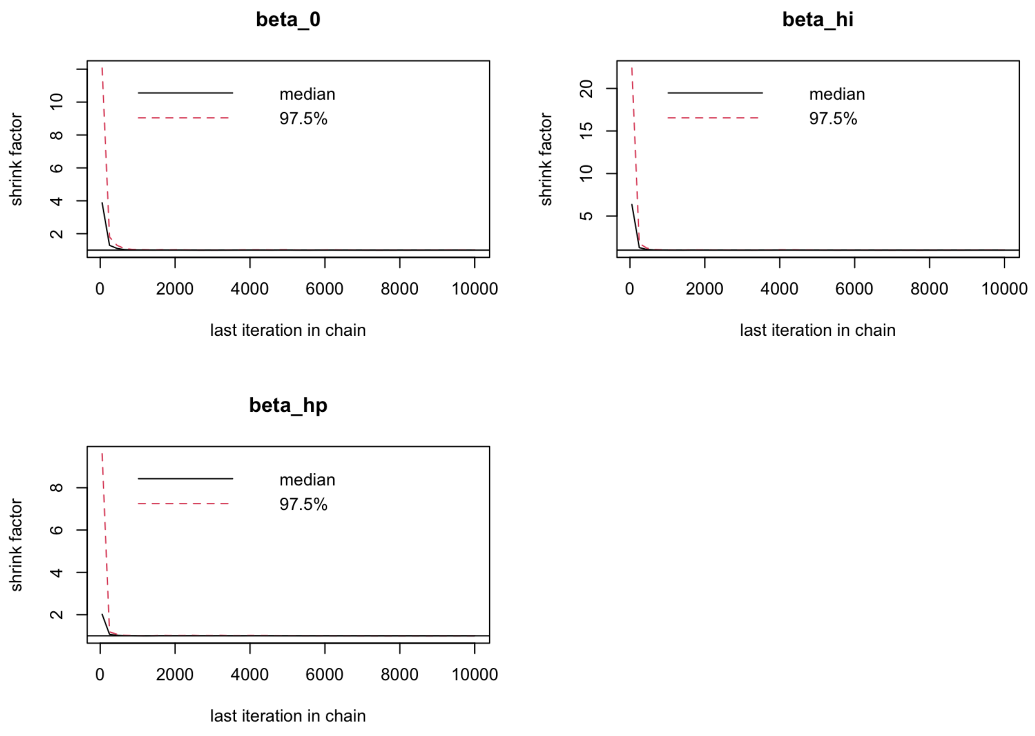

Figure 6: BGR plots for β_0, β_{hp}, β_{hi}, with `nburnin=1`.

The trace plots show that the chains are likely to converge to a unimodal distribution and the convergence is quick (before the 2000th iteration). The BGR plot supports this, since at around 1000 iterations, the shrink factor converges to 1. Hence, we choose a burn-in length of 2000 (just to be conservative). Henceforth, after re-running the command `runMCMC` with new parameters `niter = 20000` and `nburnin = 2000`, we assume that we have a convergent Markov chain with 18000 samples.

We now turn our attention to the number of samples we collect. We start by assessing the dependence between samples.

```
# ACF plots
a <- acf(posterior$samples$chain1[, "beta_0"]) # plot autocorrelation of alpha sample
b <- acf(posterior$samples$chain1[, "beta_hp"]) # plot autocorrelation of beta sample
c <- acf(posterior$samples$chain1[, "beta_hi"]) # plot autocorrelation of tau sample

par(mfrow = c(1, 3))
plot(a, main = "ACF of beta_0")
plot(b, main = "ACF of beta_hp")
plot(c, main = "ACF of beta_hi")
```

Figure 7 shows that the samples are highly correlated for the parameters β_0 and β_{hi} and moderately correlated for β_{hp}. Calculating the effective sample sizes gives

```
beta_0   beta_hi   beta_hp
5294.112 4605.316 9861.357
```

which further confirms our hypothesis. To ensure that the chain is well mixed, we could increase the parameter `thin` however, that would imply storing only every k-th sample (for some $k \in \mathbb{N}$) and so discarding information.

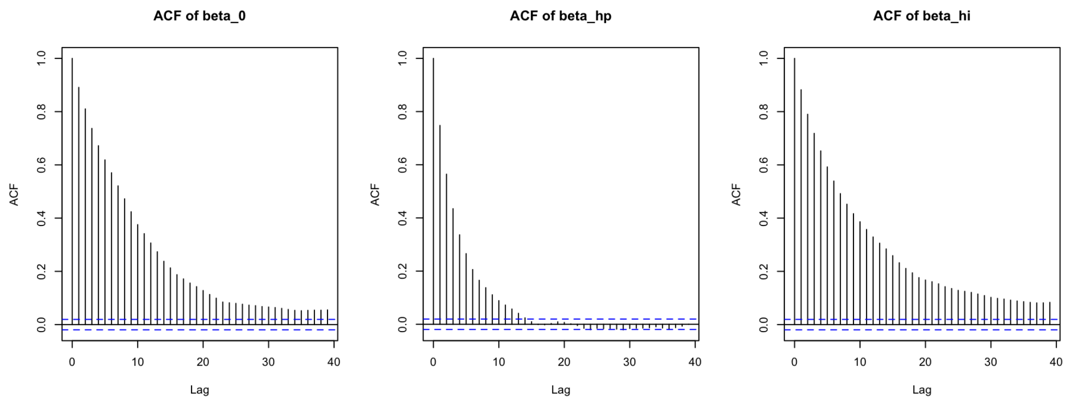

Figure 7: ACF plots for β_0, β_{hp}, β_{hi}, with `nburnin=2000`.

We wish that the Monte Carlo error satisfies

$$\frac{\text{MC error (adjusted)}}{\text{Standard error of posterior}} < 0.05.$$

Using the following `R` output:

```
> summary(combinedchains)

Iterations = 1:18000
Thinning interval = 1
Number of chains = 4
Sample size per chain = 18000

             Mean      SD  Naive SE Time-series SE
beta_0  -5.422734 0.13166 0.0004907      0.0018137
beta_hi  1.026921 0.08567 0.0003193      0.0012658
beta_hp  0.005133 0.09090 0.0003388      0.0009153
```

we calculate the ratios `Time-series SE / SD` for each parameter

```
[1] "beta_0 has a ratio of 0.0142242869435455"
[1] "beta_hi has a ratio of 0.014710292256621"
[1] "beta_hp has a ratio of 0.0101179886281656".
```

Despite the high dependence, we have collected enough samples as the ratio is below 0.05.

4.4 Posterior summary

We summarise in Table 2 the results of running the command `summary(combinedchains)`.

Parameter	**Posterior Mean**	**95% Credible Intervals**
β_0	-5.423937	(-5.6953,-5.1702)
β_{hp}	0.004649	(-0.1727,0.1816)
β_{hi}	1.027819	(0.8624,1.2024)

Table 2: Table of Posterior Means and 95% Credible Intervals for β_0, β_{hp}, β_{hi}.

Since the credible interval for β_{hp} is $(-0.1727, 0.1816)$ and zero lies inside the interval, this model suggests that the proportion of females enrolled in the HPV screening with a diagnosed cardiovascular disease (covariate `X_hp`) has no or very little impact to the probability, π_i, of being a carrier from hospital i. On the other hand, the proportion of Hispanic females enrolled in the HPV screening (covariate `X_hi`) increases the probability, π_i, of being a carrier from hospital i.

4.5 Prior sensitivity analysis

We run the same code but with different priors. Table 3 summarises the results.

Parameter	**Posterior Mean**	**95% Credible Intervals**
Model 1 (Q1) $\beta_0 \sim N(0,25)$,	$\beta_{hp} \sim N(0,25)$,	$\beta_{hi} \sim N(0,25)$
β_0	-5.423937	(-5.6953,-5.1702)
β_{hp}	0.004649	(-0.1727,0.1816)
β_{hi}	1.027819	(0.8624,1.2024)
Model 2 $\beta_0 \sim N(0,400)$,	$\beta_{hp} \sim N(0,400)$,	$\beta_{hi} \sim N(0,400)$
β_0	-5.428540	(-5.6984,-5.1782)
β_{hp}	0.005858	(-0.1739,0.1832)
β_{hi}	1.029211	(0.8635,1.1986)
Model 3 $\beta_0 \sim U(-100,100)$,	$\beta_{hp} \sim U(-100,100)$,	$\beta_{hi} \sim U(-100,100)$
β_0	-5.429076	(-5.7069,-5.1720)
β_{hp}	0.004823	(-0.1740,0.1806)
β_{hi}	1.031014	(0.8636,1.2051)

Table 3: Table of Posterior Means and 95% Credible Intervals for different models.

Table 3 shows that using different prior distributions leads to similar posterior means and credible intervals. Hence, the posteriors are very data-driven.

4.6 Model selection

We run the same code but with different likelihood. Table 4 summarises the results.

Model	**WAIC**	**lppd**	**pWAIC**
$\text{logit}(\pi_i) = \beta_0 + \beta_{hp} Z_{hp_i}$	274.6087	-100.0777	37.22659
$\text{logit}(\pi_i) = \beta_0 + \beta_{hi} Z_{hi_i}$	59.64526	-28.75534	1.067296
$\text{logit}(\pi_i) = \beta_0 + \beta_{hp} Z_{hp_i} + \beta_{hi} Z_{hi_i}$	61.20682	-29.04205	1.561359

Table 4: Table of Posterior Means and 95% Credible Intervals.

Table 4 shows that the lowest WAIC, least negative lppd, and lowest pWAIC is given by the model with $\text{logit}(\pi_i) = \beta_0 + \beta_{hi} Z_{hi_i}$. Hence we should retain only the covariate `X_hi`.

4.7 Posterior checks

The aim of this section is to check whether the inferences from the model make sense. If the model fits well, the observed data should look plausible under the posterior predictive distribution. Hence, we perform a few posterior predictive checks.

We start by generating 10,000 posterior predictive samples for the number of carriers recorded in the 15 hospitals using the same explanatory variables as in the original dataset (note that each sample will be a data vector of size 15).

```
# loading data
ni = DMDdata$ni
X_hp = DMDdata$X_hp
X_hi = DMDdata$X_hi
Y = DMDdata$ncarriers

# Number of posterior predictive samples
nsamples <- 10000

# Initialising lists to store 10000 samples for the 15 different hospitals
Y_samples <-  vector(mode = "list", length = nsamples)
```

```

for (i in 1:nsamples) {
    # Generating 10000 posterior predictive samples
    # For each parameter, sample from the empirical distribution
    beta_0_sample <- sample(as.matrix(posterior$samples)[,1], 1)
    beta_hp_sample <- sample(as.matrix(posterior$samples)[,3], 1)
    beta_hi_sample <- sample(as.matrix(posterior$samples)[,2], 1)

    # Calculate probabilities for each hospital
    pi_sample <- expit(beta_0_sample +
                    beta_hp_sample * ( ( X_hp[]-mean(X_hp[]) ) /sd(X_hp[]) ) +
                    beta_hi_sample * ( ( X_hi[]-mean(X_hi[]) ) /sd(X_hi[]) ))

    # Generate a binomial sample for each hospital
    Y_samples[[i]] <- rbinom(15,ni,pi_sample)
}
```

For each sample, we compute the following test quantities $T_j(y^{\text{rep}})$ for $j = 1, ..., 3$: the mean, median and maximum number of carriers. We obtain 3 empirical distributions.

```
# Initialising numeric vectors to store a distribution for
# the mean, median, and max carriers
Y_samples_mean <-  vector(mode = "numeric", length = nsamples)
Y_samples_median <-  vector(mode = "numeric", length = nsamples)
Y_samples_max <-  vector(mode = "numeric", length = nsamples)

# Calculates the mean, median, and max carriers for each sample
for (i in 1:nsamples) {
    Y_samples_mean[i] <- mean(Y_samples[[i]])
    Y_samples_median[i] <- median(Y_samples[[i]])
    Y_samples_max[i] <- max(Y_samples[[i]])
}
```

Now, we compare each distribution with the relative test quantity from the original sample $T_j(y)$. The discrepancy between $T_j(y^{\text{rep}})$ and $T_j(y)$ is summarised using a Bayesian p-value: we estimate both $\mathbb{P}(T_j(y^{\text{rep}}) \geqslant T_j(y))$ and $\mathbb{P}(T_j(y^{\text{rep}}) \leqslant T_j(y))$; the smallest of these probabilities should still be large enough to make the observed data plausible under the model.

```
# Calculates the Bayesian p values
pval_mean <- min(length(which(Y_samples_mean>=mean(Y))) / nsamples,
                 length(which(Y_samples_mean<=mean(Y))) / nsamples)
pval_median <- min(length(which(Y_samples_median>=median(Y))) / nsamples,
                   length(which(Y_samples_median<=median(Y))) / nsamples)
pval_max <- min(length(which(Y_samples_max>=max(Y))) / nsamples,
                 length(which(Y_samples_max>=max(Y))) / nsamples)

> print(paste("Bayesian p-value for the mean:", pval_mean))
"Bayesian p-value for the mean: 0.4806"
> print(paste("Bayesian p-value for the median:", pval_median))
"Bayesian p-value for the median: 0.0886"
> print(paste("Bayesian p-value for the max:", pval_max))
"Bayesian p-value for the max: 0.5174"
```

A non significant (at the 5% significance level) Bayesian p-value suggests that the observed data is plausible under our model. Figure 8 shows that the sample mean, median, and max are near the expectation of their empirical distributions, so it is likely that the observed data is plausible under our model.

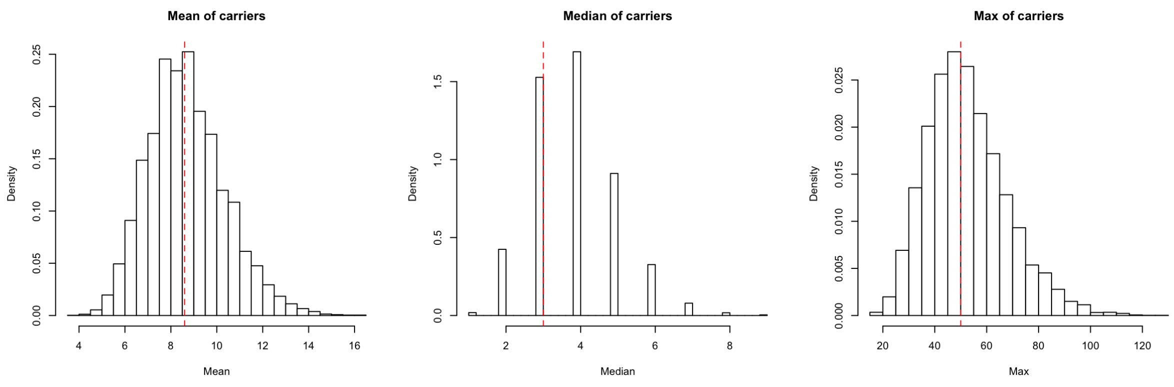

Figure 8: Histograms of the mean, median, and max for the number of carriers across the 15 hospitals. The red line indicates the sample mean, median, and max.

5 Conclusion

In this project, we first look at the background theory of Monte Carlo integration and MCMC. We show how we can check that we have a convergent chain and that we have enough samples for inference. Then, we see how we can generate such chain using the BUGS language by modelling DMD disease. Furthermore, we carry out a full inference including: model specification, model diagnostic, posterior summary, prior sensitivity analysis, model selection, and posterior checks.

A MCMC algorithms

A.1 The Gibbs sampler

The Gibbs sampler works as follows. Given a vector variable $\boldsymbol{\theta} = (\theta_1, \dots, \theta_p) \in \mathbb{R}_p$ with distribution $\pi(\boldsymbol{\theta}|\boldsymbol{x})$, the Gibbs sampler uses the set of full conditionals of π to sample indirectly from the full posterior distribution. For readability, drop the conditioning on the data and notation gives $\pi(\boldsymbol{\theta}|\boldsymbol{x}) = \pi(\boldsymbol{\theta})$.

Let $\pi(\theta_i|\boldsymbol{\theta}_{(i)})$ denote the full conditional of θ_i, given the other components $\boldsymbol{\theta}_{(i)} = (\theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_p)$, (and the data). Algorithm 1 shows one iteration of the Gibbs sampler algorithm.

Algorithm 1 One iteration of the Gibbs sampler algorithm.

```
1: Set arbitrary starting values $\boldsymbol{\theta}^0 = (\theta_1^0, ..., \theta_p^0)$.
2: Given that the Markov chain is in state $\boldsymbol{\theta}^t = (\theta_1^t, ..., \theta_p^t)$, then the Gibbs sampler successively makes random drawings from the full conditional distributions $\pi(\theta_i|\boldsymbol{\theta}_{(i)}^t)$, $i = 1, ..., p$ as follows:
```

$$\begin{array}{lll} \theta_1^{t+1} & \text{is sampled from} & \pi(\theta_1|\theta_2^t, \dots, \theta_p^t) \\ \theta_2^{t+1} & \text{is sampled from} & \pi(\theta_2|\theta_1^{t+1}, \theta_3^t, \dots, \theta_p^t) \\ \vdots & \vdots & \vdots \\ \theta_i^{t+1} & \text{is sampled from} & \pi(\theta_i|\theta_j^{t+1},\ j < i \text{ and } \theta_j^t, j > i) \\ \vdots & \vdots & \vdots \\ \theta_p^{t+1} & \text{is sampled from} & \pi(\theta_p|\theta_1^{t+1}, \dots, \theta_{p-1}^{t+1}). \end{array}$$

```
3: This completes a transition from $\boldsymbol{\theta}^t$ to $\boldsymbol{\theta}^{t+1}$. The Markov chain is then the sequence $\boldsymbol{\theta}^0, \boldsymbol{\theta}^1, ..., \boldsymbol{\theta}^T$.
```

Iteration of the full cycle of random variate generations from each of the full conditionals in turn, produces a sequence $\boldsymbol{\theta}^0, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^N, \ldots, \boldsymbol{\theta}^T$. The values $\boldsymbol{\theta}^0, \ldots, \boldsymbol{\theta}^N$ are discarded as burn-in, for same suitable value of N (see Section 2.3), and the values $\boldsymbol{\theta}^{N+1}, \ldots, \boldsymbol{\theta}^T$ can be used to obtain Monte Carlo estimates of interest.

The transition kernel for going from $\boldsymbol{\theta}^t$ to $\boldsymbol{\theta}^{t+1}$ is given by

$$\mathcal{K}_G(\boldsymbol{\theta}^t, \boldsymbol{\theta}^{t+1}) = \prod_{i=1}^{k} \pi(\theta_i^{t+1} | \theta_j^{t+1},\ j < i \text{ and } \theta_j^t,\ j > i), \tag{2}$$

and has stationary distribution π.

Conceptually, the Gibbs sampler appears to be a rather straightforward algorithmic procedure. Ideally, each of the conditionals will be of the form of a standard distribution and suitable prior specification often ensures that this is the case (for example, use of conjugate priors). However, in the cases where one or more of the conditionals is non-standard there are many ways to sample from univariate conditionals (e.g. direct sampling methods) however many of these algorithms are generally computationally intensive and inefficient. An alternative (and standard) approach is to use the Metropolis-Hastings algorithm for non-standard posterior conditionals.

A.2 The Metropolis-Hasting algorithm

A general way to construct MCMC samplers is as a form of generalised rejection sampling, where values are drawn from approximate/proposal distributions and "corrected" in order that, asymptotically, they are random observations from the target distribution. This is the motivation for methods such as the Metropolis-Hastings algorithm which sequentially draws candidate observations from a distribution, conditional only upon the last observation, thus inducing a Markov chain. The most important aspect of such algorithms is not the Markov property, but the fact that the approximating candidate distributions can be improved at each step in the simulation.

Let $\boldsymbol{\theta} = (\theta_1, ..., \theta_p)$ be a random vector with distribution $\pi(\boldsymbol{\theta}|\boldsymbol{x})$. For readability, drop the conditioning on the data and notation gives $\pi(\boldsymbol{\theta}|\boldsymbol{x}) = \pi(\boldsymbol{\theta})$. Let $\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi})$ be the transition Kernel.

The transition Kernel in the Metropolis-Hasting algorithm is set to be

$$\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi}) = q(\boldsymbol{\phi}|\boldsymbol{\theta})\alpha(\boldsymbol{\theta}, \boldsymbol{\phi})$$

where $q(\boldsymbol{\phi}|\boldsymbol{\theta})$ is the candidate generating density (of arbitrary choice) and $\alpha(\boldsymbol{\theta}, \boldsymbol{\phi})$ is the acceptance function.

To proceed we need the following Lemma: suppose that a Markov chain has a transition kernel $\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi})$ and that it satisfies detailed balance for π, i.e.

$$\pi(\boldsymbol{\theta})\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \pi(\boldsymbol{\phi})\mathcal{K}(\boldsymbol{\phi}, \boldsymbol{\theta}).$$

Then the chain has stationary density $\pi(\cdot)$.

The proof is as follows. Suppose that $\pi(\boldsymbol{\theta})\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \pi(\boldsymbol{\phi})\mathcal{K}(\boldsymbol{\phi}, \boldsymbol{\theta})$. We wish to show that $\mathbb{E}_{\boldsymbol{\theta}}(\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi})) = \pi(\boldsymbol{\phi})$. Indeed,

$$\begin{aligned}
\int \pi(\boldsymbol{\theta})\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi})d\boldsymbol{\theta} &= \int \pi(\boldsymbol{\phi})\mathcal{K}(\boldsymbol{\phi}, \boldsymbol{\theta})d\boldsymbol{\theta} \\
&= \pi(\boldsymbol{\phi}) \int \mathcal{K}(\boldsymbol{\phi}, \boldsymbol{\theta})d\boldsymbol{\theta} \\
&= \pi(\boldsymbol{\phi}).
\end{aligned}$$

Now suppose α is given by

$$\alpha(\boldsymbol{\theta}, \boldsymbol{\phi}) = \min\left(1, \frac{\pi(\boldsymbol{\phi})q(\boldsymbol{\theta}|\boldsymbol{\phi})}{\pi(\boldsymbol{\theta})q(\boldsymbol{\phi}|\boldsymbol{\theta})}\right)$$

Then the above Lemma is satisfied, i.e. the Markov chain has a stationary distribution.

The proof (assuming the form of α is given) is as follows. Using the definition of the kernel of the M-H algorithm, $\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi}) = q(\boldsymbol{\phi}|\boldsymbol{\theta})\alpha(\boldsymbol{\theta}, \boldsymbol{\phi})$ and using the above Lemma, $\pi(\boldsymbol{\theta})\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \pi(\boldsymbol{\phi})\mathcal{K}(\boldsymbol{\phi}, \boldsymbol{\theta})$, hence we must have

$$\pi(\boldsymbol{\theta})q(\boldsymbol{\phi}|\boldsymbol{\theta})\alpha(\boldsymbol{\theta}, \boldsymbol{\phi}) = \pi(\boldsymbol{\phi})q(\boldsymbol{\theta}|\boldsymbol{\phi})\alpha(\boldsymbol{\phi}, \boldsymbol{\theta}).$$

Now we have two cases. Case 1: $\boldsymbol{\theta} \neq \boldsymbol{\phi}$.

$$\begin{aligned}
\pi(\boldsymbol{\theta})\mathcal{K}(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \pi(\boldsymbol{\theta})q(\boldsymbol{\phi}\boldsymbol{\theta}) \min\left(1, \frac{\pi(\boldsymbol{\phi})q(\boldsymbol{\theta}\boldsymbol{\phi})}{\pi(\boldsymbol{\theta})q(\boldsymbol{\phi}\boldsymbol{\theta})}\right) \\
&= \min\left(\pi(\boldsymbol{\theta})q(\boldsymbol{\phi}\boldsymbol{\theta}), \pi(\boldsymbol{\phi})q(\boldsymbol{\theta}\boldsymbol{\phi})\right) \\
&= \min\left(\frac{\pi(\boldsymbol{\theta})q(\boldsymbol{\phi}\boldsymbol{\theta})}{\pi(\boldsymbol{\phi})q(\boldsymbol{\theta}\boldsymbol{\phi})}, 1\right) \pi(\boldsymbol{\phi})q(\boldsymbol{\theta}\boldsymbol{\phi}) \\
&= \pi(\boldsymbol{\phi})\mathcal{K}(\boldsymbol{\phi}, \boldsymbol{\theta}).
\end{aligned}$$

Case 2: $\boldsymbol{\theta} = \boldsymbol{\phi}$. Trivial.

To summarise, we have showed that

$$\alpha(\boldsymbol{\theta}, \boldsymbol{\phi}) = \min\left(1, \frac{\pi(\boldsymbol{\phi})q(\boldsymbol{\theta}|\boldsymbol{\phi})}{\pi(\boldsymbol{\theta})q(\boldsymbol{\phi}|\boldsymbol{\theta})}\right)$$

is the acceptance function that generates a stationary Markov chain. Algorithm 2 shows one iteration of the Metropolis-Hastings algorithm.

Algorithm 2 One iteration of the Metropolis-Hastings algorithm.

```
1: Set arbitrary starting values $\boldsymbol{\theta}^0 = (\theta_1^0, ..., \theta_p^0)$.
2: Given that the Markov chain is in state $\boldsymbol{\theta}^t = (\theta_1^t, ..., \theta_p^t)$, generate a new value $\boldsymbol{\phi}$ from the distribution
   $q(\boldsymbol{\phi}|\boldsymbol{\theta}^t)$.
3: Calculate
   $\alpha(\boldsymbol{\theta}, \boldsymbol{\phi}) = \min\left(1, \frac{\pi(\boldsymbol{\phi})q(\boldsymbol{\theta}|\boldsymbol{\phi})}{\pi(\boldsymbol{\theta})q(\boldsymbol{\phi}|\boldsymbol{\theta})}\right)$
4: With probability $\alpha(\boldsymbol{\theta}^t)$, set $\boldsymbol{\theta}^{t+1} = \boldsymbol{\phi}$, else set $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t$.
5: This completes a transition from $\boldsymbol{\theta}^t$ to $\boldsymbol{\theta}^{t+1}$. The Markov chain is then the sequence $\boldsymbol{\theta}^0, \boldsymbol{\theta}^1, ..., \boldsymbol{\theta}^T$.
```

We conclude with three remarks. Firstly, we only need to know π up to proportionality, the constants cancel in the numerator and denominator of α. Secondly, the performance of the MCMC algorithm is dependent on the choice of the proposal distribution q. If q is chosen poorly, then the number of rejections may be high, so that the efficiency of the procedure can be low; conversely if q is chosen such that only very small moves are proposed, it may take a very long time for the Markov chain to traverse the set of plausible posterior parameter values. Lastly, the Gibbs sampler is a special case of the Metropolis-Hasting algorithm.