Bayesian Assignment 4

Shubhang Periwal 19201104

4/13/2020

```
library(rjags)
```

```
## Warning: package 'rjags' was built under R version 3.6.3
```

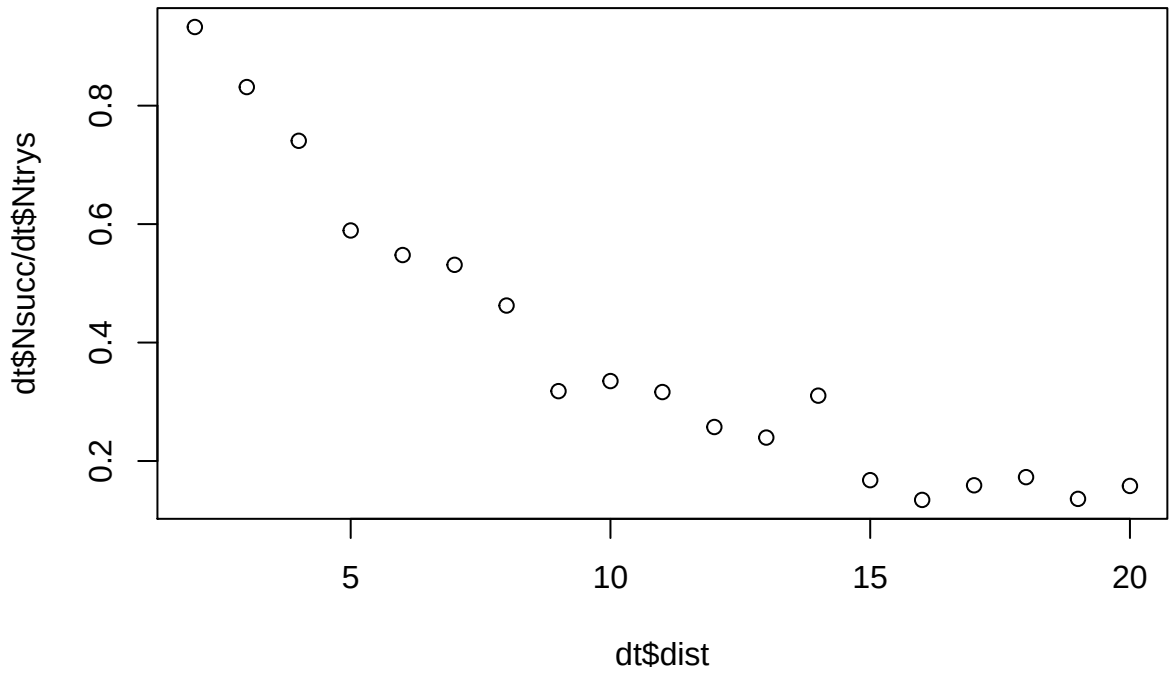
```
## Loading required package: coda
```

```
## Warning: package 'coda' was built under R version 3.6.3
```

```
## Linked to JAGS 4.3.0
```

```
## Loaded modules: basemod,bugs
```

```
dt = read.table("putting.dat", header = TRUE) # reading the data
# a
#success/tries
plot(dt$dist, dt$Nsucc/dt$Ntrys)
```



```r
# b
modinput = list(nr = nrow(dt), y1 = dt$Nsucc, y2 = dt$Ntrys,y3 = 100, x = dt$dist)
jmodel = jags.model(file = "model1.model", data = modinput)#reading model 1
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 19
##    Unobserved stochastic nodes: 21
##    Total graph size: 139
##
## Initializing model
```

```r
smp = coda.samples(jmodel, c("a", "b"), n.iter = 10000)#passing alpha, beta and values
summary(smp)#summary of produced samples
```

```
##
## Iterations = 1001:11000
## 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
## a  2.2341 0.058957 5.896e-04      0.0017831
## b -0.2561 0.006702 6.702e-05      0.0002034
##
## 2. Quantiles for each variable:
##
##      2.5%     25%    50%     75%   97.5%
## a  2.1200  2.1936  2.234  2.2728  2.3506
## b -0.2694 -0.2606 -0.256 -0.2516 -0.2434
```

```r
HPDinterval(smp[[1]], prob = 0.95)
```

```
##        lower      upper
## a  2.1173067  2.3466734
## b -0.2697305 -0.2436995
## attr(,"Probability")
## [1] 0.95
```

95% HDR intervals do you get for and

```
   lower      upper
```

a 2.1190256 2.3437664 b -0.2689385 -0.2432529

```r
# c
dt = rbind(dt, c(25, 100, NA))  #for
modinput = list(nr = nrow(dt), y1 = dt$Nsucc, y2 = dt$Ntrys,y3 = 100 ,x = dt$dist)
jmodel = jags.model(file = "model1.model", data = modinput)#passing data in model
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 19
##    Unobserved stochastic nodes: 23
##    Total graph size: 146
##
## Initializing model
```

```r
smp = coda.samples(jmodel, c("res"), n.iter = 10000)
summary(smp)
```

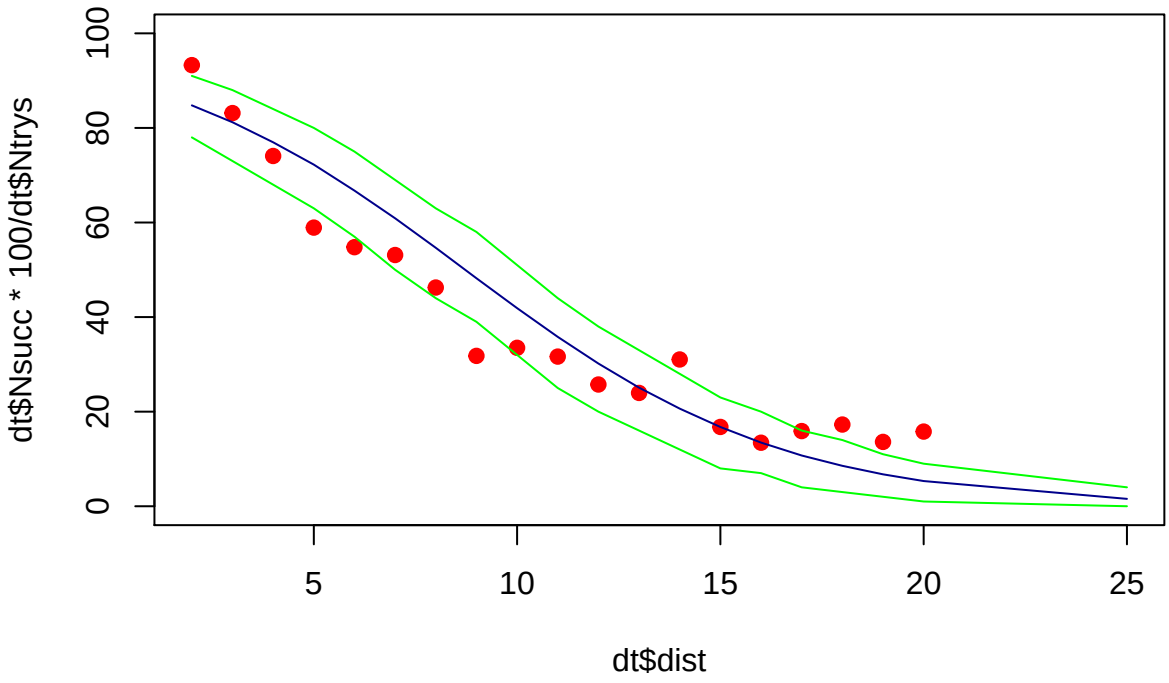
```
##
## Iterations = 1001:11000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean    SD Naive SE Time-series SE
## res[1]  84.770 3.588  0.03588        0.03674
## res[2]  81.203 3.970  0.03970        0.04200
## res[3]  76.962 4.263  0.04263        0.04263
## res[4]  72.238 4.565  0.04565        0.04639
## res[5]  66.740 4.786  0.04786        0.04786
## res[6]  60.891 4.906  0.04906        0.04906
## res[7]  54.650 4.998  0.04998        0.04866
## res[8]  48.211 5.044  0.05044        0.05044
## res[9]  41.890 5.021  0.05021        0.04920
## res[10] 35.816 4.918  0.04918        0.04968
## res[11] 30.174 4.678  0.04678        0.04678
## res[12] 25.074 4.464  0.04464        0.04464
## res[13] 20.641 4.179  0.04179        0.04179
## res[14] 16.760 3.825  0.03825        0.03895
## res[15] 13.460 3.495  0.03495        0.03777
## res[16] 10.727 3.168  0.03168        0.03515
## res[17]  8.544 2.879  0.02879        0.03092
## res[18]  6.749 2.589  0.02589        0.02770
## res[19]  5.326 2.275  0.02275        0.02420
## res[20]  1.558 1.248  0.01248        0.01248
##
## 2. Quantiles for each variable:
##
##         2.5% 25% 50% 75% 97.5%
```

```
## res[1]    78  82  85  87    91
## res[2]    73  79  81  84    89
## res[3]    68  74  77  80    85
## res[4]    63  69  72  75    81
## res[5]    57  64  67  70    76
## res[6]    51  58  61  64    70
## res[7]    45  51  55  58    64
## res[8]    38  45  48  52    58
## res[9]    32  38  42  45    52
## res[10]   26  33  36  39    46
## res[11]   21  27  30  33    40
## res[12]   17  22  25  28    34
## res[13]   13  18  21  23    29
## res[14]    9  14  17  19    25
## res[15]    7  11  13  16    21
## res[16]    5   8  11  13    17
## res[17]    3   7   8  10    15
## res[18]    2   5   7   8    12
## res[19]    1   4   5   7    10
## res[20]    0   1   1   2     4
```

```
HPDinterval(smp[[1]], prob = 0.95)
```

```
##         lower upper
## res[1]     78    91
## res[2]     73    88
## res[3]     68    84
## res[4]     63    80
## res[5]     57    75
## res[6]     50    69
## res[7]     44    63
## res[8]     39    58
## res[9]     32    51
## res[10]    25    44
## res[11]    20    38
## res[12]    16    33
## res[13]    12    28
## res[14]     8    23
## res[15]     7    20
## res[16]     4    16
## res[17]     3    14
## res[18]     2    11
## res[19]     1     9
## res[20]     0     4
## attr(,"Probability")
## [1] 0.95
```

```
plot(dt$dist, dt$Nsucc * 100/dt$Ntrys, ylim = c(0, 100), col = "red",pch = 19)
lines(dt$dist, HPDinterval(smp[[1]], prob = 0.95)[, 1], col = "green")
lines(dt$dist, HPDinterval(smp[[1]], prob = 0.95)[, 2], col = "green")
lines(dt$dist, summary(smp)$statistics[, 1], col = "darkblue")
```




```
# d
#95 percent HDR interval
HPDinterval(smp[[1]], prob = 0.95)["res[20]", ]
```

```
## lower upper
##     0     4
```

The interval means that the probability makes sense, but many points are outside the 95% interval, even though the interval values are big. So, we should try other models to fit this data.

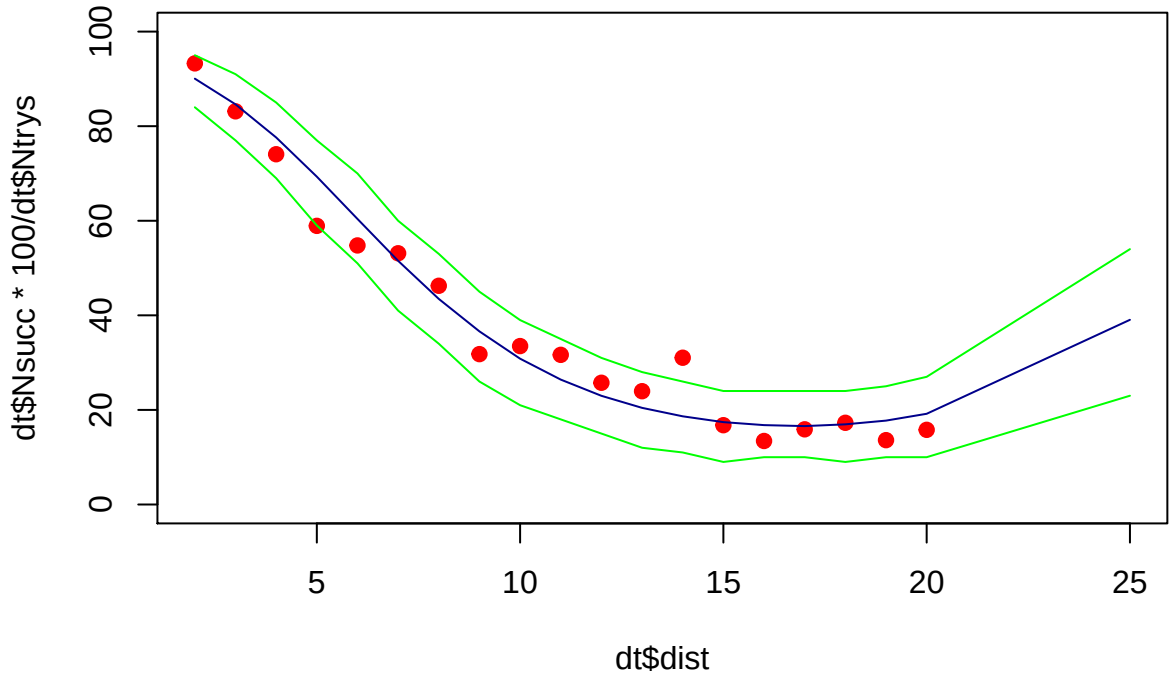
```
# e

modinput = list(nr = nrow(dt), y1 = dt$Nsucc, y2 = dt$Ntrys,y3 = 100, x = dt$dist)
jmodel = jags.model(file = "model2.model", data = modinput)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 19
##    Unobserved stochastic nodes: 24
##    Total graph size: 167
##
## Initializing model
```

```
smp = coda.samples(jmodel, c("res"), n.iter = 10000)

plot(dt$dist, dt$Nsucc * 100/dt$Ntrys, ylim = c(0, 100), col = "red",pch = 19)
lines(dt$dist, HPDinterval(smp[[1]], prob = 0.95)[, 1], col = "green")
lines(dt$dist, HPDinterval(smp[[1]], prob = 0.95)[, 2], col = "green")
lines(dt$dist, summary(smp)$statistics[, 1], col = "darkblue")
```



This model seems a better fit than the previous model more points lie close to the curve, which means lower error rates. Bu the distance increase after prior reduces.

```
#f
#95% interval
HPDinterval(smp[[1]], prob = 0.95)["res[20]", ] #95% interval
```

```
## lower upper
##    23    54
```

```
#g
modinput = list(nr = nrow(dt), y1 = dt$Nsucc, y2 = dt$Ntrys,y3 = 100,x = dt$dist )
jmodel = jags.model(file = "model3.model", data = modinput)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
```

```
##    Observed stochastic nodes: 19
##    Unobserved stochastic nodes: 23
##    Total graph size: 166
##
## Initializing model
```

```
smp = coda.samples(jmodel, c("res"), n.iter = 10000)
summary(smp)
```

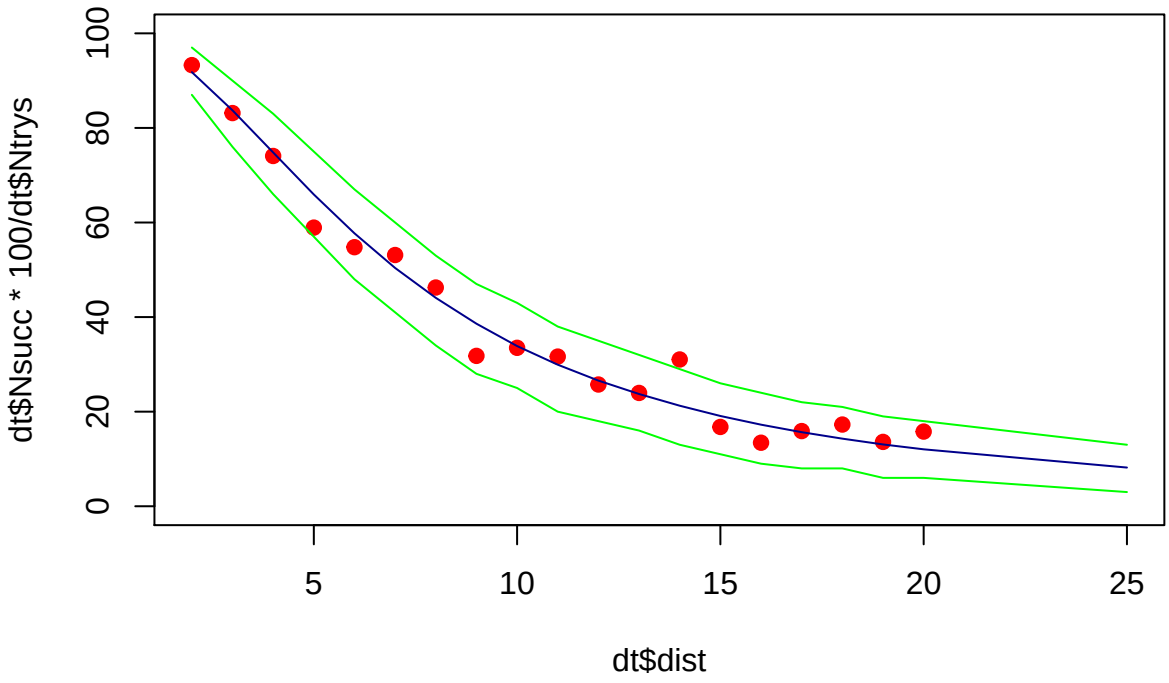
```
##
## Iterations = 1001:11000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean    SD Naive SE Time-series SE
## res[1]  91.80 2.810  0.02810        0.03361
## res[2]  83.71 3.726  0.03726        0.04510
## res[3]  74.81 4.457  0.04457        0.04709
## res[4]  65.92 4.808  0.04808        0.04808
## res[5]  57.71 4.994  0.04994        0.04994
## res[6]  50.39 5.079  0.05079        0.05079
## res[7]  44.07 5.025  0.05025        0.05025
## res[8]  38.60 4.936  0.04936        0.05015
## res[9]  33.87 4.828  0.04828        0.04903
## res[10] 29.93 4.637  0.04637        0.04637
## res[11] 26.57 4.537  0.04537        0.04537
## res[12] 23.73 4.294  0.04294        0.04407
## res[13] 21.26 4.194  0.04194        0.04473
## res[14] 19.09 3.995  0.03995        0.05480
## res[15] 17.25 3.878  0.03878        0.04456
## res[16] 15.67 3.709  0.03709        0.04168
## res[17] 14.29 3.578  0.03578        0.04154
## res[18] 13.07 3.390  0.03390        0.04077
## res[19] 12.04 3.313  0.03313        0.03708
## res[20]  8.18 2.777  0.02777        0.03266
##
## 2. Quantiles for each variable:
##
##         2.5% 25% 50% 75% 97.5%
## res[1]    86  90  92  94    97
## res[2]    76  81  84  86    91
## res[3]    66  72  75  78    83
## res[4]    56  63  66  69    75
## res[5]    48  54  58  61    67
## res[6]    40  47  50  54    60
## res[7]    34  41  44  47    54
## res[8]    29  35  39  42    48
## res[9]    25  31  34  37    44
## res[10]   21  27  30  33    39
## res[11]   18  23  26  30    36
```

```
## res[12]   15  21  24  27    32
## res[13]   13  18  21  24    30
## res[14]   12  16  19  22    27
## res[15]   10  15  17  20    25
## res[16]    9  13  16  18    23
## res[17]    8  12  14  17    21
## res[18]    7  11  13  15    20
## res[19]    6  10  12  14    19
## res[20]    3   6   8  10    14
```

```
HPDinterval(smp[[1]], prob = 0.95)
```

```
##         lower upper
## res[1]     87    97
## res[2]     76    90
## res[3]     66    83
## res[4]     57    75
## res[5]     48    67
## res[6]     41    60
## res[7]     34    53
## res[8]     28    47
## res[9]     25    43
## res[10]    20    38
## res[11]    18    35
## res[12]    16    32
## res[13]    13    29
## res[14]    11    26
## res[15]     9    24
## res[16]     8    22
## res[17]     8    21
## res[18]     6    19
## res[19]     6    18
## res[20]     3    13
## attr(,"Probability")
## [1] 0.95
```

```
plot(dt$dist, dt$Nsucc * 100/dt$Ntrys, ylim = c(0, 100), col = "red",pch = 19)
lines(dt$dist, HPDinterval(smp[[1]], prob = 0.95)[, 1], col = "green")
lines(dt$dist, HPDinterval(smp[[1]], prob = 0.95)[, 2], col = "green")
lines(dt$dist, summary(smp)$statistics[, 1], col = "darkblue")
```




```
#95% interval
HPDinterval(smp[[1]], prob = 0.95)["res[20]", ] #95% interval
```

```
## lower upper
##     3    13
```

This model (loagrithmic) is a better fit than the previous 2 models based on the output curve. The points are much closer and better explained by the model. With most of them within the 95 % interval.

```
#h
in1 = list(a = 0.8, b = 1.3, .RNG.name = "base::Mersenne-Twister",.RNG.seed = 19201104)
in2 = list(a = 1.5, b = 0.6, .RNG.name = "base::Mersenne-Twister",.RNG.seed = 19201104)
#creating init with initialized alpha and beta
jmodel1 = jags.model(file = "model1.model", data = modinput, n.chains = 2,inits = list(in1, in2))
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 19
##    Unobserved stochastic nodes: 23
##    Total graph size: 146
##
## Initializing model
```

```
lin = dic.samples(jmodel1, 10000)
jmodel2 = jags.model(file ="model2.model", data = modinput, n.chains = 2,inits = list(in1, in2))
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 19
##    Unobserved stochastic nodes: 24
##    Total graph size: 167
##
## Initializing model
```

```
quad = dic.samples(jmodel2, 10000)
jmodel3 = jags.model(file = "model3.model", data = modinput, n.chains = 2,inits = list(in1, in2))
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 19
##    Unobserved stochastic nodes: 23
##    Total graph size: 166
##
## Initializing model
```

```
logarithm = dic.samples(jmodel3, 10000)
print(lin)
```

```
## Mean deviance:  363.9
## penalty 1.963
## Penalized deviance: 365.9
```

```
print(quad)
```

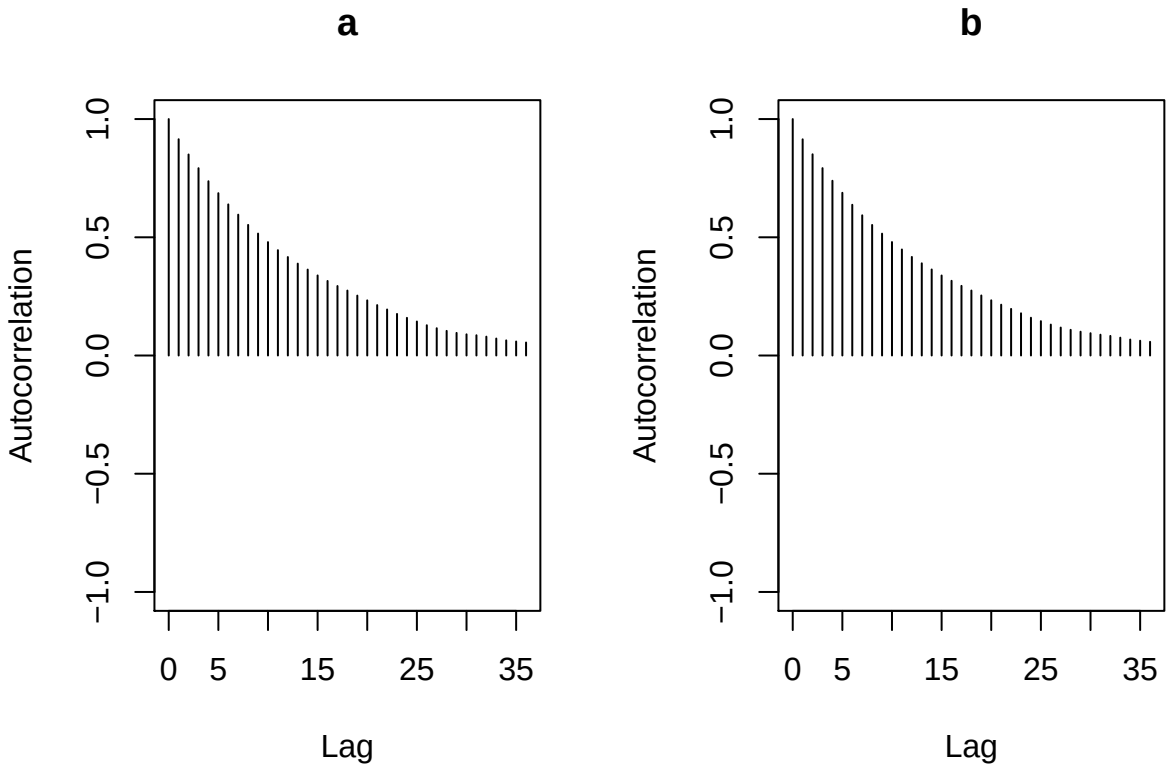
```
## Mean deviance:  182.7
## penalty 2.878
## Penalized deviance: 185.6
```

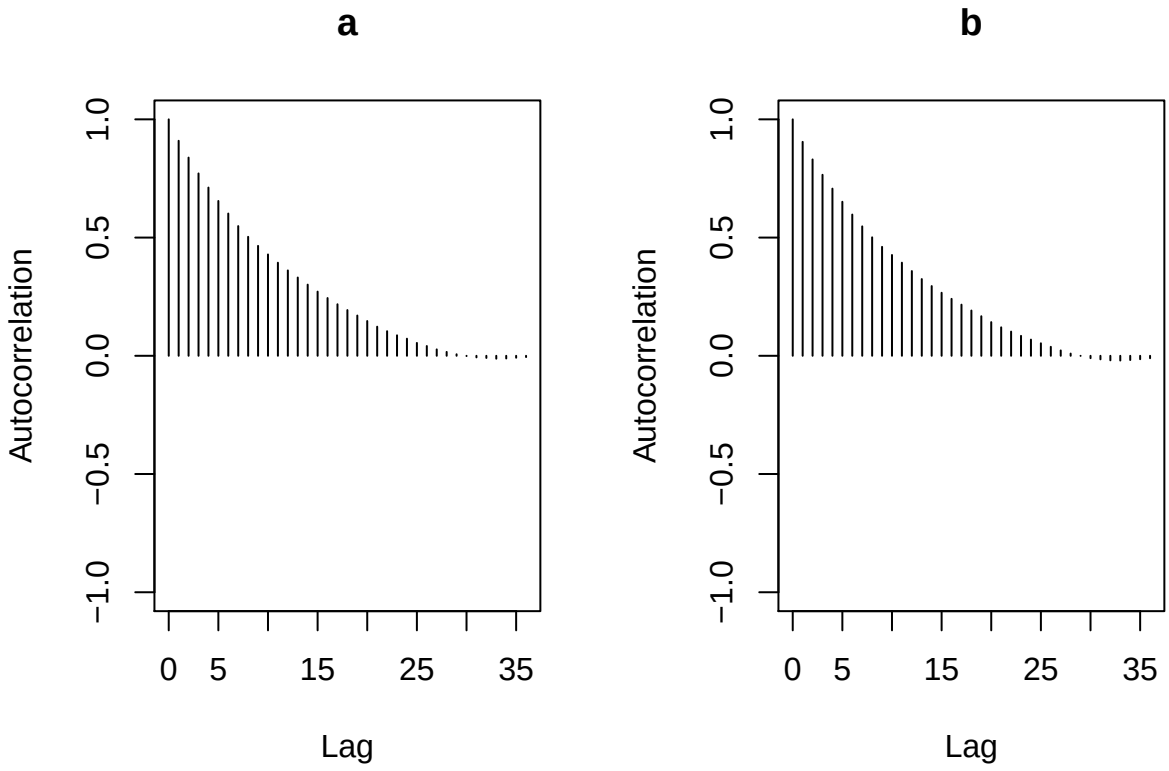
```
print(logarithm)
```

```
## Mean deviance:  143
## penalty 1.996
## Penalized deviance: 145
```

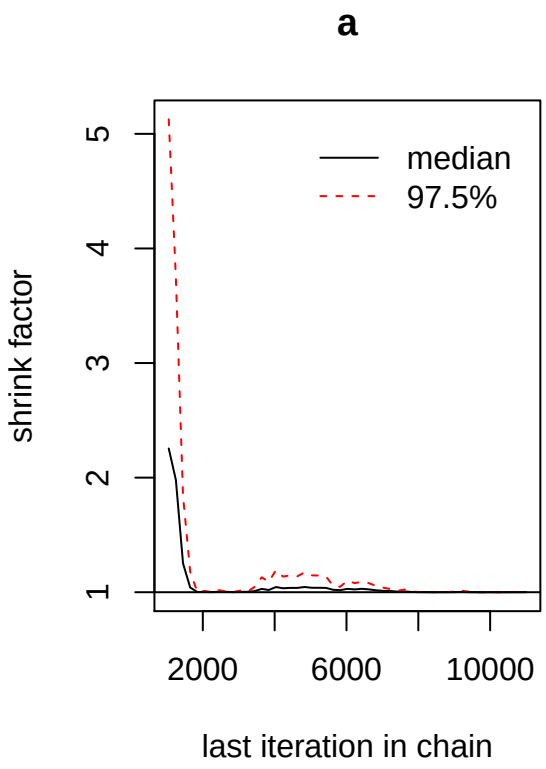
DIC (deviance information criterion) helps us to choose a model, given multiple models. The lower value accounts for a better fit, with difference of more than 10 meaning a significant amount of difference to choose one model over the other. The first 2 models have higher deviance, making the linear model, as the worst and the logarithmic model as the best model. With reducing mean deviance and penalized deviance.

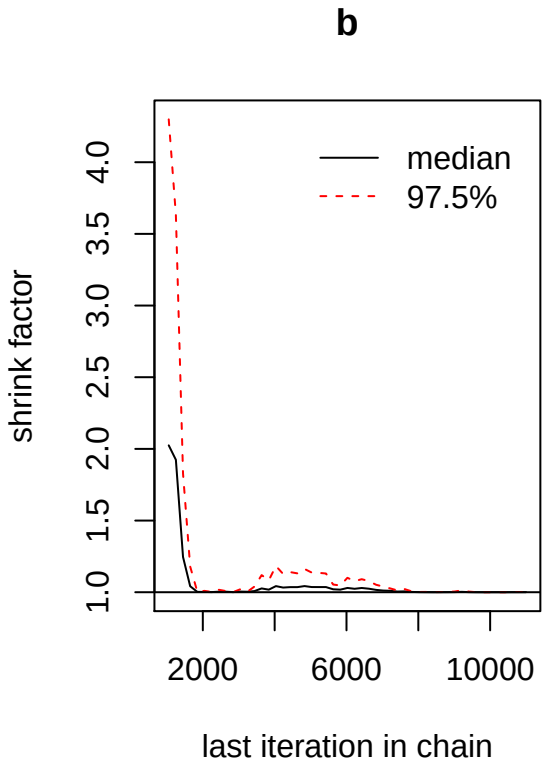
```r
# i
#creating inits usig multiple random number generators
in1 = list(.RNG.name = "base::Wichmann-Hill", .RNG.seed = 19201104)
in2 = list(.RNG.name = "base::Marsaglia-Multicarry", .RNG.seed = 19201104)
jmodel = jags.model(file = "model3.model", data = modinput, quiet = TRUE,n.chains = 2, inits = list(in1
#passing inits so get non varying outputs
smp = coda.samples(jmodel, c("a", "b"), n.iter = 10000)
autocorr.plot(smp)
```






```
gelman.plot(smp, ask = TRUE)
```






```
effectiveSize(smp)
```

```
##        a        b
## 776.4089 791.3438
```

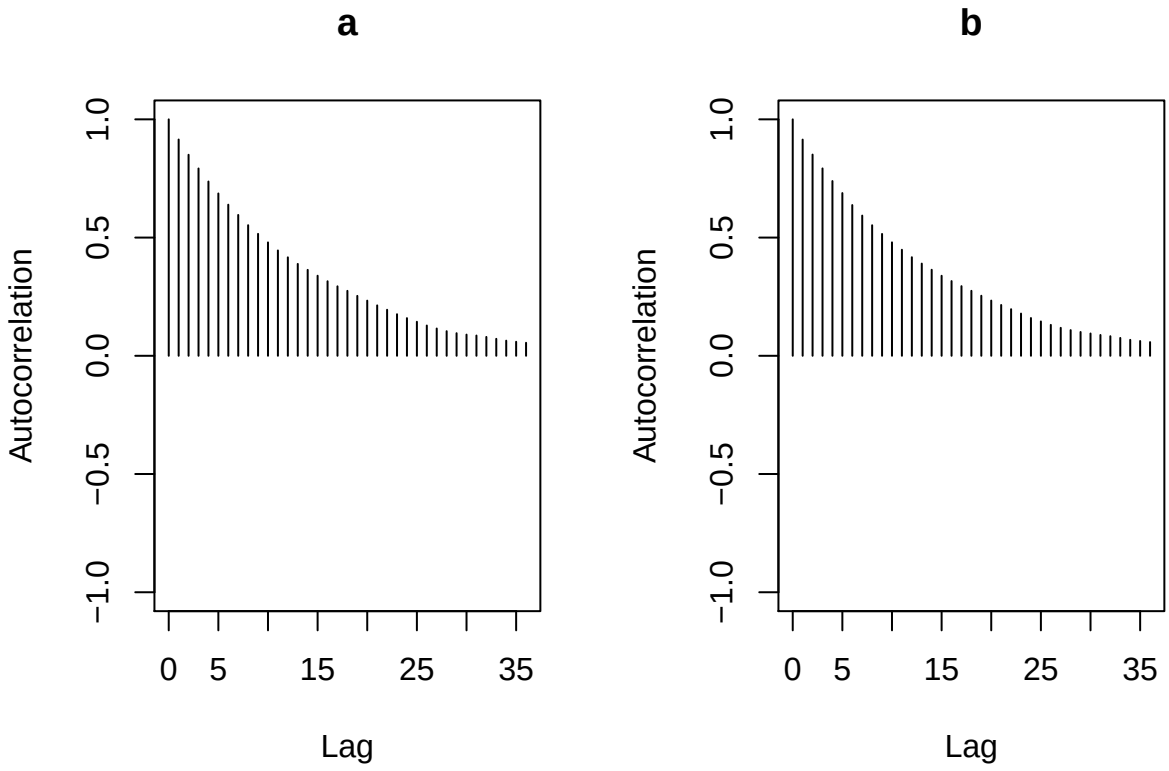
```
gelman.diag(smp, multivariate = FALSE)
```

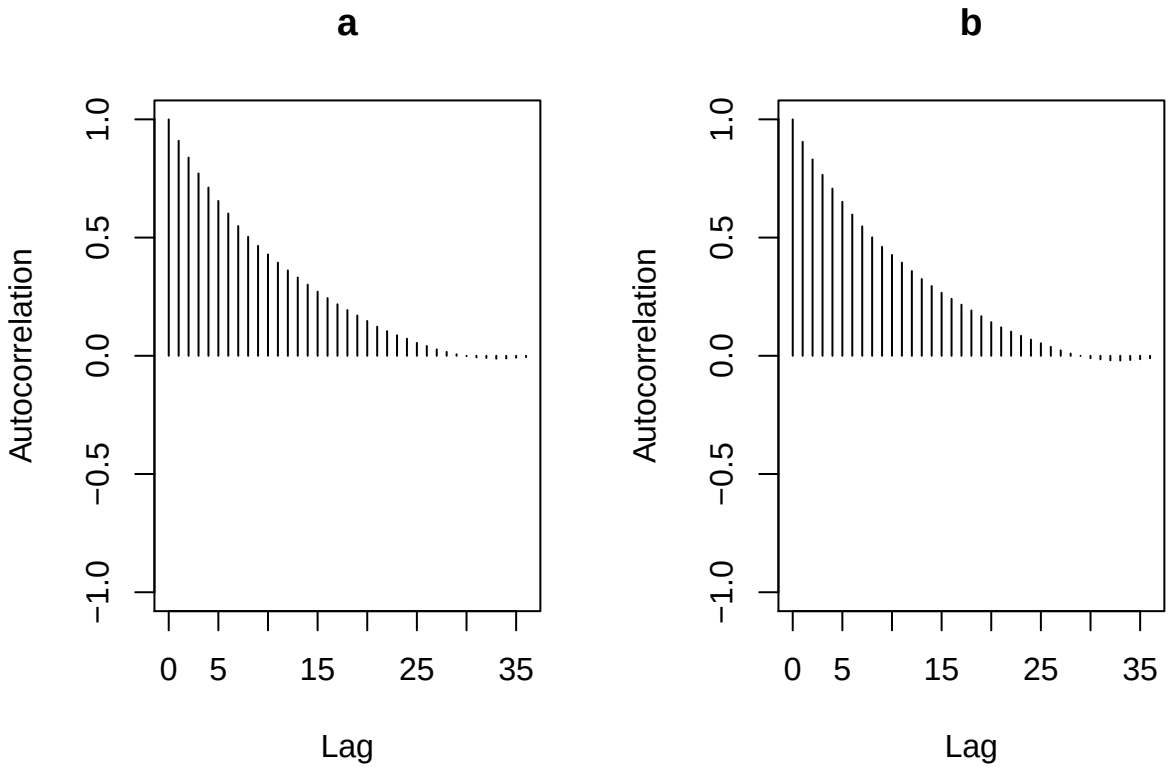
```
## Potential scale reduction factors:
##
##   Point est. Upper C.I.
## a          1          1
## b          1          1
```

```
dim(smp[[1]])
```

```
## [1] 10000     2
```

```
jmodel = jags.model(file = "model3.model", data = modinput, quiet = TRUE,n.chains = 2, inits = list(in1
csamples = coda.samples(jmodel, c("a", "b","res"), n.iter = 10000,thin = 25  )
autocorr.plot(smp)# plotting autocorrelations
```

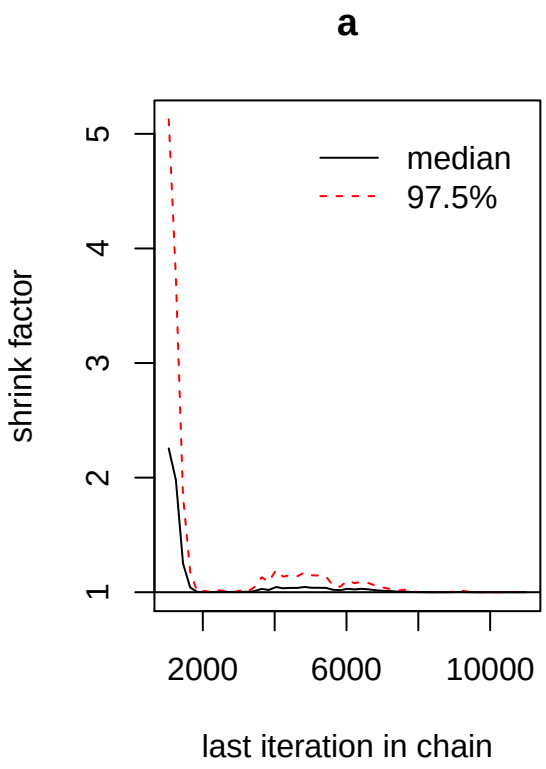

a
b
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
5
15
25
35
Lag

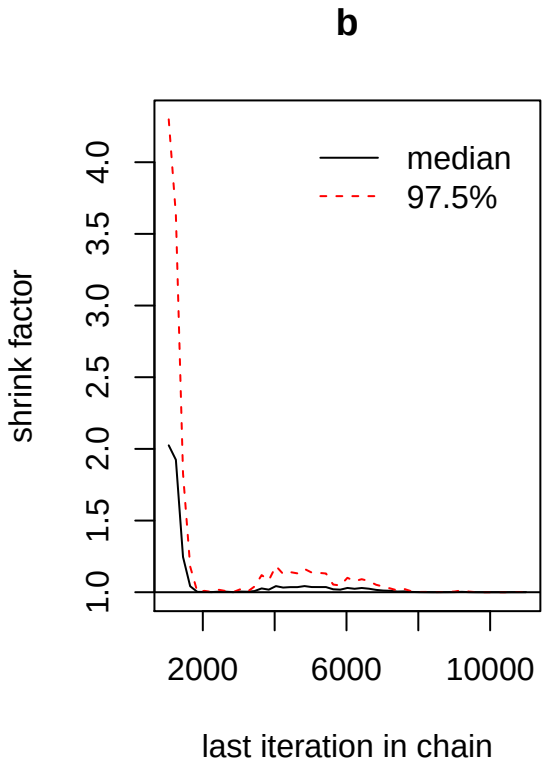



```
gelman.diag(smp) #using to check conversions
```

```
## Potential scale reduction factors:
##
##   Point est. Upper C.I.
## a          1          1
## b          1          1
##
## Multivariate psrf
##
## 1
```

```
gelman.plot(smp)
```






```
effectiveSize(smp)
```

```
##        a        b
## 776.4089 791.3438
```

```
dim(csamples[[1]])
```

```
## [1] 400  22
```

I executed 2 chains for check for convergence, we can clearly see that there is a high amount of autocorrelation atleast till the 30th lag. Even in the gelman plot we can see that there should be convergence which supports the autocorrelation plot. Effective values of alpha and beta are as follows: a b 776.4089 791.3438 , we can see that there is no mixing and no convergence after thinning and adapting