A short introduction to using JAGS and R

Søren Højsgaard
Department of Mathematical Sciences
Aalborg University, Denmark

Based on a related document by Bendix Carstensen, http://bendixcarstensen.com

May 15, 2013

Contents

1 Introduction – Bayesian modelling in short form

- In a Bayesian setting, parameters are treated as random quantities on equal footing with the random variables.
- The joint distribution of a parameter (vector) θ and data (vector) y is specified through a prior distribution $\pi(\theta)$ for θ and a conditional distribution $p(y \mid \theta)$ of data for a fixed value of θ.
- This leads to the joint distribution for data AND parameters

$$p(y, \theta) = p(y \mid \theta)\pi(\theta)$$

- The prior distribution $\pi(\theta)$ represents our knowledge (or uncertainty) about θ before data have been observed.
- After observing data y, the posterior distribution $\pi^*(\theta)$ of θ is obtained by conditioning with data which gives

$$\pi^*(\theta) = p(\theta|y) = \frac{p(y|\theta)\pi(\theta)}{p(y)} \propto L(\theta)\pi(\theta)$$

  where $L(\theta) = p(y \mid \theta)$ is the likelihood and the marginal density $p(y) = \int p(y \mid \theta)\pi(\theta)d\theta$ is the normalizing constant.
- Often we are interested in the posterior mean of some function $g(\theta)$:

$$\mathbb{E}(g(\theta)|\pi^*) = \int g(\theta)\pi^*(\theta)d\theta$$

  Examples: $\mathbb{E}(\theta|\pi^*)$ or $\mathbb{V}\mathrm{ar}(\theta|\pi^*)$.
- However, usually $\pi^*(\theta)$ can not be found analytically because the normalizing constant $p(y) = \int p(y \mid \theta)\pi(\theta)d\theta$ is intractable.
- In such cases one will often resort to sampling based methods: If we can draw samples $\theta^{(1)}, \ldots, \theta^{(N)}$ from $\pi^*(\theta)$ we can do just as well:

$$\mathbb{E}(g(\theta)|\pi^*) \approx \frac{1}{N}\sum_i g(\theta^{(i)})$$

- The question is then how to draw samples from $\pi^*(\theta)$ where $\pi^*(\theta)$ is only known up to the normalizing constant.
- There are many methods for achieving this; these methods are known as Markov Chain Monte Carlo (MCMC) methods and will be described elsewhere.
- For now, we shall just state that the `JAGS` program allows one to obtain such samples.
- Sections marked with "*" in the following can be skipped at first reading.

2 Airline fatalities

We are going to analyze the annual number of airline fatalities using a simple Poisson–Gamma model.

This model is so simple that we will know the exact form of the posterior distribution for the parameter of interest.

This way we can check that the MCMC machinery gives numerical results that are consistent with the known theoretical posterior distribution for that parameter.

First get the data and take a look at it:

```
> airline <- read.csv( "./data/airline.csv" )
> head(airline)

  year1975 year fatal miles  rate
1        1 1976    24 3.863 6.213
2        2 1977    25 4.300 5.814
3        3 1978    31 5.027 6.167
4        4 1979    31 5.481 5.656
5        5 1980    22 5.814 3.784
6        6 1981    21 6.033 3.481
```

```
> fatal <- airline$fatal
> fatal

 [1] 24 25 31 31 22 21 26 20 16 22 22 25 29 29 27 29 28 33 27 25 24 26 20 21 18 13

> (ss <- sum( fatal ))    ## We need these quantities later

[1] 634

> (nn <- length( fatal ))

[1] 26
```

3 Poisson-gamma model

3.1 Properties of the Poisson and Gamma distributions

Recall: The Poisson distribution

$$Y \sim \text{Poisson}(\mu); \quad f(y;\mu) = \frac{\mu^y e^{-\mu}}{y!}$$

Recall: If $Y \sim \text{Poisson}(\mu)$ then $\mathbb{E}(Y) = \mathbb{V}\text{ar}(Y) = \mu$.

Recall: The Gamma distribution

$$Y \sim \text{Gamma}(a, r); \quad f(y;a,r) = \frac{r^a}{\Gamma(a)} y^{a-1} e^{-ry}$$

Recall: a is the shape parameter; r is the rate parameter

Recall: If $Y \sim \text{Gamma}(a, r)$ then $\mathbb{E}(Y) = a/r$ and $\mathbb{V}\text{ar}(Y) = a/r^2 = \mathbb{E}(Y)/r$.

3.2 The Poisson–Gamma model

We shall assume

$$\begin{aligned} y_i|\mu &\sim \text{Poisson}(\mu), \quad p(y_i|\mu) \propto \mu^{y_i} e^{-\mu} \\ \mu &\sim \text{Gamma}(a, r), \quad \pi(\mu) \propto \mu^{a+1} e^{-r\mu} \end{aligned}$$

3.3 Calculating the posterior in closed form

The Poisson–Gamma model is one of the rare models where the posterior can be found in closed form.

Letting $y_+ = \sum_i y_i$, the likelihood becomes

$$p(y_1, \ldots y_n|\mu) \propto \mu^{y_+} e^{-n\mu}$$

The posterior becomes

$$\begin{aligned} \pi^*(\mu) &= p(\mu|y_1, \ldots y_n) \propto p(y_1, \ldots y_n|\mu)\pi(\mu) \\ &= \mu^{y_+} e^{-n\mu} \mu^{a+1} e^{-r\mu} = \mu^{y_+ + a - 1} e^{-(r+n)\mu} \end{aligned}$$

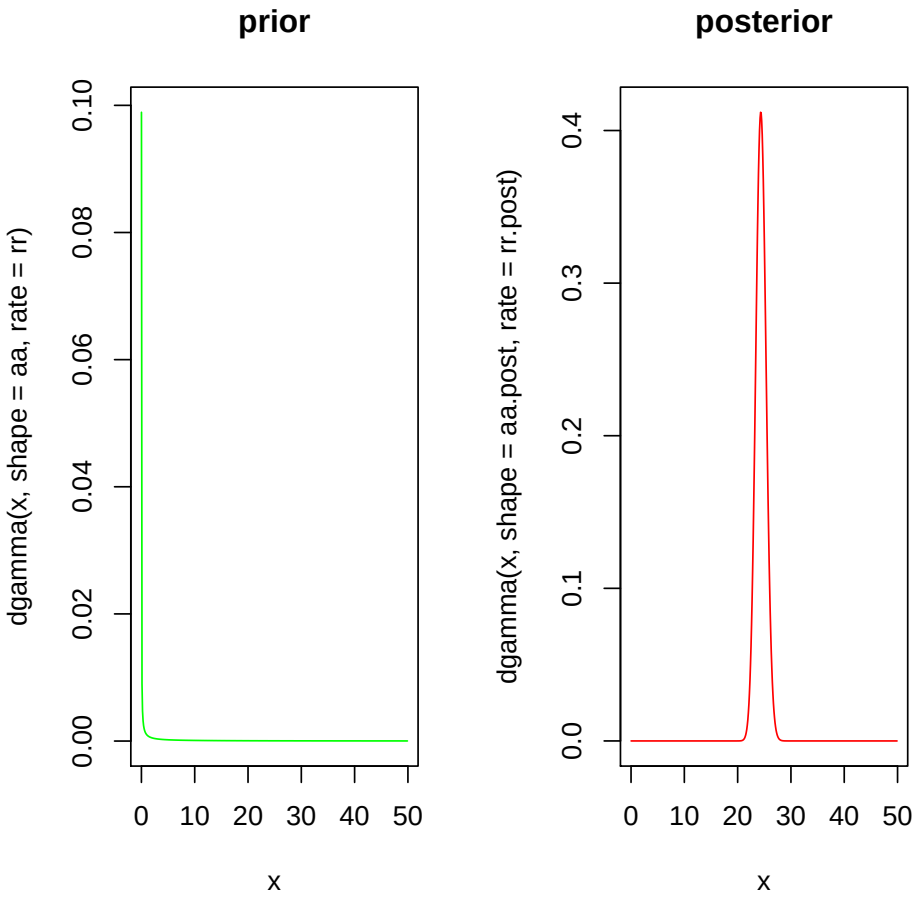
Hence the posterior $\pi^*(\mu)$ is a gamma distribution

$$\mu|y_1, \ldots, y_n \sim \text{Gamma}(a + y_+, r + n)$$

Hence if we take $a = r = 0.001$ the gamma distribution will have mean 1 and variance 1000, i.e. an "uninformative prior":

```
> aa <- 0.001
> rr <- 0.001
> aa.post <- aa + ss
> rr.post <- rr + nn
> ( EE.post <- aa.post / rr.post )

[1] 24.38372

> ( VV.post <- aa.post / rr.post^2 )

[1] 0.9377992
```

```
> x <- seq(0.01, 50, by=0.1)
> par(mfrow=c(1,2))
> plot(x, dgamma(x, shape=aa, rate=rr), type='l', col='green', main='prior')
> plot(x, dgamma(x, shape=aa.post, rate=rr.post), type='l', col='red', main='posterior')
```



3.4 Comparison with frequentist estimate*

For comparison, the usual estimate of μ in a frequentist setting (the maximum likelihood estimate) is:

$$\hat{\mu} = y_+/n;$$

The mean and variance of this estimator is

$$\mathbb{E}(\hat{\mu}) = \mu; \quad \mathbb{V}\text{ar}(\hat{\mu}) = \mu/n$$

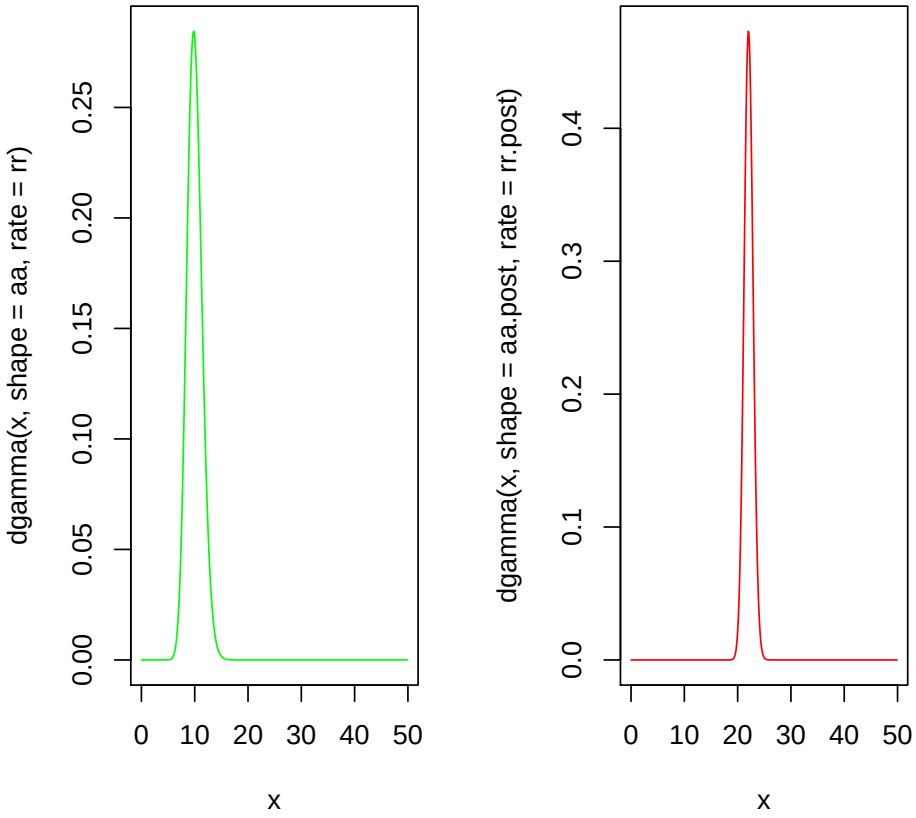
```
> mean(fatal)

[1] 24.38462

> mean(fatal)/nn # estimated variance of estimator

[1] 0.9378698
```

An informative prior could be to say that $\mathbb{E} = 10$ and $\mathbb{V}\text{ar} = 2$. Solving for r and a gives $r = \mathbb{E} \,/\, \mathbb{V}\text{ar}$ and $a = \mathbb{E}\, r = \mathbb{E}^2 \,/\, \mathbb{V}\text{ar}$.

```
> EE <- 10
> VV <-  2
> rr <- EE / VV
> aa <- EE^2 / VV
> aa.post <- aa + ss
> rr.post <- rr + nn
> ( EE.post <- aa.post / rr.post )

[1] 22.06452

> ( VV.post <- aa.post / rr.post^2 )

[1] 0.7117586
```

The posterior mean and variances now differ considerably from the mean and variance of MLE.

```
> x <- seq(0.01, 50, by=0.1)
> par(mfrow=c(1,2))
> plot(x, dgamma(x, shape=aa, rate=rr), type='l', col='green')
> plot(x, dgamma(x, shape=aa.post, rate=rr.post), type='l', col='red')
```



3.5 Sampling from the posterior - SIR*

We can obtain (samples from) the posterior as follows:

- As proposal distribution $h()$ we take a uniform distribution on a closed interval.
- Draw proposals $\mu^1, \ldots, \mu^S \sim h()$.
- The posterior $\pi^*(\mu) \propto p(y|\mu)\pi(\mu) = k(\mu)$ needs only be known up to the constant of proportionality.
- The importance weights are

$$w^j = k(\mu^j)/h(\mu^j) \propto k(\mu^j)$$

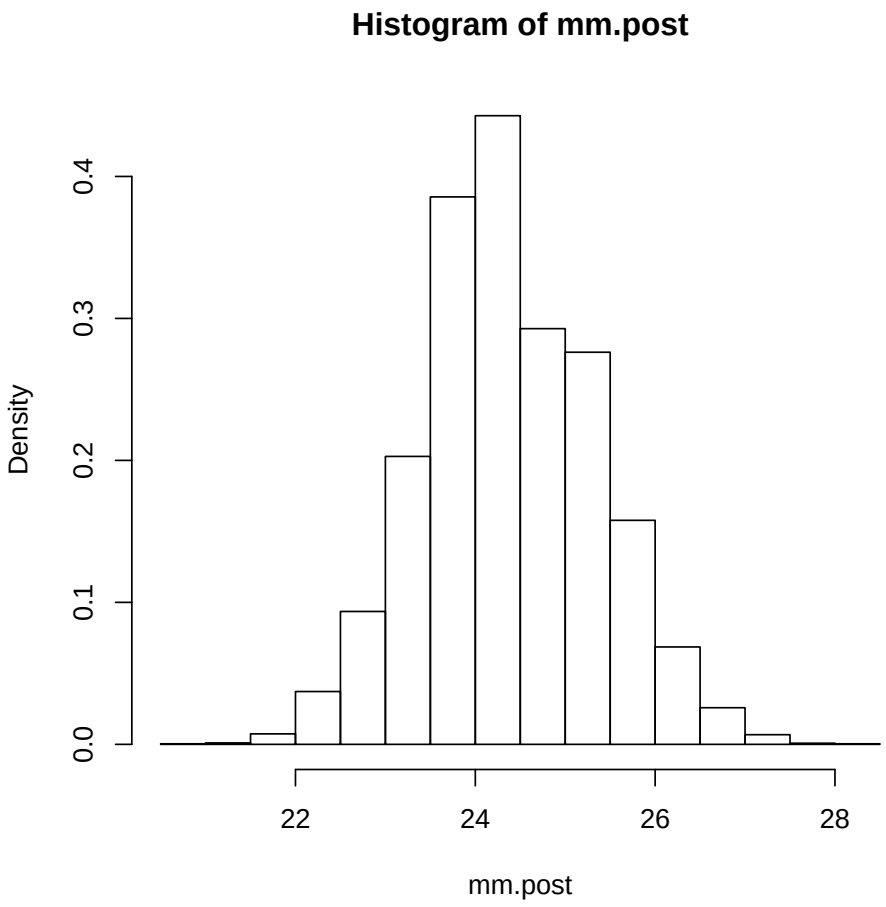
because $h()$ is uniform

```
> SS <- 10000
> mm <- sort(runif(SS, min=0, max=50))
```

```
> aa <- 0.001
> rr <- 0.001
```

```
> log.pr   <- ( aa - 1 ) * log( mm ) - rr * mm    ## log prior
> log.L    <- ss * log( mm ) - nn * mm            ## log likelihood
> log.wgt  <- log.pr + log.L                      ## log posterior (unnormalized)
> log.wgt  <- log.wgt - max( log.wgt )            ## center to make moderate sizes
> wgt      <- exp( log.wgt )                      ## unnormalized posterior
> wgt      <- wgt / sum( wgt )                    ## normalized posterior
> mm.post  <- sample(mm, replace=TRUE, prob=wgt)  ## samples from posterior
> c(mean(mm.post), var(mm.post))

[1] 24.3728214  0.9374964
```

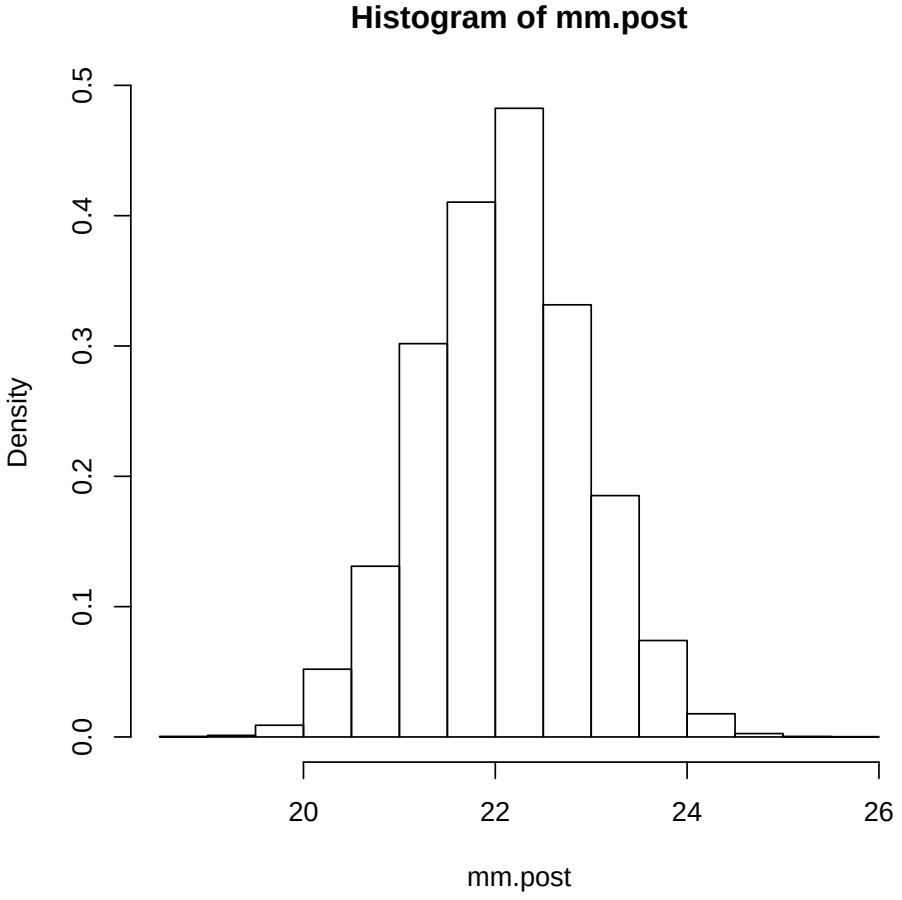
```
> hist(mm.post, prob=T)
```



```
> EE <- 10
> VV <-  2
> ( rr <- EE / VV )

[1] 5

> ( aa <- EE^2 / VV )

[1] 50
```

```
> log.pr   <- ( aa - 1 ) * log( mm ) - rr * mm    ## log prior
> log.L    <- ss * log( mm ) - nn * mm            ## log likelihood
> log.wgt  <- log.pr + log.L                      ## log posterior (unnormalized)
> log.wgt  <- log.wgt - max( log.wgt )            ## center to make moderate sizes
> wgt      <- exp( log.wgt )                      ## unnormalized posterior
> wgt      <- wgt / sum( wgt )                    ## normalized posterior
> mm.post  <- sample(mm, replace=TRUE, prob=wgt)  ## samples from posterior
> c(mean(mm.post), var(mm.post))

[1] 22.0810447  0.6963204
```

```
> hist(mm.post, prob=T)
```



3.6 Choice of proposal distribution is important*

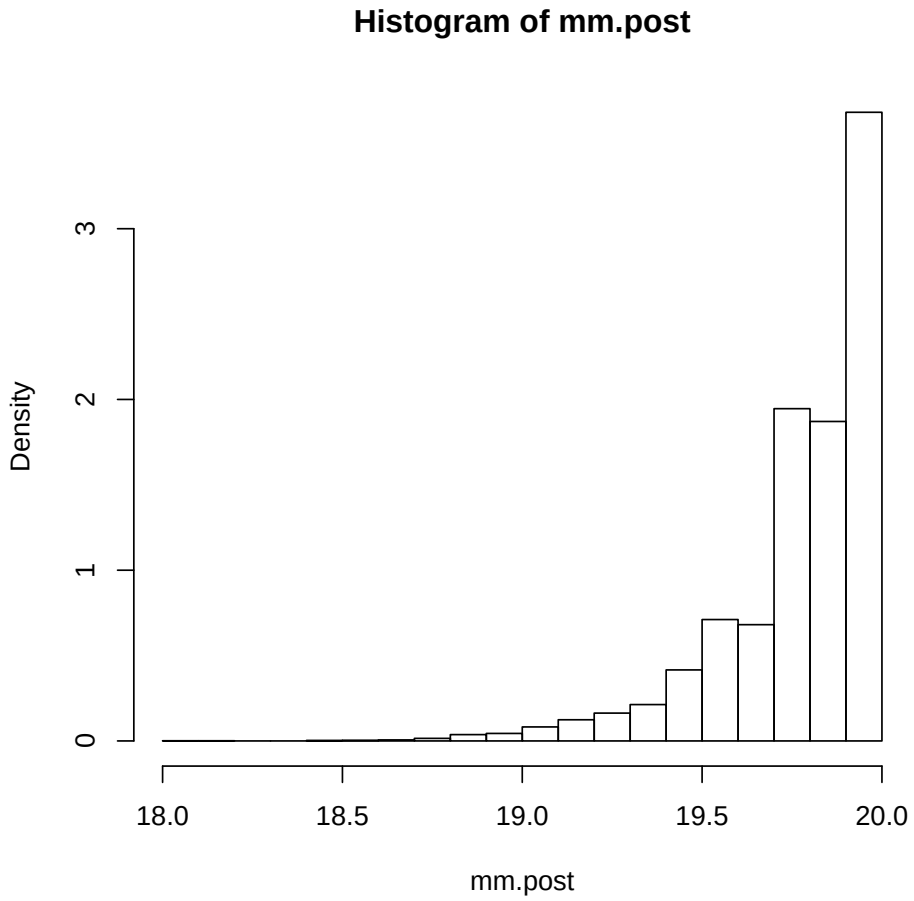
Remark: The choice of proposal distribution is always important. Consider this case where the proposal has a much smaller support than the target distribution:

```
> mm <- sort(runif(SS, min=0, max=20))
```

```
> log.pr   <- ( aa - 1 ) * log( mm ) - rr * mm    ## log prior
> log.L    <- ss * log( mm ) - nn * mm            ## log likelihood
> log.wgt  <- log.pr + log.L                      ## log posterior (unnormalized)
> log.wgt  <- log.wgt - max( log.wgt )            ## center to make moderate sizes
> wgt      <- exp( log.wgt )                      ## unnormalized posterior
> wgt      <- wgt / sum( wgt )                    ## normalized posterior
> mm.post  <- sample(mm, replace=TRUE, prob=wgt)  ## samples from posterior
> c(mean(mm.post), var(mm.post))

[1] 19.77273837  0.05027992
```

We end up accepting the best proposals, but that is not the same as the proposals beeing good.

```
> hist(mm.post, prob=T)
```



4 Running JAGS via rjags

First you must load the `rjags` package, which should automatically find your JAGS installation:

```
> library(rjags)
```

In order to run `JAGS` we must (i) supply the data; (ii) formulate the model as a BUGS program; and (iii) specify how the sampling from the chains should be done.

4.1 Data

The first thing to provide to `JAGS` is the data. This is provided in the form of a named list, one element per data-structure (usually vector or matrix). In this case we provide the vector of fatal airline accidents expanded with a NA for prediction of the number in 2002,

as well as the total number of observations:

```
> a.dat <- list( fatal = c( fatal, NA ), NN=27 )
```

4.2 Program specification of model

The program specifying the model (BUGS code) must be put in a separate file which is then read by JAGS. When working in Rthis is most conveniently done using the R-function cat() which behaves pretty much like paste() with the exception that the result is not a character object but directly written to a file you specify. If you specify file="" the output is sent to the screen. Here is the BUGS code specifying the above model, using cat to put it in the file m1.jag:

```
> cat("
+ model {
+  for( i in 1:NN ) {
+    fatal[i] ~ dpois(mu)
+  }
+  mu ~ dgamma(0.1,0.1)
+ }",
+ file="m1.jag" )
```

The code refers to data points in the variable fatal which is NN long.

The BUGS language is declarative, i.e. it is not executed as the program runs. Instead it is a specification of the model structure, and after the model is set up JAGS will decide how best to go about the MCMC-simulation.

So it would not matter if the specification of a prior of mu was put before the for statement.

Also the loop is just a compact way of writing fatal[1] ˜dpois(mu), fatal[2] ˜dpois(mu) etc.

We could have replaced NN with the number 27 in the code if we wanted. In that case the NN in the data would have been superfluous. It is, however, good practice to express model quantities as variables rather than fixed values since this makes implementing data updates much easier.

4.3 Starting values

To start the MCMC simulation we will normally supply some starting values (in most cases JAGS will however be able to generate them). In order to be able to monitor convergence we will normally run several chains, so we must supply starting values for each chain. The starting values for one chain is a named list, names are the parameters used in the model. Here we use three chains, hence the initial values is a list of three lists. Each of these list has as elements one named value for each parameter – in this case there is only one parameter μ, called mu in the BUGS program:

```
> a.ini <- list( list( mu=20 ), list( mu=23 ), list( mu=26 ) )
```

4.4 Compiling and adapting

Once these structures have been set up we ask JAGS to compile the model and run the chains for a number of cycles ("burn-in") so that the model is (hopefully) in a stable state, that is, converged to sampling from a stationary process that represents the target distribution, namely the joint posterior distribution for the unobserved quantities (stochastic nodes) in the model. In this case we ask for 3 chains and 2000 cycles of burn-in:

```
> m <- jags.model( file = "m1.jag",
+                  data = a.dat,
+                  n.chains = 3,
+                  inits = a.ini,
+                  n.adapt = 2000
+                  )

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 30

Initializing model
```

```
> m

JAGS model:


model {
 for( i in 1:NN ) {
   fatal[i] ~ dpois(mu)
 }
 mu ~ dgamma(0.1,0.1)
}
Fully observed variables:
 NN
Partially observed variables:
 fatal
```

4.5 Parameters and simulation parameters

Once the model is set up and "burnt in", we can run the chain using coda.samples, which not surprisingly produces an object of class mcmc.list that can be manipulated by the functions in the coda package.

We must specify:

- the variables (nodes) that we want to monitor in the subsequent cycles of the chain. This is done using the argument variable.names (which can be abbreviated to var if you wish).

- how many cycles (iterations) to run the chain (n.iter)
- how often we sample the specified parameters and retain the results in memory (thin)

In this case we run 10,000 cycles of the three chains, and sample every 10th value of mu and fatal[27], so we get 1000 samples from each chain, a total of 3000 samples from the posterior of the parameter(s):

```
> a1.par <- c("mu","fatal[27]")
> res <- coda.samples(m,
+                     var = a1.par,
+                     n.iter = 10000,
+                     thin = 10 )
```

The resulting object is of class mcmc.list; in this case a list with 3 elements (one per chain). Each element of the list is a 1000×2 matrix.

4.6 Results

```
> class( res )

[1] "mcmc.list"

> str( res )

List of 3
 $ : mcmc [1:1000, 1:2] 31 35 33 30 38 23 31 31 20 33 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "fatal[27]" "mu"
  ..- attr(*, "mcpar")= num [1:3] 10 10000 10
 $ : mcmc [1:1000, 1:2] 19 27 26 21 32 27 21 27 26 23 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "fatal[27]" "mu"
  ..- attr(*, "mcpar")= num [1:3] 10 10000 10
 $ : mcmc [1:1000, 1:2] 23 22 35 25 32 30 34 22 21 18 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "fatal[27]" "mu"
  ..- attr(*, "mcpar")= num [1:3] 10 10000 10
 - attr(*, "class")= chr "mcmc.list"
```

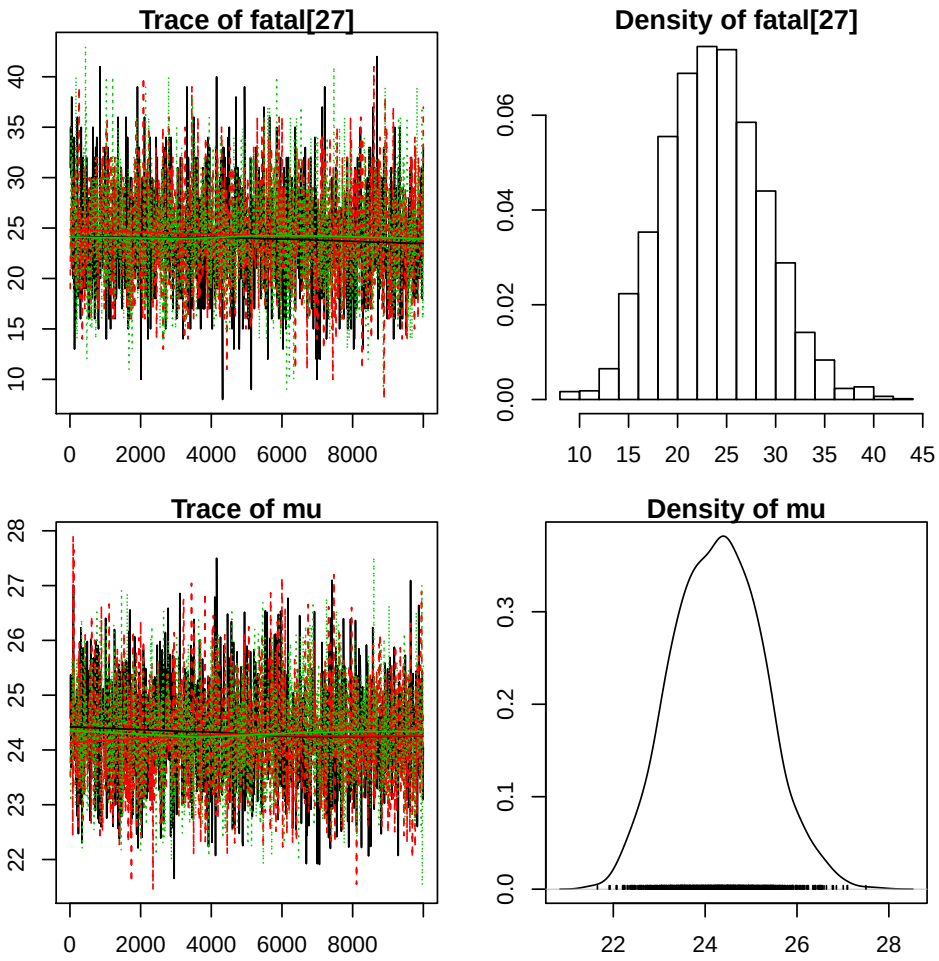
The mcpar attribute of each of the list members are the first, last and step in the sampling of the chains.

As always in R, the most useful overview comes from the summary() function:

```
> summary( res )

Iterations = 10:10000
Thinning interval = 10
Number of chains = 3
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

           Mean     SD Naive SE Time-series SE
fatal[27] 24.22 5.1830  0.09463        0.09462
mu        24.29 0.9695  0.01770        0.01798

2. Quantiles for each variable:

           2.5%   25%  50%   75% 97.5%
fatal[27] 15.00 21.00 24.0 28.00 35.00
mu        22.48 23.59 24.3 24.96 26.27
```

If we decide that the set of samples from the 3 chains provides a reasonable representation of the posterior distribution, we can get an overview of the three chains by using the function plot():

```
> par(mar=c(3,2,1,2))
> plot( res )
```

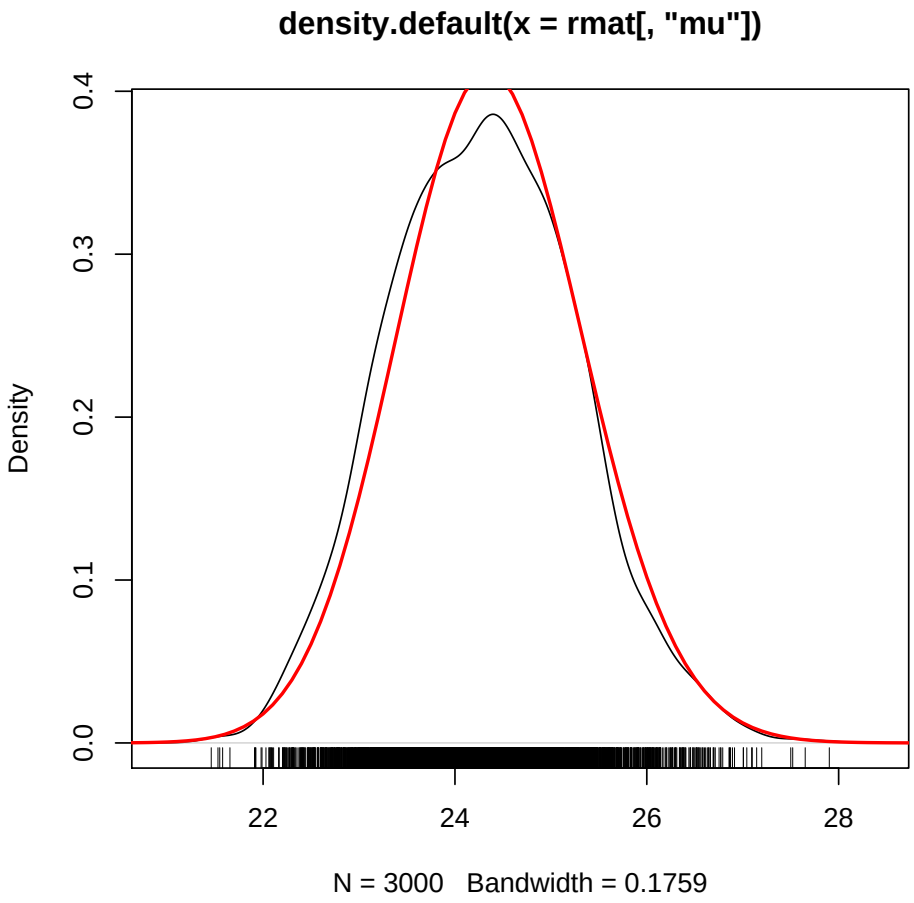


If we want a simpler structure to work with, we can collect all the posterior samples from the different chains in one matrix:

```
> rmat <- as.matrix( res )
> head( rmat )

     fatal[27]       mu
[1,]        31 24.58475
[2,]        35 25.37171
[3,]        33 24.34358
[4,]        30 23.67428
[5,]        38 25.00182
[6,]        23 24.73261
```

Since the model is so simple that we know the theoretical from of the posterior, we can add this curve to the plot in red, say:

```
> plot(density(rmat[,"mu"]))
> rug(rmat[,"mu"])
> curve( dgamma( x, 634.001, 26.001 ), from=20, to=30, lwd=2, col="red", add=TRUE )
```



5 Running JAGS directly (without using rjags)

The JAGS manual describes how to run JAGS directly and so does this somewhat old document:

http://users-deprecated.aims.ac.za/~paulhewson/runningjags.pdf