Session 1

A practical introduction to MCMC

Matt Denwood

2021-06-28

Course Outline and Practicalities

Overview

Date/time:

- 28th June to 1st July 2021
- 09.00 - 12.30 daily
- Use the same Zoom link all week!

Overview

Date/time:

- 28th June to 1st July 2021
- 09.00 - 12.30 daily
- Use the same Zoom link all week!

Teachers:

- Matt Denwood (University Of Copenhagen)
- Nils Toft (IQinAbox)
- Søren Saxmose Nielsen (University Of Copenhagen)
- Maj Beldring Henningsen (University of Copenhagen)

HARMONY

COST action CA18208: [https://harmony-net.eu/about/]

Goals are to encourage the use of latent class models/methods for:

- Diagnostic test evaluation
- Determination of true prevalence
- Certification of disease freedom

Period: October 2019 - October 2023

Contact: info@harmony-net.eu

Practical information on the course

- All of the material is on the GitHub repository
  - We may tweak material as we go along
  - Remember to pull changes at the start of each day!
  - And click *refresh* in your browser! . . .
- Recordings of the sessions will put onto our website
  - The link will be sent out following the course
  - Feel free to share this with your colleagues

Practical information on the course

- All of the material is on the GitHub repository
  - We may tweak material as we go along
  - Remember to pull changes at the start of each day!
  - And click *refresh* in your browser! . . .
- Recordings of the sessions will put onto our website
  - The link will be sent out following the course
  - Feel free to share this with your colleagues
- Attendance registration is necessary for COST meetings
  - We will take care of this via Zoom

Background

Diagnostic test evaluation: with gold standard = simple!

```
library("tidyverse")
se <- c(1, 0.6)
sp <- c(1, 0.9)
N <- 1000
prevalence <- 0.25

data <- tibble(Status = rbinom(N, 1, prevalence)) %>%
  mutate(Test1 = rbinom(N, 1, se[1]*Status + (1-sp[1])*(1-Status))) %>%
  # Which is the same as:
  mutate(Test1 = Status) %>%
  mutate(Test2 = rbinom(N, 1, se[2]*Status + (1-sp[2])*(1-Status)))

(twoXtwo <- with(data, table(Status, Test2)))
##       Test2
## Status   0   1
##      0 656  68
##      1 127 149
```

Diagnostic test evaluation: with gold standard = simple!

```
library("tidyverse")
se <- c(1, 0.6)
sp <- c(1, 0.9)
N <- 1000
prevalence <- 0.25

data <- tibble(Status = rbinom(N, 1, prevalence)) %>%
  mutate(Test1 = rbinom(N, 1, se[1]*Status + (1-sp[1])*(1-Status))) %>%
  # Which is the same as:
  mutate(Test1 = Status) %>%
  mutate(Test2 = rbinom(N, 1, se[2]*Status + (1-sp[2])*(1-Status)))

(twoXtwo <- with(data, table(Status, Test2)))
##       Test2
## Status   0   1
##      0 656  68
##      1 127 149

(sensitivity <- twoXtwo[2,2] / sum(twoXtwo[2,1:2]))
## [1] 0.5398551
(specificity <- twoXtwo[1,1] / sum(twoXtwo[1,1:2]))
## [1] 0.9060773
```

Diagnostic test evaluation: no gold standard

Now we have both values of sensitivity and specificity <1...

```
se <- c(0.9, 0.6)
sp <- c(0.95, 0.9)
N <- 1000
prevalence <- 0.25

data <- tibble(Status = rbinom(N, 1, prevalence)) %>%
  mutate(Test1 = rbinom(N, 1, se[1]*Status + (1-sp[1])*(1-Status))) %>%
  mutate(Test2 = rbinom(N, 1, se[2]*Status + (1-sp[2])*(1-Status)))

with(data, table(Status, Test1))
##       Test1
## Status   0   1
##      0 691  35
##      1  32 242
with(data, table(Status, Test2))
##       Test2
## Status   0   1
##      0 667  59
##      1 104 170
```

In real life we don't know what Status is...

```
(twoXtwo <- with(data, table(Test1, Test2)))
##      Test2
## Test1   0   1
##     0 646  77
##     1 125 152
(sensitivity_1 <- twoXtwo[2,2] / sum(twoXtwo[1:2,2]))
## [1] 0.6637555
(sensitivity_2 <- twoXtwo[2,2] / sum(twoXtwo[2,1:2]))
## [1] 0.5487365
(specificity_1 <- twoXtwo[1,1] / sum(twoXtwo[1:2,1]))
## [1] 0.8378729
(specificity_2 <- twoXtwo[1,1] / sum(twoXtwo[1,1:2]))
## [1] 0.8934993
```

In real life we don't know what Status is...

```
(twoXtwo <- with(data, table(Test1, Test2)))
##      Test2
## Test1   0   1
##     0 646  77
##     1 125 152
(sensitivity_1 <- twoXtwo[2,2] / sum(twoXtwo[1:2,2]))
## [1] 0.6637555
(sensitivity_2 <- twoXtwo[2,2] / sum(twoXtwo[2,1:2]))
## [1] 0.5487365
(specificity_1 <- twoXtwo[1,1] / sum(twoXtwo[1:2,1]))
## [1] 0.8378729
(specificity_2 <- twoXtwo[1,1] / sum(twoXtwo[1,1:2]))
## [1] 0.8934993
```

So we will *always* under-estimate the Se and Sp of both tests!

The solution

- We need to assess the sensitivity and specificity of both tests against the true (but unknown) `Status` of each individual
- This unknown `Status` is called the `latent class`
  - Therefore we need to run a `latent class model` ...

The solution

- We need to assess the sensitivity and specificity of both tests against the true (but unknown) `Status` of each individual
- This unknown `Status` is called the `latent class`
  - Therefore we need to run a `latent class model` ...
- How can we implement a latent class model?
  - Frequentist statistical methods:
    - possible, but difficult
  - Bayesian statistical methods:
    - easier and much more commonly done!

Learning outcomes

By the end of the course you should be able to:

- Understand what a latent class model is, and how they can be used for diagnostic test evaluation
- Run basic latent class models using R and JAGS for real-world problems
- Interpret the results
- Understand the nuances and complexities associated with these types of analysis and the interpretation of the `latent class`

Revision

Bayes Rule

Bayes' theorem is at the heart of Bayesian statistics:

$$P(\theta|Y) = \frac{P(\theta) \times P(Y|\theta)}{P(Y)}$$

Bayes Rule

Bayes' theorem is at the heart of Bayesian statistics:

$$P(\theta|Y) = \frac{P(\theta) \times P(Y|\theta)}{P(Y)}$$

Where: θ is our parameter value(s);

Y is the data that we have observed;

$P(\theta|Y)$ is the posterior probability of the parameter value(s);

$P(\theta)$ is the prior probability of the parameters;

$P(Y|\theta)$ is the likelihood of the data given the parameters value(s);

$P(Y)$ is the probability of the data, integrated over parameter space.

- In practice we usually work with the following:

$$P(\theta|Y) \propto P(\theta) \times P(Y|\theta)$$

- In practice we usually work with the following:

$$P(\theta|Y) \propto P(\theta) \times P(Y|\theta)$$

- Our Bayesian posterior is therefore always a combination of the likelihood of the data, and the parameter priors
- But for more complex models the distinction between what is 'data' and 'parameters' can get blurred!

MCMC

- A way of obtaining a numerical approximation of the posterior
- Highly flexible
- Not inherently Bayesian but most widely used in this context
- Assessing convergence is essential, otherwise we may not be summarising the true posterior
- Our chains are correlated so we need to consider the effective sample size

Bayesian MCMC vs Frequentist ML

Advantages:

- Very flexible modelling framework
- More natural interpretation of confidence intervals (credible intervals)
- Ability to incorporate prior information

Bayesian MCMC vs Frequentist ML

Advantages:

- Very flexible modelling framework
- More natural interpretation of confidence intervals (credible intervals)
- Ability to incorporate prior information

Disadvantages:

- More computationally intensive
- More emphasis on the practictioner to ensure the output is reliable
- Requirement to incorporate prior information

Everyone up to speed?

Any questions so far?

Anything unclear?

Session 1: A practical introduction to MCMC

MCMC in Practice

- We can write a Metropolis algorithm ourselves, but this is complex and inefficient
- There are a number of general purpose langauages that allow us to define the problem and leave the details to the software:
  - WinBUGS/OpenBUGS
    - Bayesian inference Using Gibbs Sampling
  - JAGS
    - Just another Gibbs Sampler
  - Stan
    - Named in honour of Stanislaw Ulam, pioneer of the Monte Carlo method

JAGS

- JAGS uses the BUGS language
  - This is a declarative (non-procedural) language
  - The order of statements does not matter
  - The compiler converts our model syntax into an MCMC algorithm with appropriately defined likelihood and prior
  - You can only define each variable once!!!

JAGS

- JAGS uses the BUGS language
  - This is a declarative (non-procedural) language
  - The order of statements does not matter
  - The compiler converts our model syntax into an MCMC algorithm with appropriately defined likelihood and prior
  - You can only define each variable once!!!
- Different ways to run JAGS from R:
  - rjags, runjags, R2jags, jagsUI
- See http://runjags.sourceforge.net/quickjags.html
  - This is also in the GitHub folder

A simple JAGS model might look like this:

```
model{
  # Likelihood part:
  Positives ~ dbinom(prevalence, N)

  # Prior part:
  prevalence ~ dbeta(1, 1)

  # Hooks for automatic integration with R:
  #data# Positives, N
  #monitor# prevalence
  #inits# prevalence
}
```

There are two model statements:

- The first:

```
Positives ~ dbinom(prevalence, N)
```

states that the number of `Positive` test samples is Binomially distributed with probability parameter `prevalence` and total trials `N`

There are two model statements:

- The first:

```
Positives ~ dbinom(prevalence, N)
```

states that the number of `Positive` test samples is Binomially distributed with probability parameter `prevalence` and total trials `N`

- The second:

```
prevalence ~ dbeta(1,1)
```

states that our prior probability distribution for the parameter `prevalence` is Beta(1,1), which is the same as Uniform(0,1)

There are two model statements:

- The first:

```
Positives ~ dbinom(prevalence, N)
```

states that the number of `Positive` test samples is Binomially distributed with probability parameter `prevalence` and total trials `N`

- The second:

```
prevalence ~ dbeta(1,1)
```

states that our prior probability distribution for the parameter `prevalence` is Beta(1,1), which is the same as Uniform(0,1)

These are very similar to the likelihood and prior functions defined in the preparatory exercise (although this prior is less informative)

The other lines in this model:

```
#data# Positives, N
#monitor# prevalence
#inits# prevalence
```

are automated hooks that are only used by runjags

The other lines in this model:

```
#data# Positives, N
#monitor# prevalence
#inits# prevalence
```

are automated hooks that are only used by runjags

Compared to our Metropolis algorithm, this JAGS model is:

- Eaiser to write and understand
- More efficient (lower autocorrelation)
- Faster to run

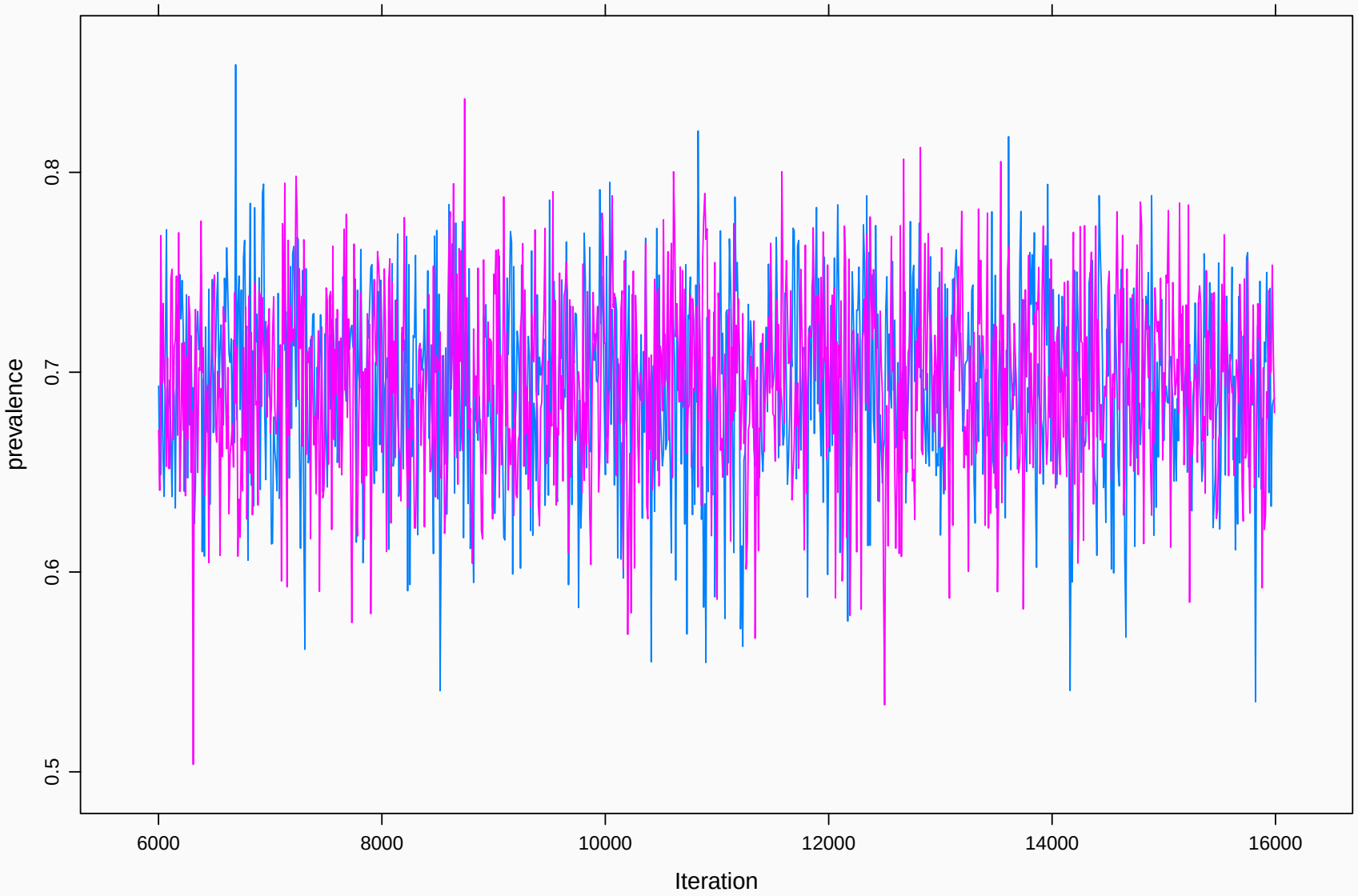
To run this model, copy/paste the code above into a new text file called "basicjags.txt" in the same folder as your current working directory. Then run:

```
library('runjags')

# data to be retrieved by runjags
Positives <- 70
N <- 100

# initial values to be retrieved by runjags:
prevalence <- list(chain1=0.05, chain2=0.95)

results <- run.jags('basicjags.txt', n.chains=2, burnin=5000,
↪  sample=10000)
```

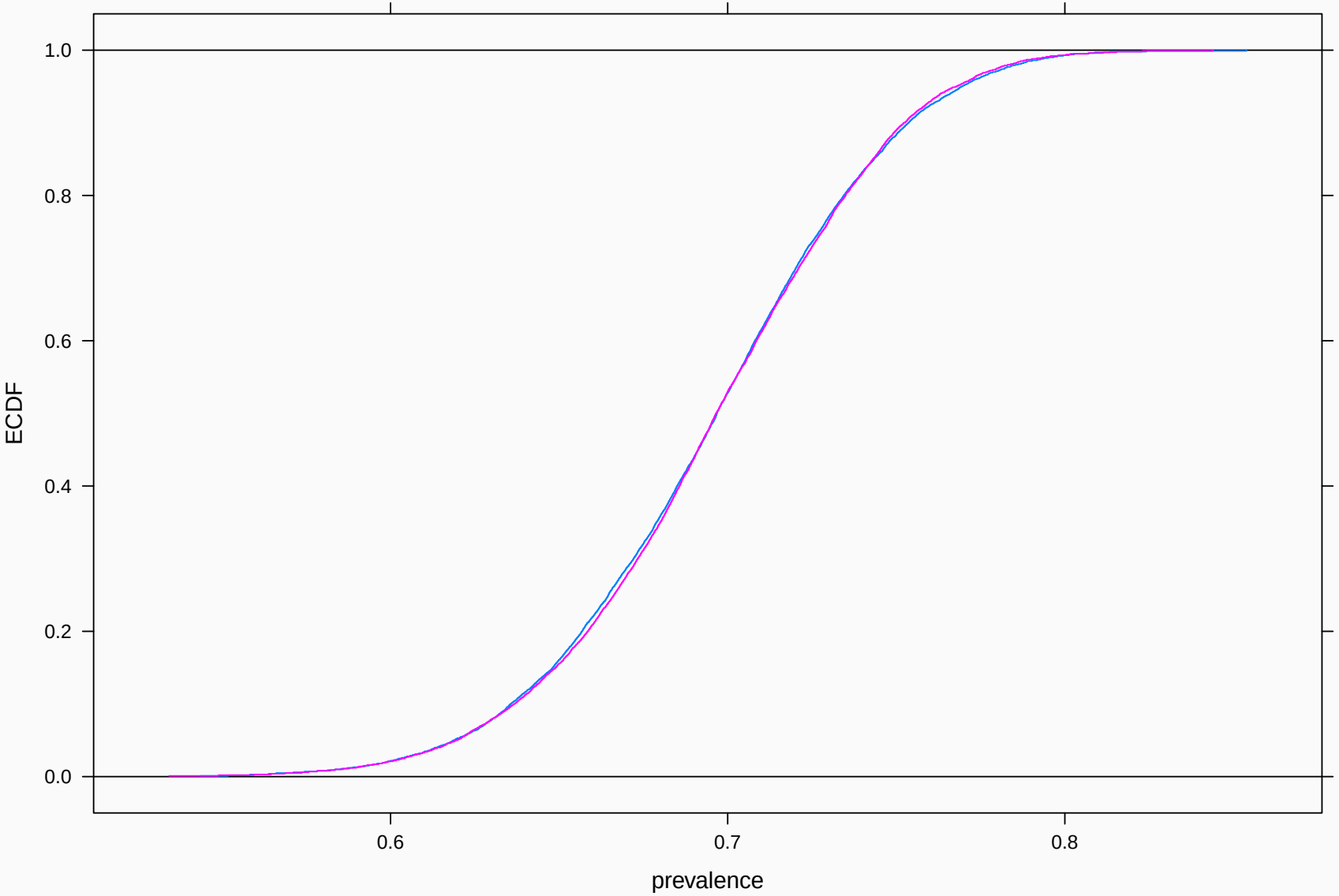
First check the plots for convergence:

```
plot(results)
```

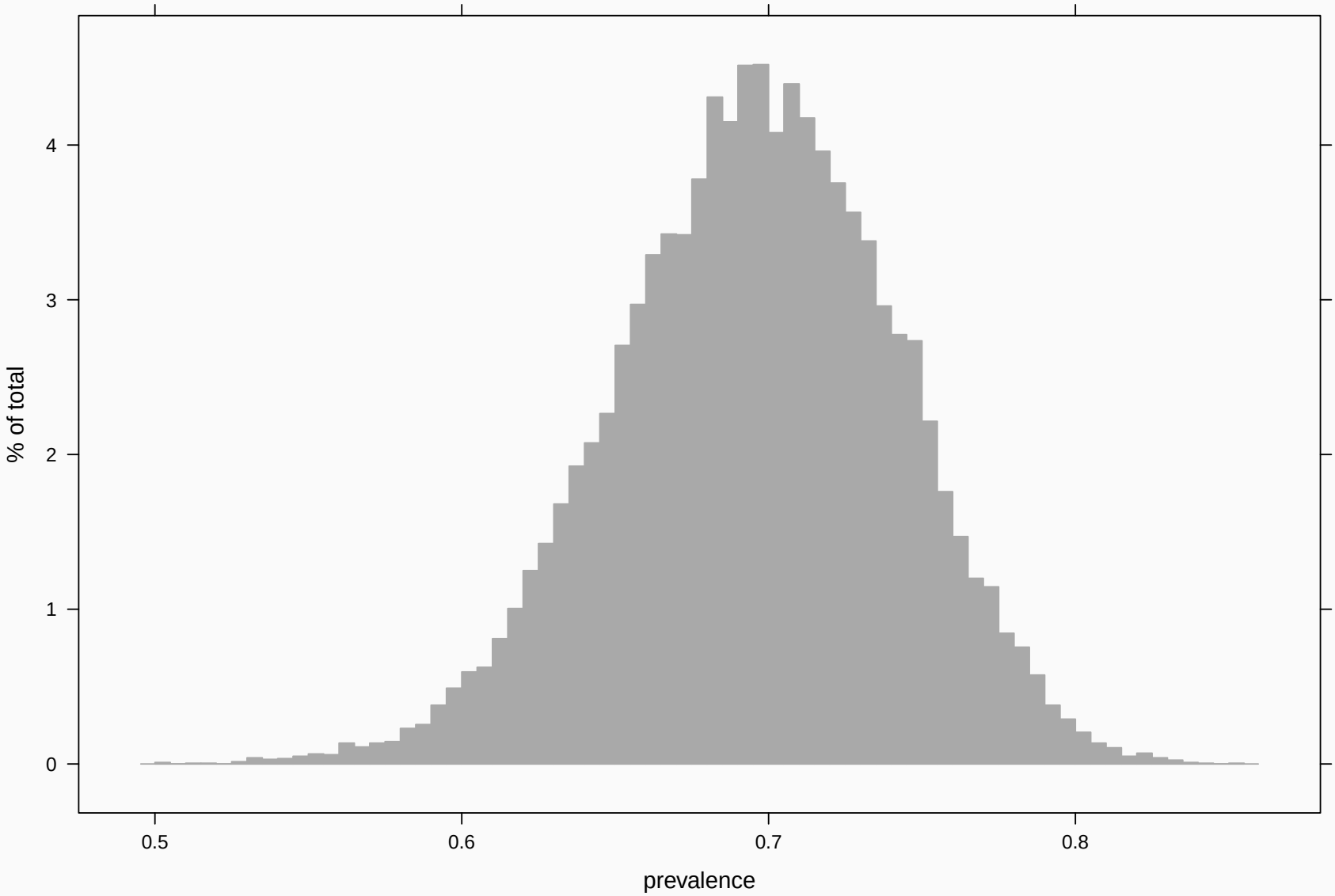
Trace plots: the two chains should be stationary:



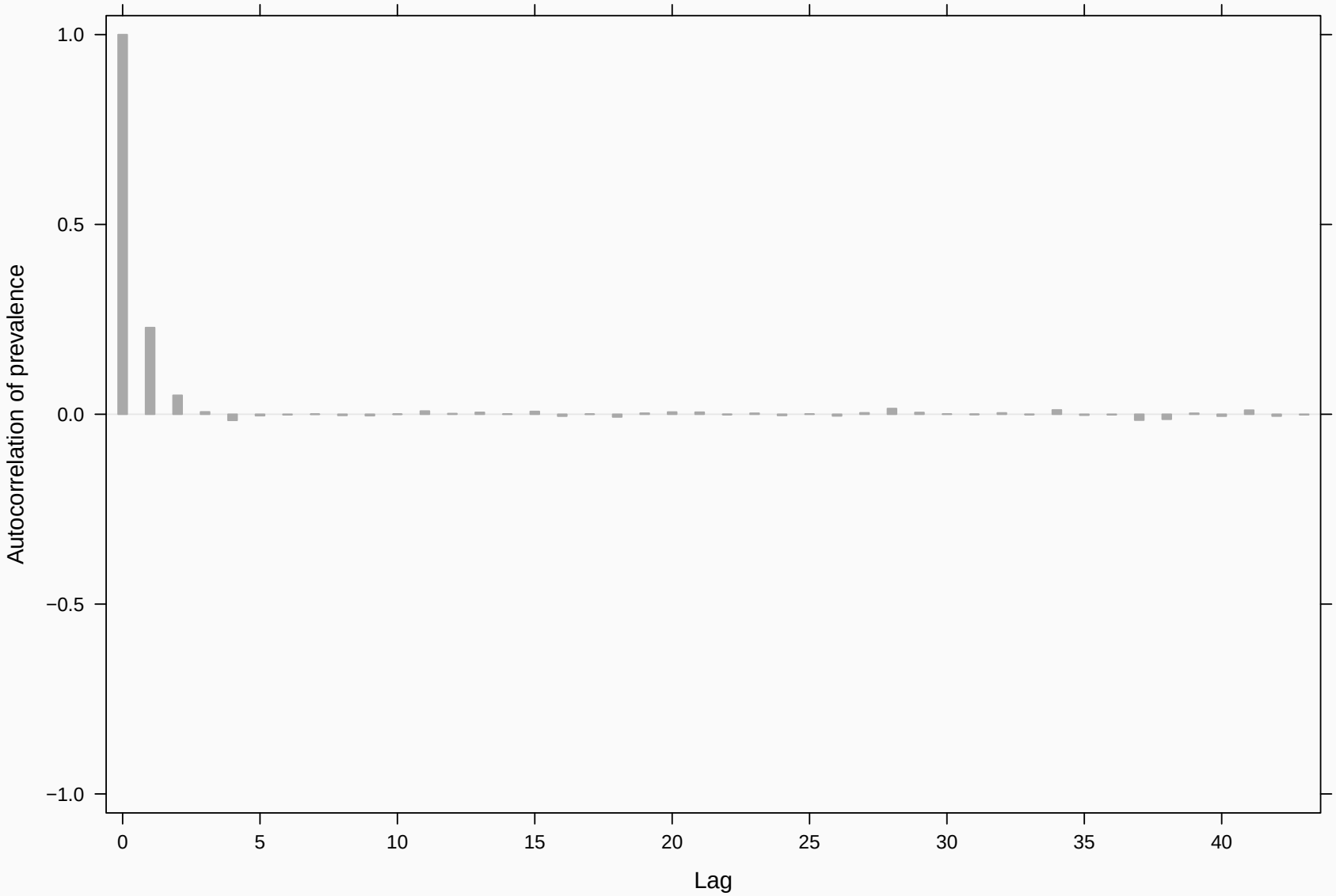
ECDF plots: the two chains should be very close to each other:



Histogram of the combined chains should appear smooth:



Autocorrelation plot tells you how well behaved the model is:



Note: for multiple parameters, use the ‘back’ button to cycle through plots!

Note: for multiple parameters, use the 'back' button to cycle through plots!

Then check the effective sample size and Gelman-Rubin statistic:

```
results
##
## JAGS model summary statistics from 20000 samples (chains = 2;
↪  adapt+burnin = 6000):
##
##            Lower95  Median Upper95    Mean       SD Mode
## prevalence 0.60911 0.69667 0.78618 0.69557 0.045581   --
##
##                 MCerr MC%ofSD SSeff      AC.10   psrf
## prevalence 0.00040654     0.9 12571 0.00012156 1.0001
##
## Total time taken: 0.2 seconds
```

Reminder: we want SSeff $>$ 1000 and psrf $<$ 1.05

Introduction to practical sessions

Each practical session will consist of:

1. Some general/philosophical points to consider
2. One or more practical exercises for everyone to complete
3. One or more additional (optional) exercise for those that finish the main exercise early
4. A wrap-up discussion to reinforce the key messages

Introduction to practical sessions

Each practical session will consist of:

1. Some general/philosophical points to consider
2. One or more practical exercises for everyone to complete
3. One or more additional (optional) exercise for those that finish the main exercise early
4. A wrap-up discussion to reinforce the key messages

Consideration points are given in the PDF

The exercises (and solutions) are only in the HTML versions

- We have approximately 1 hour per practical session
  - 30-45 minutes for exercises, 15-30 minutes for discussion

- We have approximately 1 hour per practical session
  - 30-45 minutes for exercises, 15-30 minutes for discussion
- We will allocate you in pairs / threes to breakout rooms for the exercises (partly randomly, partly based on your institution)
  - If you need help please use the "Ask For Help" feature from your breakout room
  - Otherwise we will drop into the breakout rooms periodically to see how you are getting on!

Practical Session 1

Points to consider

1. What are the advantages and disadvantages of Bayesian MCMC relative to more standard frequentist likelihood-based methods?

2. `Identifiability` refers to the ability of a model to extract useful information from a dataset for a particular set of parameters. What 3 things affect whether or not a model/parameter will be identifiable?

The exercises can be found in Session_1.html!

Summary

- MCMC allows flexibility in models BUT requires more computational resource and user awareness
  - Convergence
  - Effective sample size
- Bayesian methods allow priors to be used BUT necessitate that priors are used
- Models are less likely to be identifiable if they:
  - Are more complex
  - Have less informative priors
  - Do not have sufficient data
- There is often a disparity between the model we would like to run and the model we can run given the data available