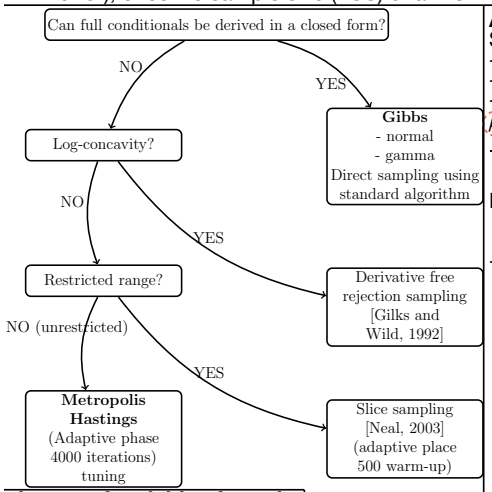
Stages of Bayesian analysis (an iterative process):
Stage 1: model building (likelihood(classical), parameters(classical),priors(Bayesian))
Stage 2: calculation of the posterior distribution (or target distribution)
– Analytical computation (Conjugate Bayes)
– Bayesian numerical approximation (INLA, bayesmeta)
– MCMC sampling: we get a sample either from own samplers or from JAGS, Stan, OpenBUGS, and WinBUGS
Stage 3: CODA convergence diagnostics are required for MCMC sampling. If CODA indicates any problems with MCMC samples, go to Stage 1.
Stage 4: Analysis of the posterior distribution. Compute descriptive statistics (mean, sd, quantiles, CrI, tail probabilities) of marginal posterior distributions.
Stage 5: Description of the results for the client.

Properties of Markov chain : • irreducible • aperiodic • positive-recurrent

General steps of MCMC algorithm

1. Select an initial value $\theta^{(0)}$.
2. Generate T values until the equilibrium is reached.
3. Monitor convergence diagnostics (if CODA fails, generate more observations, that is, increase T).
4. Cutoff the first B observations (in BUGS it is called burn-in and in Stan warming up).
5. Consider $\theta^{(B+1)}, \theta^{(B+2)}, \ldots, \theta^{(T)}$ as the sample for the posterior analysis (possibly after some tuning).
6. Plot the posterior distribution (usually focus on univariate marginal distributions).
7. Obtain summaries of the posterior distribution (classical: sample mean, median, quantiles, MC-error), effective sample size (ESS) of a MCMC simulation.



change of variables formula

$$f_Y(y) = f_X\{g^{-1}(y)\} \cdot \left|\frac{dg^{-1}(y)}{dy}\right|$$

$$= f_X(x) \cdot \left|\frac{dg(x)}{dx}\right|^{-1},$$

Eg: derive the density function of a lognormal random variable $Y = exp(X)$, where $X \sim \mathrm{N}(\mu, \sigma^2)$. From $y = g(x) = exp(x)$ we find that the inverse transformation is $x = g^{-1}(y) = log(y)$, which has the derivative $dg^{-1}(y)/dy = 1/y$.

$$f_Y(y) = f_X\{g^{-1}(y)\} \cdot \left|\frac{dg^{-1}(y)}{dy}\right|$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\frac{(\log(y)-\mu)^2}{\sigma^2}\right\}\left|\frac{1}{y}\right|$$

$$= \frac{1}{\sigma}\frac{1}{y}\varphi\left\{\frac{\log(y)-\mu}{\sigma}\right\},$$

Application of the Gibbs sampler

Setting:
- Model: sampling distribution $y_1, \ldots, y_n \sim \mathrm{N}(\mu, \sigma^2)$
- Data: set.seed(44566), n=30 from N(μ=4, σ²=16)
- Priors: independent mean and precision:

$\mu \sim \mathrm{N}(\mu_0, \sigma_0^2), \mu_0 = -3, \sigma_0^2 = 4$

$\frac{1}{\sigma^2} \sim \mathrm{G}(a_0, b_0), a_0 = 1.6, b_0 = 0.4$

$$\text{Mean} = \frac{a_0}{b_0} = \frac{1.6}{0.4} = 4 \text{ and Var} = \frac{a_0}{b_0^2} = \frac{1.6}{0.4^2} = 10$$

- Likelihood

$$f(y_1, \ldots, y_n \mid \mu, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_i-\mu)^2\right)$$

$$= \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2\right)$$

- Posterior

$$f(\mu, \sigma^2 \mid y_1, \ldots, y_n) \propto f(y_1, \ldots, y_n \mid \mu, \sigma^2) f(\mu, \sigma^2)$$

- Prior ($y = \sigma^2, x = \frac{1}{\sigma^2}$)

$f(\mu, \sigma^2) = f(\mu) f(\sigma^2)$ informative and independent

$-\infty < \mu_0 < \infty, \sigma_0^2, a_0, b_0 > 0$

$$f(x) = \frac{b_0^{a_0}}{\Gamma(a_0)} x^{a_0-1} \exp(-b_0 x)$$

$$y = \frac{1}{x} = g(x), \; x = \frac{1}{y} = g^{-1}(y), \; \frac{dg^{-1}(y)}{dy} = -\frac{1}{y^2}$$

$$f(y) = \frac{b_0^{a_0}}{\Gamma(a_0)} \left(\frac{1}{y}\right)^{a_0-1} \exp\left\{-\frac{b_0}{y}\right\} \left|-\frac{1}{y^2}\right|$$

$$= \frac{b_0^{a_0}}{\Gamma(a_0)} y^{-(a_0+1)} \exp\left\{-\frac{b_0}{y}\right\}$$

- Posterior

$$f(\mu, \sigma^2 \mid y_1, \ldots, y_n) \propto (\sigma^2)^{-n/2} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2\right)$$

$$\times (\sigma_0^2)^{-1/2} \exp\left(-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right)$$

$$\times (\sigma^2)^{-(a_0+1)} \exp\left(-\frac{b_0}{\sigma^2}\right)$$

接下来约掉所有σ^2项，以得到$f(\mu \mid \sigma^2, y_1, \ldots, y_2)$
或者约掉μ项，以得到$f(\sigma^2 \mid \mu, y_1, \ldots, y_2)$

$$f(\mu \mid \sigma^2, y_1, \ldots, y_n)$$

$$\propto \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2\right) \times \exp\left(-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right)$$

$$= \exp\left(-\frac{1}{2}\left[\frac{1}{\sigma^2}\left\{\sum_{i=1}^{n} y_i^2 + n\mu^2 - 2\mu\sum_{i=1}^{n} y_i\right\} + \frac{1}{\sigma_0^2}\left\{\mu^2 + \mu_0^2 - 2\mu_0\mu\right\}\right]\right)$$

$$\propto \exp\left(-\frac{1}{2}\left[\mu^2\left\{\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}\right\} - 2\mu\left\{\frac{\sum_{i=1}^{n} y_i}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}\right\}\right]\right)$$

$$p(x) \propto exp[-0.5(ax^2 - 2xb)]$$

$$\propto exp[-0.5a(x^2 - 2xb/a)]$$

$$\propto exp\{-0.5a[(x-b/a)^2 - (b/a)^2]\}$$

$$\propto exp[-0.5a(x-b/a)^2]exp[0.5a(b/a)^2]$$

$$\propto exp[-0.5a(x-b/a)^2]\underbrace{exp(\frac{b^2}{2a})}_{\text{constant}}$$

$$f(x) = \frac{p(x)}{\int_{-\infty}^{\infty} p(x)dx}$$

$$= \frac{\exp\left(\frac{b^2}{2a}\right)\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)}{\int_{-\infty}^{\infty}\exp\left(\frac{b^2}{2a}\right)\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)dx}$$

$$= \frac{\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)}{\int_{-\infty}^{\infty}\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)dx}$$

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

$$= \sqrt{\frac{a}{2\pi}}\exp\left(-\frac{a}{2}\left(x-\frac{b}{a}\right)^2\right)$$

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

This corresponds to a Gaussian kernel with expectation b/a and variance 1/a

$p(\theta) \propto \exp\left(-\frac{1}{2}(a\theta^2 - 2b\theta)\right)$, then $\theta \sim \mathrm{N}\left(\frac{b}{a}, \frac{1}{a}\right)$

$$\mu^{(t)} \mid (\sigma^2)^{(t-1)}, y_1, \ldots, y_n \sim \mathrm{N}\left(\frac{\frac{\sum_{i=1}^{n} y_i}{(\sigma^2)^{(t-1)}} + \frac{\mu_0}{\sigma_0^2}}{\frac{n}{(\sigma^2)^{(t-1)}} + \frac{1}{\sigma_0^2}}, \frac{1}{\frac{n}{(\sigma^2)^{(t-1)}} + \frac{1}{\sigma_0^2}}\right)$$

$$f(\sigma^2 \mid \mu, y_1, \ldots, y_n)$$

$$\propto (\sigma^2)^{-n/2}\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2\right)(\sigma^2)^{-(a_0+1)}\exp\left(-\frac{b_0}{\sigma^2}\right)$$

$$= (\sigma^2)^{-\frac{n}{2}-(a_0+1)}\exp\left(-\frac{1}{\sigma^2}\left[b_0 + \frac{1}{2}\sum_{i=1}^{n}(y_i-\mu)^2\right]\right),$$

which is the kernel of an inverse Gamma distribution

$$(\sigma^2)^{(t)} \mid \mu^{(t)}, y_1, \ldots, y_n \sim \mathrm{InvG}\left(\frac{n}{2} + a_0, b_0 + \frac{1}{2}\sum_{i=1}^{n}(y_i - \mu^{(t)})^2\right)$$

Gibbs algorithm $\theta = (\mu, \sigma^2)$, $p = 2$
- Set initial values
- For t=1, ..., T

μ-step, calculate:

$$\text{mean} = \frac{\frac{\sum_{i=1}^{n} y_i}{(\sigma^2)^{(t-1)}} + \frac{\mu_0}{\sigma_0^2}}{\frac{n}{(\sigma^2)^{(t-1)}} + \frac{1}{\sigma_0^2}}, \; \text{var} = \frac{1}{\frac{n}{(\sigma^2)^{(t-1)}} + \frac{1}{\sigma_0^2}}$$

generate one random value from N(mean,var), set $\mu^{(t)} = \mu$

σ^2-step, calculate:

$$\text{shape} = \frac{n}{2} + a_0, \; \text{scale} = b_0 + \frac{1}{2}\sum_{i=1}^{n}(y_i - \mu^{(t)})^2$$

generate σ^2 from InvG(shape, scale), set $(\sigma^2)^{(t)} = \sigma^2$

```r
set.seed(44566)
n <- 30
mu <- 4
sigma2 <- 16 #\sigma^2
y <- rnorm(n=n, mean=mu, sd=sqrt(sigma2))
mu0 <- -3
sigma2_0 <- 4
a0 <- 1.6
b0 <- 0.4
set.seed(44566)
n.iter <- 10000
n.burnin <- 4000
n.thin <- 1
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
```

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

Explanation of the code: First of all we set the number of thinning, the number of burn-in iterations and the length of the MCMC chain. n.tot is the number of total iterations the Gibbs sampler must be run. Next, we define a matrix with the number of rows equal to the number of iterations and the number of columns equal to the number of parameters that need to be estimated by MCMC algorithm. Gibbs sampler starts by initialising the MCMC chains. The for loop goes through n.tot iterations and computes parameters according to the full conditional posterior distributions for μ and σ^2 at each iteration. Moreover, the code generates the samples for the precision $1/\sigma^2$ as the reciprocal of σ^2. The code also cuts the first n.burnin many iterations and saves every n.thin-th iterations. Moreover, the code prints in which iteration the chain is after each 1000 iteration

Visualisation & quantitative analysis

```r
mu_gibbs_samples <- gibbs_samples[,"mu"]
sigma2_gibbs_samples <- gibbs_samples[,"sigma2"]
inv_sigma2_gibbs_samples <- gibbs_samples[,"inv_sigma2"]
summary(gibbs_samples)
for(ii in 1:3) {
  print(colnames(gibbs_samples)[ii])
  print(quantile(gibbs_samples[, ii], probs = c(0.025, 0.5, 0.975)))
}
```

APPLICATION OF THE METROPOLIS-HASTINGS SAMPLER

1. Data

• Suppose we have N binomial observations from $\frac{y_i}{n_i}$, $i = 1,\ldots,N$

• Expectation $\mathbb{E}(y_i) = n_i p_i$, where p_i is the corresponding response probability ($\hat{p}_i$ estimated relative frequency of deaths)

2. Logistic model :

Transformation to make linear: $\text{logit}(p_i) = \ln(\frac{p_i}{1-p_i}) = \alpha + \beta x_i \quad p_i = \frac{\exp(\alpha+\beta x_i)}{1+\exp(\alpha+\beta x_i)}$

3. Likelihood

$$f((\mathbf{y}_i, \mathbf{n}_i, \mathbf{x}_i) \mid \alpha, \beta) = \prod_{i=1}^{N} \binom{n_i}{y_i} p_i^{y_i}(1-p_i)^{n_i-y_i} = \prod_{i=1}^{N} \binom{n_i}{y_i} \left(\frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}\right)^{y_i} \left(1-\frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}\right)^{n_i-y_i}$$

4. Priors independent

$$f(\alpha) = \mathrm{N}(0,\sigma^2),\ f(\beta) = \mathrm{N}(0,\sigma^2),\ \sigma^2 = 10^4$$

5. Posterior distribution

$$f(\alpha,\beta \mid (\mathbf{y}_i, \mathbf{n}_i, \mathbf{x}_i)) \propto \prod_{i=1}^{N} \binom{n_i}{y_i} \left(\frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}\right)^{y_i} \left(1-\frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}\right)^{n_i-y_i} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\alpha^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\beta^2}{2\sigma^2}}$$

6. Random walk univariate proposal

$$\alpha' \sim \mathrm{N}(\alpha, \sigma_\alpha^2),\ q(\alpha \mid \alpha') = \frac{1}{\sqrt{2\pi\sigma_\alpha^2}} \exp\left(-\frac{1}{2\sigma_\alpha^2}(\alpha-\alpha')^2\right),\ q(\alpha' \mid \alpha) = \frac{1}{\sqrt{2\pi\sigma_\alpha^2}} \exp\left(-\frac{1}{2\sigma_\alpha^2}(\alpha-\alpha')^2\right)$$

Log-acceptance

$$\ln(A^\alpha) = \ln\left(\frac{f(\alpha',\beta \mid \mathbf{y},\mathbf{n},\mathbf{x})q(\alpha \mid \alpha')}{f(\alpha,\beta \mid \mathbf{y},\mathbf{n},\mathbf{x})q(\alpha' \mid \alpha)}\right) = \ln(f(\alpha',\beta \mid \mathbf{y},\mathbf{n},\mathbf{x})) - \ln(f(\alpha,\beta \mid \mathbf{y},\mathbf{n},\mathbf{x}))$$

$$\text{If } \ln(\text{runif}(1)) \le \ln A^\alpha \text{ then } \alpha \leftarrow \alpha', \text{ accept } \alpha' \text{ with probability } \ln(A^\alpha)$$

7. Random walk univariate proposal. $\beta' \sim \mathrm{N}(\beta, \sigma_\beta^2)$

$$\ln(A^\beta) = \ln\left(\frac{f(\alpha,\beta' \mid \mathbf{y},\mathbf{n},\mathbf{x})q(\beta \mid \beta')}{f(\alpha,\beta \mid \mathbf{y},\mathbf{n},\mathbf{x})q(\beta' \mid \beta)}\right) \text{ If } \log(\text{runif}(1)) \le \log A^\beta \text{ then } \beta \leftarrow \beta'.$$

Remarks:

• The user is responsible for tuning of σ_α^2 and σ_β^2.

• Random walk bivariate proposal. $\begin{pmatrix}\alpha'\\ \beta'\end{pmatrix} \leftarrow \text{rmvnorm}\left(\begin{pmatrix}\alpha\\ \beta\end{pmatrix}, \sigma\right)$ If $\log(\text{runif}(A)) \le \log A^{\alpha,\beta}$ then $\binom{\alpha}{\beta} \leftarrow \binom{\alpha'}{\beta'}$.

$$\ln(A)^{\alpha,\beta} = \ln\left(\frac{f(\alpha',\beta' \mid \mathbf{y},\mathbf{n},\mathbf{x})q(\alpha,\beta \mid \alpha',\beta')}{f(\alpha,\beta \mid \mathbf{y},\mathbf{n},\mathbf{x})q(\alpha',\beta' \mid \alpha,\beta)}\right)$$

Motivation adjust the shape of the probing proposal density to the shape of the posterior.

In case of a Gibbs sampler, we get A = 1. To see this, consider a single component Metropolis algorithm with

$$A = \min\left(1, \frac{f(\theta_j' \mid \boldsymbol{\theta}_{-j},\mathbf{y})q(\theta_j \mid \theta_j', \boldsymbol{\theta}_j)}{f(\theta_j \mid \boldsymbol{\theta}_{-j},\mathbf{y})q(\theta_j' \mid \theta_j, \boldsymbol{\theta}_{-j})}\right)$$

Take $q(\theta_j \mid \theta_j', \boldsymbol{\theta}_{-j}) = f(\theta_j \mid \boldsymbol{\theta}_{-j}, \mathbf{y})$ the full conditional posterior distribution, and $q(\theta_j' \mid \theta_j, \boldsymbol{\theta}_{-j}) = f(\theta_j' \mid \boldsymbol{\theta}_{-j}, \mathbf{y})$.

$$\frac{f(\theta_j' \mid \theta_{-\mathbf{j}},\mathbf{y})f(\theta_j \mid \theta_{-\mathbf{j}},\mathbf{y})}{f(\theta_j \mid \theta_{-\mathbf{j}},\mathbf{y})f(\theta_j' \mid \theta_{-\mathbf{j}},\mathbf{y})} = 1,$$ so that the Gibbs proposal will be always accepted.

```r
## Two independent normal proposals
rm(list=ls())
#Data from Collett, D. (2003) Modelling Binary Data 2nd Edition
#Table 1.6 p. 7 Number of deaths from pneumonia amongst batches
#of 40 mice exposed to different doses of an anti-pneumococcus serum
#(grouped)
#Table 3.1 p. 71 (original) Number of deaths from pneumonia in mice
#exposed to various doses of an anti-pneumococcus serum
# the covariate values (dose)
x_original <- c(0.0028, 0.0028, 0.0056, 0.0112, 0.0225, 0.0450)
# the centered covariate values (centered dose)
x <- x_original - mean(x_original)
# number of mice deaths
y <- c(26, 9, 21, 9, 6, 1)
# total number of mice
n <- c(28, 12, 40, 40, 40, 40)
# Assumption
# variance of normal priors
sigma2 <- 10^(4)
```

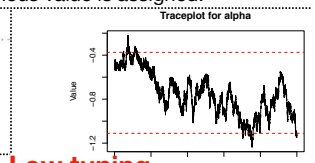
```r
## Bayesian analysis
####################
# inverse logit: logit^(-1)(alpha + beta*x)
mypi <- function(alpha, beta, x){
  tmp <- exp(alpha + beta*x)
  pi <- tmp/(1+tmp)
  return(pi)
}
#####################################
```

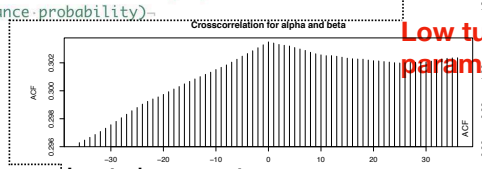
```r
## Step 1: R: (univariate proposal)
## Metropolis MCMC settings
#####################################
set.seed(44566)
n.iter <- 10000
n.burnin <- 4000
n.thin <- 1
#####################################
## univariate random walk proposals ##
#####################################
alpha_samples <- c()
beta_samples <- c()
# number of accepted proposals
alpha_yes <- 0
beta_yes <- 0
# starting values
alpha <- 0
beta <- 0
# standard deviations for the
# normal proposal
s_alpha <- 1
s_beta <- 60
# counter
count <- 0
```

What the code is doing?

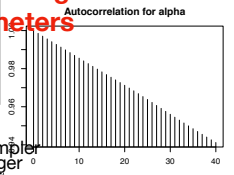
```r
# start the MCMC algorithm (the first iteration after the burn-in is 1)
for(i in -n.burnin:(n.iter*n.thin)){
  count <- count +1
  ## update alpha
  # generate a new proposal for alpha
  alpha_star <- rnorm(1, alpha, sd=s_alpha)
  # NOTE: it is more stable to calculate everything on the log scale
  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
  }
  ## 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
  }
  # 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")
  }
}
return(list(
  samples = data.frame(alpha = alpha_samples
                       beta = beta_samples),
  alpha_accepted = alpha_yes,
  beta_accepted = beta_yes,
  iterations = count))
mhSampler_low <- mhSampler(s_alpha = 0.01, s_beta = 1)
mhSampler_middle <- mhSampler(s_alpha = 1, s_beta = 100)
mhSampler_high <- mhSampler(s_alpha = 50, s_beta = 5000)
```

At each iteration, we draw a sample from one of the proposal distributions and compute the acceptance rate by dividing the posterior with the new parameter by the posterior with the old parameter. To ensure numerical stability, the code uses a log transformation. Samples are accepted only if the log acceptance rate is larger or equal to the log of a random uniform value. Otherwise, the sample is rejected and the previous value is assigned.



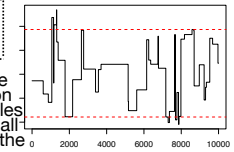


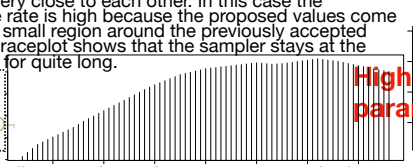
Low tuning parameters



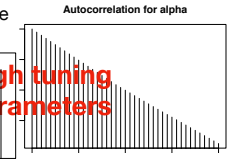
Low tuning parameters

When using the lower tuning settings, the MH sampler is heavily auto- and cross- correlated even for larger lags. This leads to highly dependent chains, and a failure to fully explore the parameter space. The auto-correlation and cross-corellation plots show that in case of low tuning parameters the samples are heavily correlated because the algorithm takes only small steps when exploring the parameter space and, therefore, the draws are very close to each other. In this case the acceptance rate is high because the proposed values come from a very small region around the previously accepted value. The traceplot shows that the sampler stays at the same value for quite long.





High tuning parameters

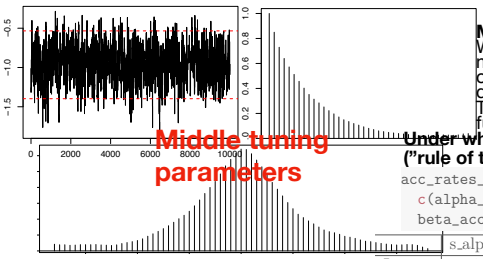


High tuning parameters

The steps of the random walk are too large, the acceptance rate is close to 0 and most of the proposed values get rejected. This means that the algorithm accepts the proposed values not often enough so that the chain remains at the same value for a large number of iterations. Therefore, the auto-correlations and cross-correlation of the resulting sample are high. The traceplot shows steps which means that the sampler remained in the same value for too long.

Interesting to note, that the auto-correlation and cross-correlation plots in cases of low and high tuning parameters show that the samples are heavily correlated but because of different reasons. Furthermore, the traceplots in the low and high tuning parameters case show that the algorithm stays at the same point for a long time, but because of different reasons.

As noted previously too low or too high tuning parameters are problematic. Proposal distributions with too low spread are very uninformative, leading to a log acceptance rates that are very high, which in turn means that the algorithm will accept samples too often, even if the change is rather small. On the other hand, proposal distributions with too large spread lead to samples close to the previous value. The algorithm will only accept new samples whenever the uniform random sample happens to be extremely small. With this the sample moves too slow and doesn't mix well.



Middle tuning parameters

With middle tuning parameters, the MH sampler appears much more well-behaved. The traceplots show that a large proportion of the parameter space is explored. Furthermore, we observe a distinct reduction in auto- and cross-correlation for larger lags. Thinning of the chain could reduce the correlations even further.

Under which condition the optimal acceptance rate of about 0.2-0.4 ("rule of thumb") is attained?

```r
acc_rates_middle <- with(mhSampler_middle, {
  c(alpha_accepted/iterations,
  beta_accepted/iterations)})
```

	s_alpha	s_beta	acc_alpha	acc_beta
Low	0.01	1	0.976	0.974
Middle	1.00	100	0.217	0.231
High	50.00	5000	0.005	0.005

JAGS: Just Another Gibbs Sampler

`glm.out <- glm(`$y \sim x$`, family=., data=.)` $y \sim x$ is the description of a model, queried via extractor functions,

```
summary(), coef, vcov
library(rjags)
m <- jags.model("code",data,inits,n.chain=2)
list.samplers(m)
# Burn-in
updates(m,n.iter=4000)
```

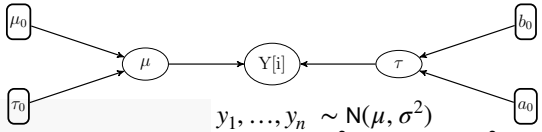
m is not a fitted model, it is a dynamic object that can be queried to generate from the posterior; **"code"** name of the file containing a description of the model in BUGS language; **data** named list of data for observed variables; **inits** is a list of lists of initial values, one for each chain; **n.chain** number of chains to run; **updates** is used for adaptation, burn-in

To generate samples from a given model **m** `x <- coda.samples(m, variable.names=".", n.iter=1000)`
• `x` is an object of class `mcmc.list -> coda` • `coda.samples` is a wrapper function for `"jags.samples"`

NORMAL EXAMPLE IN JAGS

```
# Define the parameters of the prior distributions
mu0 <- -3
sigma2_0 <- 4
a0 <- 1.6
b0 <- 0.4
suppressPackageStartupMessages(library(INLA))
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))
```

INLA used to compare



$y_1, \ldots, y_n \sim \mathrm{N}(\mu, \sigma^2)$

$\mu \sim \mathrm{N}(\mu_0, \sigma_0^2)$, $\mu_0 = -3$, $\sigma_0^2 = 4$

precision: $\tau = 1/\sigma^2$ (1/variance) $\sim \mathrm{G}(a_0, b_0)$

$a_0 = 1.6$, $b_0 = 0.4$

R/Stan: N(mean, sd)

BUGS/JAGS/INLA: $\mathrm{N}(\mu, \tau^{-1})$, $\tau^{-1} = \sigma^2$, $\tau = \dfrac{1}{\sigma^2}$

```
set.seed(44566)
suppressPackageStartupMessages(library(rjags))
suppressPackageStartupMessages(library(coda))
wb_data <- list( N=30,
                 y=c(3.048,2.980,2.029,7.249,-0.259,3.061,4.059,6.370,7.902,1.926,
                     9.094,10.489,-0.384,-3.096,2.315,5.830,-1.542,-1.544,5.714,
                     -5.182,3.828,-4.038,2.169,5.087,-0.201,4.880,3.302,3.859,
                     11.144,5.564)
)
wb_inits <- list( mu=-0.2381084, inv_sigma2=0.3993192 )
modelString = " # open quote for modelString
model{
# likelihood
for (i in 1:N){
y[i] ~ dnorm( mu, inv_sigma2 )
}
# Priors
mu ~ dnorm( -3, 0.25 ) # prior for mu N(mu0, prec=1/sigma2_0)
inv_sigma2 ~ dgamma( 1.6, 0.4 ) # prior for precision G(a0, b0)
# transformations
# deterministic definition of variance
sigma2 <- 1/inv_sigma2

# deterministic definition of standard deviation
sigma <- sqrt(sigma2)
}
" # close quote for modelString
writeLines(modelString, con="TempModelexe3.txt") # write to a file
```

JAGS CODE

```
model {
# likelihood
for(i in 1:N) {
Y[i] ~ dnorm(mu, tau)}
# priors
mu ~ dnorm(-3, 0.25) # (mu_0, tau_0)
tau ~ dgamma(1.6, 0.4) # (a_0, b_0)
}
```

list.factories and **set.factories** show and control over the status of factories in JAGS modules. **jags.modules** shows the names of the currently loaded modules and also loads or unloads JAGS modules.

```
list.factories(type = "rng")
##          factory status
## 1 base::BaseRNG   TRUE
```

```
load.module("glm")
list.modules()
## [1] "basemod"
    "bugs"    "glm"
unload.module("glm")
```

```
list.factories(type = "monitor")
##          factory status
## 1 base::Variance   TRUE
## 2     base::Mean   TRUE
## 3    base::Trace   TRUE
```

```
list.factories(type = "sampler")
##             factory status
## 1  bugs::BinomSlice   TRUE
## 2         bugs::RW1   TRUE
## 3    bugs::Censored   TRUE
```

```
set.factory(name = "base::Slice",
            type = "sampler", state = FALSE)
```

```
# model initiation
set.seed(44566)
model.jags <- jags.model(
  file = "TempModelexe3.txt",
  data = wb_data,
  inits = wb_inits,
  n.chains = 1,
  n.adapt = 4000
)
update(model.jags, n.iter = 4000) # burn-in
# sampling
fit.jags.coda <- coda.samples(
  model = model.jags,
  variable.names = c("mu", "sigma2", "inv_sigma2"),
  n.iter = 10000,
  thin = 1
)
summary(fit.jags.coda)
plot(fit.jags.coda)
```

one chain

```
wb_inits <- function() {
  list(mu = rnorm(1),
       inv_sigma2 = runif(1)
  )
}
# model initialisation
model.jags <- jags.model(
  file = "TempModelexe3.txt",
  data = wb_data,
  inits = wb_inits,
  n.chains = 4,
  n.adapt = 4000
)
```

several chains

```
effectiveSize(fit.jags.coda)
lapply(fit.jags.coda, effectiveSize)
gelman.diag(fit.jags.coda,autoburnin=TRUE)
gelman.plot(fit.jags.coda,autoburnin=TRUE)
geweke.diag(fit.jags.coda) heidel.diag(fit.jags.coda)
geweke.plot(fit.jags.coda)raftery.diag(fit.jags.coda)
coda:::traceplot(fit.jags.coda)
```

CODA

```
# "DIC" penalised expected deviance computation
dic1<-dic.samples(model=model.jags, n.iter=1000, type="popt")
```

Steps (JAGS follows it): • initialization • burn-in 1,…,b • monitoring (sampling) b+1, . . . , b+M (assumption stationary)

Q1 What should I use for starting values?
1. two (or four) different starting values 2. Idea: Start/initialize the chain somewhere near a measure of center of the relevant posterior distribution (mean, mode) (use maximum likelihood justified by weakly informative priors). 3. Remark: This can be problematic if the posterior is multimodal. For example, in a bimodal normal mixture $Y = \alpha X_1 + (1-\alpha)X_2$.

Q2 **Frequentistic Hypothesis testing:** How long should burn-in period be? (How do you know when the Markov chain reaches equilibrium (stationary distribution)?) **WE DON'T KNOW**
• We can only look for evidence that it has not converged • This is the usual situation in hypothesis testing • But the consequences of the type II error are severe: Invalid INFERENCE

Gelman / Rubin / Brooks BGR (Convergence to ergodic average)

$$\sqrt{\hat{R}} \approx \sqrt{\frac{W+B}{W}} \to 1, \quad \text{if} \quad B \to 0.$$

Idea: Run multiple chains from widely dispersed starting values and preform an Analysis of Variance to see if the between-chain variability (B) is large in relation to the average variability within (W + B) the (pooled) chain (if so this would indicate more than one mode). The idea is that if separate chains have not mixed well, the variance of all chains (W + B) taken together (pooled) should be higher than the variance of individual chains (W). Run many (at least 2 short chains and compare late parts (second half) of the chains).
Chains mix well and converge, if $\hat{R} \to 1$. $\hat{R}$ is also called psrf the "potential scale reduction factor".
• If chains have not converged, they will be over-dispersed • No testing • "Shrink factor" $\hat{R}$ and Upper Confidence Interval • Uses normal approximation to derive $\hat{R}$.
Vehtari et al. [2021]: traditional $\hat{R}$ can fail to correctly diagnose convergence failures when the chain has a heavy tail or when the variance varies across the chains and proposed an alternative rank-based diagnostic. They recommend that at least four chains should be run by default and the threshold applied to $\hat{R}$ should be 1.01($\hat{R}$ < 1.01). They only recommend relying on the $\hat{R}$ estimate to make decisions about the quality of the chains if the ESS is large enough.

$$\hat{R} \approx \sqrt{1 + \frac{\texttt{n.chain}}{\mathrm{ESS}}}$$

Vats and Knudson [2021]: a cutoff of $\hat{R} \le 1.1$ is too high to yield reasonable estimates of target quantities. They show that $\hat{R}$ = 1.1 corresponds to ESS = 5 × n.chain (5 independent samples per chain), which is too low to estimate the mean with any reasonable certainty. In contrast, $\hat{R}$ = 1.01 corresponds to ESS = 50 × n.chain (50 independent samples per chain), which is more appropriate for estimation of the mean.

Geweke (Convergence to stationarity) Detects cases when equilibrium has not been reached.

$$Z = \frac{\bar{\theta}^{*A} - \bar{\theta}^{*B}}{\sqrt{\mathrm{Var}(\theta^{*A}) + \mathrm{Var}(\theta^{*B})}}$$

Idea: • consider a single long run • test for equality of means between early and late sections of the chain • essentially, two independent sample t-tests. Example : Assume: Ⓐ corresponds to early 10 % (frac1) of the chain Ⓑ corresponds to next 50 % (frac2) of the chain. Does | Z |> 2? **geweke.plot**: 2.5 % of Z-scores may be below −2 and 2.5 % above 2. Properties: Univariate. It is unclear how to choose the width of the early and the late window?

Q3 How long do you need to monitor the chain to get results of sufficient MC accuracy? (< 0.001)

RELATED TO Q2 & Q3

Raftery & Lewis (Convergence to ergodic average) Idea: Run diagnostics based on a criterion of accuracy of estimation of the quantile q. It is a non-parametric approach. • Two state Markov-Chain theory, discretisation of a continuous chain to get a binary control $Z^{(t)} = I_{\theta^t \le \theta_q}$ $\quad P = \left[\begin{smallmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{smallmatrix}\right]$
• Pseudo-transition Matrix
• Needs a pre-run to set up P preliminary chain to estimate α, β
Q2 (Burn-in) (M) needed; Q3 Total N sample size N needed; N_{min} minimal sample needed $I = \frac{M+N}{N_{\min}}$ (independent),
The (in)dependence factor I, indicates to which extend autocorrelation inflates the required sample size. I > 5 strong autocorrelation. It is a crude estimate of the thinning interval. Quantiles which are closer to median need more N than quantiles further away.

Heidelberger & Welch (Convergence to stationarity)
Q2 Stationarity test: Basically a Kolmogorov-Smirnov Test. At each iteration 10% are removed of the first half of the chain. Information: if passed, iteration and p-value are provided. If all iterations have been used and did not pass, then failed.
Q3 When all passed then a Halfwidth test is applied on the retained chain.
95 % CI (mean) with portion of the chain which has passed stationarity test. Coefficient of variation = $\frac{S}{\mu}$
< eps = 0.1 desired accuracy, where S is half width of 95 % CI and μ is mean. If > eps then fails.
If the sample is not sufficient to estimate the mean with sufficient accuracy then increase it by 1.5.

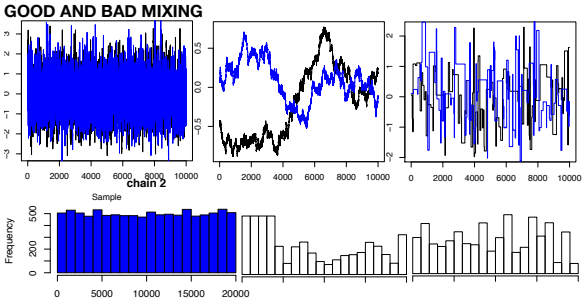
Why MCMC sampling can be dangerous?
Diagnostic tests are frequentist tests and there is a danger of false negative results, type II error β.

Truth, convergence to stationarity	Diagnostic test: non-significant	significant
yes	$1-\alpha$	α
no	β	$1-\beta$

CODA (Overview)

		Convergence to stationarity	Convergence to ergodic average
number of chains 1	graphical	traceplots	NA
	numerical	H&W (run length control based on mean)	ESS
		Geweke (lack of convergence)	R&L
number of chains > 1	graphical	rank plots	NA
		NA	BGR, $\hat{R}$ (Lack of convergence using multiple parallel chains)

GOOD AND BAD MIXING

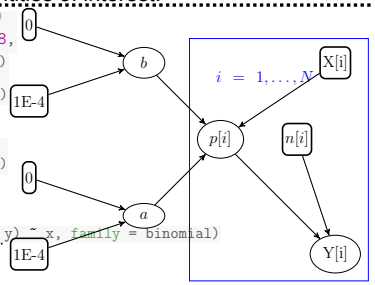


Left panel: good mixing obtained for a suitable standard deviation.Rank plot indicates uniformity for the chain. This shows that the chain has converged and is mixing well.
Middle panel: bad mixing obtained for a too small standard deviation.
Right panel: bad mixing obtained for a too large standard deviation.
ESS shows how many independent draws contain the same amount of information as the dependent sample obtained by the MCMC algorithm. The higher ESS the better. ESS must be large enough to get stable inferences for quantities of interest.

LOGISTIC REGRESSION IN RJAGS

```
#create Data
data <- matrix(NA, nrow=6, ncol=3)
colnames(data) <- list("x", "y", "n")
data[,1] <- x
data[,2] <- y
data[,3] <- n
mice.data <- list(y=data[,2], x=data[,1],
                  n=data[,3], N = nrow(data))
#define parameters
mice.params <- c("a", "b")
# define initial values for 2 chains
inits1 <- list(a=rnorm(1), b=rnorm(1),
               .RNG.name="base::Super-Duper", .RNG.seed=1)
inits2 <- list(a=rnorm(1), b=rnorm(1),
               .RNG.name="base::Wichmann-Hill", .RNG.seed=2)
mice.inits <- list(inits1, inits2)
modelString = "
# open quote for modelString
model{
  for(i in 1:N){
    y[i] ~ dbin(p[i],n[i])
    logit(p[i]) <- a+b*x[i]
  }
  a ~ dnorm(0,0.0001)
  b ~ dnorm(0,0.0001)
}" # close quote for modelString
writeLines(modelString, con="05_mice_logistic_regression_JAGS.txt")
# model initiation
mice.rjags <- jags.model(
  file = "05_mice_logistic_regression_JAGS.txt",
  data = mice.data,
  inits = mice.inits,
  n.chains = 2,
  n.adapt = 4000)
N.iter <- 10000
N.thin <- 1
N.burnin <- 4000
# burn-in
update(mice.rjags, n.iter = N.burnin)
# sampling/monitoring
fit.rjags.coda.mice <- coda.samples(
  model = mice.rjags,
  variable.names = mice.params,
  n.iter = N.iter,
  thin = N.thin)
# store samples for each parameter f
a.sample <- fit.rjags.coda.mice[,"a"]
b.sample <- fit.rjags.coda.mice[,"b"]
summary(fit.rjags.coda.mice)
```

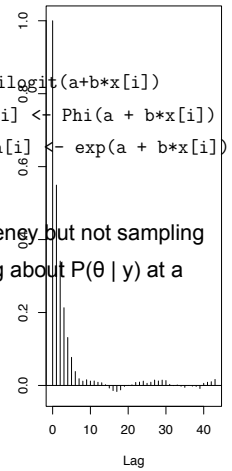
```
# the covariate values (dose)
x_original <- c(0.0028, 0.0028,
  0.0056, 0.0112, 0.0225, 0.0450)
# center covariate values
x <- x_original - mean(x_original)
# number of mice deaths
y <- c(26, 9, 21, 9, 6, 1)
  # total number of mice
n <- c(28, 12, 40, 40, 40, 40)
## Fit a classical generalized linear model
  (logistic regression for binomial data)
m <- glm(formula = cbind(y, n - y) ~ x, family = binomial)
```

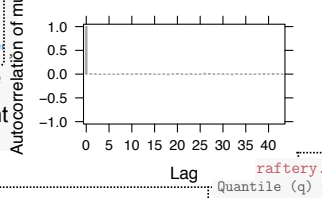



```
logit(p[i]) <- a+b*x[i] - binary - logit ↔ p[i] <- ilogit(a+b*x[i])
probit(mu[i]) <- a + b*x[i] - binary - probit ↔ mu[i] <- Phi(a + b*x[i])
log(lambda[i]) <- a+b*x[i] - Poisson - log ↔ lambda[i] <- exp(a + b*x[i])
```

```
a.sample <- fit.rjags.coda.mice[,"a"]
acf(as.matrix(a.sample), main = "a", ci = "")
```

ACF plots shows high dependence. –> Autocorrelation in the draws affects only efficiency but not sampling validity.
If autocorrelation is large, then we are learning about P(θ | y) at a slower rate than iid.





Autocorrelation plots give an overview of autocorrelation in different thinning interval. When the lag is larger than 1, the autocorrelation is close to 0.

```
heidel.diag(fit.rjags.coda.mice)
##   Stationarity start     p-value
##   test          iteration
## a passed        1         0.688
## b passed        1         0.442
##
##   Halfwidth Mean      Halfwidth
##   test
## a passed      -0.964 0.009
## b passed    -142.693 0.979
```

heidel.diag checks the convergence to stationarity and shows that the chains passed the stationarity and halfwidth

```
raftery.diag(fit.rjags.coda.mice)
Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

  Burn-in  Total Lower bound  Dependence
  (M)      (N)   (Nmin)       factor (I)
a 10       12690 3746         3.39
b 12       13200 3746         3.52
```

3746: spl size per chain
n.thin*3750=n.iter

raftery.diag checks the convergence to ergodic average and suggests a thinning equal to 4 (maximum of all dependence

$$\mathrm{ACF}(k) = \mathrm{corr}(\theta_t^*, \theta_{t+k}^*)$$

$$\mathrm{ESS} = N_{\mathrm{eff}} = \frac{M}{1 + 2\sum_{k>1} \mathrm{ACF}(k)}$$

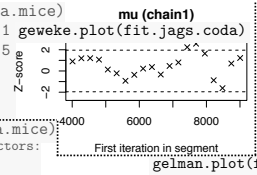
```
effectiveSize(fit.jags.coda)
```

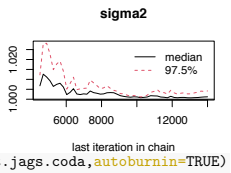
geweke.diag and geweke.plot check the convergence to stationarity: geweke.diag shows that the z-scores are not larger than 1.96 and geweke.plot shows that only a few values fall out of the bound (-1.96, 1.96).

```
geweke.diag(fit.rjags.coda.mice)
Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

     a      b
-0.1719 -0.3351
```

```
geweke.plot(fit.jags.coda)
```






```
gelman.diag(fit.rjags.coda.mice)
## Potential scale reduction factors:
##
##   Point est. Upper C.I.
## a          1          1
## b          1          1
##
## Multivariate psrf
##
## 1
```

```
gelman.plot(fit.jags.coda,autoburnin=TRUE)
```

gelman.diag and gelman.plot deal with the convergence to ergodic average
Figure demonstrates that the shrink factor ($\hat{R}$) decreases to 1 with increasing number of iterations.

```
stable.GR(fit.rjags.coda.mice)
## $psrf
## [1] 1.000062 1.000062
```

stable.GR from package stableGR (Vats and Knudson). psrf is the "potential scale reduction factor" which is the R and these are smaller than 1.01.

After an adaptation phase of 1000 and a burn-in of 4000 draw a sample of 1000 observations in one chain with thinning set to 1.

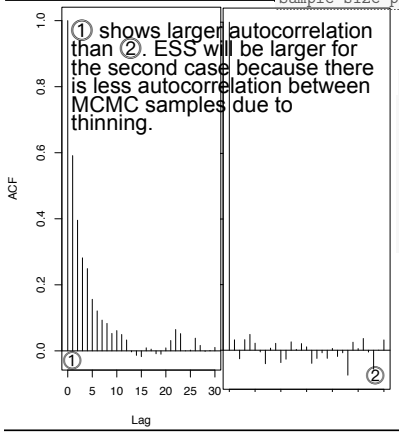
```
Iterations = 5001:6000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 1000
```

①

After an adaptation phase of 1000 and a burn-in of 4000 draw a sample of 10000 observations in one chain with thinning set to 10.

```
Iterations = 5010:15000
Thinning interval = 10
Number of chains = 1
Sample size per chain = 1000
```

②



① shows larger autocorrelation than ②. ESS will be larger for the second case because there is less autocorrelation between MCMC samples due to thinning.

```
(n.eff(as.matrix(fit.rjags.coda.mice.1[, "b"])))
## $n.eff    $converged $n.target
##        se [1] FALSE      se
## 196.3447             31303
(n.eff(as.matrix(fit.rjags.coda.mice.2[, "b"])))
## $n.eff $converged $n.target
## var1   [1] FALSE var1
## 1000             6147
```

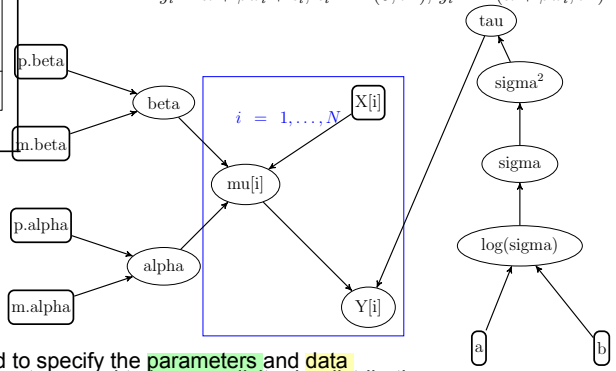
n.eff from the R package stableGR informs about the convergence status.

BUGS CODE FOR A LINEAR REGRESSION MODEL

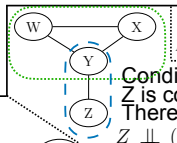
Remark: In R: `lm(y ~ x)`.

$$y_i = \alpha + \beta x_i + \epsilon_i,\ \epsilon_i \overset{iid}{\sim} \mathrm{N}(0, \sigma^2),\ y_i \sim \mathrm{N}(\alpha + \beta x_i, \sigma^2).$$

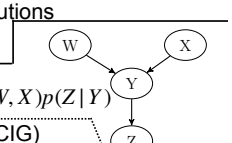
```
model {
# likelihood
for(i in 1:length(Y)) {
Y[i] ~ dnorm(mu[i], tau)
mu[i] <- alpha + beta * x[i]
}
# priors
alpha ~ dnorm(m.alpha, p.alpha)
beta ~ dnorm(m.beta, p.beta)
log.sigma ~ dunif(a,b)
sigma <- exp(log.sigma)
sigma.sq <- pow(sigma, 2)
tau <- 1/sigma.sq
}
```



- Need to specify the parameters and data
- parameters need to have explicit prior distributions
- not vectorized "for loops"
- model includes parameter transformations



Directed Acyclic Graph (DAG)
$p(W, X, Y, Z) = p(W)p(X)p(Y \mid W, X)p(Z \mid Y)$



Conditional Independence Graphs (CIG)
Z is conditionally independent of (W, X) given Y.
There are two cliques in a CIG: (W, X, Y) and (Y, Z)
$Z \perp\!\!\!\perp (W, X) \mid Y$

(a) DAG (b) CIG

Figure 5.6: Marrying parent nodes to get CIG from a DAG