Session 1

A practical introduction to MCMC

Matt Denwood

2021-06-21

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!

Teachers:

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

Motivation

Diagnostic test evaluation: with gold standard

- Evaluation of a new test against a gold standard is simple!
- Let's simulate some data to illustrate:

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

library("tidyverse")
```

```
## - Attaching packages --------------------- tidyverse 1.3.1 -
```

```
## v ggplot2 3.3.4     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## - Conflicts ---------------------- tidyverse_conflicts() -
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Diagnostic test evaluation: no gold standard

- What happens when the reference test is imperfect?

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

library("tidyverse")

data <- tibble(Status = rbinom(N, 1, prevalence)) %>%
  mutate(Test_1 = rbinom(N, 1, se[1]*Status +
  ↪  (1-sp[1])*(1-Status))) %>%
  mutate(Test_2 = rbinom(N, 1, se[2]*Status +
  ↪  (1-sp[2])*(1-Status))) %>%
  print()
```

```
## # A tibble: 1,000 x 3
##    Status Test_1 Test_2
##     <int>  <int>  <int>
##  1      0      0      0
```

The solution

- We can

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

Preparation

- Any questions so far? Anything unclear?
- Do we all have R and JAGS installed?
- Can we all access the teaching material from GitHub?

Preparation

- Any questions so far? Anything unclear?
- Do we all have R and JAGS installed?
- Can we all access the teaching material from GitHub?

Any problems: see us during the first practical session!

Session 1a: Theory and application of MCMC

MCMC in Practice

- 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 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, 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*

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)

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:

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

are automated hooks that are only used by runjags

The other lines in this model:

```
#data# Positives, TotalTests
#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.bug" in the same folder as your current working directory. Then run:

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

```
## Attaching runjags (version 2.2.0-2) and setting user-specified
↪  options
```

```
##
## Attaching package: 'runjags'
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

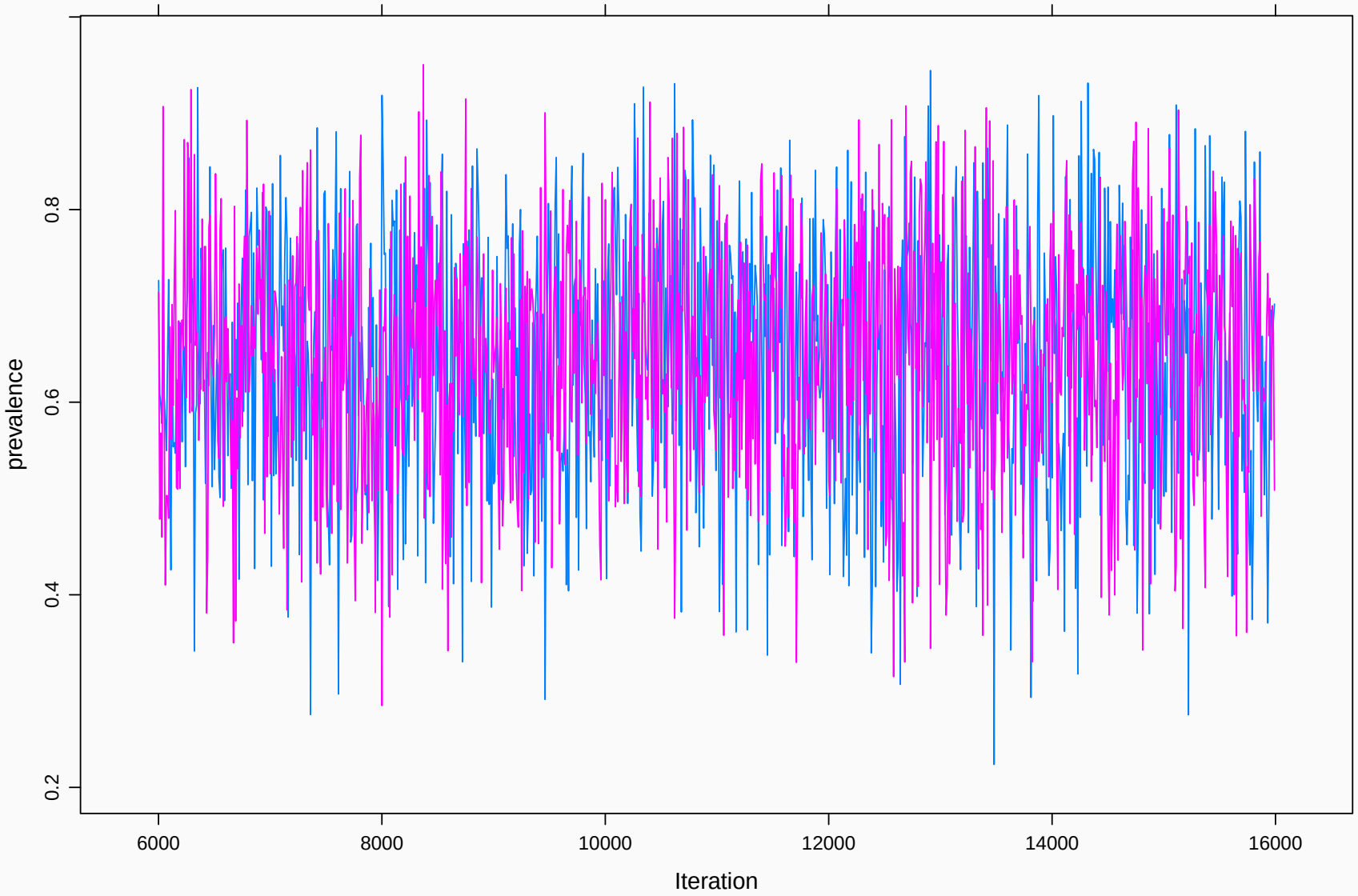
```
# data to be retrieved by runjags:
Positives <- 7
TotalTests <- 10

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

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

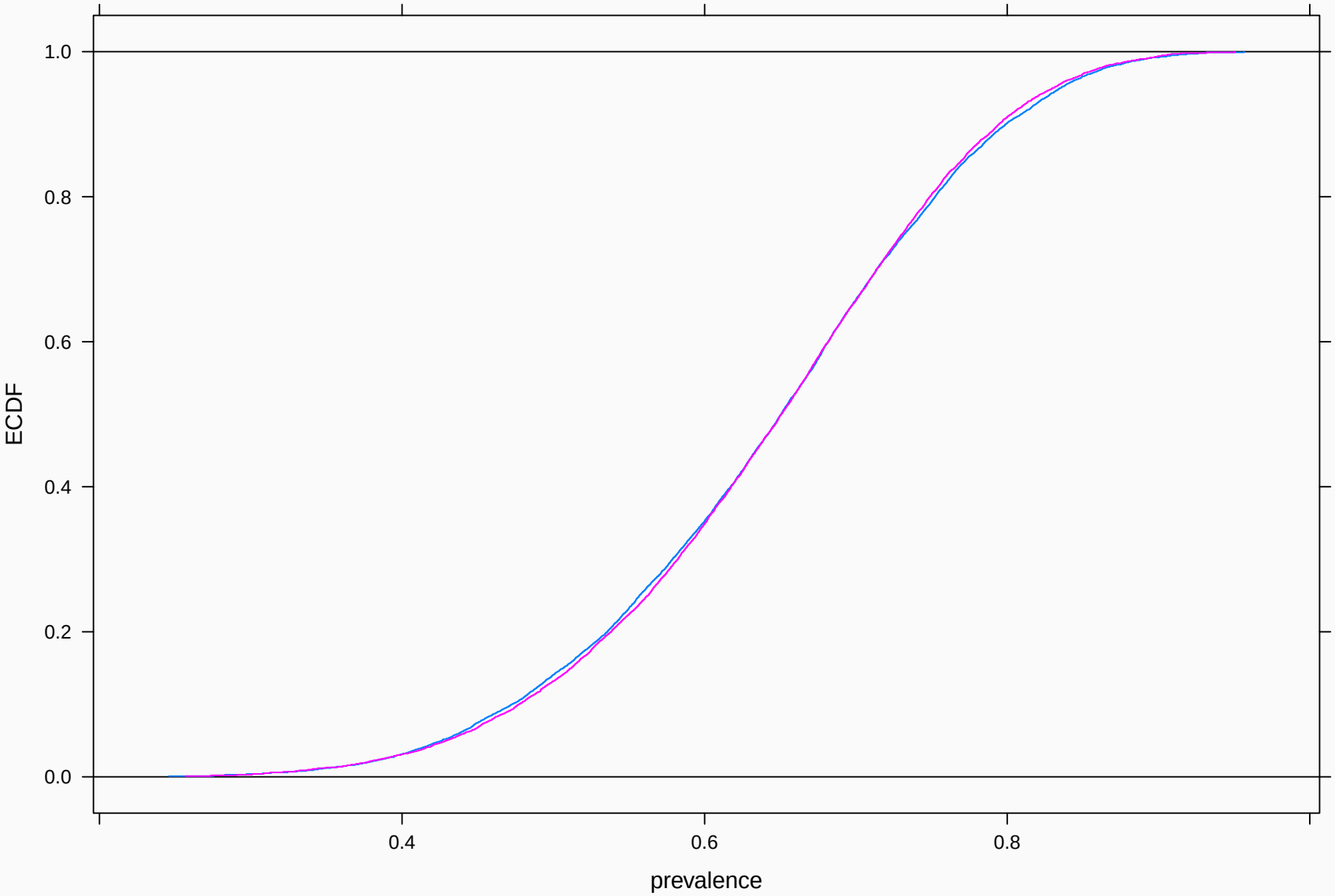
First check the plots for convergence:

```
plot(results)
```

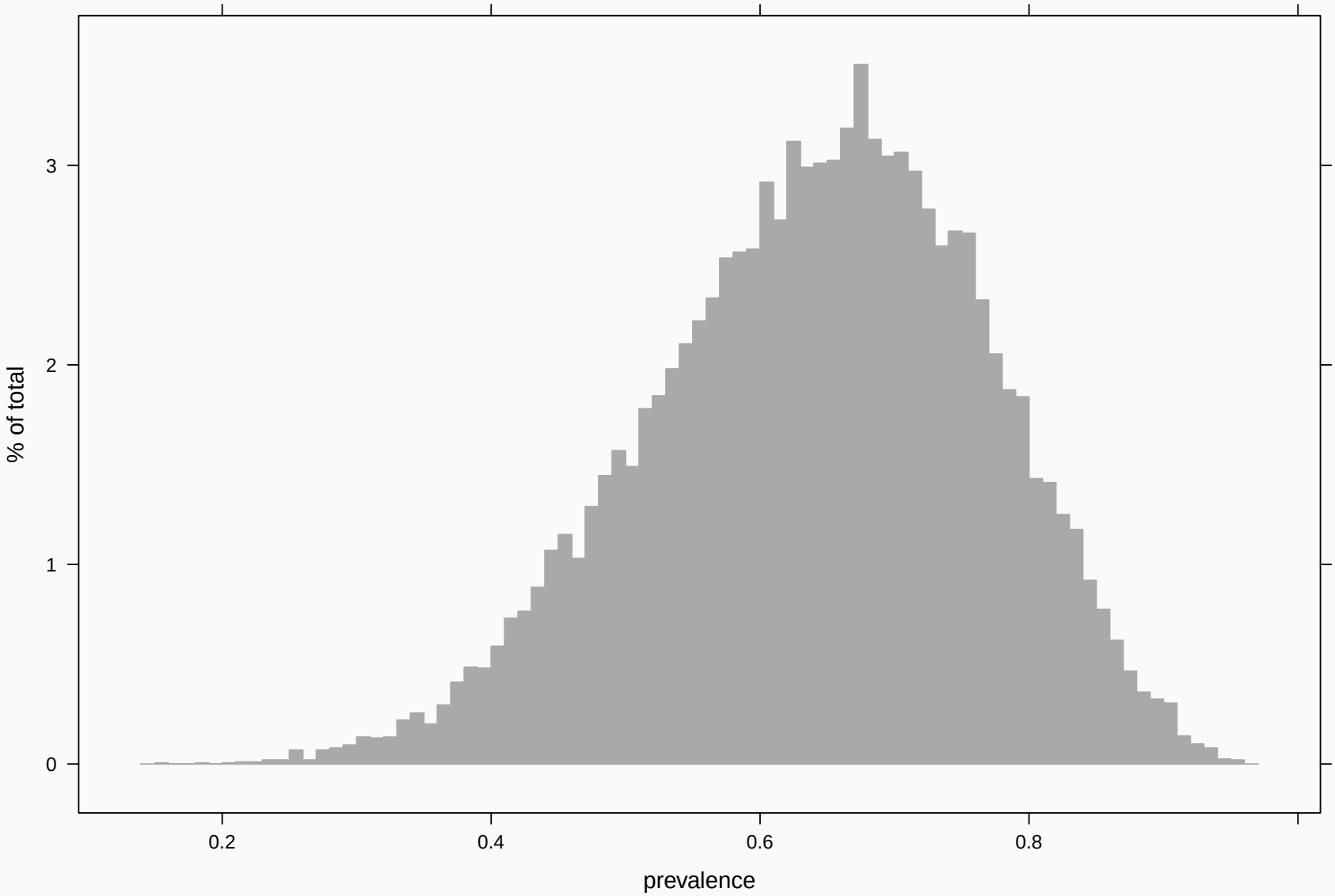
Trace plots: the two chains should be stationary:



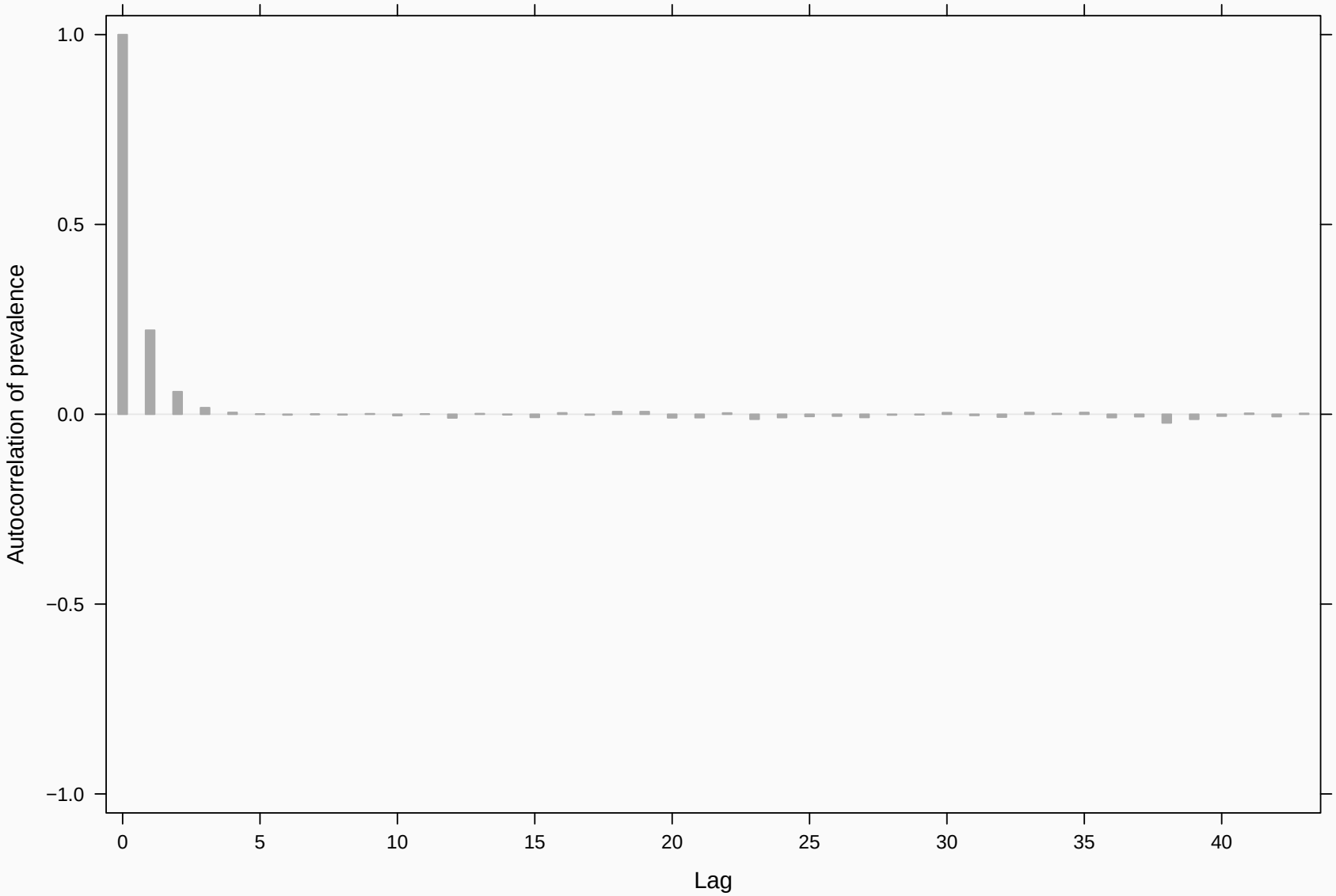
ECDF plots: the two chains should be very close to each other:



Histogram of the combined chains should appear smooth:



Autocorrelation plot tells you how well behaved the model is:



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

```
results
```

```
##
## JAGS model summary statistics from 20000 samples (chains = 2;
↪  adapt+burnin = 6000):
##
##            Lower95  Median Upper95    Mean      SD Mode
## prevalence  0.3993 0.65047 0.86804 0.64202 0.12317   -
##
##                MCerr MC%ofSD SSeff       AC.10   psrf
## prevalence 0.0010989     0.9 12564 -0.0037817 1.0002
##
## Total time taken: 0.2 seconds
```

Reminder: we want psrf < 1.05 and SSeff > 1000

Exercise

- Run this model yourself in JAGS
- Change the initial values for the two chains and make sure it doesn't affect the results
- Reduce the burnin length - does this make a difference?
- Change the sample length - does this make a difference?

Optional Exercise

- Change the number of chains to 1 and 4
  - Remember that you will also need to change the initial values
  - What affect does having different numbers of chains have?
- Try using the `run.jags` argument `method='parallel'` - what affect does this have?

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

Other runjags options

There are a large number of other options to runjags. Some highlights:

- The method can be parallel or background or bgparallel
- You can use extend.jags to continue running an existing model (e.g. to increase the sample size)
- You can use coda::as.mcmc.list to extract the underlying MCMC chains
- Use the summary() method to extract summary statistics
  - See `?summary.runjags` and `?runjagsclass` for more information

Using embedded character strings

- For simple models we might not want to bother with an external text file. Then we can do:

```
mt <- "
model{
  Positives ~ dbinom(prevalence, TotalTests)
  prevalence ~ dbeta(2, 2)

  #data# Positives, TotalTests
  #monitor# prevalence
  #inits# prevalence
}
"

results <- run.jags(mt, n.chains=2)
```

- But I would advise that you stick to using a separate text file!

Setting the RNG seed

- If we want to get numerically replicable results we need to add `.RNG.name` and `.RNG.seed` to the initial values, and an additional `#modules#` lecuyer hook to our basicjags.bug file:

```
model{
  Positives ~ dbinom(prevalence, TotalTests)
  prevalence ~ dbeta(2, 2)

  #data# Positives, TotalTests
  #monitor# prevalence
  #inits# prevalence, .RNG.name, .RNG.seed
  #modules# lecuyer
}
```

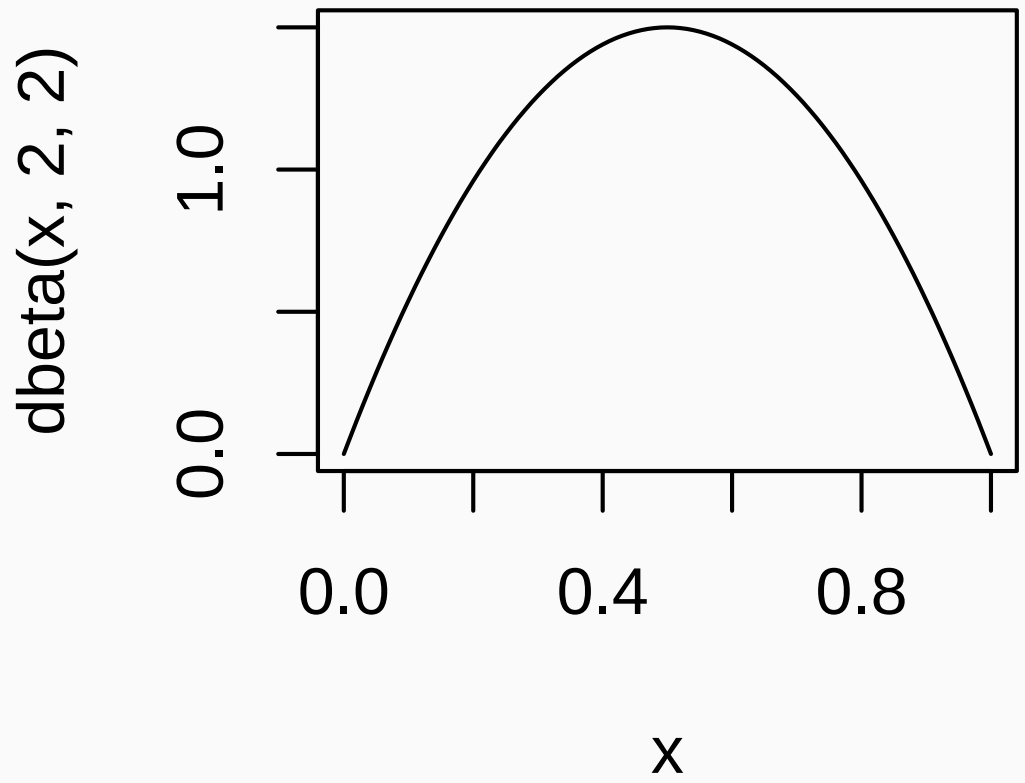
```
.RNG.name <- "lecuyer::RngStream"
.RNG.seed <- list(chain1=1, chain2=2)
results <- run.jags('basicjags.bug', n.chains=2)
```


- Every time this model is run the results will now be identical

A different prior

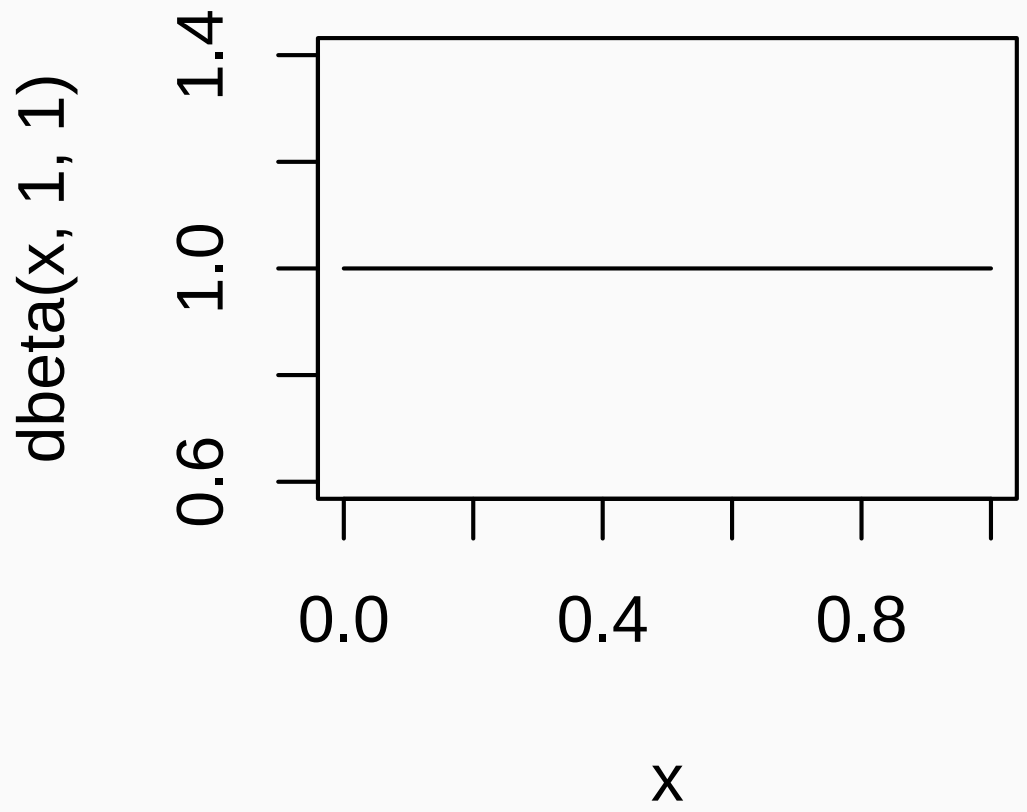
- A quick way to see the distribution of a prior:

```
1  curve(dbeta(x, 2, 2), from=0, to=1)
```



- A minimally informative prior might be:

```
curve(dbeta(x, 1, 1), from=0, to=1)
```



- Let's change the prior we are using to `dbeta(1,1)`:

```
model{
  Positives ~ dbinom(prevalence, TotalTests)
  prevalence ~ dbeta(1, 1)

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

An Equivalent Model

- We could equivalently specify an observation-level model:

```
model{
  # Likelihood part:
  for(i in 1:TotalTests){
    Status[i] ~ dbern(prevalence)
  }

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

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

- But we need the data in a different format: a vector of 0/1 rather than total positives!

A GLM Model

```
model{
  # Likelihood part:
  for(i in 1:TotalTests){
    Status[i] ~ dbern(predicted[i])
    logit(predicted[i]) <- intercept
  }

  # Prior part:
  intercept ~ dnorm(0, 10^-6)

  # Derived parameter:
  prevalence <- ilogit(intercept)

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

- This is the start of a generalised linear model, where we could add covariates at individual animal level.
- We introduce a new distribution `dnorm()` - notice this is mean and precision, not mean and sd!
- For a complete list of the distributions available see:
  - https://sourceforge.net/projects/mcmc-jags/files/Manuals/4.x/
  - This document is also provided on the GitHub repository
- However, notice that the prior is specified differently...

Exercise

- Run the original version, the observation-level version, and the GLM version of the model and compare results with the same data
- Now try a larger sample size: e.g. 70 positives out of 100 tests - are the posteriors from the two models more or less similar than before?
- Now try running the GLM model with a prior of `dnorm(0, 0.33)` (and the original data) - does this make a difference?

Optional Exercise

Another way of comparing different priors is to run different models with no data - as there is no influence of a likelihood, the posterior will then be identical to the priors (and the model will run faster).

One way to do this is to make all of the response data (i.e. either Positives or Status) missing. Try doing this for the following three models, and compare the priors for prevalence:

- The original model with prior `prevalence ~ dbeta(1,1)`
- The GLM model with prior `intercept ~ dnorm(0, 10^-6)`
- The GLM model with prior `intercept ~ dnorm(0, 0.33)`

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

Imperfect tests

- Up to now we have ignored issues of diagnostic test sensitivity and specificity
- Usually, however, we do not have a perfect test, so we do not know how many are truly positive or truly negative, rather than just testing positive or negative.
- But we know 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

- We can incorporate the imperfect sensitivity and specicifity into our model:

```
model{
  Positives ~ dbinom(obsprev, TotalTests)
  obsprev <- (prevalence * se) + ((1-prevalence) * (1-sp))

  prevalence ~ dbeta(1, 1)
  se ~ dbeta(1, 1)
  sp ~ dbeta(1, 1)

  #data# Positives, TotalTests
  #monitor# prevalence, obsprev, se, sp
  #inits# prevalence, se, sp
}
```

- And run it:

```
prevalence <- list(chain1=0.05, chain2=0.95)
se <- list(chain1=0.5, chain2=0.99)
sp <- list(chain1=0.5, chain2=0.99)
Positives <- 70
TotalTests <- 100
results <- run.jags('basicimperfect.bug', n.chains=2)
```

```
## Finished running the simulation
```

[Remember to check convergence and effective sample size!]

What do these results tell us?

	Lower95	Median	Upper95	SSeff	psrf
prevalence	0.000	0.479	0.931	2294	1.000
obsprev	0.604	0.694	0.781	19697	1.000
se	0.141	0.686	1.000	1588	1.001
sp	0.000	0.303	0.837	1613	1.000

What do these results tell us?

	Lower95	Median	Upper95	SSeff	psrf
prevalence	0.000	0.479	0.931	2294	1.000
obsprev	0.604	0.694	0.781	19697	1.000
se	0.141	0.686	1.000	1588	1.001
sp	0.000	0.303	0.837	1613	1.000

- We can estimate the observed prevalence quite well
- But not the prevalence, se or sp!
  - The model is unidentifiable.

Priors

- We cannot estimate `se`, `sp` and `prevalence` simultaneously
  - We need strong priors for se and sp
- We can use the PriorGen package to generate Beta priors based on published results, for example:

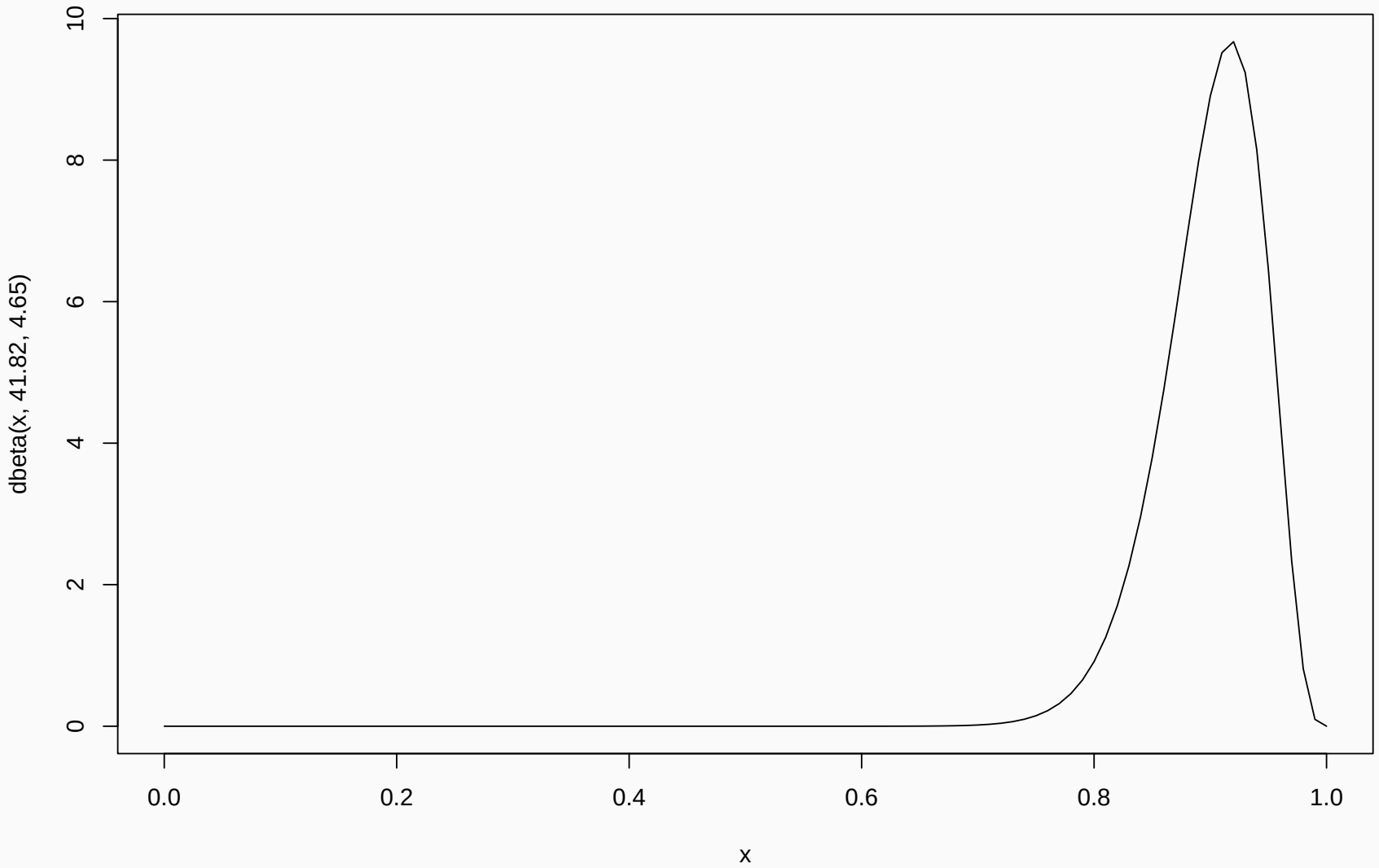
```
PriorGen::findbeta(themean=0.9, percentile = 0.975,
↪  percentile.value = 0.8)
```

```
## [1] "The desired Beta distribution that satisfies the
↪  specified conditions is: Beta( 41.82 4.65 )"
## [1] "Here is a plot of the specified distribution."
## [1] "Descriptive statistics for this distribution are:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.6830  0.8744  0.9051  0.9002  0.9322  0.9925
## [1] "Verification: The percentile value 0.8 corresponds to the
↪  0.025 th percentile"
```

```r
qbeta(c(0.025, 0.5, 0.975), 41.82, 4.65)
```

```
## [1] 0.7999144 0.9056630 0.9677283
```

```r
curve(dbeta(x, 41.82, 4.65), from=0, to=1)
```



Exercise

- Find beta distribution priors for:
  - Sensitivity = 0.9 (95% CI: 0.85 - 0.95)
  - Specificity = 0.95 (95%CI: 0.92-0.97)
- Look at these distributions using curve and qbeta
- Modify the imperfect test model using these priors and re-estimate prevalence

Optional Exercise

- Run the same model with se and sp fixed to the mean estimate
  - How does this affect CI for prevalence?
- Run the same model with se and sp fixed to 1
  - How does this affect estimates and CI for prevalence?

Summary

- Using JAGS / runjags allows us to work with MCMC more easily, safely and efficiently than writing our own sampling algorithms
- But we must *never forget* to check convergence and effective sample size!
- More complex models become easy to implement
    - For example imperfect diagnostic tests
- But just because a model can be defined does not mean that it will be useful for our data
    - We need to be realistic about the information available in the data, what parameters are feasible to estimate, and where we will need to use strong priors