Computer Lab 02 — BRugs

Contents

1 Introduction

If you are on a system on which you cannot run OpenBUGS but the R package BRugs is available (e.g. a Linux operating system), then the following shows how to run the analysis using the BRugs package. It is essentially a compilation of the code shown in the lecture.

2 Douglas firs example with BRugs

2.1 Flat prior

Assume we have the BUGS code for that example in a file called Geometric-flat.bug:

```
model{
  ## likelihood
  for(i in 1:N){
    y[i] ~ dgeom(p)
  }

  ## prior
  p ~ dbeta(1, 1)
}
```

Note that this code is using a flat prior for p.

First, we load the package BRugs:

```
library(BRugs)
```

```
## Welcome to BRugs connected to OpenBUGS version 3.2.3
```

Then we check whether the code in the file Geomtric-flat.bug is syntactically correct[1] :

```
modelCheck("Geometric-flat.bug")
```

```
## model is syntactically correct
```

This time, as we have observed data, we first have to write it out into a file and then load it into our program. The names of the objects that contain the data has to be specified in a list to bugsData(), which also needs the name of a file to write too:

[1] If you encounter an error on the computer in the Mathematics Computing Lab, look up in last week's lab on how to use the shortPathName() command to construct a file name that the modelCheck() command can handle.

```
y.freq <- c(71, 28, 5, 2, 2, 1)
y.obs <- rep(1:6, times = y.freq)
bugsData(list(y = y.obs, N = length(y.obs)), file = "data.txt")
modelData("data.txt")
```

```
## data loaded
```

Next, we compile four chains:

```
nch <- 4
modelCompile(numChains = nch)
```

```
## model compiled
```

And generate initial values for each of the chains:

```
modelGenInits()
```

```
## initial values generated, model initialized
```

Next, we specify the node~p that we want to monitor:

```
samplesSet("p")
```

```
## monitor set for variable 'p'
```

and then update the model 10,000 times:

```
modelUpdate(10000)
```

```
## 10000 updates took 0 s
```

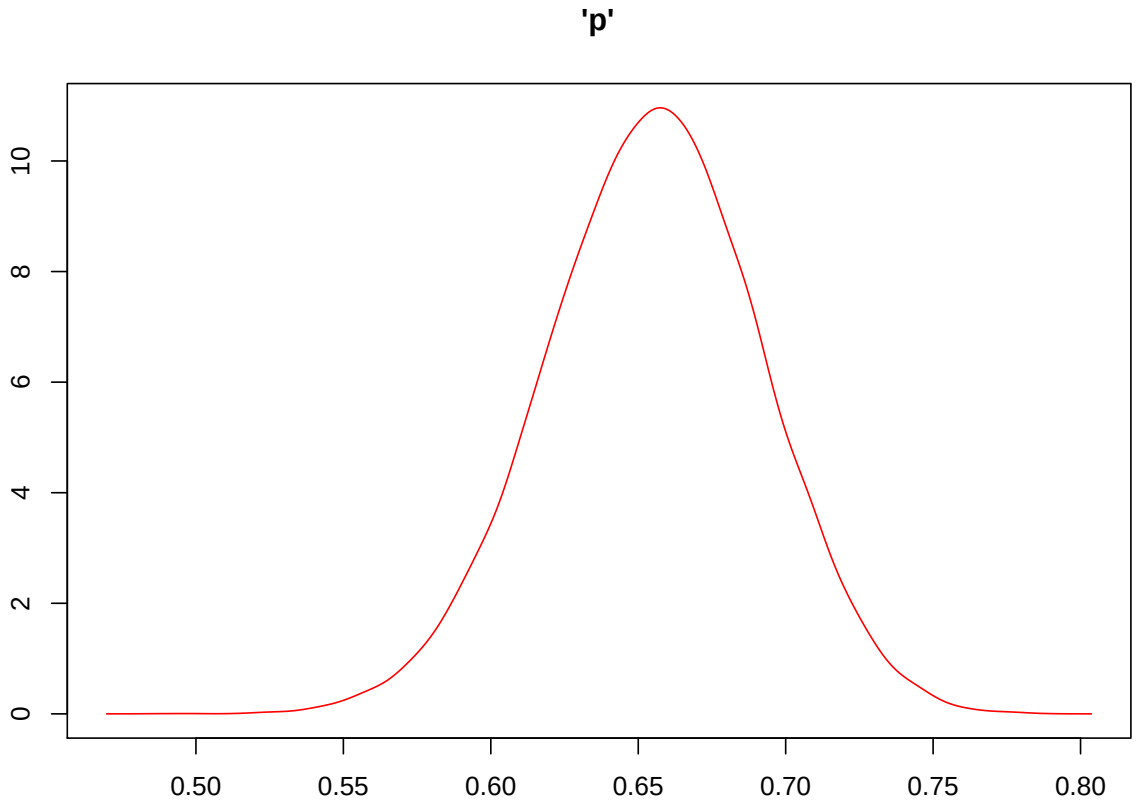
The statistics on each monitored node we can obtain as follows:

```
samplesStats("*", beg = 1001)
```

```
##     mean      sd  MC_error val2.5pc median val97.5pc start sample
## p 0.6548 0.03646 0.0001875   0.5815 0.6555    0.7247  1001  36000
```

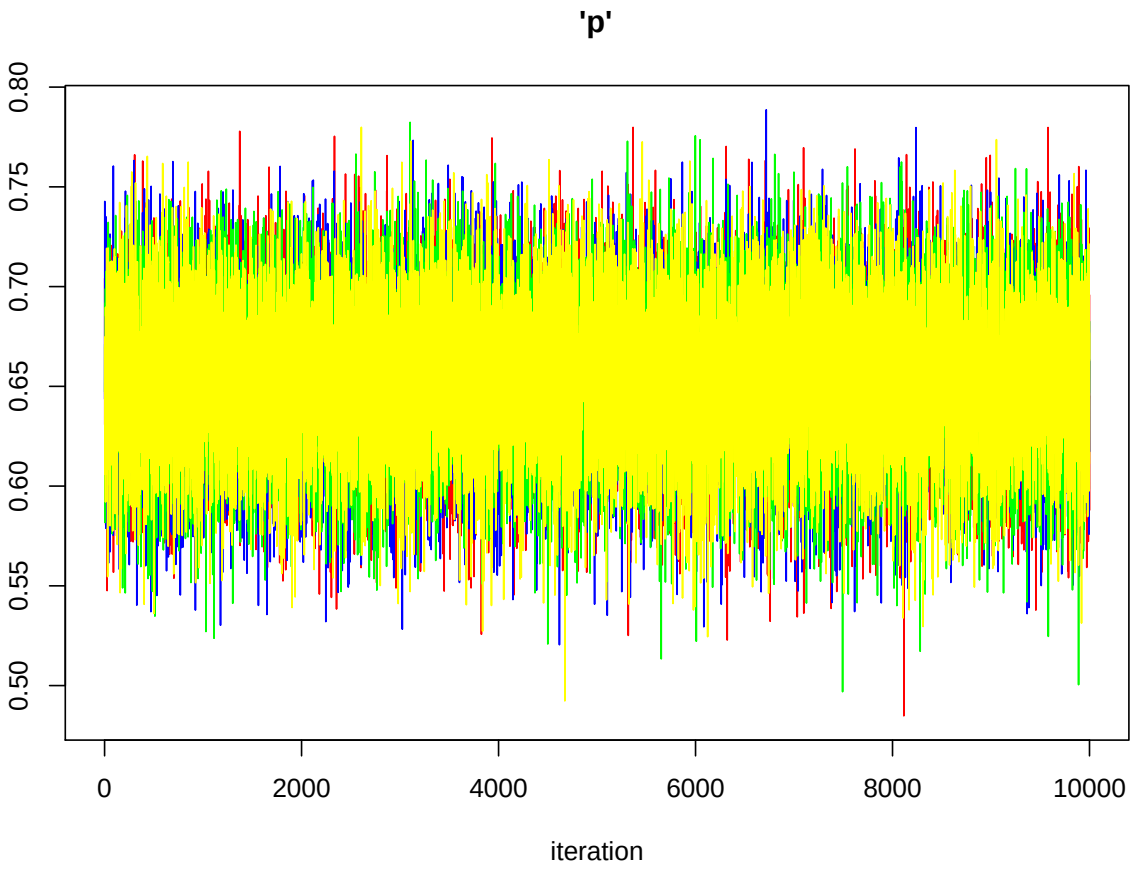
While the empirical densities of the simulated values are produced by the following command:

```
samplesDensity("*", beg = 1001, ask = FALSE, mfrow = c(1, 1))
```

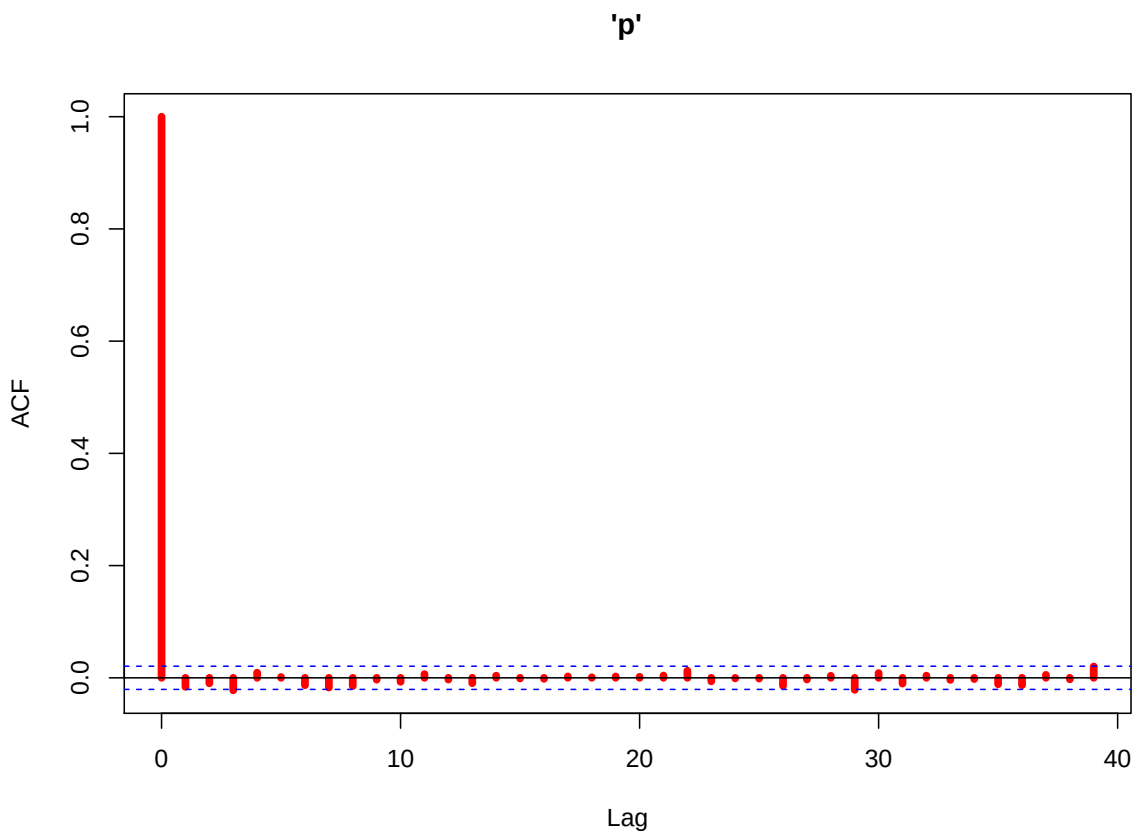


The traceplot and the estimated auto-correlation function can be obtained with the following commands:

```
samplesHistory("*", ask = FALSE, mfrow = c(1, 1))
```




```
samplesAutoC("*", chain = 1, ask = FALSE, beg = 1001, mfrow = c(1, 1))
```



2.2 Jeffreys' prior

Assume we have the BUGS code for that example in a file called `Geometric-Jeffreys.bug`:

```
model{
  ## likelihood
  for(i in 1:N){
    y[i] ~ dgeom(p)
  }

  ## prior
  u ~ dflat()
  u.abs <- abs(u)
  p <- 1 - pow( (1-exp(-u.abs))/(1+exp(-u.abs)), 2)
}
```

We follow pretty much the same steps as above:

```
modelCheck("Geometric-Jeffreys.bug")
```

```
## model is syntactically correct
```

```
y.freq <- c(71, 28, 5, 2, 2, 1)
y.obs <- rep(1:6, times = y.freq)
bugsData(list(y = y.obs, N = length(y.obs)), file = "data.txt")
modelData("data.txt")
```

```
## data loaded
```

```
nch <- 4
modelCompile(numChains = nch)
```

```
## model compiled
```

However, since `u` has a flat prior, `OpenBUGS` cannot generate automatically[2] initial values for the chains. We have to specify initial values. Note that initial values have to be specified via a list with named objects for each compiled chain. The lists for all the chains should be in a list that is then written out by the `bugInits()` command. In other words, that command expects a list of list(s) as it first argument:

```
inits <- list(list(u = 1), list(u = 2), list(u = 4), list(u = 8))
fnames <- paste0("init", 1:nch, ".txt")
bugsInits(inits, numChains = nch, fileName = fnames)
modelInits(fnames)
```

```
## Initializing chain 1:
## initial values loaded and chain initialized but another chain contain uninitialized variables
## Initializing chain 2:
## initial values loaded and chain initialized but another chain contain uninitialized variables
## Initializing chain 3:
## initial values loaded and chain initialized but another chain contain uninitialized variables
## Initializing chain 4:
## model is initialized
```

```
samplesSet("p")
```

```
## monitor set for variable 'p'
```

```
modelUpdate(10000)
```

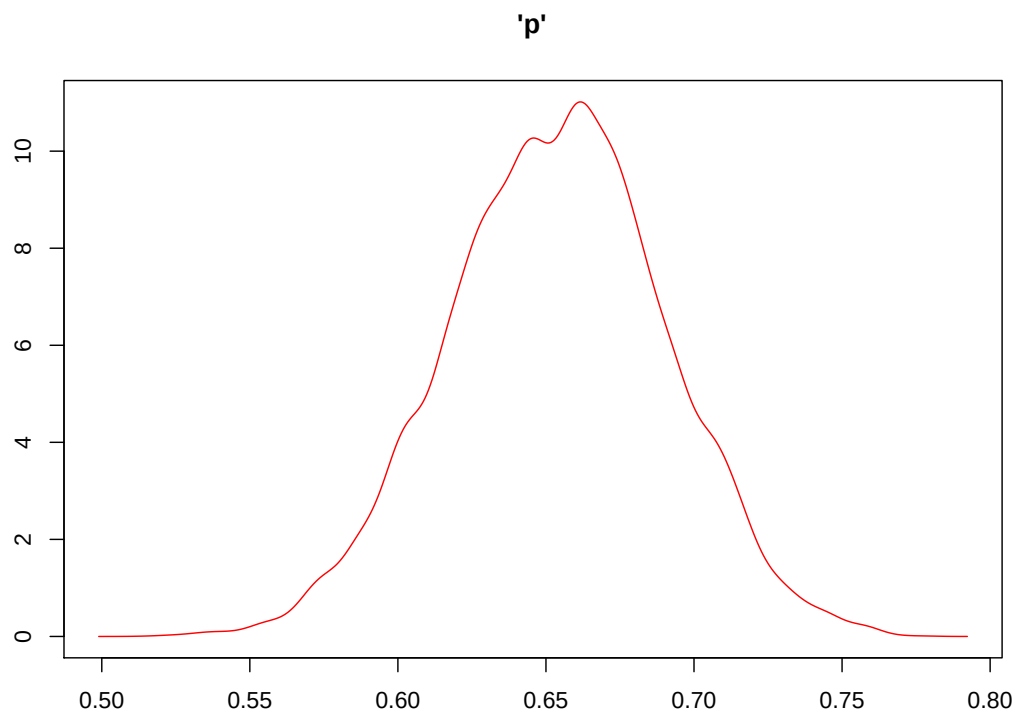
```
## 10000 updates took 1 s
```

```
samplesStats("*", beg = 1001)
```

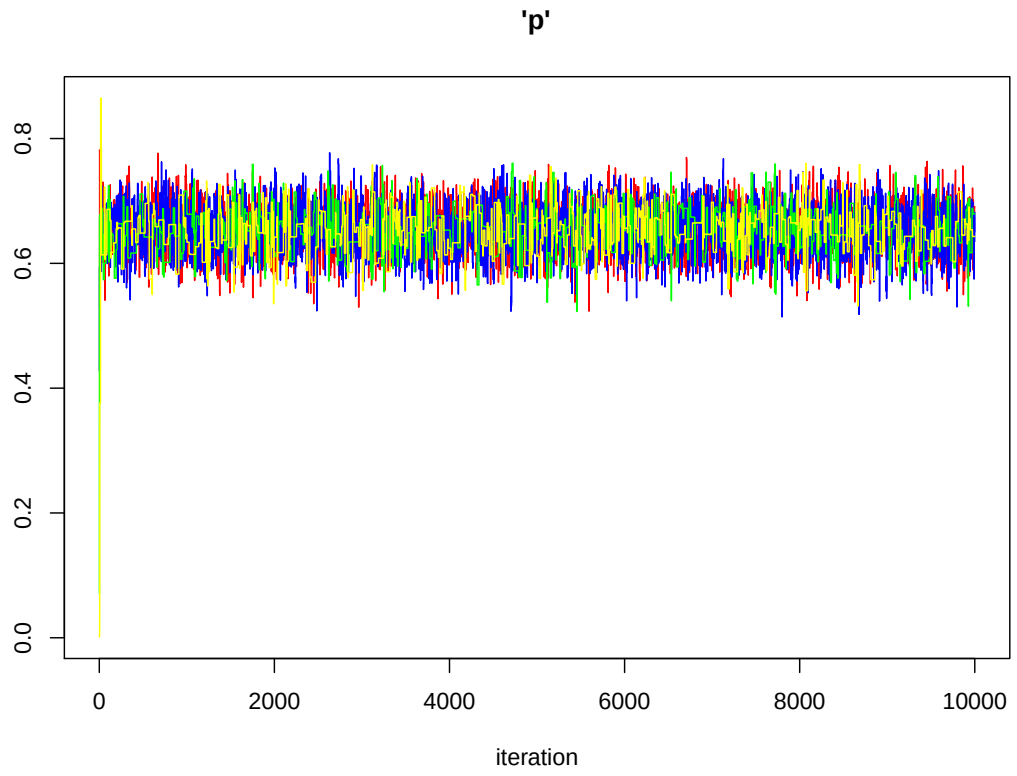
```
##     mean      sd  MC_error val2.5pc median val97.5pc start sample
## p 0.6536 0.03654 0.0006539   0.5811 0.6545    0.7233  1001  36000
```

```
samplesDensity("*", beg = 1001, ask = FALSE, mfrow = c(1, 1))
```

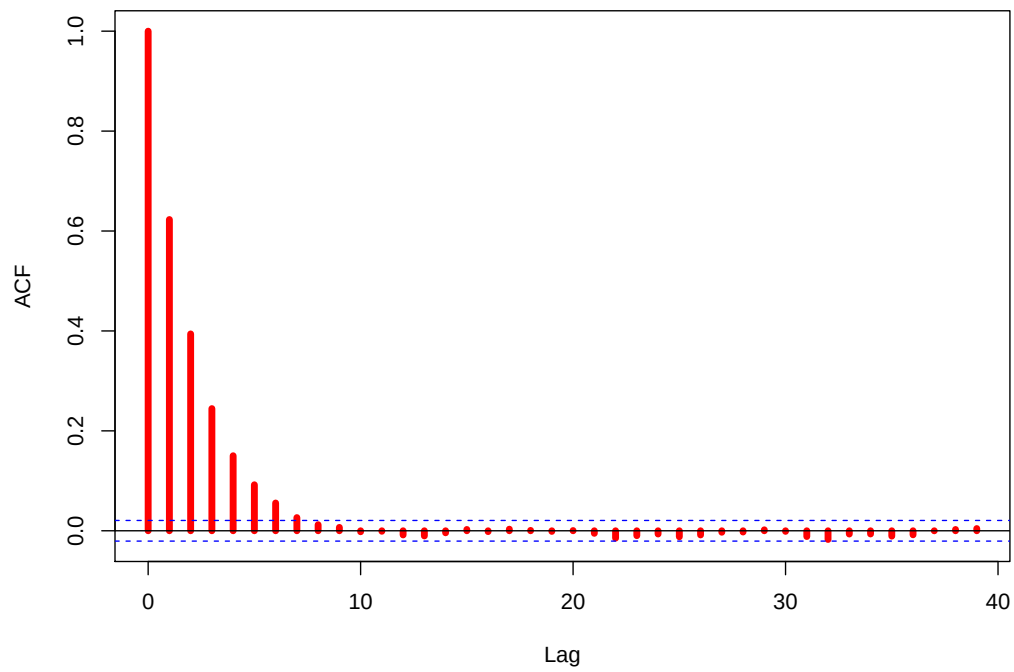
[2] Or should this be automagically?




```
samplesHistory("*", ask = FALSE, mfrow = c(1, 1))
```




```
samplesAutoC("*", chain = 1, beg = 1001, ask = FALSE, mfrow = c(1, 1))
```




```
samplesAutoC("*", chain = 4, beg = 1001, ask = FALSE, mfrow = c(1, 1))
```