An (extremely brief) introduction to Bayesian Markov chain Monte Carlo

Matt Denwood

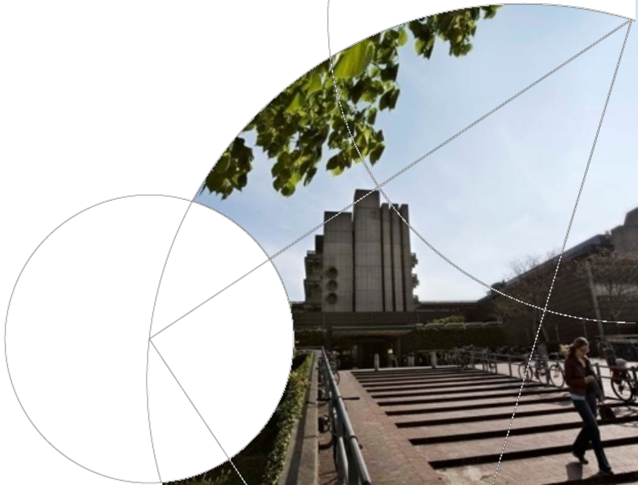
md@sund.ku.dk

Department of Veterinary and Animal Sciences

University of Copenhagen

Denmark

UNIVERSITY OF COPENHAGEN



Overview

- What is Bayesian statistics?
- What is the theory behind MCMC?
- How does MCMC work in practice?
- A basic example: prevalence estimation

What is Bayesian statistics?

Bayes theorem

Discrete form:

$$P(A_j \mid B) = \frac{P(B \mid A_j) \cdot P(A_j)}{\sum_i P(B \mid A_i) \cdot P(A_i)}$$

Continuous form:

$$f_X(x \mid Y = y) = \frac{f_Y(y \mid X = x) \cdot f_X(x)}{\int_{-\infty}^{\infty} f_Y(y \mid X = \varepsilon) \cdot f_X(\varepsilon) \cdot d\varepsilon}$$

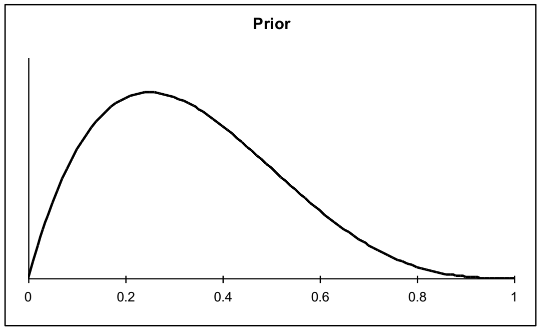
General form:

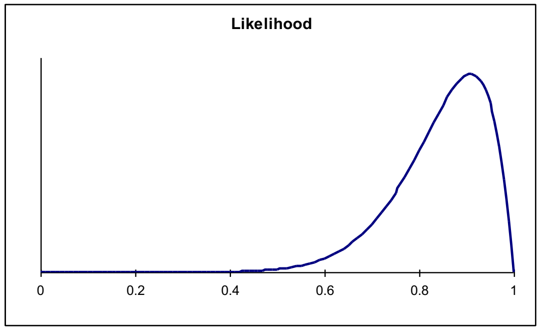
Posterior $\propto$ Likelihood x Prior

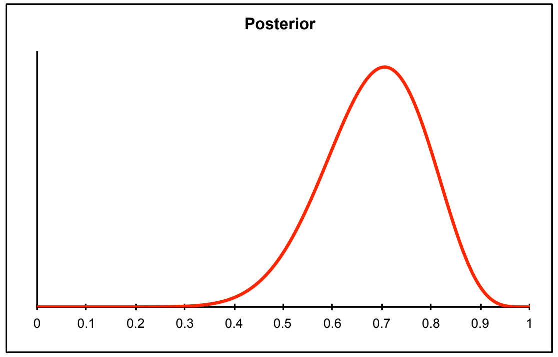
Bayesian statistics

- ”Belief” is expressed as a probability distribution
  - i.e. We don’t know exactly what the parameter value is, but we can quantify our uncertainty
  - This is the key difference to frequentist statistics

- We usually have continuous probability distributions for both prior and posterior







What we knew before + What the data tell us = What we know now

What is the theory behind MCMC?

MCMC: the theory

First define a function to calculate a posterior probability corresponding to a Binomial likelihood and Beta prior:

```
log_posterior_fun <- function(parameter){
   # ... Some R code e.g. ...
   ll <- dbinom(data$Pos, data$N, parameter, log=TRUE)
   lp <- dbeta(parameter, 1, 1, log=TRUE)
   return(ll + lp)
}
```

Then choose a ‘reasonable’ place to start looking for our parameter values, and calculate the posterior at that parameter value:

```
parameter[1] <- 0.25
log_post[1] <- log_posterior_fun(parameter[1])
log_post[1]
        -27.5222
```

MCMC: the theory

Now sample another parameter value using a random walk:

```
new_par <- rnorm(1, mean=parameter[1], sd=sigma)
       0.2606076
new_lpost <- log_posterior_fun(new_par)
       -18.00832
```

That’s a bit better! Let’s store this new value:

```
parameter[2] <- new_par
log_post[2] <- new_lpost
```

And resample:

```
new_par <- rnorm(1, mean=parameter[2], sd=sigma)
       0.2572526
new_lpost <- log_posterior_fun(new_par)
       -20.72975
```

MCMC: the theory

This is a bit worse. Should we keep it or not?

We will let fate decide … but it makes sense to have a higher chance of accepting the new parameter value if it is not much worse than the old value:

```
probability_ratio <- exp(new_lpost – log_post[2])
       0.06578108
accept <- rbinom(1, 1, probability_ratio)
       0
```

So we reject this new value, and keep the old one

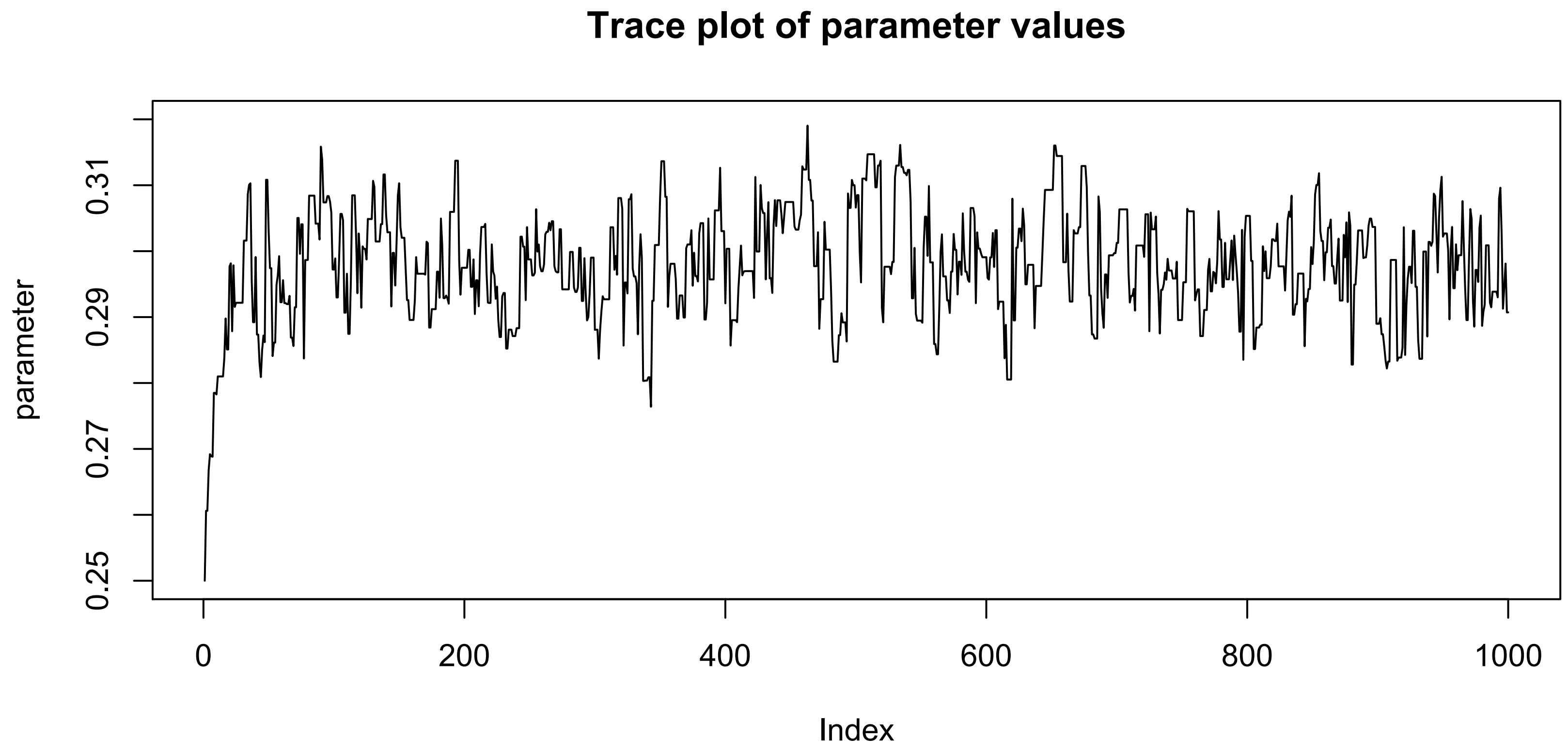
We also reward the old parameter for being hard to beat by counting it twice (so this counts as the 2nd *and* 3rd values in the chain)

```
parameter[3] <- parameter[2]
log_post[3] <- log_post[2]
```

MCMC: the theory

Repeat for the rest of the 1000 iterations, then look at the results:

```
plot(parameter, type='l')
```



Markov chain Monte Carlo

This algorithm is a basic implementation of MCMC!

- MCMC is used to approximate a posterior distribution by sampling from it

- Dependence on previous value is critical
  - Consecutive samples are *not* independent
  - Eventually a Markov chain will forget where it started, but it might take a while before it does this

This non-randomness of Markov chains has 2 important consequences:

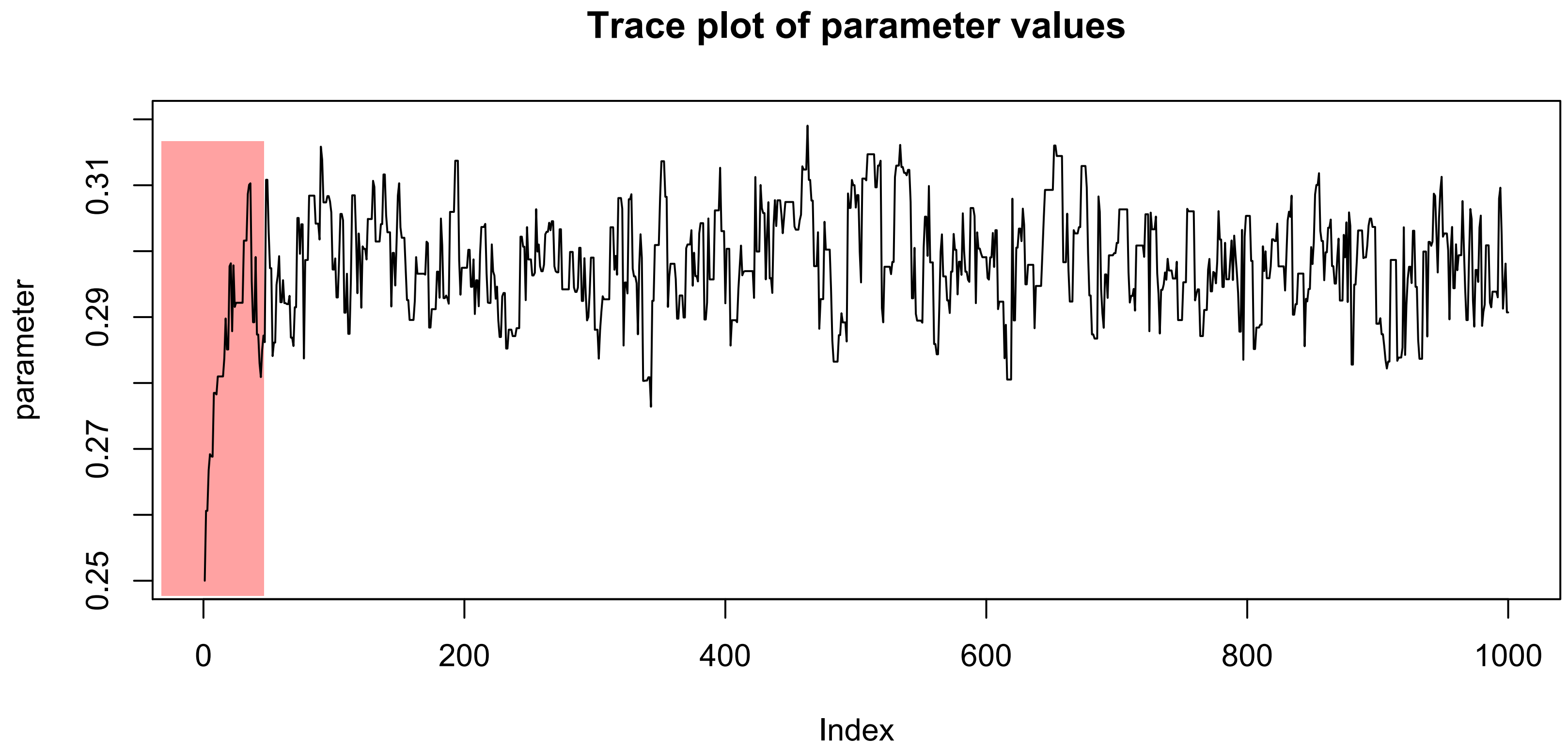
1. The starting values are not immediately forgotten
   - We need a *burn-in* period before sampling from the chains (aka identifying convergence on the stationary distribution)

2. A sample of 1000 auto-correlated iterations gives us less information than a sample of 1000 independent iterations
   - We need to know the *effective sample size* or *Monte Carlo error*

1. Burn-in

It is ESSENTIAL to identify and remove the burn-in period

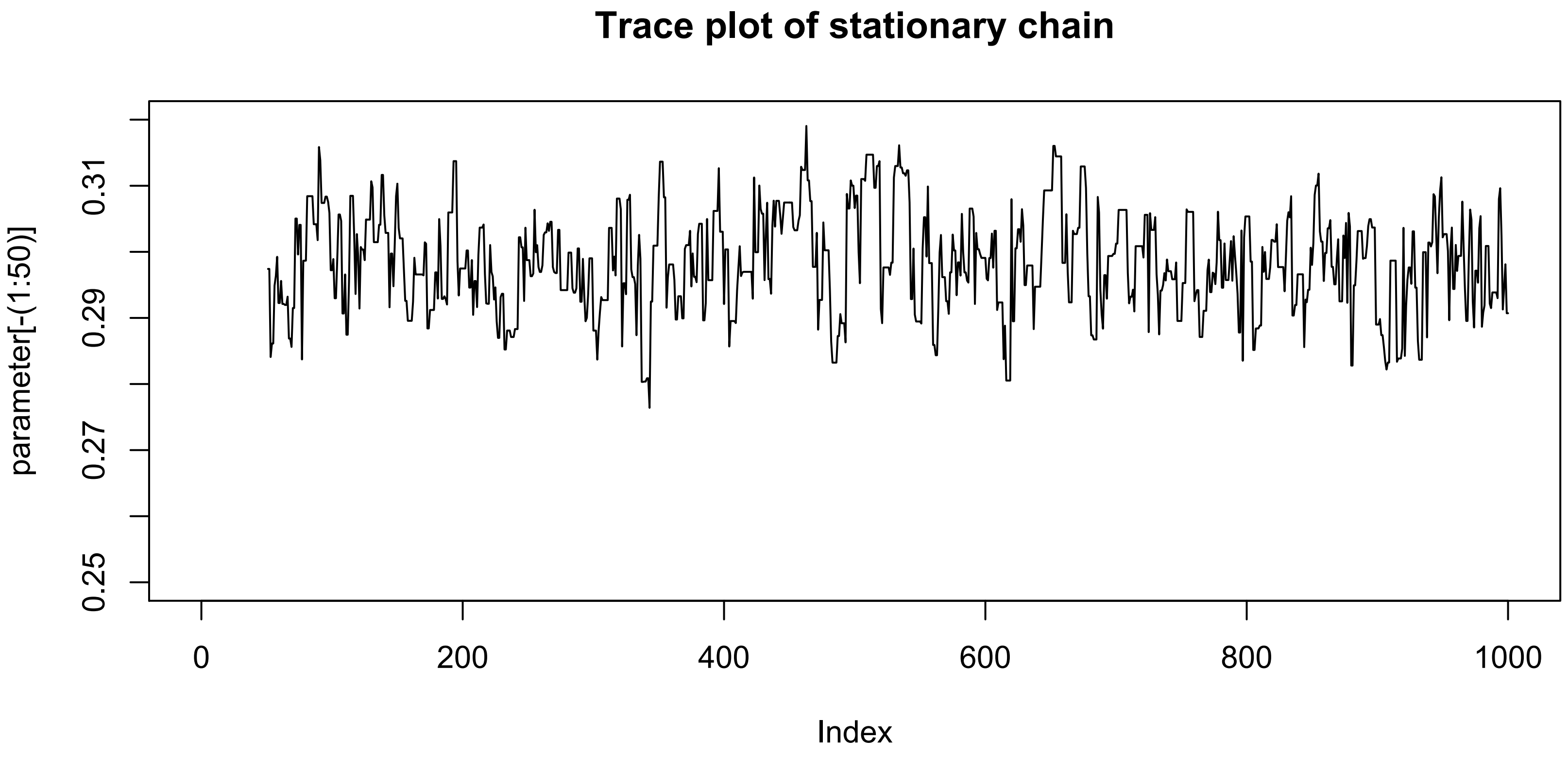
```
plot(parameter, type='l')
```



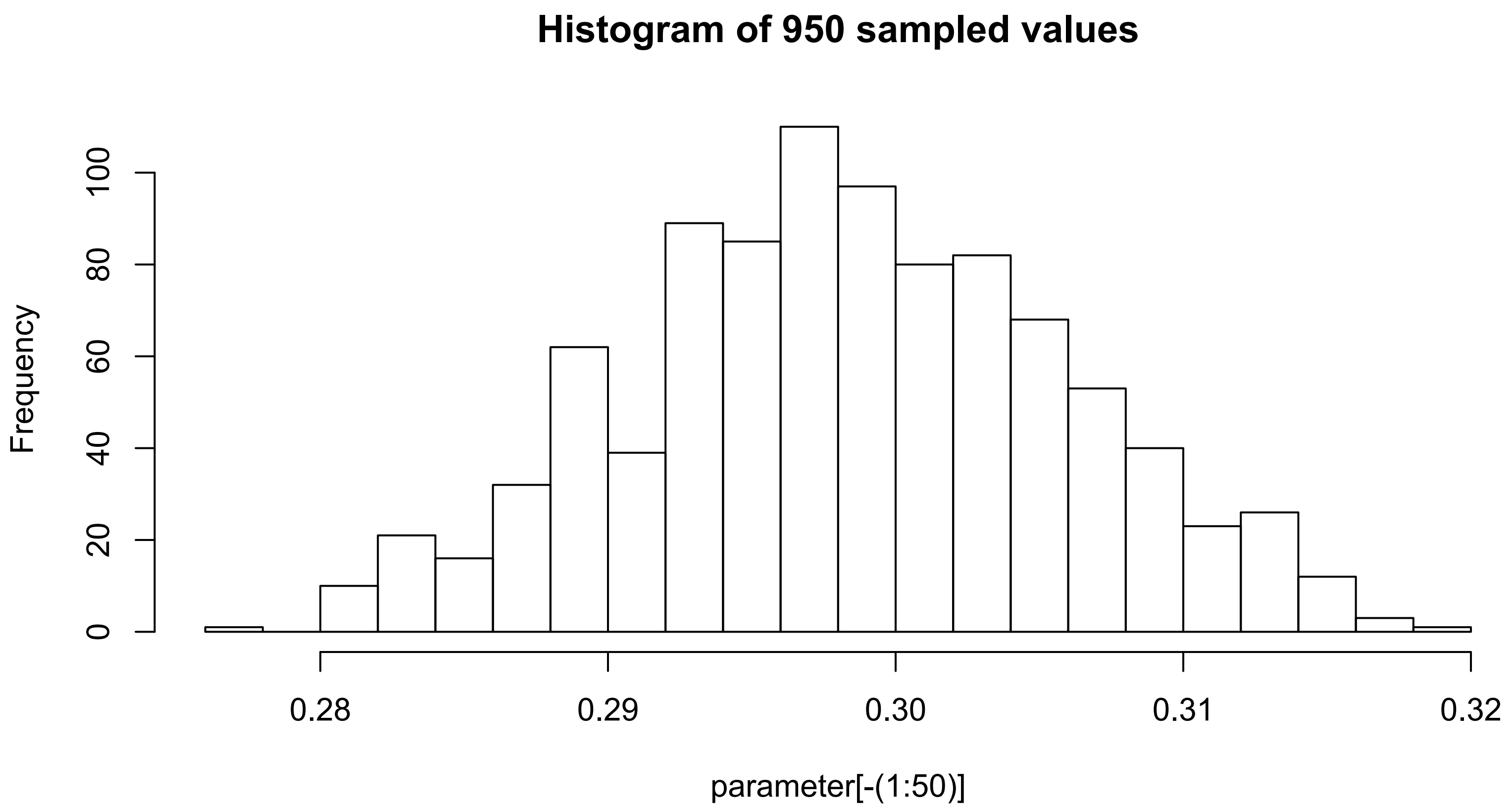
1. Burn-in

It is ESSENTIAL to identify and remove the burn-in period

```
plot(parameter[-(1:50)], type='l')
```

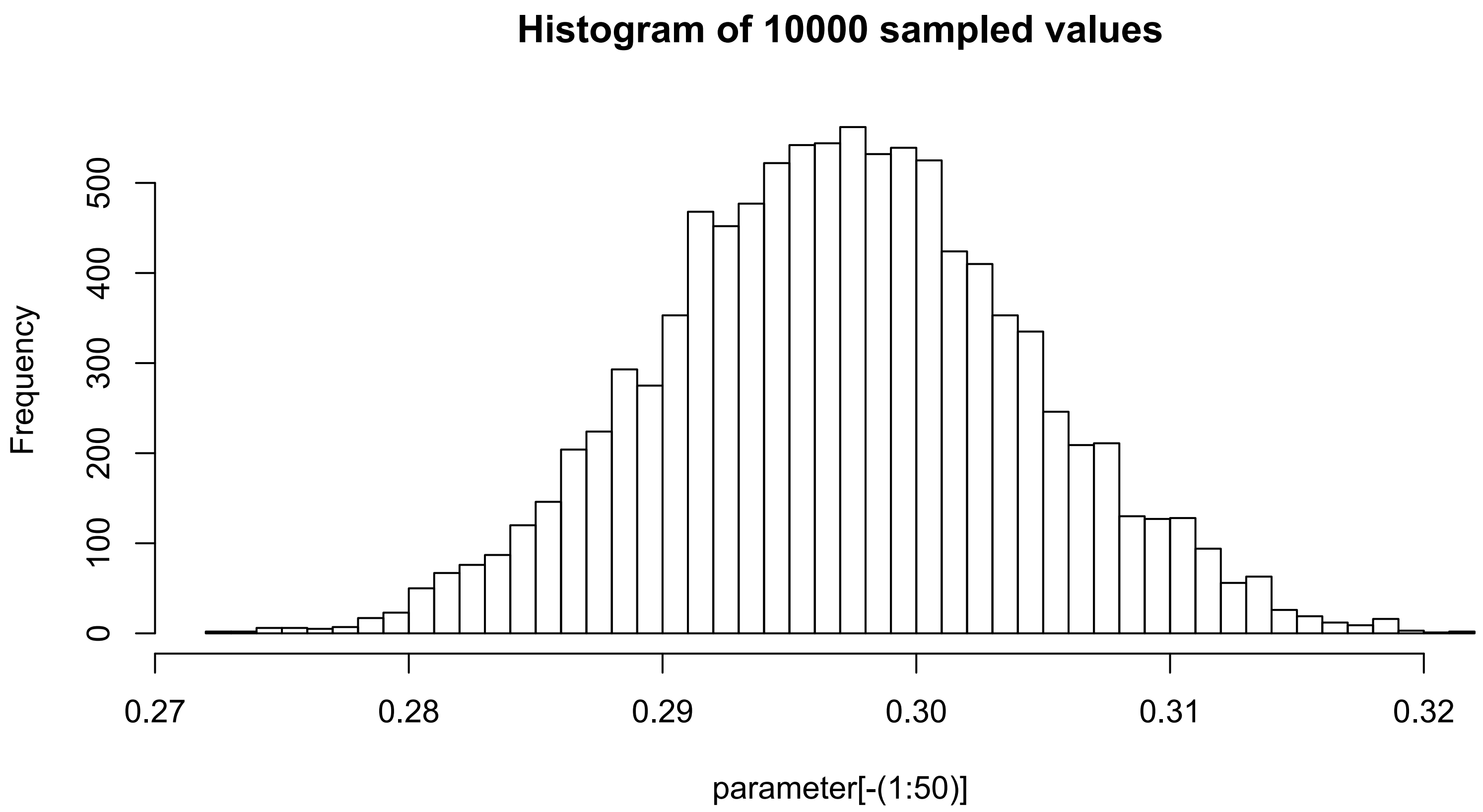


2. Effective sample size



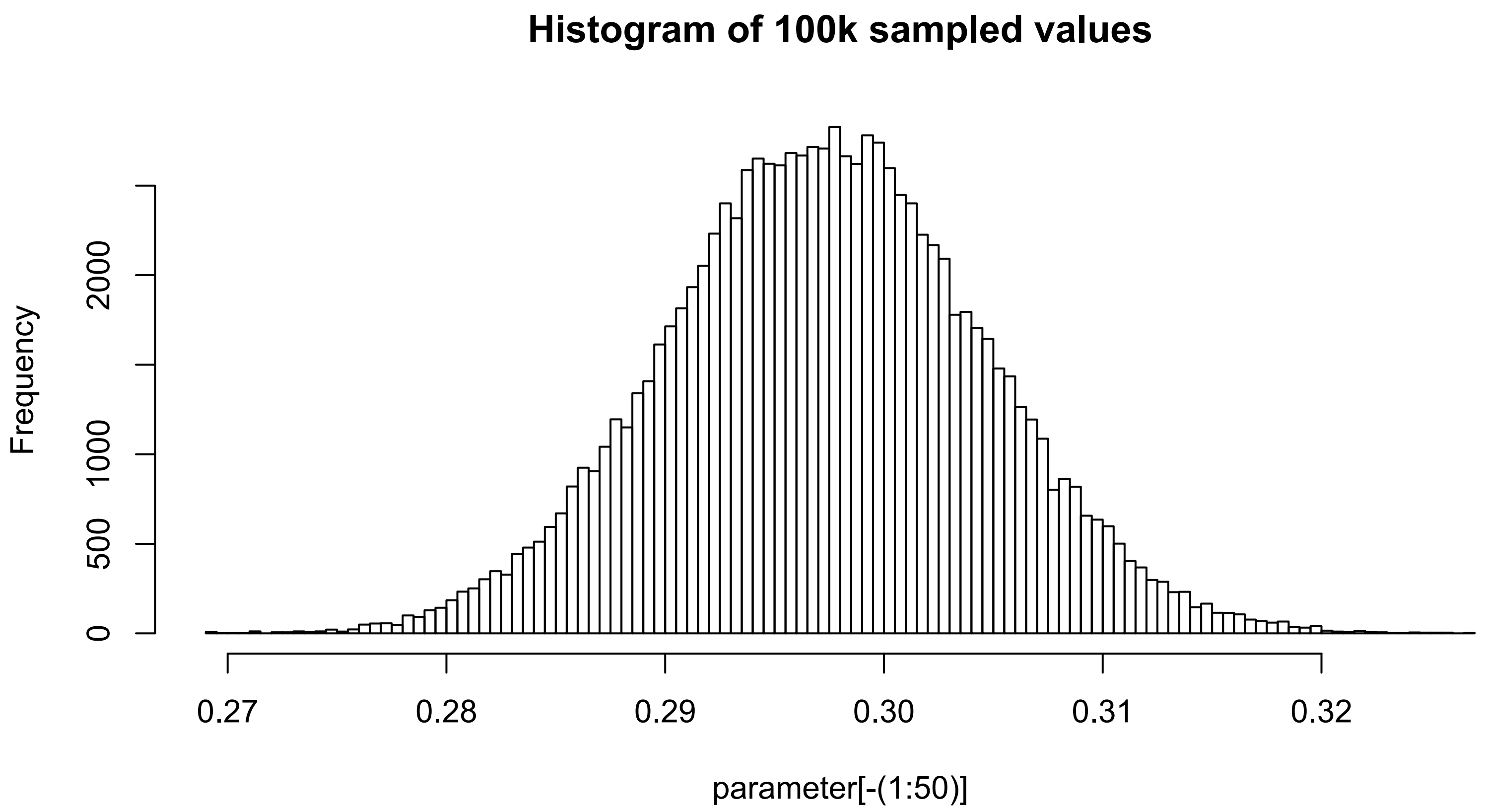
An effective sample size of 137 – not a good approximation

2. Effective sample size



An effective sample size of 1752 – a pretty good approximation

2. Effective sample size



An effective sample size of 16759 – a very good approximation

Dangers of MCMC

1. Problems related to the Markov chain
   - We didn’t remove enough burn-in iterations so our posterior is skewed by the parameter space from where we happened to start the initial values
   - Our chain hasn’t really found the stationary posterior, and is just sampling from a local solution
   - CONVERGENCE

2. Problems related to the Monte Carlo integration
   - We didn’t take enough samples to adequately represent the true posterior
   - EFFECTIVE SAMPLE SIZE / MONTE CARLO ERROR

How does MCMC work in practice?

Using BUGS

Bayesian (analysis) Using Gibbs Sampling

- ✓ Generic MCMC framework with ready-made algorithms
- ❖ Does not automatically resolve any of the dangers with MCMC sampling!

A BUGS model is compiled by the software into an MCMC sampler

- All variables in the model must be defined EXACTLY once
- NB: it is a symbolic language NOT a programming language

OpenBUGS

- Available for Windows, Linux and Mac (under emulation)
- No longer under active development

JAGS (Just Another Gibbs Sampler)

- Technically not BUGS, but almost identical
- Primarily designed to be called from within R
  - Packages available: rjags, runjags, R2jags, jagsUI

Procedure

1) Define a syntactically valid model (used to calculate the posterior)

2) Supply data and initial values for at least 2 chains (these are often optional)

3) Decide which variables to monitor

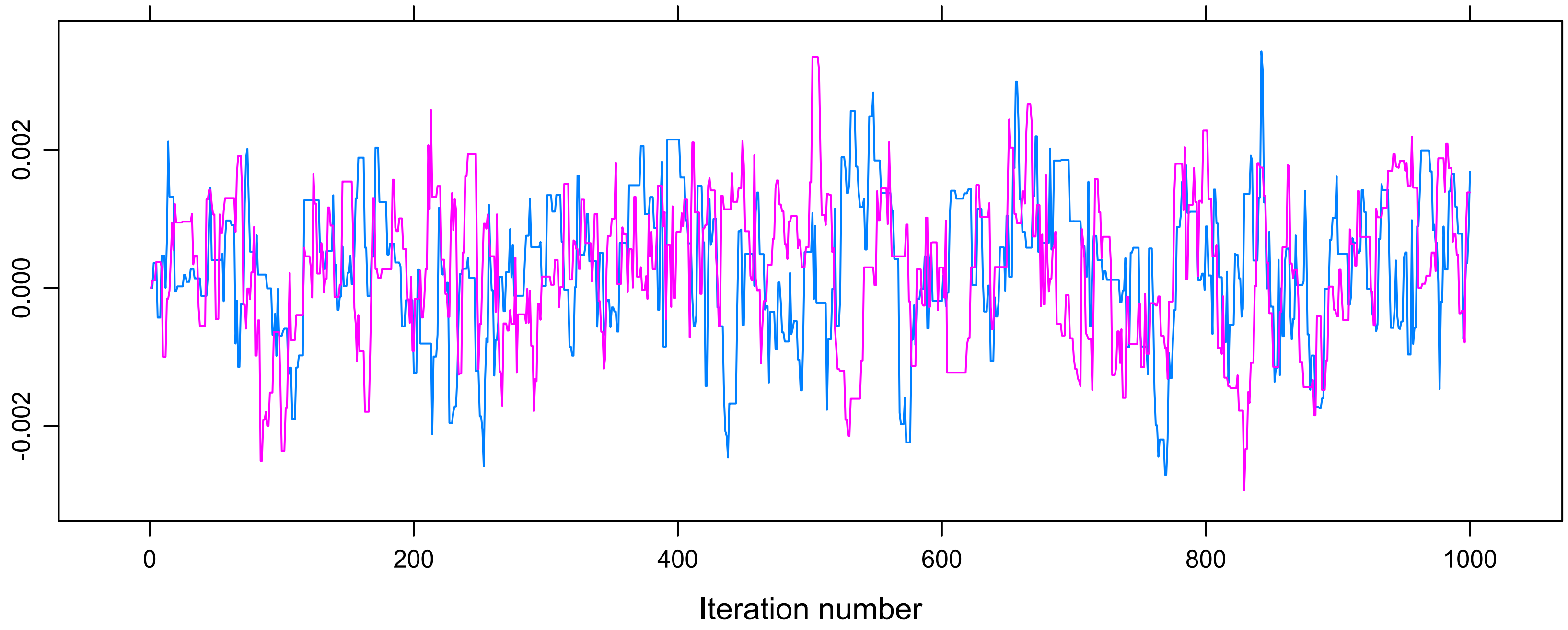
4) Run the model

5) Make sure the chains have converged
    - Brooks-Gelman-Rubin statistic compares the variance within chains to the variance between chains – ratio < 1.05 indicates that chains are sampling the same values so have probably converged
    - But always check the trace plots visually as well!

6) Make sure the number of sampled iterations is high enough
    - Minimum effective sample size of >400 for parameters of interest
    - Or equivalently when MC_error < 0.05 x sample SD

Visual assessment of convergence

Have these two chains (pink and blue) converged?

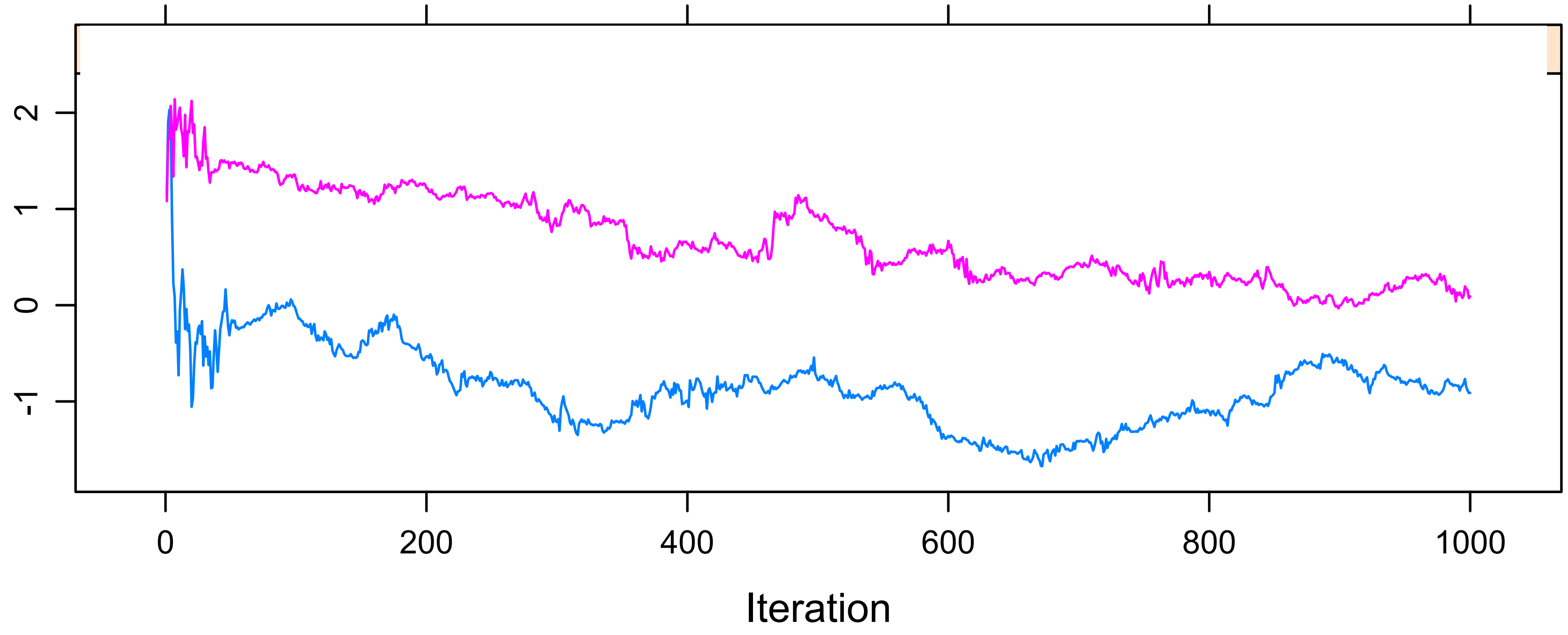


Looks good

[We are OK to look at the model results]

Visual assessment of convergence

Have these two chains (pink and blue) converged?



Looks bad

[Model results cannot be used]

A basic example: prevalence estimation

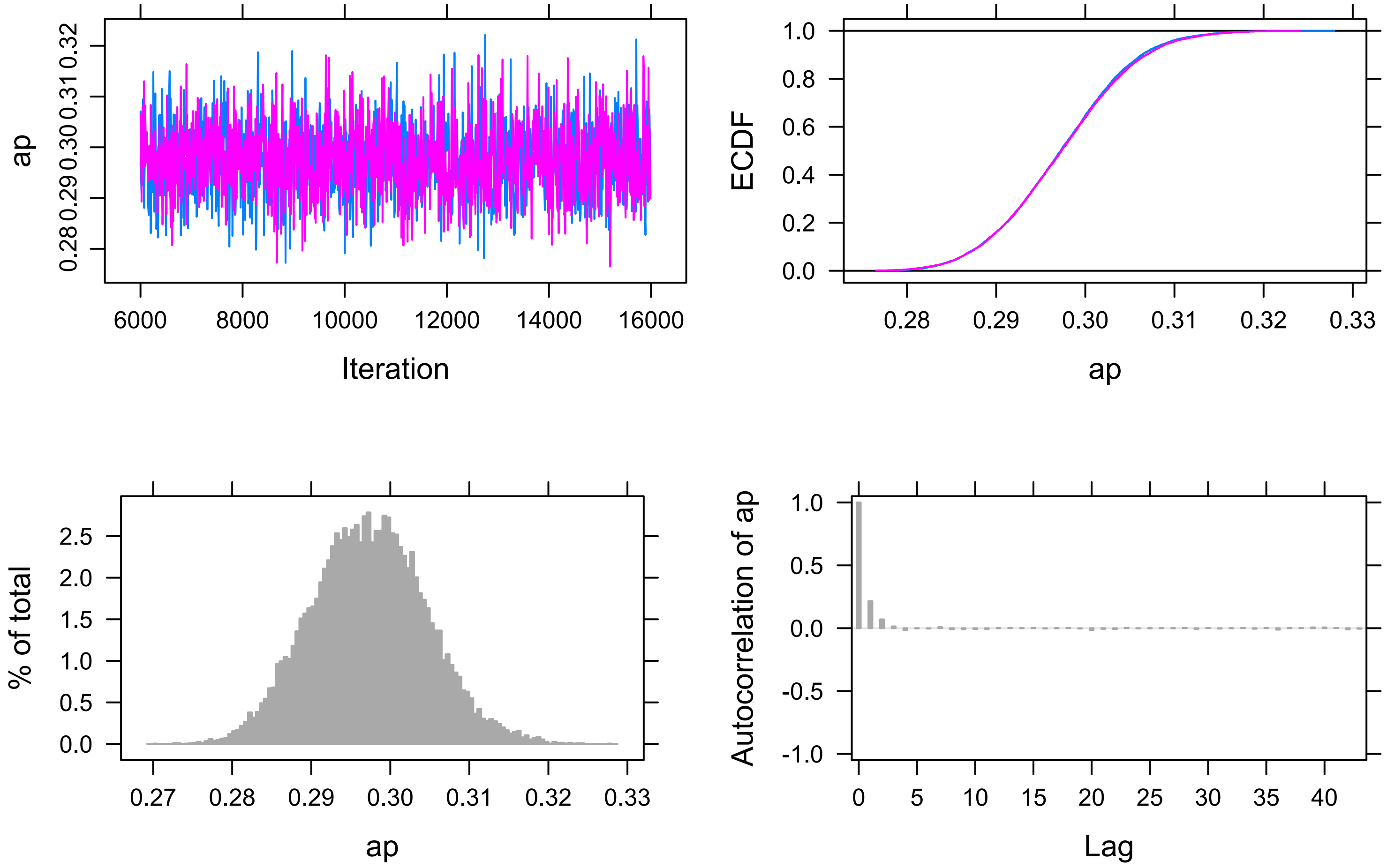
Define and run the model

Define the model, load the runjags library, and define the data in R:

```
bugs_model <- "
        model{
                Pos ~ dbinom(ap, N)
                # Uniform (non-informative) prior for apparent prev.
                ap ~ dbeta(1,1)

                #data# Pos, N
                #inits# ap
                #monitor# ap
        }
"
library('runjags’)

# Data:
Pos <- 1210
N <- 4072

# Initial values:
ap <- list(chain1=0.1, chain2=0.9)

results <- run.jags(bugs_model, n.chains=2, burnin=5000, sample=10000)
```

Check convergence

```
plot(results)
```



Check sample size and results

```
> results


JAGS model summary statistics from 20000 samples (chains = 2;
adapt+burnin = 6000):

       Lower95  Median Upper95    Mean        SD Mode
ap     0.28339 0.29719 0.31155 0.2972 0.0071436   --

       MCerr           MC%ofSD SSeff    AC.10   psrf
ap     0.000063509      0.9    12652 0.018843 1.0002

Total time taken: 0.8 seconds

# See also:
?runjags
```