Bertinelli_Gabriele_Rlab05

June 1, 2024

1 RLab05 - Gabriele Bertinelli (2103359)

[ ]:
```r
library(tidyverse)
library(gridExtra)
library(latex2exp)
library(emdbook)
library(bayestestR)
library(coda)
library(magrittr)
library(rjags)

set.seed(2103359)
```

1)

[2]:
```r
my.mcmc <- function(func = func, mu.init = mu.init, n.iter = n.iter, burn.in = burn.in, thinning = thinning, norm = FALSE) {
  # Initialize chain and acceptance counter
  mu.cur <- mu.init
  func.samp <- matrix(data=NA, nrow=n.iter, ncol=4)
  func.samp[1:2, 2] <- c(mu.cur, mu.cur)
  n.acc <- 0
  acc.rate <- 0

  func.samp[1:2, 1] <- c(1, 2)

  # Loop for MCMC iterations
  for (i in 3:n.iter) {
    # Propose new value for mu
    mu.prop <- rnorm(1, func.samp[i-1, 2], 1)

    # Calculate acceptance probability ratio
    rho <- min(1, func(mu.prop, norm) / func(func.samp[i-1, 2], norm))

    # Accept or reject proposal based on uniform random draw
    if (runif(1) < rho) {
      func.samp[i, 2] <- mu.prop
```

```
      n.acc <- n.acc + 1
    } else {
      func.samp[i, 2] <- func.samp[i-1, 2]
    }
        func.samp[i, 1]        <- i
        func.samp[i, 3] <- n.acc
        func.samp[i, 4] <- n.acc / i
  }

    # Remove burn-in (initial samples for convergence)
      start <- round(n.iter * burn.in)
      func.samp <- func.samp[start:n.iter, ]

      # Apply thinning
      func.samp <- func.samp[seq(1, nrow(func.samp), by=thinning), ]

      # Return thinned MCMC chain, acceptance rate, and ACF
      return(func.samp)
}
```

[3]:
```
post.func <- function(theta, norm=FALSE) {

    if (norm==FALSE) {

                return(1/2*exp(-((theta+3)^2)/2) + 1/2*exp(-((theta-3)^2)/2))␣
 ↪#unnormalized post
        }

    else {
        # integ <- integrate(u.post, -Inf, +Inf)$value
        # print(integ)
    integ <- sqrt(2*pi)

    return((1/2*exp(-((theta+3)^2)/2) + 1/2*exp(-((theta-3)^2)/2)) / integ)␣
 ↪#normalized post
    }
}
```

[4]:
```
theta.init <- rnorm(1, 0, 1)  # Initial value for theta
thinning <- 1
burn.in <- 0.15
test.chain <- my.mcmc(func = post.func, mu.init = rnorm(1,0,1), n.iter =␣
 ↪100000, burn.in = burn.in, thinning=thinning, norm=F)
test.chain <- as.data.frame(test.chain)
colnames(test.chain) <- c('it', 'theta', 'n.acc', 'acc.rate')

print(length(test.chain[,1]))
```

```
[1] 85001
```

[5]:
```
posterior.mean <- mean(test.chain$theta) # must be centered in 0

posterior.sd <- sd(test.chain$theta)
x_min <- posterior.mean - posterior.sd # true value is x=-3
x_max <- posterior.mean + posterior.sd # true value is x=+3

cat("Posterior Mean:", round(posterior.mean, 2), "\n")
cat("Posterior Standard Deviation:", round(posterior.sd, 2), "\n")
print(sprintf("Firs max is at x=%.2f Second max is at x=%.2f", x_min, x_max))
```

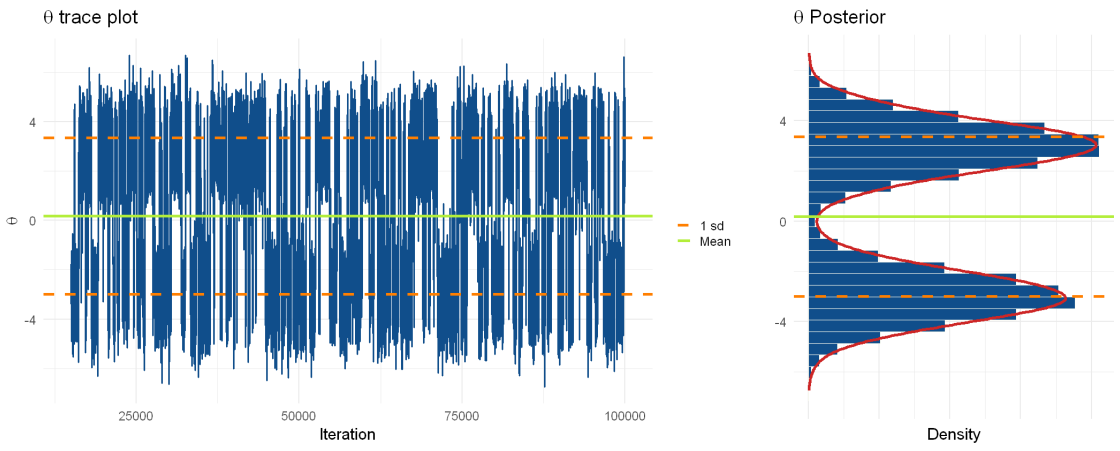
```
Posterior Mean: 0.18
Posterior Standard Deviation: 3.17
[1] "Firs max is at x=-2.99 Second max is at x=3.35"
```

[6]:
```
# cred.int <- quantile(test.chain$theta, c(0.025, 0.975))

# cat("95% Credible Interval:", round(cred.int[1], 2), '-', round(cred.int[2],␣
↪2), "\n")
```

[7]:
```
# Print acceptance rate
cat("Acceptance Rate:", tail(test.chain$acc.rate, 1), "\n")

options(repr.plot.width = 17.5, repr.plot.height = 7)

# Custom layout
layout <- matrix(c(1, 1, 2), ncol = 3, byrow = TRUE)

# Visualize posterior distribution
post.hist <- ggplot(data = test.chain) +
       geom_histogram(aes(x = theta, after_stat(density)), bins = 30, color =␣
↪"ivory", fill = "dodgerblue4") +
       geom_density(aes(x = theta), color = "firebrick3", lwd = 1.5) +
       geom_vline(aes(xintercept = posterior.mean, color='Mean'), lwd=1.5)+
       geom_vline(aes(xintercept = posterior.mean - posterior.sd, color='1␣
↪sd'), lwd=1.5, linetype='dashed') +
       geom_vline(aes(xintercept = posterior.mean + posterior.sd, color='1␣
↪sd'), lwd=1.5, linetype='dashed') +
       # geom_vline(aes(xintercept = posterior.median, color='Median'), lwd=1.
↪5) +
       # geom_vline(aes(xintercept = cred.int[1], color='95% CI'), lwd=1.5,␣
↪linetype='dotdash') +
       # geom_vline(aes(xintercept = cred.int[2], color='95% CI'), lwd=1.5,␣
↪linetype='dotdash') +
       labs(title = TeX(r"(\theta Posterior)"), x = '', y = "Density") +
```

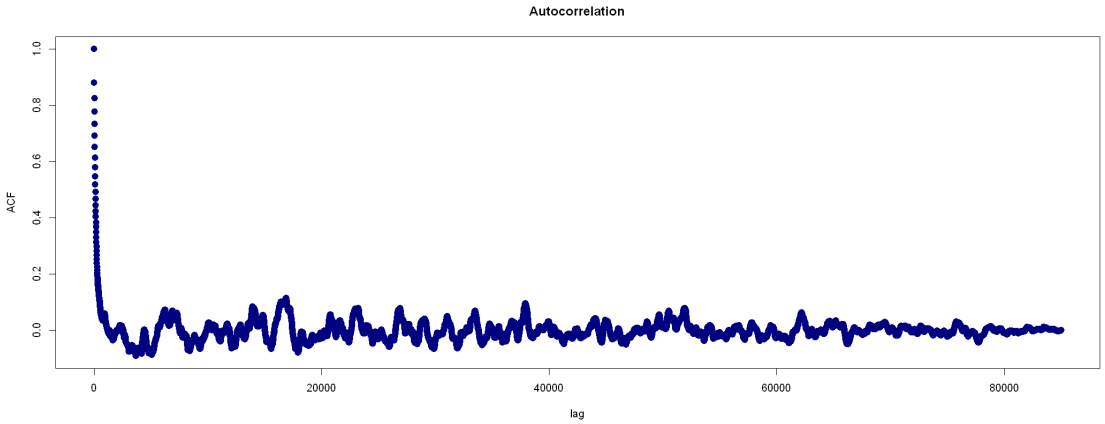
```
        scale_color_manual(values = c('Mean' = 'olivedrab2', '1 sd' =␣
↪'#FF7F00', '95% CI' = 'violetred3')) +
        theme_minimal(base_size = 18) + theme(axis.text.x=element_blank(),␣
↪legend.position="none") + coord_flip()

trace.plot <- ggplot(data = test.chain) +
        geom_line(aes(x = it, y = theta), color = "dodgerblue4", lwd=1) +
        geom_hline(aes(yintercept = posterior.mean, color='Mean'), lwd=1.5)+
        geom_hline(aes(yintercept = posterior.mean - posterior.sd, color='1␣
↪sd'), lwd=1.5, linetype='dashed') +
        geom_hline(aes(yintercept = posterior.mean + posterior.sd, color='1␣
↪sd'), lwd=1.5, linetype='dashed') +
        # geom_hline(aes(yintercept = posterior.median, color='Median'), lwd=1.
↪5) +
        # geom_hline(aes(yintercept = cred.int[1], color='95% CI'), lwd=1.5,␣
↪linetype='dotdash') +
        # geom_hline(aes(yintercept = cred.int[2], color='95% CI'), lwd=1.5,␣
↪linetype='dotdash') +
        labs(title = TeX(r"(\theta trace plot)"), x = "Iteration", y =␣
↪TeX(r"(\theta)"), color='', lwd=1.5) +
        scale_color_manual(values = c('Mean' = 'olivedrab2', '1 sd' =␣
↪'#FF7F00', '95% CI' = 'violetred3')) +
        theme_minimal(base_size = 18)

grid.arrange(trace.plot, post.hist, layout_matrix = layout)
```

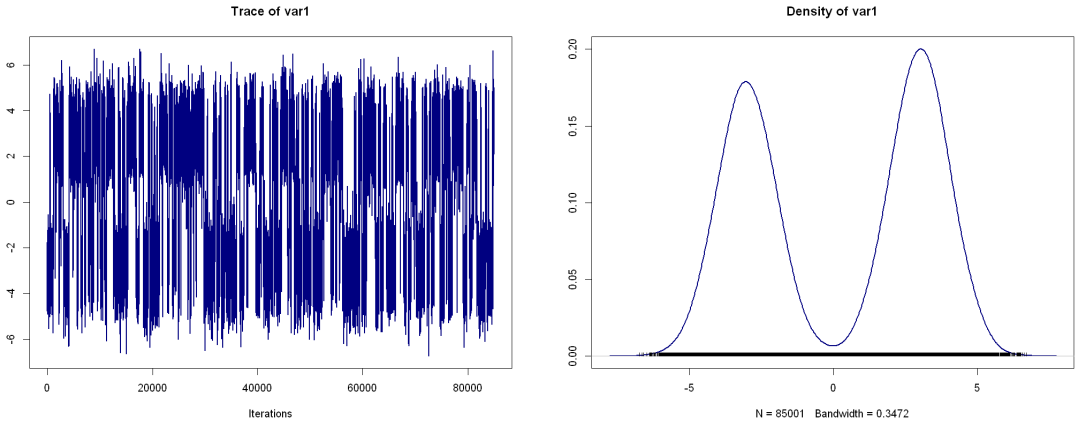
```
Acceptance Rate: 0.71276
```



[8]:
```
test.mcmc <- coda::mcmc(test.chain$theta)
my.lags <- seq(0, length(test.mcmc)-1, 10)
y1 <- autocorr(test.mcmc, lags=my.lags)
```

[9]:
```
plot(my.lags, y1, pch=19, col='navy', xlab='lag', ylab='ACF', cex=1.3,␣
 ↪main='Autocorrelation')

# Additional diagnostic plots
plot(test.mcmc, col='navy', lwd=2)
```





Instead of looking for the best set of parameters by printing the plots, I built a simple grid-search. In the first GS I only check which couple of `burn.in` and `thinning` returns the mean that is closest to 0 and the sd that is closest to 3.
Checking also autocorrelation is a bit too stringent. We can find one ACF value lower than another but it is not certain that the mean and sd values are closest to the true values (and most of the time we are stuck in the first parameters' couple). Moreover, the trend of the ACF is similar in all cases for the chosen values.

Must be noted that even if I set a seed for random generation, each mcmc with the same parameters

will give slightly different results from the ones from the GS.

[10]:
```
# Grid search function
grid_search <- function(burn.in.values, thinning.values, n.iter, post.func, mu.
↪init) {



   # chain.i <- my.mcmc(post.func, mu.init, n.iter, burn.in.values[1],␣
↪thinning.values[1])
   # chain.i <- as.data.frame(chain.i)
   # colnames(chain.i) <- c('it', 'theta', 'n.acc', 'acc.rate')

   # theta_mcmc.i <- coda::mcmc(chain.i$theta)
   # my.lags <- seq(0, length(theta_mcmc.i)-1, 10)

   # acf.i <- autocorr.diag(theta_mcmc.i, lags=my.lags)
   # acf1.i <- mean(acf.i[10:length(acf.i)])
   # mean_val.i <- mean(chain.i$theta)
   # sd_val.i <- sd(chain.i$theta)

   # cat('meanA', mean_val.i)
   # cat('sdA', sd_val.i)
   # cat('acfA', acf1.i)
   # cat('\n')

   # if (is.na(mean_val.i) | is.na(sd_val.i)) {break}

   # best_params <- list(burn.in = burn.in.values[1], thinning = thinning.
↪values[1],
   #                     acf = acf1.i, mean = mean_val.i, sd = sd_val.i)

   best_params <- list(burn.in = NULL, thinning = NULL,
                       acf = Inf, mean = Inf, sd = NULL)

   for (burn.in in burn.in.values) {
       for (thinning in thinning.values) {
           chain <- my.mcmc(post.func, mu.init, n.iter, burn.in, thinning)
           chain <- as.data.frame(chain)
           colnames(chain) <- c('it', 'theta', 'n.acc', 'acc.rate')

           theta_mcmc <- coda::mcmc(chain$theta)

           acf <- autocorr.diag(theta_mcmc, lags=my.lags)
           acf1 <- mean(acf[10:length(acf)])


           mean_val <- mean(chain$theta)
```

```
            sd_val <- sd(chain$theta)

            if (is.na(mean_val) | is.na(sd_val)) {break}

            # cat('mean', abs(best_params$mean - mean_val))
            # cat('min', abs(mean_val - sd_val + 3))
            # cat('max', abs(mean_val + sd_val - 3))
            # cat('\n')

            # cat('mean', mean_val)
            # cat('sd', sd_val)
            # cat('acf', acf1)
            # cat('\n')

            if (abs(mean_val) < abs(best_params$mean) &&
                abs(sd_val - 3) < 0.15) {

                best_params <- list(burn.in = burn.in, thinning = thinning, acf␣
 ↪= acf1, mean = mean_val, sd = sd_val)
            }
        }
    }

    return(best_params)
}
```

[11]:
```
# Define the grid values for burn-in and thinning
burn.in.values <- seq(0, 0.5, by=0.05)
thinning.values <- c(1, 25, 50, 100)

# Initial value for theta
theta.init <- rnorm(1, 0, 1)

# Perform grid search
best_params <- grid_search(burn.in.values, thinning.values, n.iter = 100000,␣
 ↪post.func, theta.init)

# Output the best parameters
print(best_params)
```

```
$burn.in
[1] 0.2

$thinning
[1] 100

$acf
```

```
[1] -0.003863594

$mean
[1] -0.003156569

$sd
[1] 3.134345
```

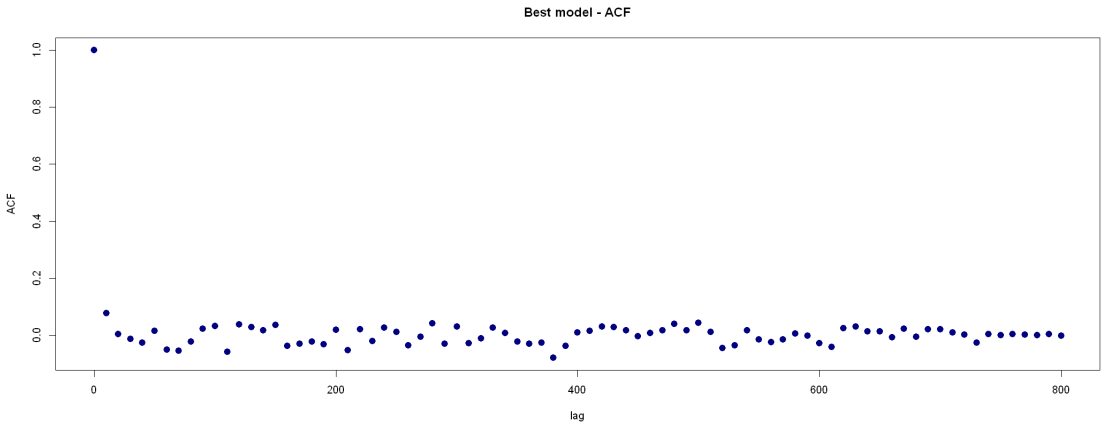
[12]:
```r
best.chain <- my.mcmc(func = post.func, mu.init = rnorm(1,0,1), n.iter =␣
↪100000,
                      burn.in = best_params$burn.in, thinning =␣
↪best_params$thinning, norm=F)

best.chain <- as.data.frame(best.chain)
colnames(best.chain) <- c('it', 'theta', 'n.acc', 'acc.rate')
```

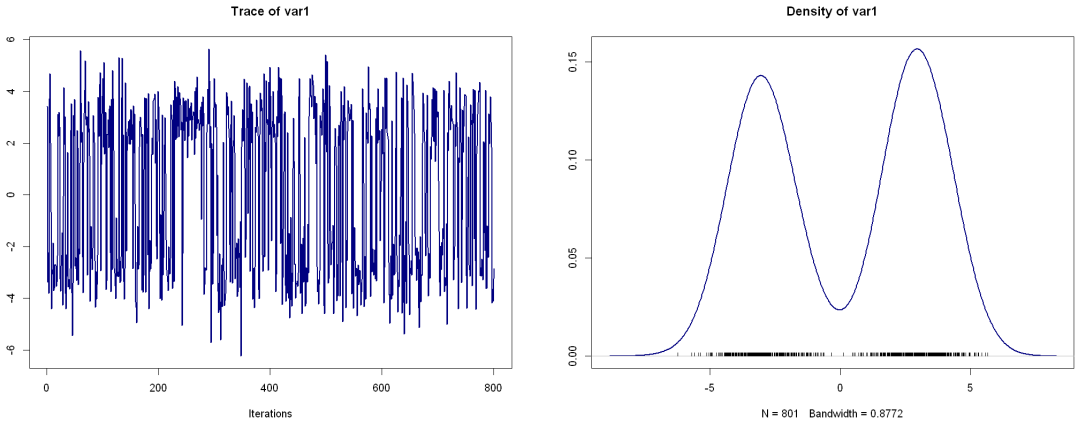
[13]:
```r
posterior.mean <- mean(best.chain$theta) # must be centered in 0

posterior.sd <- sd(best.chain$theta)
x_min <- posterior.mean - posterior.sd # true value is x=-3
x_max <- posterior.mean + posterior.sd # true value is x=+3

cat("Posterior Mean:", round(posterior.mean, 2), "\n")
cat("Posterior Standard Deviation:", round(posterior.sd, 2), "\n")
print(sprintf("Firs max is at x=%.2f Second max is at x=%.2f", x_min, x_max))
```

```
Posterior Mean: 0.13
Posterior Standard Deviation: 3.15
[1] "Firs max is at x=-3.02 Second max is at x=3.28"
```

[14]:
```r
best.mcmc <- coda::mcmc(best.chain$theta)
my.lags <- seq(0, length(best.mcmc)-1, 10)
best.acf <- autocorr(best.mcmc, lags=my.lags)
```

[15]:
```r
plot(my.lags, best.acf, pch=19, col='navy', xlab='lag', ylab='ACF', cex=1.3,␣
↪main='Best model - ACF')

# Additional diagnostic plots
plot(best.mcmc, col='navy', lwd=2)
```





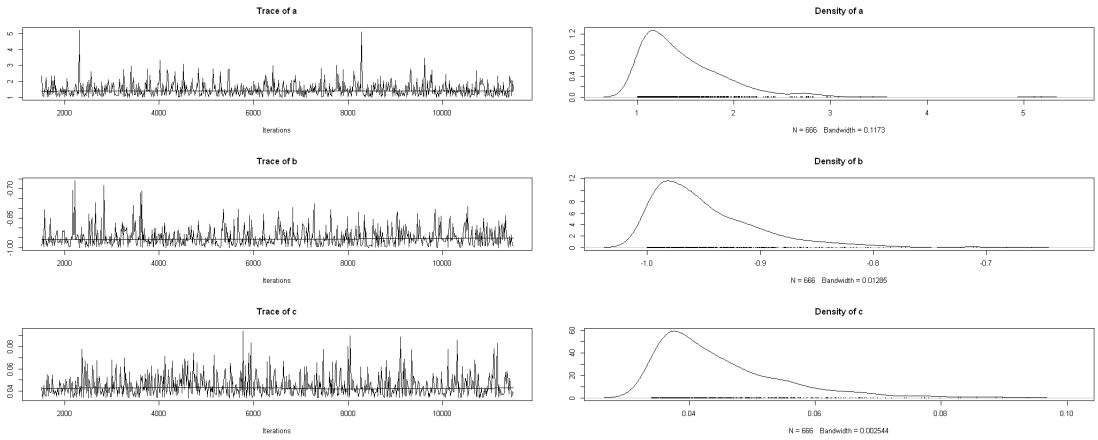
2)

[16]:
```
# Create data object
data1 <- list(
  Y = c(-7.821, -1.494, -15.444, -10.807, -13.735, -14.442, -15.892, -18.326),
  X = c(5, 6, 7, 8, 9, 10, 11, 12)
)

# Define the model in JAGS syntax
model1 <- 'jm_ex5.2.bug'

jm1 <- jags.model(model1, data1)
update(jm1, 500) #burnin

chain1 <- coda.samples(jm1, c("a", "b", "c"), n.iter=10000, thin=15)
```

```
# Summarize the results
print(summary(chain1))
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 8
   Unobserved stochastic nodes: 3
   Total graph size: 41

Initializing model


Iterations = 1515:11490
Thinning interval = 15
Number of chains = 1
Sample size per chain = 666

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

      Mean      SD  Naive SE Time-series SE
a  1.48292 0.47117 0.0182575      0.0182575
b -0.94727 0.04978 0.0019288      0.0019288
c  0.04479 0.01003 0.0003888      0.0003443

2. Quantiles for each variable:

      2.5%      25%      50%      75%    97.5%
a  1.01070  1.14722  1.35361  1.69128  2.72062
b -0.99872 -0.98399 -0.96122 -0.92438 -0.81944
c  0.03431  0.03736  0.04182  0.04917  0.07029
```

[17]:
```
plot(chain1)
```

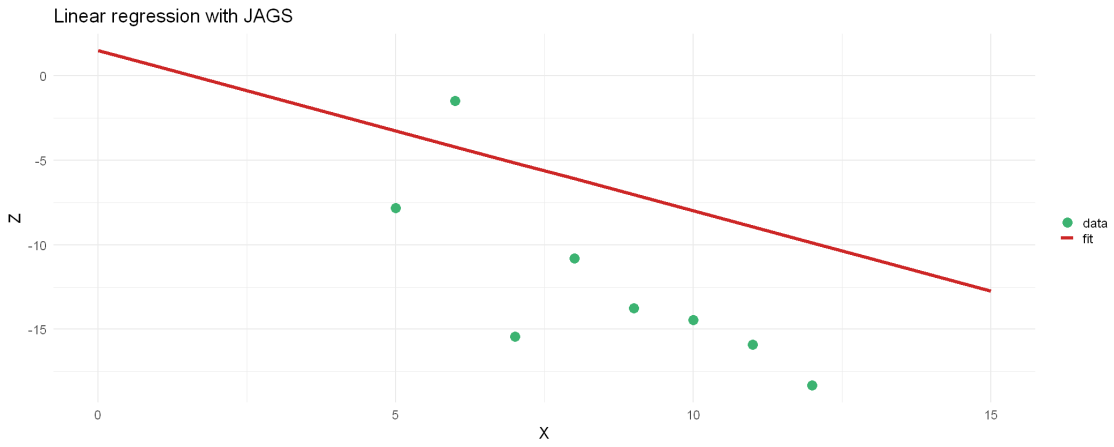


[18]:
```
a.mean <- summary(chain1)$statistics[1,1]
b.mean <- summary(chain1)$statistics[2,1]
c.mean <- summary(chain1)$statistics[3,1]

print(sprintf('Candidate value for a is: %.2f', a.mean))
print(sprintf('Candidate value for b is: %.2f', b.mean))
print(sprintf('Candidate value for c is: %.2f', c.mean))
cat('\n')

a.sd <- summary(chain1)$statistics[2,1]
b.sd <- summary(chain1)$statistics[2,2]
c.sd <- summary(chain1)$statistics[2,3]

a.ci.lo <- summary(chain1)$quantiles[1,1]
a.ci.hi <- summary(chain1)$quantiles[1,5]
b.ci.lo <- summary(chain1)$quantiles[2,1]
b.ci.hi <- summary(chain1)$quantiles[2,5]
c.ci.lo <- summary(chain1)$quantiles[3,1]
c.ci.hi <- summary(chain1)$quantiles[3,5]

print(sprintf('a 0.95 CI is [%.2f, %.2f]', a.ci.lo, a.ci.hi))
print(sprintf('b 0.95 CI is [%.2f, %.2f]', b.ci.lo, b.ci.hi))
print(sprintf('c 0.95 CI is [%.2f, %.2f]', c.ci.lo, c.ci.hi))
```

```
[1] "Candidate value for a is: 1.48"
[1] "Candidate value for b is: -0.95"
[1] "Candidate value for c is: 0.04"

[1] "a 0.95 CI is [1.01, 2.72]"
[1] "b 0.95 CI is [-1.00, -0.82]"
[1] "c 0.95 CI is [0.03, 0.07]"
```

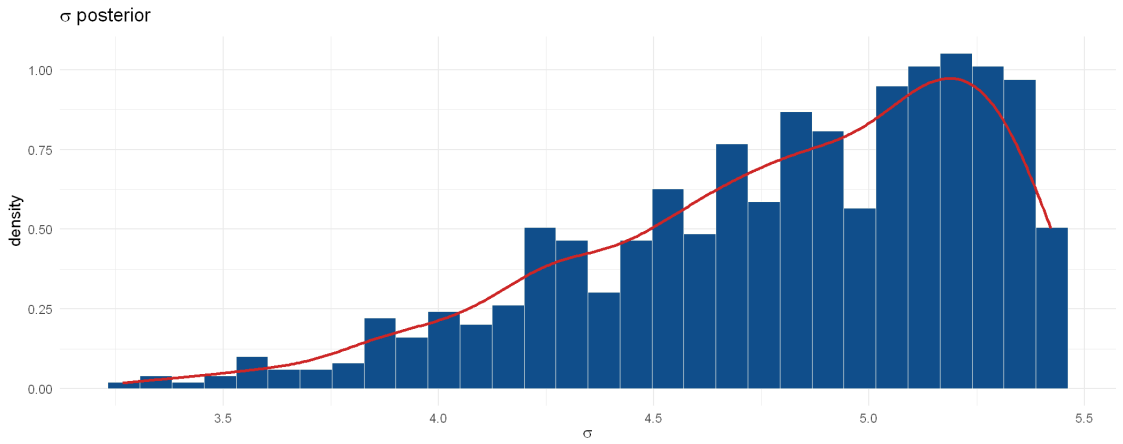
[19]:
```r
lin.reg <- function(X, a, b) {
    return(a + b * X)
}

num <- seq(0, 15, 1)
z.fit <- lin.reg(num, a.mean, b.mean)
data1.fit <- data.frame(num = num, z.fit = z.fit)
```

[20]:
```r
interp <- ggplot() +
    geom_point(aes(x=data1$X, y=data1$Y, color='data'), size=5) +
    geom_line(aes(x=data1.fit$num, y=data1.fit$z.fit, color='fit'), lwd=1.7) +
    labs(title='Linear regression with JAGS', x='X', y='Z', color='') +
    theme_minimal(base_size = 18) +
        scale_color_manual(values=c('data'= 'mediumseagreen', 'fit'=␣
 ↪'firebrick3' ))

interp
```



[21]:
```r
chain1.df <- as.data.frame(as.mcmc(chain1))

c_samples <- chain1.df$c  # Assuming 'sigma.sq' represents c in your model


sigma_samples <- 1 / sqrt(c_samples)

sigma.post <- ggplot() +
    geom_histogram(aes(x=sigma_samples, after_stat(density)), bins = 30, color␣
 ↪= "ivory", fill = "dodgerblue4") +
    geom_density(aes(x=sigma_samples), color = "firebrick3", lwd = 1.5) +
    labs(title=TeX(r'(\sigma posterior)'), x=TeX(r'(\sigma)')) +
    theme_minimal(base_size = 18)
```

```
sigma.post
```



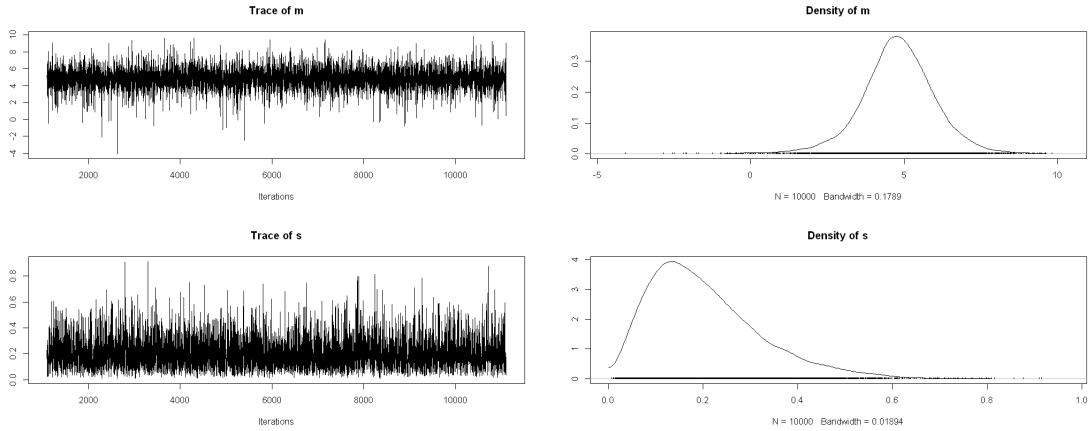
3)

[22]:
```
# Create data object
data2 <- list(
  x = c(2.06, 5.56, 7.93, 6.56, 2.05)
)

# Define the model in JAGS syntax
model2 <- 'jm_ex5.3.bug'

jm2 <- jags.model(model2, data2)
update(jm2, 100) #burnin

chain2 <- coda.samples(jm2, c("m", "s"), n.iter=10000, thin=1)

# Summarize the results
print(summary(chain2))
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 5
   Unobserved stochastic nodes: 2
   Total graph size: 11

Initializing model
```

```
Iterations = 1101:11100
Thinning interval = 1
Number of chains = 1
Sample size per chain = 10000

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

    Mean     SD Naive SE Time-series SE
m 4.7985 1.2423 0.012423       0.019091
s 0.2056 0.1197 0.001197       0.002073

2. Quantiles for each variable:

     2.5%    25%    50%    75%  97.5%
m 2.20061 4.1089 4.8180 5.5361 7.1910
s 0.04149 0.1177 0.1822 0.2688 0.5002
```

[23]:
```
plot(chain2)
```



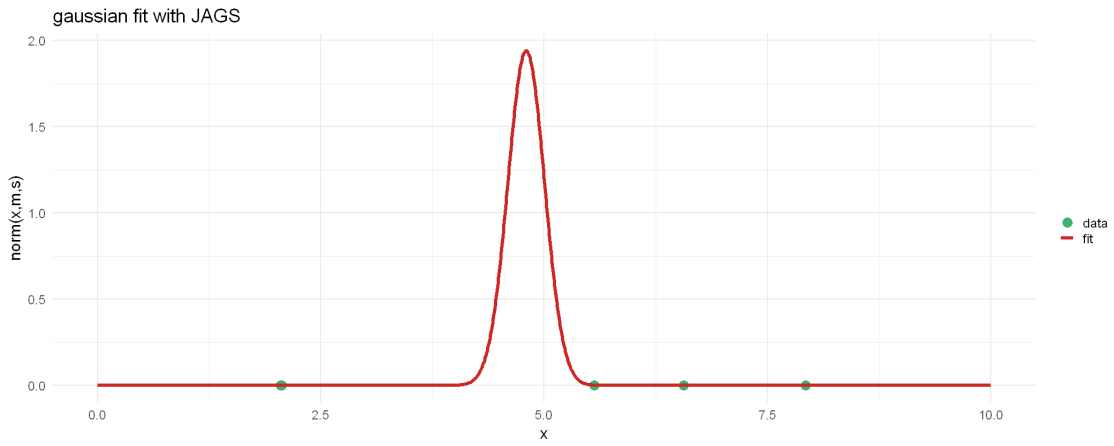
[24]:
```
m.mean <- summary(chain2)$statistics[1,1]
s.mean <- summary(chain2)$statistics[2,1]

print(sprintf('Candidate value for m is: %.2f', m.mean))
print(sprintf('Candidate value for s is: %.2f', s.mean))
cat('\n')

m.sd <- summary(chain2)$statistics[2,1]
s.sd <- summary(chain2)$statistics[2,2]

m.ci.lo <- summary(chain2)$quantiles[1,1]
```

```
m.ci.hi <- summary(chain2)$quantiles[1,5]
s.ci.lo <- summary(chain2)$quantiles[2,1]
s.ci.hi <- summary(chain2)$quantiles[2,5]

print(sprintf('m 0.95 CI is [%.2f, %.2f]', m.ci.lo, m.ci.hi))
print(sprintf('s 0.95 CI is [%.2f, %.2f]', s.ci.lo, s.ci.hi))
```

```
[1] "Candidate value for m is: 4.80"
[1] "Candidate value for s is: 0.21"

[1] "m 0.95 CI is [2.20, 7.19]"
[1] "s 0.95 CI is [0.04, 0.50]"
```

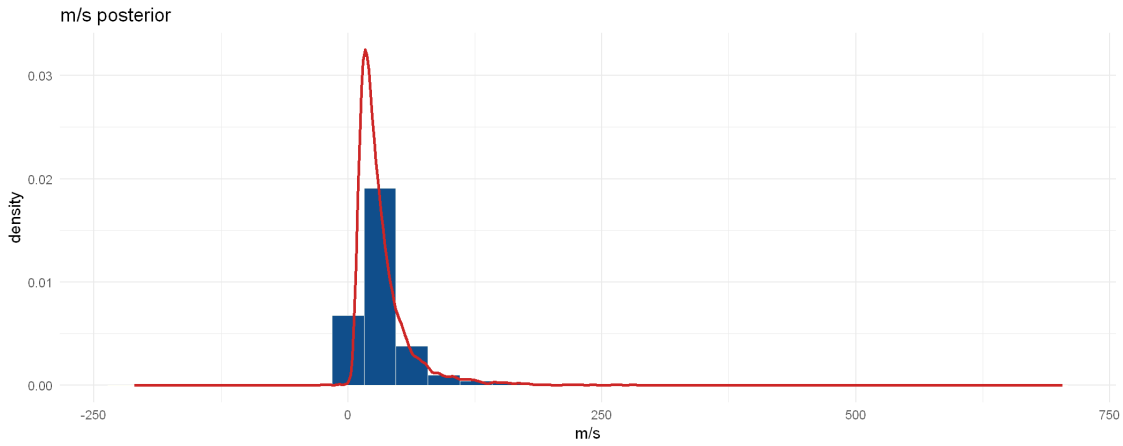
[25]:
```
y.data <- dnorm(data2$x, m.mean, s.mean)
num <- seq(0, 10, 0.01)
y.norm <- dnorm(num, m.mean, s.mean)
data2.fit <- data.frame(num = num,y.norm = y.norm)
```

[26]:
```
norm.plot <- ggplot() +
    geom_point(aes(x=data2$x, y=y.data, color='data'), size=5) +
    geom_line(aes(x=data2.fit$num, y=data2.fit$y.norm, color='fit'), lwd=1.7) +
    labs(title = 'gaussian fit with JAGS', x='x', y='norm(x,m,s)', colour='')+
    theme_minimal(base_size=18) +
    scale_color_manual(values = c('data' = 'mediumseagreen',␣
 ↪'fit'='firebrick3'))

norm.plot
```



[27]:
```
chain2.df <- as.data.frame(as.mcmc(chain2))

param.ratio <- chain2.df$m / chain2.df$s
```

```
ratio.post <- ggplot() +
    geom_histogram(aes(x=param.ratio, after_stat(density)), bins = 30, color =␣
 ↪"ivory", fill = "dodgerblue4") +
    geom_density(aes(x=param.ratio), color = "firebrick3", lwd = 1.5) +
    labs(title='m/s posterior', x='m/s') +
    theme_minimal(base_size = 18)

ratio.post
```



4)

[28]:
```
D <- c(0.0032, 0.0034, 0.214, 0.263, 0.275, 0.275, 0.45, 0.5, 0.5, 0.63, 0.8, 0.
 ↪9, 0.9, 0.9, 0.9, 2, 2, 2, 2)
V <- c(170, 290, -130, -70, -185, -220, 200, 290, 270, 200, 920, 450, 500, 500,␣
 ↪960, 500, 850, 800, 1090)

data3 <- data.frame('D' = D, 'V' = V)

# Define the model in JAGS syntax
model3 <- 'jm_ex5.4.bug'

jm3 <- jags.model(model3, data3)
update(jm3, 1000) #burnin

chain3 <- coda.samples(jm3, c("b", "c"), n.iter=100000, thin=50)

# Summarize the results
print(summary(chain3))
```

```
Compiling model graph
   Resolving undeclared variables
```

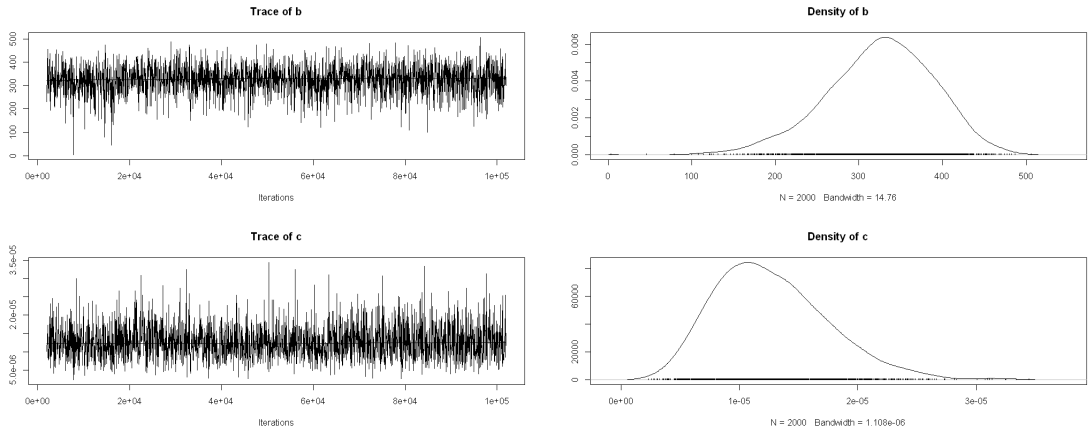
```
   Allocating nodes
Graph information:
   Observed stochastic nodes: 19
   Unobserved stochastic nodes: 2
   Total graph size: 54

Initializing model


Iterations = 2050:102000
Thinning interval = 50
Number of chains = 1
Sample size per chain = 2000

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

       Mean        SD  Naive SE Time-series SE
b 3.253e+02 6.468e+01 1.446e+00      1.446e+00
c 1.272e-05 4.805e-06 1.074e-07      1.074e-07

2. Quantiles for each variable:

       2.5%       25%       50%       75%     97.5%
b 1.869e+02 2.852e+02 3.293e+02 3.705e+02 4.361e+02
c 5.120e-06 9.194e-06 1.214e-05 1.560e-05 2.355e-05
```

[29]:
```r
b.mean <- summary(chain3)$statistics[1,1]
c.mean <- summary(chain3)$statistics[2,1]

print(sprintf('Candidate value for b is: %s', b.mean))
print(sprintf('Candidate value for c is: %s', c.mean))
cat('\n')

b.sd <- summary(chain3)$statistics[2,1]
c.sd <- summary(chain3)$statistics[2,2]

b.ci.lo <- summary(chain3)$quantiles[1,1]
b.ci.hi <- summary(chain3)$quantiles[1,5]
c.ci.lo <- summary(chain3)$quantiles[2,1]
c.ci.hi <- summary(chain3)$quantiles[2,5]

print(sprintf('m 0.95 CI is [%s, %s]', b.ci.lo, b.ci.hi))
print(sprintf('s 0.95 CI is [%s, %s]', c.ci.lo, c.ci.hi))
```

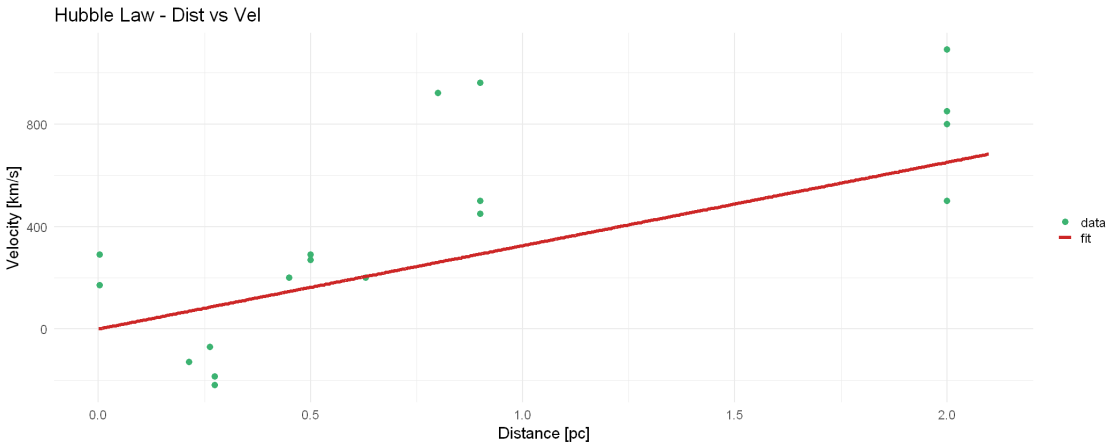
```
[1] "Candidate value for b is: 325.346146247471"
[1] "Candidate value for c is: 1.27240934268202e-05"
```

```
[1] "m 0.95 CI is [186.870470369208, 436.061497879245]"
[1] "s 0.95 CI is [5.11968421317999e-06, 2.35538636702016e-05]"
```

[30]:
```
plot(chain3)
```



[31]:
```
# Calculate predicted velocity based on model estimates
chain3.df <- as.data.frame(as.mcmc(chain3))

dist.cand <- sort(runif(1000, 0, 2.1))
predicted.velocity <- b.mean * dist.cand + c.mean

# Create the ggplot object
hlaw.plot <- ggplot() +
    geom_point(aes(x = data3$D, y =data3$V, color='data'), size=3) +  # Scatter␣
↪plot for data points
    geom_line(aes(x = dist.cand, y = predicted.velocity, color = 'fit'), lwd=1.
↪7) +  # Regression line
    labs(title = "Hubble Law - Dist vs Vel", x = "Distance [pc]", y = "Velocity␣
↪[km/s]", color = '') +
    theme_minimal(base_size = 18) +
    scale_color_manual(values = c('data' = 'mediumseagreen', 'fit' =␣
↪'firebrick3'))

hlaw.plot
```


Hubble Law - Dist vs Vel
Velocity [km/s]
800
400
0
0.0
0.5
1.0
1.5
2.0
Distance [pc]
data
fit

[ ]: