Hands-on training session 1

Introduction, revision and running basic models

Matt Denwood Giles Innocent

2020-02-09

Course Outline and Practicalities

Overview

Date/time:

- 19th February 2020
- 14.00 - 15.30

Teachers:

- Matt Denwood (presenter)
- Giles Innocent

Accessing the teaching material

Demonstration of GitHub and where the files are...

Preparation

- Do we all have R and JAGS installed?
- Have we all looked at the preparation material?

Session 1a: Theory and application of MCMC

Bayes Rule

Bayes' theorem is at the heart of Bayesian statistics. This states that:

$P(\theta|Y) = \frac{P(\theta) \times P(Y|\theta)}{P(Y)}$ 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) given the data and priors; $P(\theta)$ is the prior probability of the parameters BEFORE we had observed the data; $P(Y|\theta)$ is the likelihood of the data given the parameters value(s), as discussed above; and $P(Y)$ is the probability of the data, integrated over all parameter space.

Note that $P(Y)$ is rarely calculable except in the simplest of cases, but is a constant for a given model. So 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 (revision)

TODO

- Brief highlight of important points re convergence and effective sample size

Exercise

TODO

Using the same function to estimate prevalence for different priors and datas

Including positives $= 0$

Care with convergence and sample size

Solution

TODO

Session 1b: Working with basic models (apparent prevalence)

MCMC software

We have seen that we can write a Metropolis algorithm but this is complex and inefficient

There are a number of general purpose langauages that allow us to define the problem in a faily intuitive manner and leave the details to the software. Among the latter are BUGS (Bayesian inference Using Gibbs Sampling) JAGS (Just another Gibbs Sampler) and STAN (named in honour of Stanislaw Ulam, pioneer of the Monte Carlo method).

JAGS

JAGS is a declarative (non-procedural) programming language. That is the order of statements does not matter. When implemented the interpreted just considers the appropriate line of code to interpret in terms of likelihood and prior. That is why you can only define each variable (LHS) once.

Different ways to run JAGS from R:

- rjags
- runjags
- R2jags
- jagsUI

A simple JAGS model might look like this:

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

  # Prior part:
  prevalence ~ dbeta(2, 2)

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

There are two model statements:

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

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

```
prevalence ~ dbeta(2,2)
```

states that our prior probability distribution for the parameter *prevalence* is Beta(2,2)

These are very similar to the likelihood and prior functions defined in the preparatory exercise

The other lines in this model are automated hooks to allow the runjags package to make sure that the correct R objects are passed to R as data, and that we get samples for the parameter we want to monitor. These statements are ignored by JAGS.

One of the advantages of using something like JAGS or STANis that much of the detail of the coding is shielded from us which means that the code is accessible to a wider variety of audiences. Running this model is much easier (and more efficient) than the Metropolis algorithm code.

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

Then run:

```
library('runjags')
```

```
## Attaching runjags (version 2.0.4-6) and setting
↪  user-specified options
```

```
# data to be retrieved by runjags:
Positives <- 7
TotalTests <- 10

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

```
results <- run.jags('basicjags.bug', n.chains=2)
```

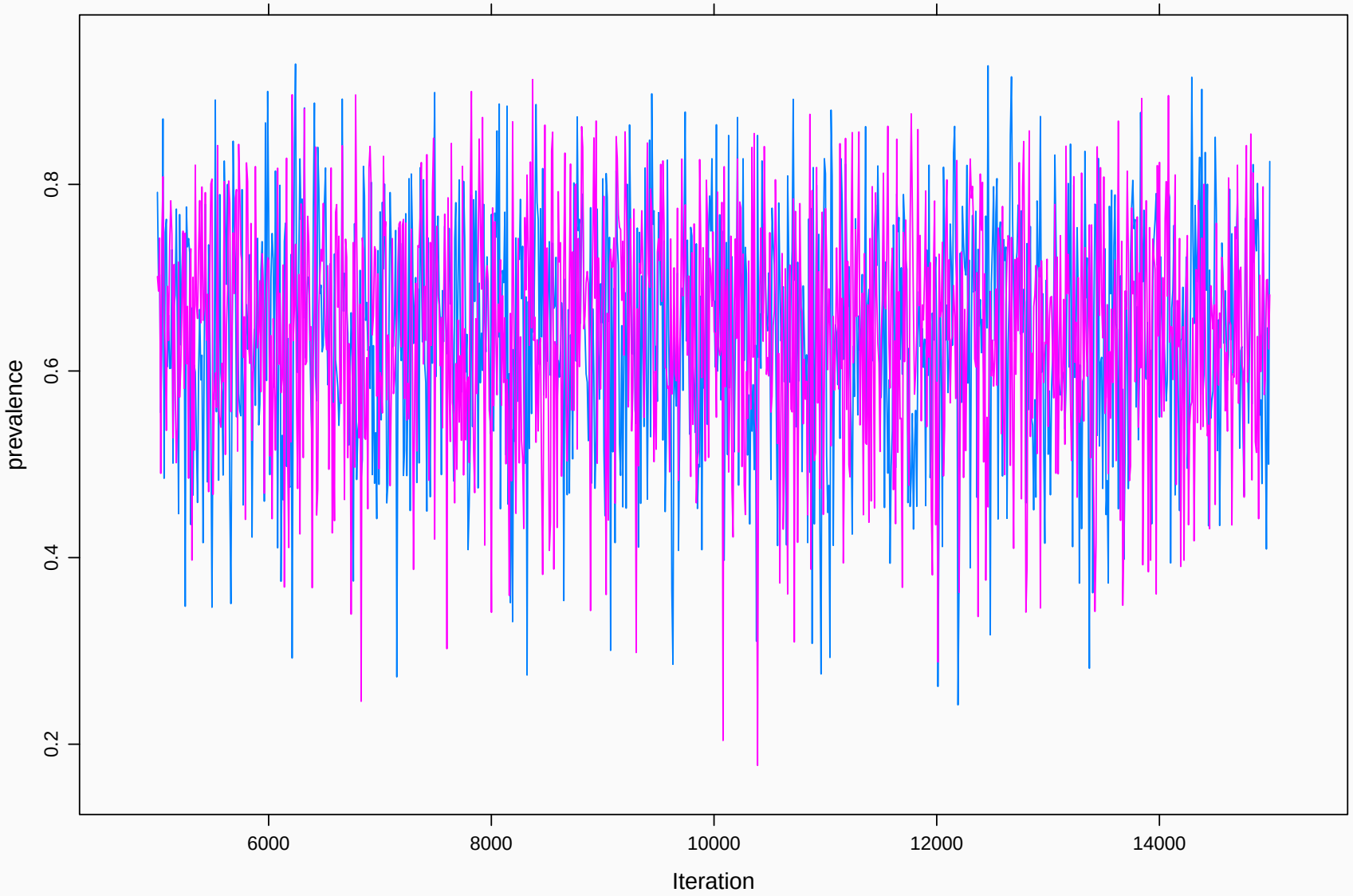
```
## Loading required namespace: rjags
```

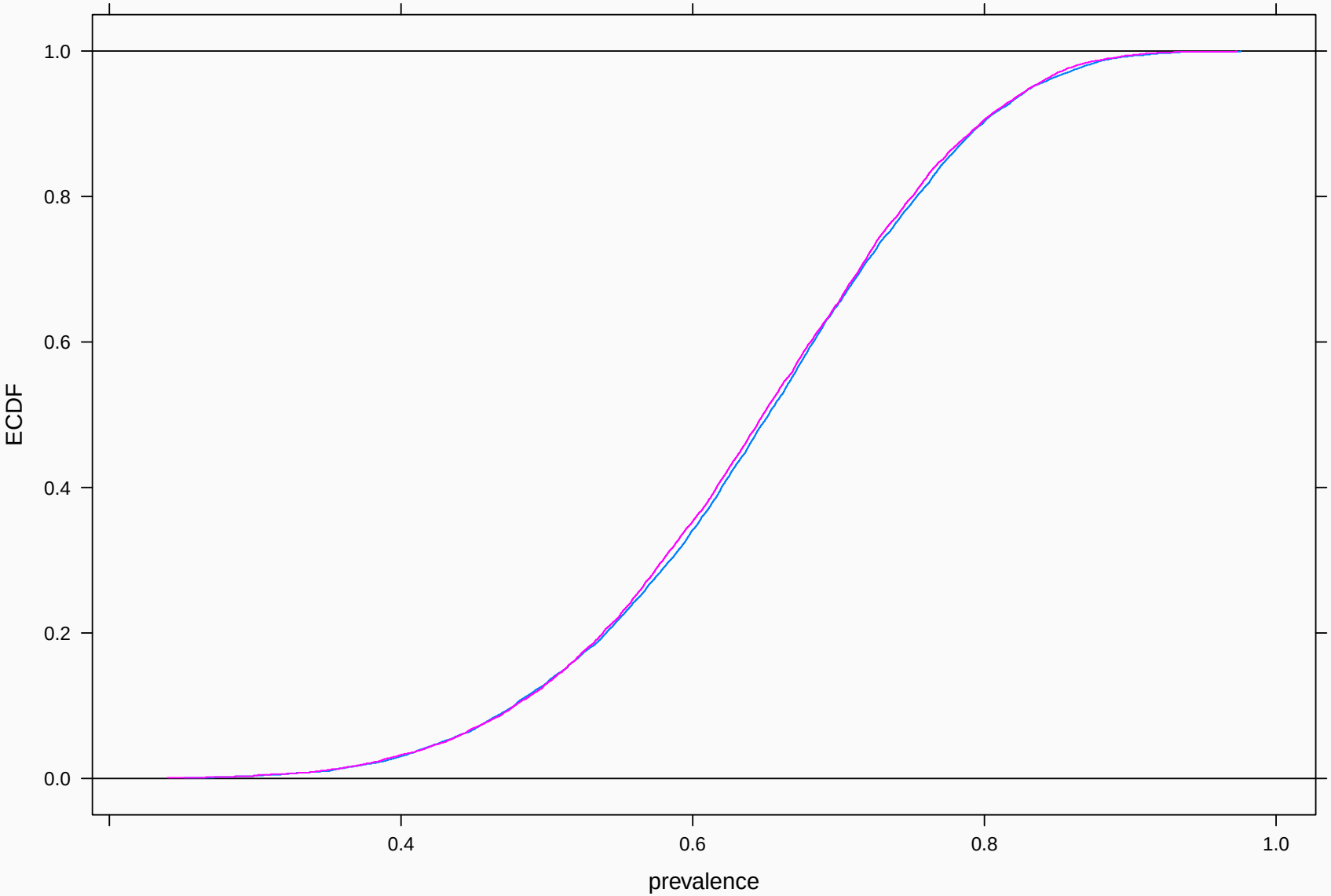
```
## Compiling rjags model...
## Calling the simulation using the rjags method...
## Adapting the model for 1000 iterations...
## Burning in the model for 4000 iterations...
## Running the model for 10000 iterations...
## Simulation complete
## Calculating summary statistics...
## Calculating the Gelman-Rubin statistic for 1
↪  variables....
## Finished running the simulation
```

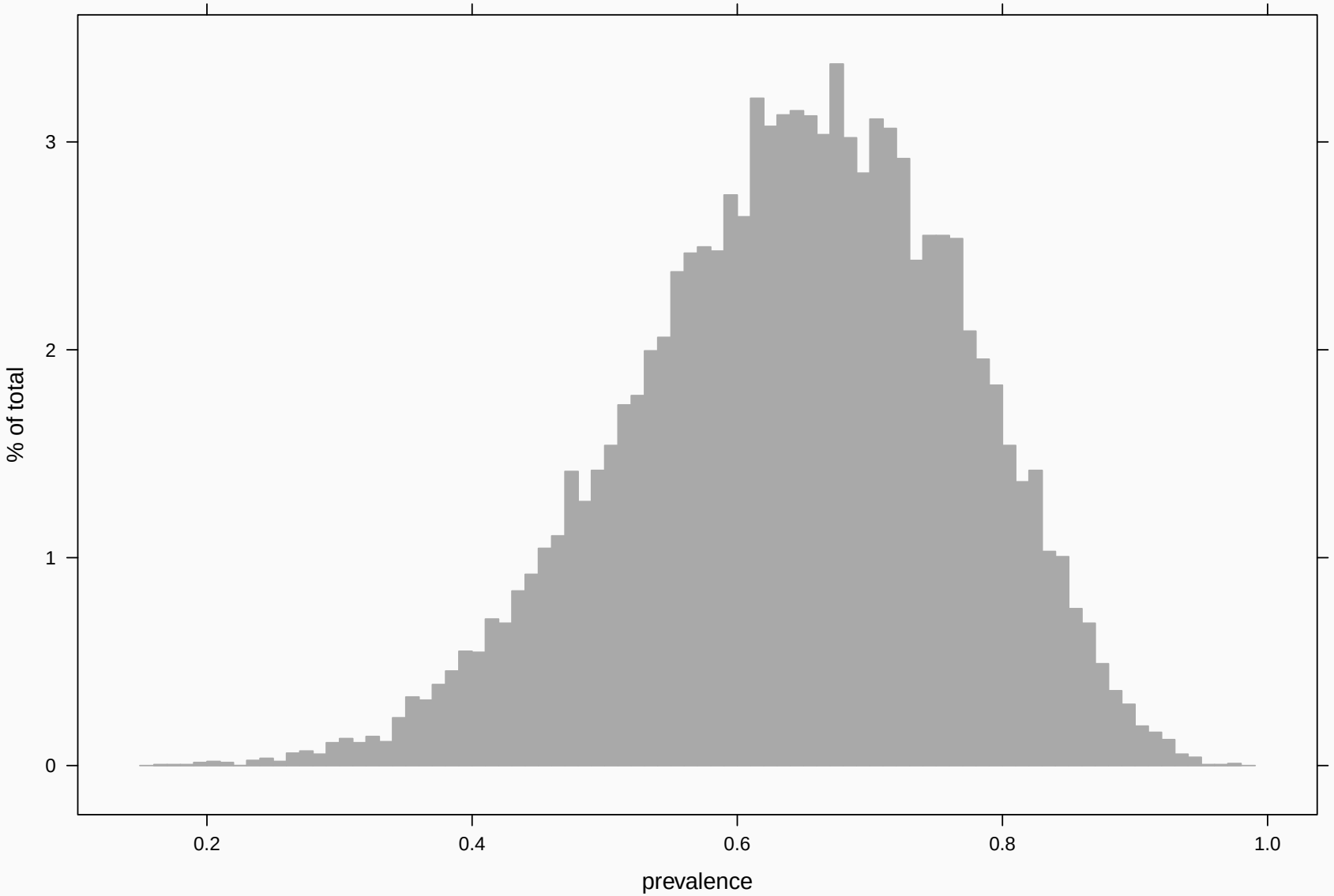
You should see some updates then it will be finished

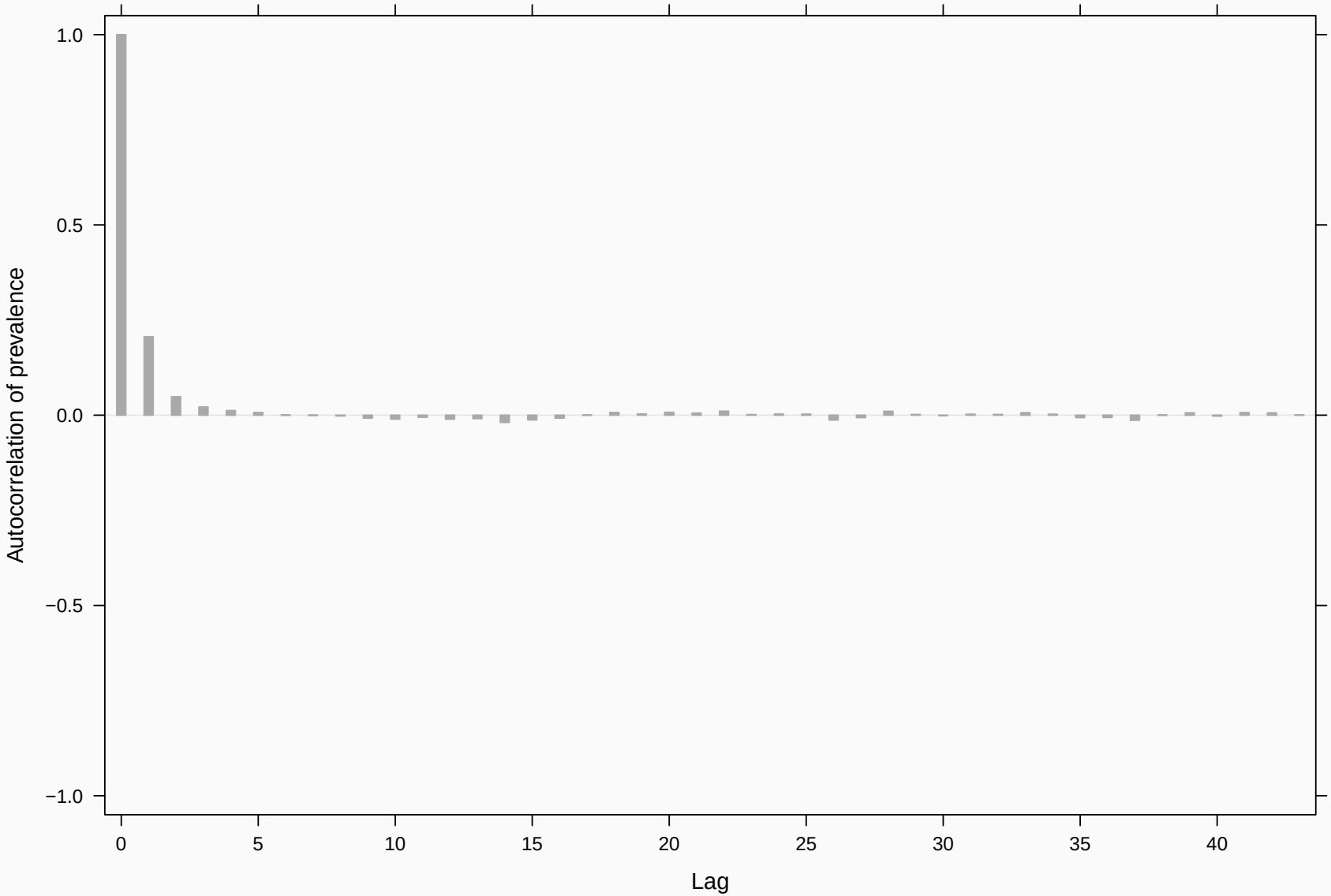
First check for convergence based on trace plots:

```
plot(results)
```


prevalence
0.2
0.4
0.6
0.8
6000
8000
10000
12000
14000
Iteration


1.0
0.8
0.6
0.4
0.2
0.0
ECDF
0.4
0.6
0.8
1.0
prevalence


% of total
0
1
2
3
0.2
0.4
0.6
0.8
1.0
prevalence


Autocorrelation of prevalence
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25
30
35
40
Lag

Then check the effective sample size (SSeff) and Gelman-Rubin statistic (psrf):

```
results
```

```
##
## JAGS model summary statistics from 20000 samples (chains = 2; adapt+burnin = 5000):
##
##             Lower95  Median Upper95    Mean      SD Mode     MCerr MC%ofSD
## prevalence 0.40896 0.64994 0.87733 0.64299 0.12271 0.6614 0.0010695     0.9
##
##             SSeff     AC.10   psrf
## prevalence 13165 -0.010679 1.0002
##
## Total time taken: 0.9 seconds
```

If these both look OK (psrf < 1.05 and SSeff > 1000) then

Other runjags options

Using .RNG.seed

Parallel chains

extend.jags

Exercise

TODO

Run this model for yourselves

Adjust data and priors

Solution

TODO

Session 1c: Basics of latent-class models (imperfect test)

Imperfect tests

TODO

Usually, however, we do not have a perfect test to know how many are truly positive (negative). Say we have a test with known sensitiveity of 0.8, and known specificity of 0.95. We know therefore that:

$$Prev_{obs} = (Prev_{true} \times Se) + ((1 - Prev_{true}) \times (1 - Sp))$$
$$\implies Prev_{true} = \frac{Prev_{obs} - (1 - Sp)}{Se - (1 - Sp)}$$

Model specification

How to write the model in JAGS, and specify priors for Se and Sp

Priors

Where to get priors in real life

PriorGen pacakge

Exercise

Run the same model in runjags with imperfect Se/Sp with priors we give them based on mean and 95% CI

Compare results