Slice sampling and JAGS

Ege Rubak and Jesper Møller
Based on material by Søren Højsgaard

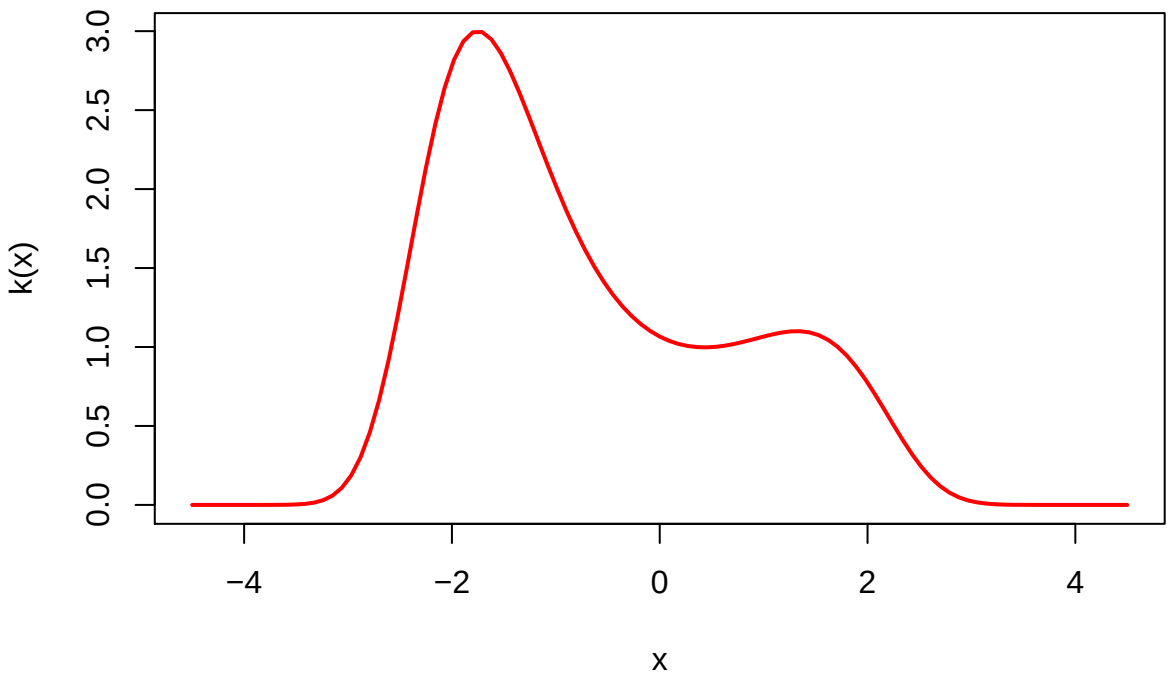
Slice sampling

Suppose we want to sample from $p(x_i|x_{-i})$ where $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_K)$.

Since x_{-i} is fixed we can regard $p(x_i|x_{-i})$ as a function of x_i alone; call this function $k_i(x_i)$ and recall that $k_i()$ is an unnormalized density.

Slice sampling is a simple approach for sampling from an unnormalized density.

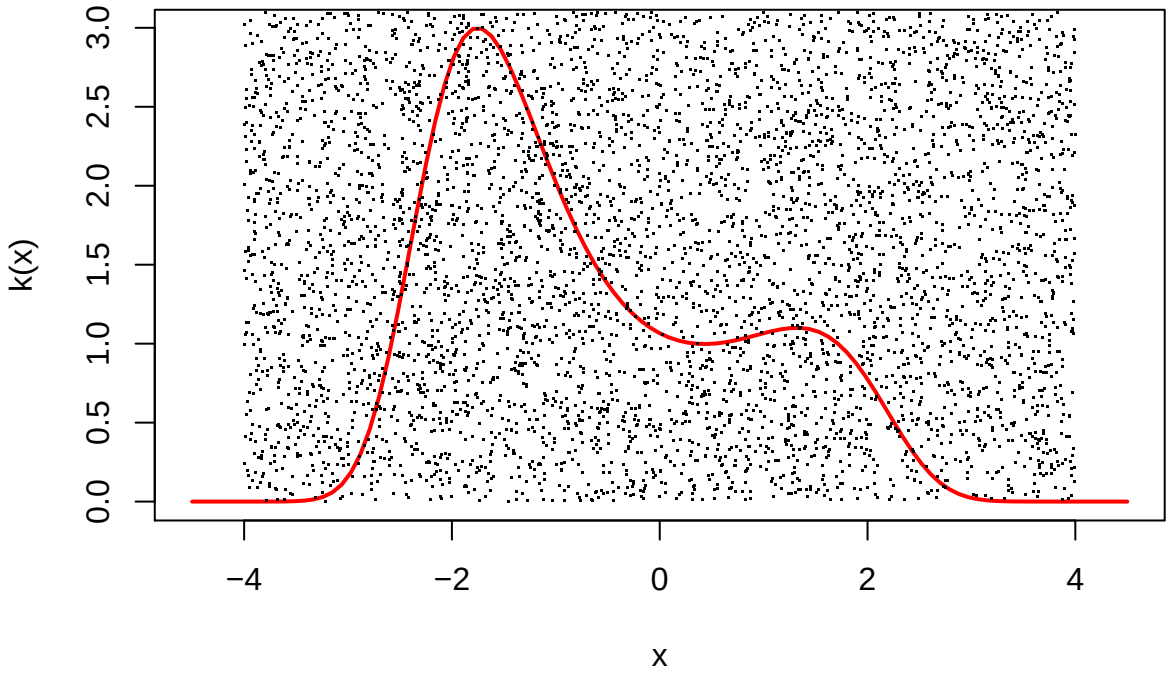
```
k <- function(x, a=.4, b=.08){exp( a * (x - a)^2 - b * x^4)}
curve(k(x), from = -4.5, to = 4.5, lwd=2, col=2)
```



Notice: $k()$ is practically zero outside $[-4, 4]$ and in this interval $k()$ takes values in, say $[0, 3]$.

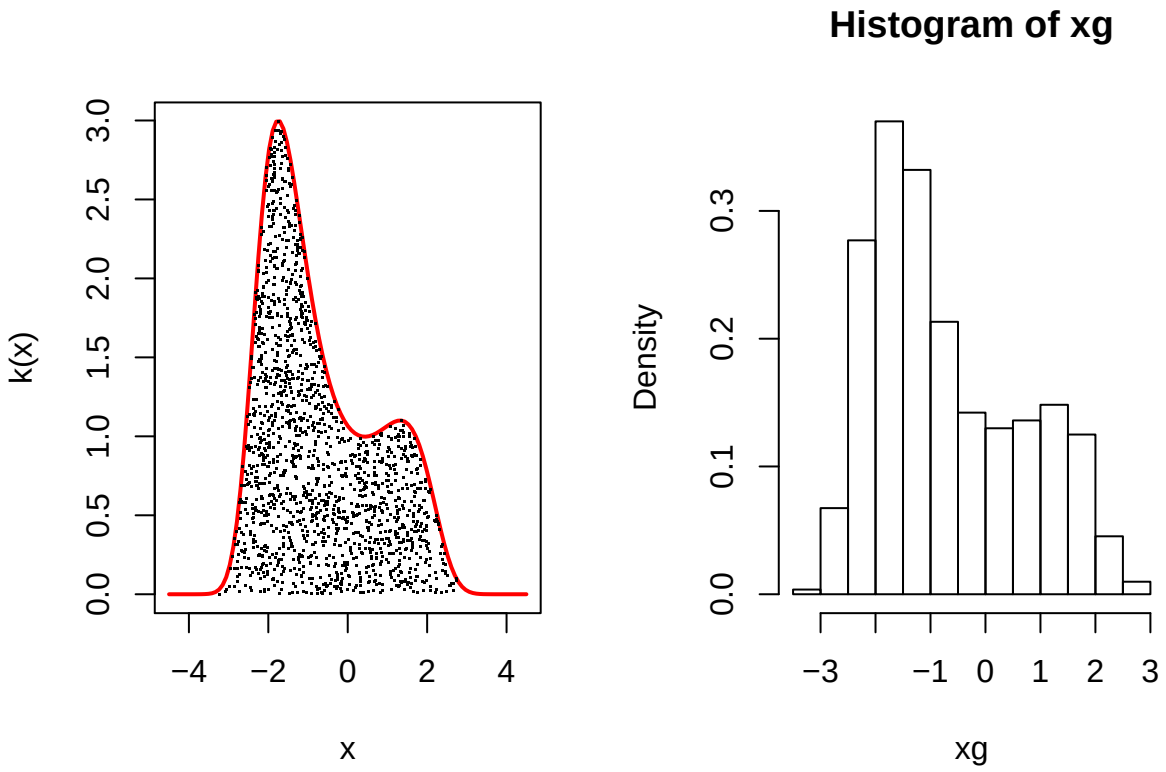
Slice sampling is based on the following idea: Sample uniformly in a "large enough" window:

```
N <- 5000
xs <- runif(N, -4, 4)
ys <- runif(N, 0, 3.1)
curve(k(x), from = -4.5, to = 4.5, lwd=2, col=2)
points(xs, ys, pch=".")
```



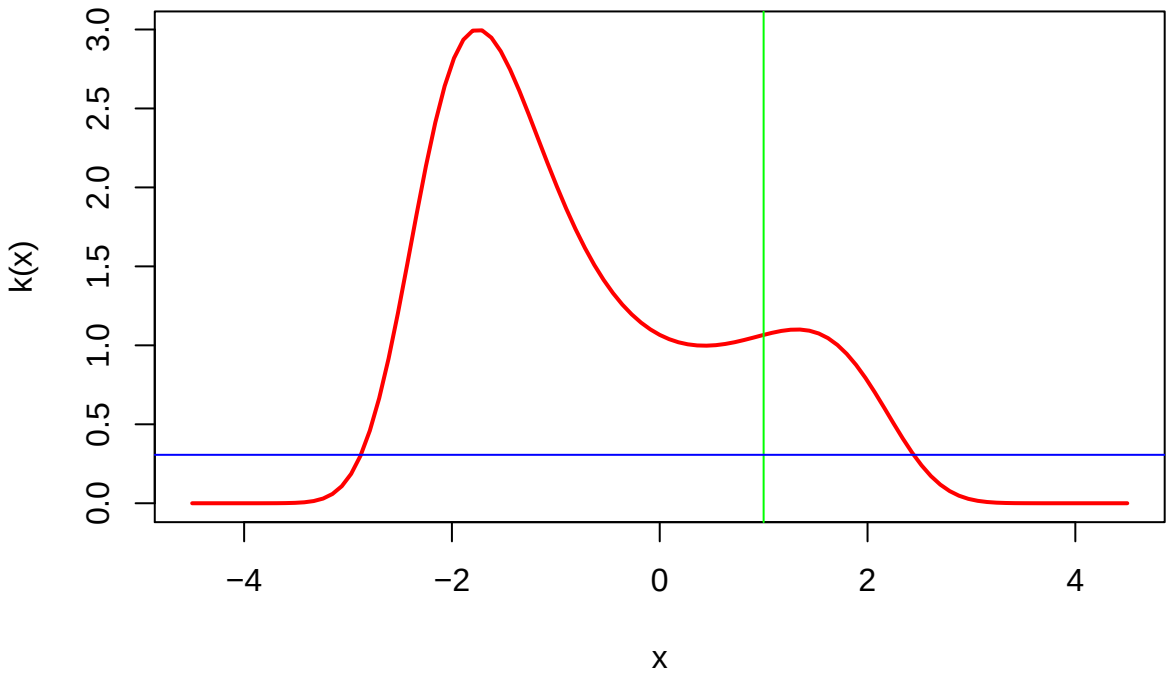
Keep those samples that fall under the curve.

```
good <- ys < k(xs)
xg <- xs[good]
yg <- ys[good]
par(mfrow=c(1, 2))
curve(k(x), -4.5, 4.5, lwd=2, col=2)
points(xg, yg, pch='.')
hist(xg, prob=TRUE)
```



Algorithm goes as follows: Given sample x^t. Pick y uniformly in $(0, k(x^t))$.

```
set.seed(123)
xt <- 1; y <- runif(1, 0, k(xt))
curve(k(x), -4.5, 4.5, lwd=2, col=2)
abline(v=xt, col='green'); abline(h=y, col='blue')
```



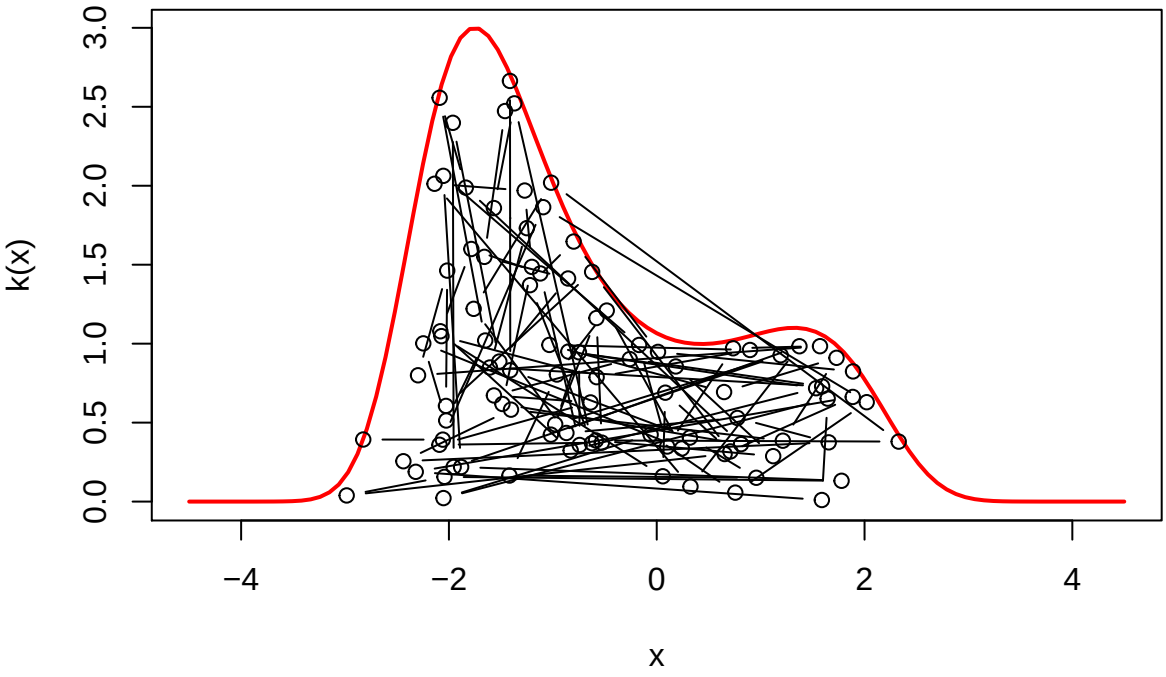
Let set $S = \{x : k(x) \geq y\}$ of x-values for which $k(x)$ is larger than y. Sample x^{t+1} uniformly from S.

- Sample y uniformly from $(0, k(x^t))$. This y defines a horizontal "slice" $S = \{x : k(x) \geq y\}$.
- Find interval $I = [L, R]$ containing all (or much) of the slice.
- Sample x^{t+1} uniformly from the part of the slice within I.

Various strategies exist for step two, but in any case it is problematic if the target is multi-modal.

```
## Inputs: function k, current x-value xc, interval width w to place around xc
slicesample <- function(k, xc, w){
    kc <- k(xc)
    y <- runif(1, 0, kc)
    ## place w randomly around xc from lower bound l to upper bound u
    a <- runif(1)
    l <- xc - a*w
    u <- xc + (1 - a)*w
    ## expand interval to the left if necessary
    kl <- k(l)
    while (kl > y){
        l <- l - w; kl <- k(l)
    }
    ## expand interval to the right if necessary
    ku <- k(u)
    while(ku > y){
        u <- u + w; ku <- k(u)
    }
    xp <- runif(1, l, u) ## propose xp
    kp <- k(xp)
    ## If k(xp) is not above y we shrink the interval and propose again
    while(kp<y){
        if (xp<xc) l <- xp else u <- xp
        xp <- runif(1, l, u)
        kp <- k(xp)
    }
    return(list(x=xp, y=y))
}
```

```
N   <- 100
y <- out <- rep(0, N)
x   <- 1
for (i in 1:N){
    rslt <- slicesample(k, x, w=1)
    x <- rslt$x
    y[i] <- rslt$y
    out[i] <- x
}
curve(k(x), lwd=2, col=2, from = -4.5, to = 4.5)
lines(out[-N], y[-1], type = "b")
```



Towards omnibus software

- With the slice sampling method, it is now clear why general purpose software can be constructed.
- JAGS (Just Another Gibbs Sampler) is one example of such software (another is STAN).
- JAGS builds on the BUGS language from WinBUGS (Windows only).
- From the WinBUGS website at Cambridge University:

  The programs are reasonably easy to use and come with a wide range of examples. There is, however, a need for caution. A knowledge of Bayesian statistics is assumed, including recognition of the potential importance of prior distributions, and MCMC is inherently less robust than analytic statistical methods. There is no in-built protection against misuse.

JAGS

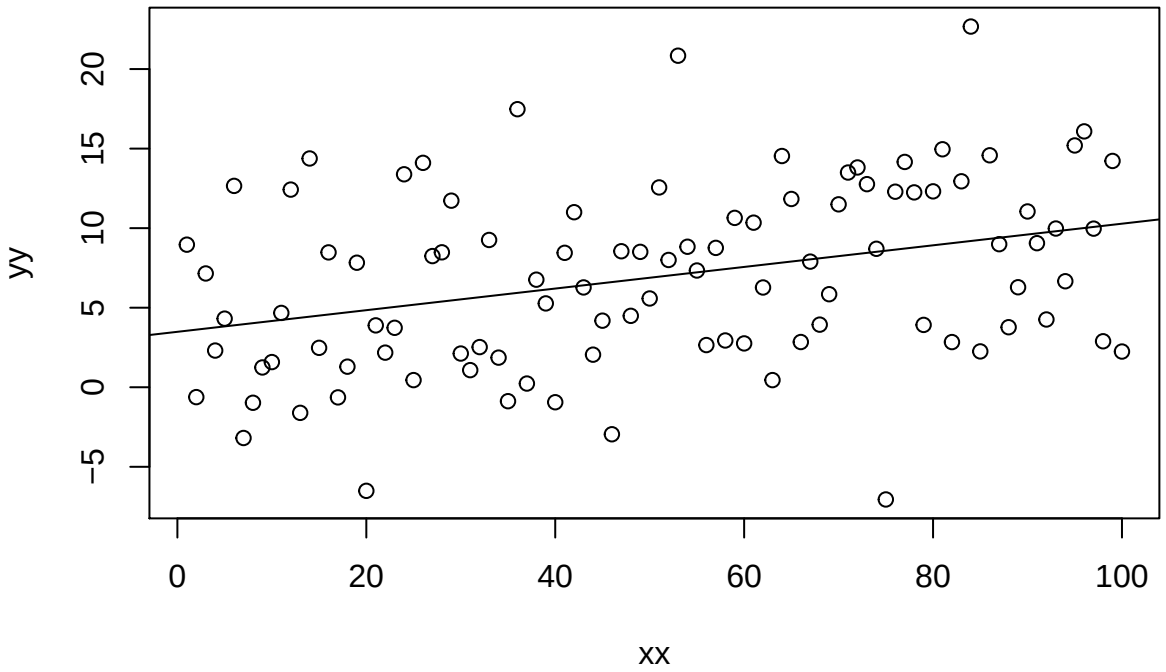
In JAGS you need to

- specify a model in a text file (can be written on the fly in R);
- supply the data;
- supply initial values (optional);
- compile the model;
- run the sampler;
- profit :-)

Linear regression in JAGS

Let's do a Bayesian analysis of simple linear regression with the following fake data:

```
xx <- 1:100
yy <- rnorm(length(xx), 2+.1*xx, sd = 5)
plot(xx, yy)
abline(coef(lm(yy~xx)))
```



Linear regression in JAGS (defining model)

```
cat("
model {
for (i in 1:N) {
Y[i]   ~ dnorm(mu[i], tau)
mu[i] <- alpha + beta * x[i]
}
alpha    ~ dnorm(0.0, 1.0E-4)
beta     ~ dnorm(0.0, 1.0E-4)
tau      ~ dgamma(1.0E-3, 1.0E-3)
}
", file="linear.jag")
```

Linear regression in JAGS (compiling model)

```
dat <- list(Y = yy, x = xx, N = length(yy))
ini <- list(alpha = 0, beta = 1, tau = 1)
library( rjags )
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.0

## Loaded modules: basemod,bugs
```

```
linmod <- jags.model("linear.jag",
    data=dat,
    n.adapt = 1000,
    inits = ini,
    n.chains = 1)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 3
##    Total graph size: 407
##
## Initializing model
```

```
print(linmod)
```

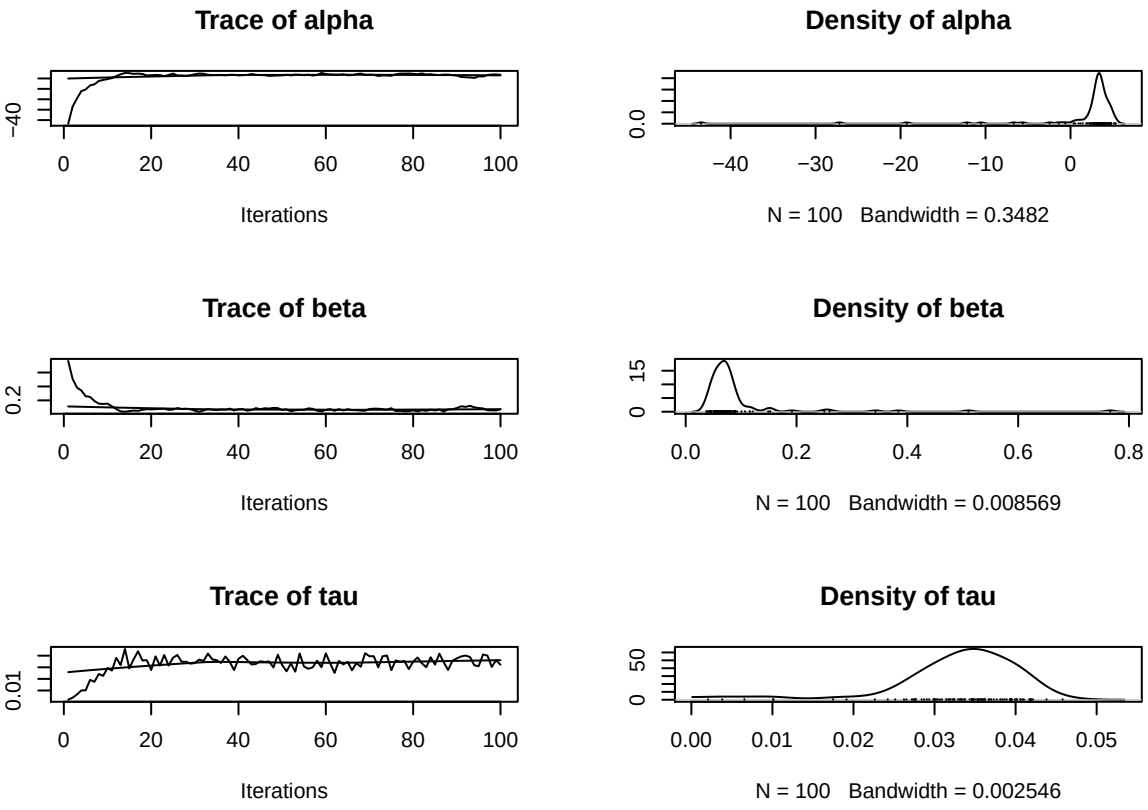
```
## JAGS model:
##
##
## model {
## for (i in 1:N) {
## Y[i]   ~ dnorm(mu[i], tau)
## mu[i] <- alpha + beta * x[i]
## }
## alpha    ~ dnorm(0.0, 1.0E-4)
## beta     ~ dnorm(0.0, 1.0E-4)
## tau      ~ dgamma(1.0E-3, 1.0E-3)
## }
## Fully observed variables:
##  N Y x
```

Linear regression in JAGS (running sampler)

```
chain <- coda.samples(linmod,
  var = c("alpha", "beta", "tau"),
  n.iter = 100,
  thin = 1 )
```

Linear regression in JAGS (profitting)

```
plot(chain)
```



Beetles in JAGS

- Note: A Cauchy distribution with scale γ is the same as a t-distribution with location 0, precision $1/\gamma^2$ and degrees of freedom 1.

```
cat("
model {
for (i in 1:N) {
mu[i] <- alpha + beta * x[i]
p[i] <- exp(mu[i])/(1+exp(mu[i]))
Y[i]   ~ dbin(p[i], 1)
}
alpha    ~ dt(0.0, 1/(10^2), 1)
beta     ~ dt(0.0, 1/(2^2), 1)
}
", file="logistic.jag")
```

Beetles in JAGS

```
data_url <- "https://asta.math.aau.dk/course/bayes/2021/?file=beetles.dat"
beetles <- read.table(data_url)
dat <- list(Y = beetles[,2], x = beetles[,1], N = nrow(beetles))
ini <- list(
  list(alpha = 0, beta = 1),
```

```
  list(alpha = 60, beta = -30))
logisticmod <- jags.model("logistic.jag",
    data=dat,
    n.adapt = 1000,
    inits = ini,
    n.chains = 2)
```

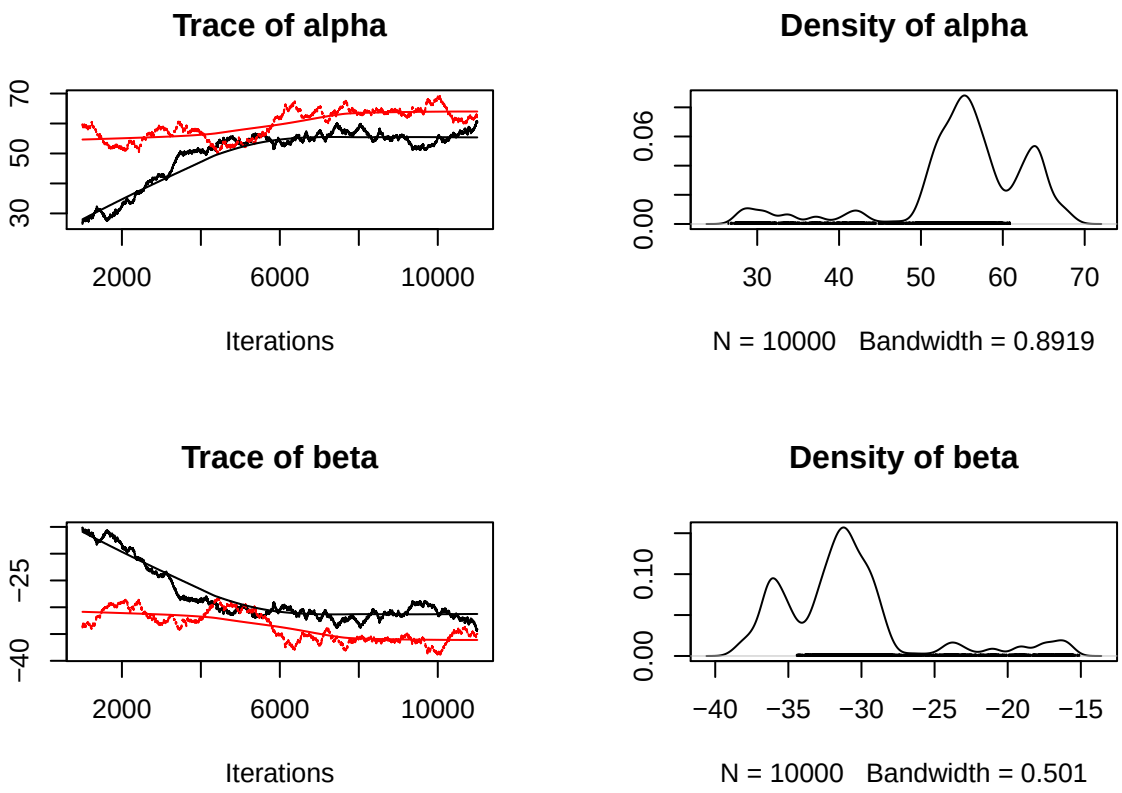
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 481
##    Unobserved stochastic nodes: 2
##    Total graph size: 1013
##
## Initializing model
```

Beetles in JAGS

```
chain <- coda.samples(logisticmod,
  var = c("alpha", "beta"),
  n.iter = 10000,
  thin = 1 )
```

Beetles in JAGS

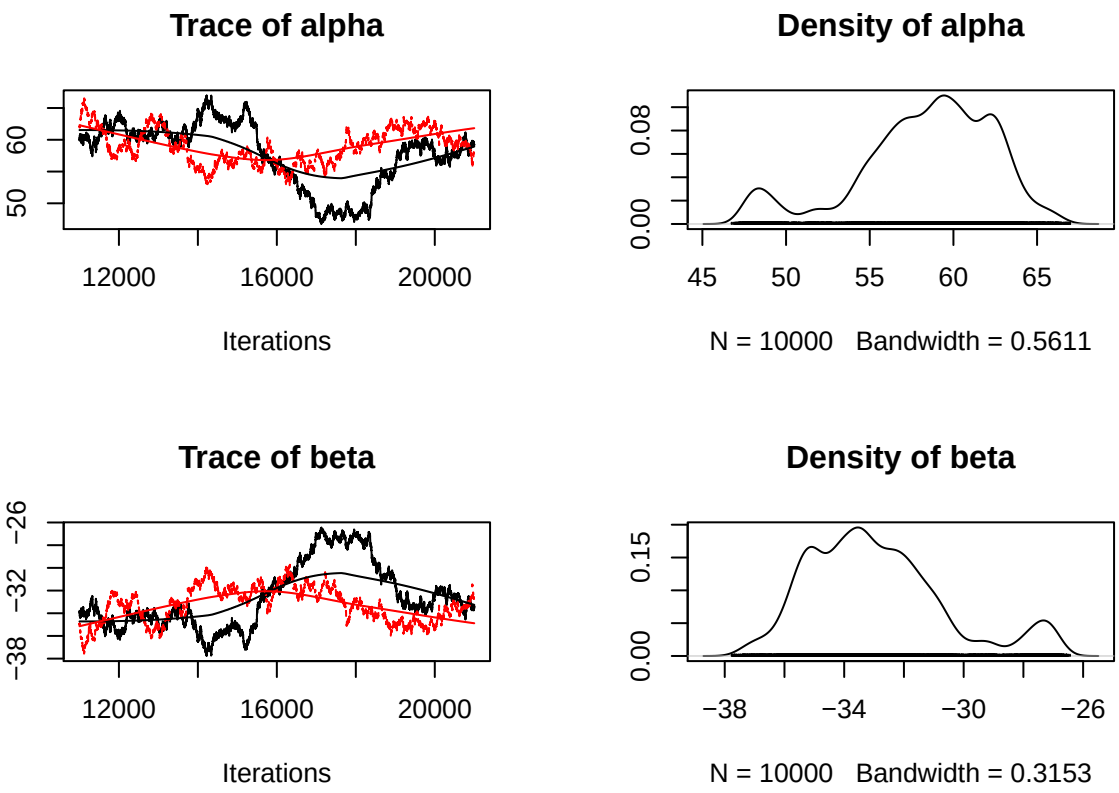
```
plot(chain)
```



Beetles in JAGS - Continuing the sampler

```
# Continuing sampling from earlier
chain2 <- coda.samples(logisticmod,
  var = c("alpha", "beta"),
  n.iter = 10000,
  thin = 1 )
```

```
plot(chain2)
```



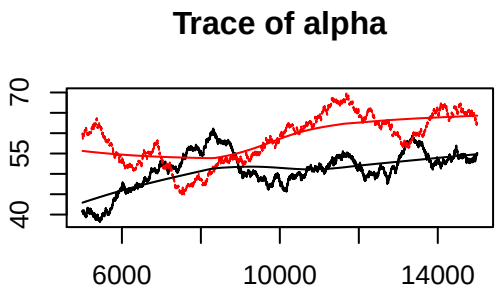
Beetles in JAGS - Longer adaptation

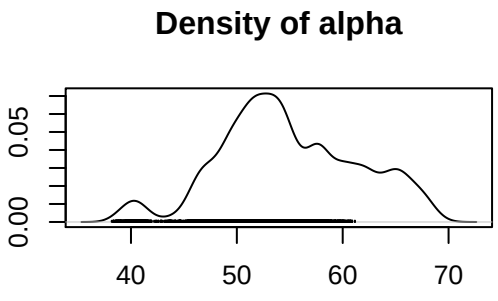
```
logisticmod <- jags.model("logistic.jag",
    data=dat,
    n.adapt = 5000,
    inits = ini,
    n.chains = 2)
```

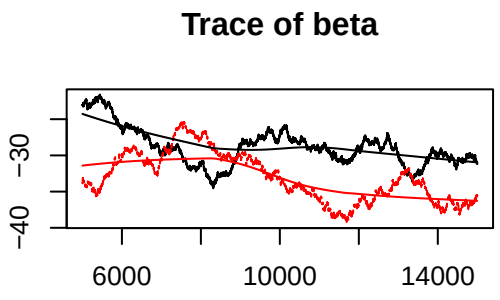
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 481
##    Unobserved stochastic nodes: 2
##    Total graph size: 1013
##
## Initializing model
```

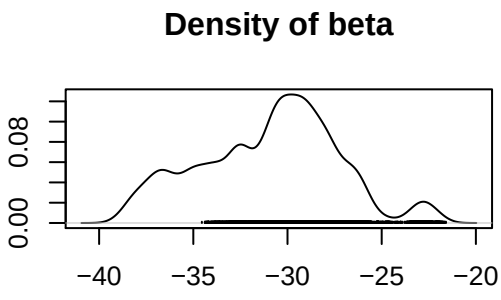
```
chain_new <- coda.samples(logisticmod,
  var = c("alpha", "beta"),
  n.iter = 10000,
  thin = 1 )
```

```
plot(chain_new)
```









Mixture in JAGS

```
cat("
model {
    # Likelihood:
    for( i in 1 : N ) {
      Y[i] ~ dnorm( mu[i] , 1 )
      mu[i] <- muvec[ Z[i] ]
      Z[i] ~ dcat( lambda[1:k] )
    }
    # Prior:
    for ( j in 1:k ) {
      muvec[j] ~ dnorm( 0 , 1.0E-10 )
    }
    lambda[1:k] ~ ddirch( rep(1, k ))
}
", file="mixture.jag")
```

Mixture in JAGS

```
data_url <- "https://asta.math.aau.dk/course/bayes/2021/?file=simmix.csv"
simmix <- read.csv(data_url)
k <- 3
dat <- list(Y = simmix[,1], k = k, N = nrow(simmix))
```

```
ini <- list(
  # list(lambda = rep(1/k, k), muvec = seq(-2,2,length.out = k)),
  list(lambda = rep(1/k, k), muvec = seq(-5,5,length.out = k)))
mixturemod <- jags.model("mixture.jag",
    data=dat,
    n.adapt = 1000,
    inits = ini,
    n.chains = 1)
```

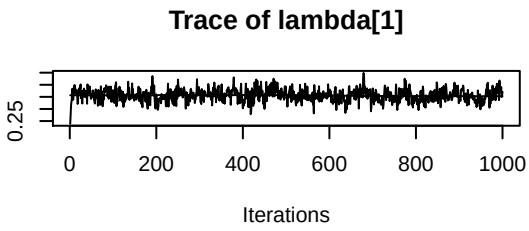
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 500
##    Unobserved stochastic nodes: 504
##    Total graph size: 1510
##
## Initializing model
```

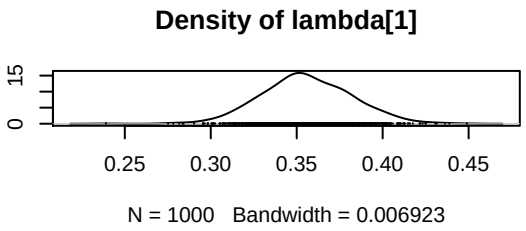
Mixture in JAGS

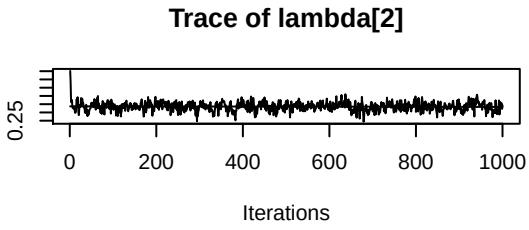
```
chain <- coda.samples(mixturemod,
  var = c("muvec", "lambda"),
  n.iter = 1000,
  thin = 1 )
```

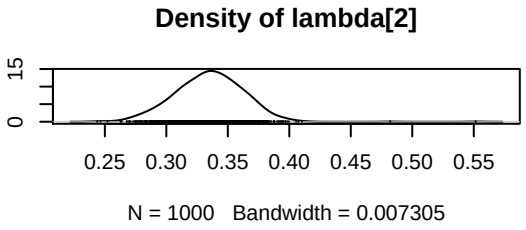
Mixture in JAGS

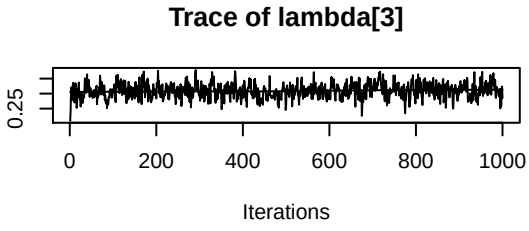
```
plot(chain[,1:3])
```

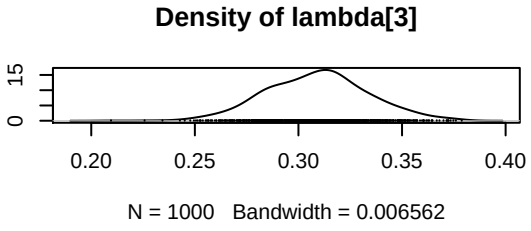













```
plot(chain[,4:6])
```