> ########## ########## MCMC Comparison of the different samplers and their packages ########## ##########

> ########## using the Blocker example with the Random Effects Model

>

>

>

> ########## testing running time

> #### to test the running time, the Samplers are running separately and the package "tictoc" are used

>

>

> ########## testing other diagnostica

> #### because of development of compareMCMCs, similar functions from other packages e.g. coda-package are used

>

>

>

>

> #### Clear data

> rm(list=ls())

>

>

>

> #### Setting working directory

> setwd("C:/Users/IvanB/Desktop/Masterarbeit/Ergebnisse/zusätzliche Experimente/MCMC Vergleiche")

>

>

>

> ### loading libraries

> library(R2WinBUGS)

> library(rjags)

Linked to JAGS 4.3.0

Loaded modules: basemod,bugs,glm,lecuyer,dic

> library(jagsUI)

Lade nötiges Paket: lattice

Attache Paket: ‘lattice’

The following object is masked from ‘package:boot’:

melanoma

Attache Paket: ‘jagsUI’

The following object is masked from ‘package:coda’:

traceplot

The following object is masked from ‘package:utils’:

View

> library(R2jags)

Attache Paket: ‘R2jags’

The following objects are masked from ‘package:jagsUI’:

autojags, jags, traceplot

The following object is masked from ‘package:coda’:

traceplot

> library(runjags)

> library(nimble)

nimble version 0.8.0 is loaded.

For more information on NIMBLE and a User Manual,

please visit http://R-nimble.org.

Attache Paket: ‘nimble’

The following object is masked from ‘package:boot’:

logit

The following object is masked from ‘package:stats’:

simulate

> #library("rstan")

> #library("rstantools")

>

> library(coda)

> library(tictoc)

> library(lattice)

> library(random)

> library(matrixStats) # zus?tzl Paket, berechnet Median

> library(car)

Lade nötiges Paket: carData

Attache Paket: ‘car’

The following object is masked from ‘package:nimble’:

logit

The following object is masked from ‘package:boot’:

logit

> library(fitR)

> library(mcmcr)

> #library(BayesianTools)

>

> # load RNGs

> load.module("glm")

> load.module("lecuyer")

> #list.factories(type="rng")

> #parallel.seeds("lecuyer::RngStream", 5);

> load.module("dic")

>

>

>

> ##### Read the data into R.

> #data = read.table("Blocker\_Data\_neu sortiert.txt", sep = "", header=F)

> data = as.matrix(read.table("Blocker\_Data\_neu sortiert.txt", sep = "", header=F))

> head(data) # Shows the first six entries

V1 V2 V3 V4 V5 V6 V7

[1,] 3 3 39 38 1 2 2

[2,] 14 7 116 114 1 2 2

[3,] 11 5 93 69 1 2 2

[4,] 127 102 1520 1533 1 2 2

[5,] 27 28 365 355 1 2 2

[6,] 6 4 52 59 1 2 2

> data2 = read.table("Data\_Blocker\_Rest.txt")

> head(data2) # Shows the first six entries

V1 V2 V3

1 nt <- 2

2 ns <- 22

>

>

>

> ##### Values for simulation, prepare dat for JAGS (allocation values from data)

> ns <- nrow(data)

> # ns # check

> nt <- ncol(data[,5:6])

> #nt # check

> na <- data[,7]

> # na # check

> r <- data[,1:2]

> # r # Check

> n <- data[,3:4]

> # n # Check

> t <- data[,5:6]

> # t # Check

>

> dat <- list("ns", "nt", "na", "r", "n", "t") # names list of numbers

> dat2 <- list(ns=ns, nt=nt, na=na, r=r, n=n, t=t)

>

>

>

> ##### Parameter to monitor/save

> # analog https://nature.berkeley.edu/~pdevalpine/MCMC\_comparisons/some\_BUGS\_comparisons/blocker/nimble\_blocker\_comparisons.html,

> # all known parameters were monitored

> params <- c("mu", "d", "sd", "A" )

>

>

>

> ##### read in inits with chains

> inits1 <- list(d=c( NA, 0),

+ sd=1,

+ mu=c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),

+ .RNG.name="lecuyer::RngStream", .RNG.seed=1)

>

> inits2 <- list(d=c( NA, -1),

+ sd=4,

+ mu=c(-3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3),

+ .RNG.name="lecuyer::RngStream", .RNG.seed=2)

>

> inits3 <- list(d=c( NA, 2),

+ sd=2,

+ mu=c(-3, 5, -1, -3, 7, -3, -4, -3, -3, 0, -3, -3,0, 3, 5, -3, -3, -1, -3, -7, -3, -3),

+ .RNG.name="lecuyer::RngStream", .RNG.seed=3 )

>

> all.inits <- list(inits1, inits2, inits2)

>

>

>

> ##### model Code

> # the model code is named as "Blocker\_Model\_Random.txt"

>

>

>

>

>

> # Part 1: Testing running time of R code of WinBUGS ----------------------------------

>

>

>

>

> tic("Total WinBUGS")

> tic("Compiling and Simulation/Sampling")

> test\_WinBUGS <- bugs(data=dat, inits=all.inits, parameters=params, model.file="Blocker\_Model\_Random.bug",

+ n.chains=3, n.iter=20000, n.burnin =10000,

+ bugs.directory="C:/Users/IvanB/Desktop/Masterarbeit/Statistische Programme und Gibbs Sampler/WinBUGS/WinBUGS - Programm und Skript/winbugs143\_unrestricted/winbugs14\_full\_patched/WinBUGS14/"

+ )

> toc()

Compiling and Simulation/Sampling: 24.4 sec elapsed

> tic("generating summary")

> print(test\_WinBUGS)

Inference for Bugs model at "Blocker\_Model\_Random.bug", fit using WinBUGS,

3 chains, each with 20000 iterations (first 10000 discarded), n.thin = 30

n.sims = 1002 iterations saved

mean sd 2.5% 25% 50% 75% 97.5% Rhat n.eff

mu[1] -2.4 0.4 -3.4 -2.7 -2.4 -2.1 -1.6 1 1000

mu[2] -2.2 0.3 -2.7 -2.4 -2.2 -2.0 -1.7 1 720

mu[3] -2.1 0.3 -2.7 -2.3 -2.1 -1.9 -1.6 1 1000

mu[4] -2.4 0.1 -2.6 -2.4 -2.4 -2.3 -2.2 1 1000

mu[5] -2.4 0.2 -2.7 -2.5 -2.4 -2.3 -2.1 1 210

mu[6] -2.2 0.3 -2.9 -2.5 -2.2 -2.0 -1.6 1 1000

mu[7] -1.7 0.1 -1.9 -1.8 -1.7 -1.6 -1.5 1 1000

mu[8] -2.1 0.1 -2.4 -2.2 -2.1 -2.0 -1.9 1 1000

mu[9] -2.0 0.2 -2.2 -2.1 -2.0 -1.8 -1.7 1 1000

mu[10] -2.2 0.1 -2.4 -2.3 -2.2 -2.2 -2.1 1 860

mu[11] -2.3 0.1 -2.6 -2.4 -2.3 -2.2 -2.1 1 1000

mu[12] -1.5 0.1 -1.7 -1.5 -1.5 -1.4 -1.2 1 380

mu[13] -3.0 0.2 -3.4 -3.1 -3.0 -2.8 -2.5 1 1000

mu[14] -2.7 0.1 -3.0 -2.8 -2.7 -2.6 -2.5 1 1000

mu[15] -1.4 0.2 -1.7 -1.5 -1.4 -1.2 -1.0 1 750

mu[16] -1.5 0.1 -1.8 -1.6 -1.5 -1.4 -1.2 1 1000

mu[17] -2.0 0.2 -2.4 -2.1 -2.0 -1.9 -1.6 1 310

mu[18] -3.0 0.3 -3.5 -3.2 -3.0 -2.8 -2.5 1 610

mu[19] -3.5 0.4 -4.3 -3.7 -3.4 -3.2 -2.8 1 1000

mu[20] -1.5 0.1 -1.8 -1.6 -1.5 -1.4 -1.2 1 740

mu[21] -2.1 0.1 -2.4 -2.2 -2.1 -2.0 -1.9 1 760

mu[22] -2.9 0.1 -3.2 -3.0 -2.9 -2.8 -2.6 1 1000

d[2] -0.3 0.1 -0.4 -0.3 -0.3 -0.2 -0.1 1 290

sd 0.1 0.1 0.0 0.1 0.1 0.2 0.3 1 110

A -2.2 0.5 -3.3 -2.6 -2.2 -1.8 -1.1 1 1000

deviance 259.4 8.2 245.0 253.8 259.2 264.4 276.4 1 570

For each parameter, n.eff is a crude measure of effective sample size,

and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = Dbar-Dhat)

pD = 28.6 and DIC = 288.0

DIC is an estimate of expected predictive error (lower deviance is better).

> plot(test\_WinBUGS)

> toc()

generating summary: 0.3 sec elapsed

> toc()

Total WinBUGS: 24.72 sec elapsed

>

>

>

>

>

> # Part 2: Testing running time of R code of JAGS ----------------------------------

>

>

>

>

> ##### ##### jagsUI package ##### #####

>

>

> ##### Inits for jagsUI

> inits1 <- list(d=c( NA, 0),

+ sd=1,

+ mu=c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),

+ A=0,

+ .RNG.name="base::Wichmann-Hill", .RNG.seed=1)

>

> inits2 <- list(d=c( NA, -1),

+ sd=4,

+ mu=c(-3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3),

+ A=1,

+ .RNG.name="base::Wichmann-Hill", .RNG.seed=2)

>

> inits3 <- list(d=c( NA, 2),

+ sd=2,

+ mu=c(-3, 5, -1, -3, 7, -3, -4, -3, -3, 0, -3, -3,0, 3, 5, -3, -3, -1, -3, -7, -3, -3),

+ A=2,

+ .RNG.name="base::Wichmann-Hill", .RNG.seed=3 )

>

> # Note: "lecuyer::RngStream" does not work with jagsUI

>

> all.inits.jagsUI <- list(inits1, inits2, inits2)

>

>

>

> tic("Total jagsUI")

> tic("Compiling and Simulation/Sampling")

> jags.m.jagsUI <- jags(data=dat,

+ inits=all.inits,

+ parameters.to.save=params,

+ model.file="Blocker\_Model\_Random.txt",

+ n.chains=3,

+ n.iter=20000, n.burnin=10000) # store.data=TRUE

Compiling model graph

Resolving undeclared variables

Allocating nodes

Graph information:

Observed stochastic nodes: 44

Unobserved stochastic nodes: 47

Total graph size: 964

Initializing model

|++++++++++++++++++++++++++++++++++++++++++++++++++| 100%

|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| 100%

> toc()

Compiling and Simulation/Sampling: 10.81 sec elapsed

> tic("generating summary")

> jags.View(jags.m.jagsUI)

Error in if (x$mcmc.info$n.chains != 1) { : argument is of length zero

> jags.m.jagsUI

Inference for Bugs model at "Blocker\_Model\_Random.txt", fit using jags,

3 chains, each with 20000 iterations (first 10000 discarded), n.thin = 10

n.sims = 3000 iterations saved

mu.vect sd.vect 2.5% 25% 50% 75% 97.5% Rhat n.eff

A -2.203 0.559 -3.285 -2.603 -2.201 -1.809 -1.122 1.001 3000

d[1] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 1.000 1

d[2] -0.248 0.065 -0.377 -0.291 -0.247 -0.207 -0.114 1.003 700

mu[1] -2.458 0.451 -3.384 -2.749 -2.438 -2.138 -1.630 1.001 2800

mu[2] -2.191 0.238 -2.666 -2.349 -2.183 -2.027 -1.742 1.001 3000

mu[3] -2.129 0.280 -2.718 -2.317 -2.119 -1.933 -1.614 1.001 2400

mu[4] -2.396 0.081 -2.558 -2.449 -2.396 -2.342 -2.237 1.001 3000

mu[5] -2.412 0.158 -2.737 -2.513 -2.402 -2.303 -2.122 1.001 3000

mu[6] -2.240 0.345 -2.959 -2.467 -2.225 -1.997 -1.608 1.003 970

mu[7] -1.709 0.085 -1.879 -1.766 -1.708 -1.650 -1.549 1.001 3000

mu[8] -2.117 0.119 -2.359 -2.195 -2.117 -2.036 -1.887 1.001 3000

mu[9] -1.960 0.149 -2.249 -2.057 -1.958 -1.860 -1.672 1.002 1100

mu[10] -2.242 0.070 -2.382 -2.289 -2.242 -2.196 -2.102 1.001 2600

mu[11] -2.318 0.117 -2.557 -2.394 -2.314 -2.236 -2.099 1.002 1800

mu[12] -1.474 0.133 -1.736 -1.563 -1.472 -1.386 -1.212 1.001 3000

mu[13] -2.994 0.214 -3.424 -3.139 -2.994 -2.844 -2.577 1.001 3000

mu[14] -2.737 0.129 -3.007 -2.820 -2.734 -2.648 -2.499 1.002 1700

mu[15] -1.356 0.162 -1.680 -1.466 -1.354 -1.244 -1.049 1.001 3000

mu[16] -1.494 0.144 -1.778 -1.589 -1.495 -1.395 -1.215 1.001 2100

mu[17] -2.007 0.195 -2.405 -2.135 -2.002 -1.876 -1.638 1.002 1200

mu[18] -2.965 0.285 -3.547 -3.143 -2.950 -2.768 -2.432 1.003 970

mu[19] -3.443 0.363 -4.206 -3.678 -3.420 -3.182 -2.791 1.002 1900

mu[20] -1.492 0.144 -1.778 -1.586 -1.490 -1.393 -1.215 1.001 3000

mu[21] -2.128 0.141 -2.403 -2.225 -2.129 -2.032 -1.848 1.001 3000

mu[22] -2.918 0.142 -3.204 -3.012 -2.916 -2.823 -2.643 1.003 960

sd 0.139 0.079 0.018 0.079 0.132 0.184 0.317 1.014 410

deviance 259.090 8.053 244.807 253.410 258.778 264.222 276.539 1.001 3000

For each parameter, n.eff is a crude measure of effective sample size,

and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)

pD = 32.4 and DIC = 291.5

DIC is an estimate of expected predictive error (lower deviance is better).

> toc()

generating summary: 0.03 sec elapsed

> toc()

Total jagsUI: 10.84 sec elapsed

>

>

>

>

> ##### ##### R2jags package ##### #####

>

>

> tic("Total R2jags")

> tic("Compiling and Simulation/Sampling")

> jags.m.R2jags <- jags(data=dat, inits=all.inits, parameters.to.save=params, n.chains = 3, n.iter = 30000, n.burnin = 10000,

+ model.file="Blocker\_Model\_Random.txt", DIC=TRUE, jags.module = c("glm","dic") )

Compiling model graph

Resolving undeclared variables

Allocating nodes

Graph information:

Observed stochastic nodes: 44

Unobserved stochastic nodes: 47

Total graph size: 964

Initializing model

|++++++++++++++++++++++++++++++++++++++++++++++++++| 100%

|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| 100%

> toc()

Compiling and Simulation/Sampling: 15.82 sec elapsed

> tic("generating summary")

> print(jags.m.R2jags)

Inference for Bugs model at "Blocker\_Model\_Random.txt", fit using jags,

3 chains, each with 30000 iterations (first 10000 discarded), n.thin = 20

n.sims = 3000 iterations saved

mu.vect sd.vect 2.5% 25% 50% 75% 97.5% Rhat n.eff

A -2.206 0.552 -3.293 -2.573 -2.209 -1.825 -1.150 1.002 3000

d[1] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 1.000 1

d[2] -0.248 0.064 -0.372 -0.291 -0.248 -0.207 -0.116 1.001 3000

mu[1] -2.442 0.446 -3.392 -2.711 -2.418 -2.133 -1.633 1.001 3000

mu[2] -2.188 0.242 -2.683 -2.344 -2.182 -2.021 -1.748 1.001 3000

mu[3] -2.125 0.275 -2.706 -2.303 -2.113 -1.932 -1.630 1.001 2400

mu[4] -2.397 0.082 -2.559 -2.454 -2.396 -2.340 -2.242 1.002 1400

mu[5] -2.416 0.157 -2.742 -2.520 -2.411 -2.306 -2.132 1.002 1700

mu[6] -2.228 0.340 -2.913 -2.443 -2.222 -1.995 -1.577 1.001 3000

mu[7] -1.710 0.083 -1.866 -1.766 -1.711 -1.655 -1.543 1.001 3000

mu[8] -2.116 0.122 -2.359 -2.197 -2.114 -2.033 -1.884 1.002 1200

mu[9] -1.960 0.148 -2.254 -2.056 -1.957 -1.860 -1.677 1.001 3000

mu[10] -2.242 0.070 -2.377 -2.289 -2.243 -2.193 -2.105 1.001 3000

mu[11] -2.313 0.116 -2.543 -2.391 -2.311 -2.233 -2.086 1.002 1800

mu[12] -1.473 0.129 -1.725 -1.558 -1.471 -1.387 -1.222 1.001 3000

mu[13] -2.994 0.220 -3.430 -3.139 -2.992 -2.841 -2.577 1.002 1700

mu[14] -2.739 0.129 -3.008 -2.823 -2.732 -2.648 -2.499 1.003 830

mu[15] -1.356 0.160 -1.677 -1.461 -1.357 -1.248 -1.049 1.001 3000

mu[16] -1.492 0.143 -1.786 -1.586 -1.490 -1.396 -1.212 1.001 3000

mu[17] -2.010 0.196 -2.421 -2.135 -2.001 -1.874 -1.648 1.001 2100

mu[18] -2.979 0.282 -3.569 -3.162 -2.969 -2.782 -2.461 1.002 1100

mu[19] -3.452 0.361 -4.185 -3.684 -3.432 -3.199 -2.810 1.001 3000

mu[20] -1.492 0.140 -1.774 -1.584 -1.489 -1.399 -1.215 1.002 1900

mu[21] -2.137 0.140 -2.415 -2.227 -2.138 -2.046 -1.857 1.001 3000

mu[22] -2.913 0.145 -3.201 -3.009 -2.913 -2.813 -2.627 1.001 3000

sd 0.132 0.082 0.008 0.070 0.124 0.185 0.310 1.006 420

deviance 258.907 8.046 244.318 253.445 258.383 264.025 276.572 1.003 810

For each parameter, n.eff is a crude measure of effective sample size,

and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)

pD = 32.3 and DIC = 291.2

DIC is an estimate of expected predictive error (lower deviance is better).

> jags.m.R2jags[["BUGSoutput"]][["median"]]

$A

[1] -2.208886

$d

[1] 0.0000000 -0.2484517

$deviance

[1] 258.3833

$mu

[1] -2.417829 -2.182389 -2.113350 -2.395505 -2.411393 -2.222102 -1.710524 -2.114056 -1.956887 -2.243180 -2.310854 -1.470940 -2.991918

[14] -2.731687 -1.357383 -1.489721 -2.000636 -2.968725 -3.431991 -1.489049 -2.137789 -2.913446

$sd

[1] 0.1241696

> toc()

generating summary: 0.04 sec elapsed

> toc()

Total R2jags: 15.86 sec elapsed

>

>

>

>

> ##### ##### rjags package ##### #####

>

>

> ##### read in inits with chains for rjags

> inits.rjags <- function(){

+ #chain 1

+ list(d=c( NA, 0),

+ sd=1,

+ mu=c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),

+ .RNG.name="lecuyer::RngStream", .RNG.seed=1

+ )

+ #chain 2

+ list(d=c( NA, -1),

+ sd=4,

+ mu=c(-3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3, -3),

+ .RNG.name="lecuyer::RngStream", .RNG.seed=2

+ )

+ #chain 3

+ list(d=c( NA, 2),

+ sd=2,

+ mu=c(-3, 5, -1, -3, 7, -3, -4, -3, -3, 0, -3, -3,0, 3, 5, -3, -3, -1, -3, -7, -3, -3),

+ .RNG.name="lecuyer::RngStream", .RNG.seed=3

+ )

+ }

>

>

> tic("Total rjags")

> tic("Compiling")

> jags.m.rjags <- jags.model( file = "Blocker\_Model\_Random.txt", data=dat2, inits=inits.rjags, n.chains=3, n.adapt=1500)

Compiling model graph

Resolving undeclared variables

Allocating nodes

Graph information:

Observed stochastic nodes: 44

Unobserved stochastic nodes: 47

Total graph size: 964

Initializing model

|++++++++++++++++++++++++++++++++++++++++++++++++++| 100%

> update(jags.m.rjags, 10000) # burn in

|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| 100%

> toc()

Compiling: 6.27 sec elapsed

> tic("Simulation/Sampling")

> samps\_coda <- coda.samples( jags.m.rjags, variable.names=c("mu", "d", "sd", "A"), n.iter=30000, DIC=T )

|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| 100%

> toc()

Simulation/Sampling: 16.54 sec elapsed

> tic("generating summary")

> summary(window(samps\_coda, start=10001)) # burnin = 10000

Iterations = 11501:41500

Thinning interval = 1

Number of chains = 3

Sample size per chain = 30000

1. Empirical mean and standard deviation for each variable,

plus standard error of the mean:

Mean SD Naive SE Time-series SE

A -2.1946 0.55225 0.0018408 0.0018661

d[1] 0.0000 0.00000 0.0000000 0.0000000

d[2] -0.2505 0.06513 0.0002171 0.0008655

mu[1] -2.4415 0.44730 0.0014910 0.0019784

mu[2] -2.1833 0.23992 0.0007997 0.0011201

mu[3] -2.1347 0.27088 0.0009029 0.0012604

mu[4] -2.3949 0.08164 0.0002721 0.0005212

mu[5] -2.4135 0.15626 0.0005209 0.0009876

mu[6] -2.2298 0.34791 0.0011597 0.0016289

mu[7] -1.7091 0.08428 0.0002809 0.0008187

mu[8] -2.1147 0.12223 0.0004074 0.0009183

mu[9] -1.9571 0.14736 0.0004912 0.0007968

mu[10] -2.2414 0.07002 0.0002334 0.0004550

mu[11] -2.3134 0.11572 0.0003857 0.0007274

mu[12] -1.4691 0.13145 0.0004382 0.0008926

mu[13] -2.9930 0.21628 0.0007209 0.0010377

mu[14] -2.7392 0.12883 0.0004294 0.0015266

mu[15] -1.3538 0.16049 0.0005350 0.0008214

mu[16] -1.4923 0.14445 0.0004815 0.0008226

mu[17] -2.0064 0.19792 0.0006597 0.0014094

mu[18] -2.9721 0.28621 0.0009540 0.0013887

mu[19] -3.4434 0.35753 0.0011918 0.0016903

mu[20] -1.4881 0.14340 0.0004780 0.0007752

mu[21] -2.1295 0.14279 0.0004760 0.0008739

mu[22] -2.9121 0.14579 0.0004860 0.0008498

sd 0.1356 0.07963 0.0002654 0.0022020

2. Quantiles for each variable:

2.5% 25% 50% 75% 97.5%

A -3.27657 -2.5646 -2.1940 -1.8241 -1.1088

d[1] 0.00000 0.0000 0.0000 0.0000 0.0000

d[2] -0.37410 -0.2932 -0.2528 -0.2086 -0.1169

mu[1] -3.39452 -2.7247 -2.4155 -2.1297 -1.6469

mu[2] -2.66801 -2.3418 -2.1781 -2.0185 -1.7317

mu[3] -2.69457 -2.3124 -2.1269 -1.9480 -1.6252

mu[4] -2.55965 -2.4488 -2.3937 -2.3394 -2.2388

mu[5] -2.73241 -2.5161 -2.4100 -2.3061 -2.1157

mu[6] -2.95035 -2.4525 -2.2137 -1.9915 -1.5940

mu[7] -1.87048 -1.7665 -1.7106 -1.6525 -1.5409

mu[8] -2.36381 -2.1951 -2.1123 -2.0324 -1.8829

mu[9] -2.24899 -2.0559 -1.9553 -1.8564 -1.6764

mu[10] -2.37866 -2.2885 -2.2416 -2.1946 -2.1042

mu[11] -2.54813 -2.3894 -2.3116 -2.2348 -2.0923

mu[12] -1.73274 -1.5568 -1.4660 -1.3794 -1.2190

mu[13] -3.43385 -3.1355 -2.9870 -2.8441 -2.5850

mu[14] -3.00897 -2.8224 -2.7333 -2.6493 -2.5016

mu[15] -1.67543 -1.4605 -1.3527 -1.2446 -1.0429

mu[16] -1.78531 -1.5870 -1.4897 -1.3939 -1.2191

mu[17] -2.41939 -2.1327 -1.9989 -1.8707 -1.6392

mu[18] -3.56451 -3.1576 -2.9588 -2.7740 -2.4455

mu[19] -4.19358 -3.6730 -3.4261 -3.1961 -2.7933

mu[20] -1.77580 -1.5839 -1.4869 -1.3907 -1.2116

mu[21] -2.41473 -2.2231 -2.1292 -2.0348 -1.8497

mu[22] -3.20153 -3.0088 -2.9106 -2.8145 -2.6298

sd 0.01018 0.0758 0.1282 0.1840 0.3159

Warning messages:

1: In FUN(X[[i]], ...) : start value not changed

2: In FUN(X[[i]], ...) : start value not changed

3: In FUN(X[[i]], ...) : start value not changed

> #### Median Calculation

> # Median for T1

> median(as.matrix(samps\_coda[,1]))

[1] -2.194048

> # Median for T2

> median(as.matrix(samps\_coda[,2]))

[1] 0

> # Median for d[2]

> median(as.matrix(samps\_coda[,3]))

[1] -0.2528315

> # Median for sd

> median(as.matrix(samps\_coda[,4]))

[1] -2.415499

> toc()

generating summary: 3.79 sec elapsed

> toc()

Total rjags: 26.64 sec elapsed

>

>

>

>

|  |
| --- |
| >  > ##### ##### runjags package ##### #####  >  >  > tic("Total runjags")  > tic("Compiling and Simulation/Sampling")  > jags.m.runjags <- run.jags(model="Blocker\_Model\_Random.txt", monitor=c("mu", "d", "sd", "A" ),  + data=list("ns"=ns, "nt"=nt, "na"=na, "r"=r, "n"=n, "t"=t) , n.chains=3, inits=all.inits, burnin = 10000, sample = 30000, adapt = 1500)  Compiling rjags model...  Calling the simulation using the rjags method...  Adapting the model for 1500 iterations...  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%  Burning in the model for 10000 iterations...  |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| 100%  Running the model for 30000 iterations...  |\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| 100%  Simulation complete  Calculating summary statistics...  Note: The monitored variable 'd[1]' appears to be non-stochastic; it will not be included in the convergence diagnostic  Calculating the Gelman-Rubin statistic for 26 variables....  Finished running the simulation  > toc()  Compiling and Simulation/Sampling: 36.89 sec elapsed  > tic("generating summary")  > print(jags.m.runjags)  JAGS model summary statistics from 90000 samples (chains = 3; adapt+burnin = 11500):    Lower95 Median Upper95 Mean SD Mode MCerr MC%ofSD SSeff AC.10 psrf  mu[1] -3.3279 -2.4146 -1.5818 -2.4407 0.44963 -- 0.0027677 0.6 26393 0.0076456 1.0001  mu[2] -2.655 -2.1743 -1.7056 -2.1825 0.24264 -- 0.0015327 0.6 25061 0.01548 0.99995  mu[3] -2.6667 -2.1278 -1.6035 -2.1357 0.27352 -- 0.001699 0.6 25918 0.0025946 0.99996  mu[4] -2.5524 -2.3963 -2.2368 -2.3968 0.080669 -- 0.00060476 0.7 17793 0.029291 1.0001  mu[5] -2.7129 -2.407 -2.0981 -2.4125 0.15593 -- 0.0012794 0.8 14853 0.039075 0.99998  mu[6] -2.9018 -2.2122 -1.5527 -2.2241 0.34589 -- 0.002101 0.6 27104 0.0036069 1.0001  mu[7] -1.8685 -1.7122 -1.5434 -1.7105 0.083874 -- 0.0011684 1.4 5153 0.11446 1.0005  mu[8] -2.3577 -2.1096 -1.8824 -2.1137 0.1219 -- 0.0010793 0.9 12755 0.060161 1  mu[9] -2.2461 -1.9552 -1.6639 -1.9575 0.14784 -- 0.001038 0.7 20286 0.0147 0.99997  mu[10] -2.3792 -2.2408 -2.1064 -2.2408 0.069251 -- 0.00056448 0.8 15050 0.041473 1.0002  mu[11] -2.5418 -2.311 -2.0893 -2.3132 0.11521 -- 0.0009159 0.8 15824 0.043016 1.0003  mu[12] -1.7254 -1.4666 -1.2149 -1.4687 0.13011 -- 0.0011543 0.9 12706 0.037145 1  mu[13] -3.4057 -2.9845 -2.5703 -2.9911 0.21443 -- 0.0013131 0.6 26666 0.011119 1.0001  mu[14] -3.0061 -2.7284 -2.4966 -2.7353 0.13021 -- 0.001997 1.5 4251 0.17248 1.0011  mu[15] -1.6665 -1.3512 -1.03 -1.353 0.16242 -- 0.0010363 0.6 24563 0.012426 0.99997  mu[16] -1.7677 -1.4859 -1.1997 -1.4893 0.14453 -- 0.0010972 0.8 17351 0.010906 1.0002  mu[17] -2.3915 -1.9963 -1.6248 -2.0026 0.19509 -- 0.0016947 0.9 13252 0.046284 1.0001  mu[18] -3.5619 -2.9558 -2.4359 -2.9688 0.28854 -- 0.0017747 0.6 26435 0.013215 0.99997  mu[19] -4.1266 -3.4215 -2.7404 -3.4418 0.35492 -- 0.0023182 0.7 23442 0.0076658 0.99997  mu[20] -1.7703 -1.4868 -1.2071 -1.4886 0.14365 -- 0.00095599 0.7 22580 0.019204 0.99996  mu[21] -2.4133 -2.1308 -1.8612 -2.1323 0.13989 -- 0.0010723 0.8 17020 0.040942 0.99995  mu[22] -3.2032 -2.9095 -2.6363 -2.9115 0.1454 -- 0.0010771 0.7 18224 0.031606 1.0001  d[1] 0 0 0 0 0 0 -- -- -- -- --  d[2] -0.37106 -0.25107 -0.11356 -0.2507 0.064935 -- 0.00098199 1.5 4373 0.28946 1.0005  sd 0.000058089 0.12139 0.28493 0.12935 0.086752 -- 0.0028505 3.3 926 0.72505 1.0057  A -3.2397 -2.1948 -1.082 -2.1928 0.54715 -- 0.0030814 0.6 31528 0.0011168 1.0003  Total time taken: 34.9 seconds  > summary(jags.m.runjags$mcmc) # f?r 2.5 - 97.5 CrI  Iterations = 11501:41500  Thinning interval = 1  Number of chains = 3  Sample size per chain = 30000  1. Empirical mean and standard deviation for each variable,  plus standard error of the mean:  Mean SD Naive SE Time-series SE  mu[1] -2.4390 0.45083 0.0015028 0.0020152  mu[2] -2.1837 0.24178 0.0008059 0.0011930  mu[3] -2.1342 0.27407 0.0009136 0.0012287  mu[4] -2.3965 0.08103 0.0002701 0.0005447  mu[5] -2.4130 0.15661 0.0005220 0.0011074  mu[6] -2.2244 0.34421 0.0011474 0.0015337  mu[7] -1.7108 0.08449 0.0002816 0.0010608  mu[8] -2.1130 0.12218 0.0004073 0.0009909  mu[9] -1.9580 0.14775 0.0004925 0.0008249  mu[10] -2.2412 0.06924 0.0002308 0.0004988  mu[11] -2.3134 0.11569 0.0003856 0.0007782  mu[12] -1.4683 0.13019 0.0004340 0.0010378  mu[13] -2.9905 0.21452 0.0007151 0.0010591  mu[14] -2.7351 0.13020 0.0004340 0.0018430  mu[15] -1.3525 0.16124 0.0005375 0.0008472  mu[16] -1.4901 0.14371 0.0004790 0.0008532  mu[17] -2.0022 0.19518 0.0006506 0.0014103  mu[18] -2.9705 0.28860 0.0009620 0.0014181  mu[19] -3.4407 0.35525 0.0011842 0.0018610  mu[20] -1.4883 0.14370 0.0004790 0.0008121  mu[21] -2.1320 0.13926 0.0004642 0.0009339  mu[22] -2.9118 0.14548 0.0004849 0.0009124  d[1] 0.0000 0.00000 0.0000000 0.0000000  d[2] -0.2507 0.06496 0.0002165 0.0010081  sd 0.1293 0.08658 0.0002886 0.0026439  A -2.2008 0.54835 0.0018278 0.0018064  2. Quantiles for each variable:  2.5% 25% 50% 75% 97.5%  mu[1] -3.401587 -2.72255 -2.4132 -2.1251 -1.6293  mu[2] -2.680014 -2.34296 -2.1763 -2.0177 -1.7305  mu[3] -2.700266 -2.31179 -2.1256 -1.9450 -1.6244  mu[4] -2.558019 -2.45032 -2.3953 -2.3416 -2.2403  mu[5] -2.732840 -2.51555 -2.4079 -2.3058 -2.1169  mu[6] -2.939019 -2.44652 -2.2109 -1.9860 -1.5907  mu[7] -1.872569 -1.76865 -1.7122 -1.6543 -1.5434  mu[8] -2.361275 -2.19319 -2.1095 -2.0299 -1.8824  mu[9] -2.252515 -2.05619 -1.9560 -1.8577 -1.6731  mu[10] -2.377599 -2.28744 -2.2413 -2.1950 -2.1051  mu[11] -2.547599 -2.38852 -2.3115 -2.2354 -2.0904  mu[12] -1.733136 -1.55337 -1.4659 -1.3796 -1.2209  mu[13] -3.430860 -3.13122 -2.9841 -2.8422 -2.5876  mu[14] -3.010579 -2.81920 -2.7282 -2.6450 -2.4977  mu[15] -1.674459 -1.45867 -1.3509 -1.2433 -1.0384  mu[16] -1.780267 -1.58429 -1.4878 -1.3927 -1.2151  mu[17] -2.408710 -2.12633 -1.9965 -1.8690 -1.6380  mu[18] -3.570893 -3.15627 -2.9584 -2.7705 -2.4391  mu[19] -4.185876 -3.66616 -3.4219 -3.1949 -2.7942  mu[20] -1.774388 -1.58398 -1.4863 -1.3911 -1.2101  mu[21] -2.409099 -2.22557 -2.1306 -2.0382 -1.8611  mu[22] -3.203062 -3.00773 -2.9101 -2.8131 -2.6335  d[1] 0.000000 0.00000 0.0000 0.0000 0.0000  d[2] -0.375118 -0.29406 -0.2513 -0.2101 -0.1177  sd 0.002742 0.06141 0.1209 0.1831 0.3213  A -3.278405 -2.57322 -2.2009 -1.8312 -1.1241  > toc()  generating summary: 3.78 sec elapsed  > toc()  Total runjags: 40.67 sec elapsed  >  >  >  >  >  > # Part 3: Testing running time of R code of NIMBLE ----------------------------------  >  >  >  >  > ##### Adaptions for NIMBLE  >  > Nimble\_constants = list(ns=ns, nt=nt, na=na, t=t)  > Nimble\_data = list(r=r, n=n)  > Nimble\_inits = list(d = c( NA, 0),  + mu = c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  + # zus?tzlich noch einzuf?gen:  + A=0)  > Code\_Model<- nimbleCode( {  + for(i in 1:ns){ # LOOP THROUGH STUDIES  + w[i,1] <- 0 # adjustment for multi-arm trials is zero for control arm  + delta[i,1] <- 0 # treatment effect is zero for control arm  + mu[i] ~ dnorm(0,.0001) # vague priors for all trial baselines  + for (k in 1:na[i]) { # LOOP THROUGH ARMS  + r[i,k] ~ dbin(p[i,k],n[i,k]) # binomial likelihood  + logit(p[i,k]) <- mu[i] + delta[i,k] # model for linear predictor  + rhat[i,k] <- p[i,k] \* n[i,k] # expected value of the numerators  + dev[i,k] <- 2 \* (r[i,k] \* (log(r[i,k])-log(rhat[i,k]))+ (n[i,k]-r[i,k]) \* (log(n[i,k]-r[i,k]) - log(n[i,k]-rhat[i,k]))) #Deviance contribution  + }  + resdev[i] <- sum(dev[i,1:na[i]]) # summed residual deviance contribution for this trial  + for (k in 2:na[i]) { # LOOP THROUGH ARMS  + delta[i,k] ~ dnorm(md[i,k],taud[i,k]) # trial-specific LOR distributions  + md[i,k] <- d[t[i,k]] - d[t[i,1]] + sw[i,k] # mean of LOR distributions (with multi-arm trial correction)  + taud[i,k] <- tau \*2\*(k-1)/k # precision of LOR distributions (with multi-arm trial correction)  + w[i,k] <- (delta[i,k] - d[t[i,k]] + d[t[i,1]]) # adjustment for multi-arm RCTs  + sw[i,k] <- sum(w[i,1:(k-1)])/(k-1) # cumulative adjustment for multi-arm trials  + }  + }  + totresdev <- sum(resdev[1:22]) #Total Residual Deviance  + d[1] <- 0 # treatment effect is zero for reference treatment  + sd ~ dunif(0,5) # vague prior for between-trial SD.  + tau <- pow(sd,-2) # between-trial precision = (1/between-trial variance)  +  + # Provide estimates of treatment effects T[k] on the natural (probability) scale  + # Given a Mean Effect, meanA, for 'standard' treatment 1, with precision (1/variance) precA  +  + for (k in 2:nt){ d[k] ~ dnorm(0,0.0001) }  + for (k in 1:nt) { logit(T[k]) <- A + d[k] }  + A ~ dnorm(-2.2, 3.3)  + })  >  >  >  >  > ##### ##### nimbleModel package ##### #####  >  >  > tic("Total nimbleModel")  > tic("Compiling and Simulation/Sampling")  > Model\_Nimble\_nimbleModel <- nimbleModel(code = Code\_Model, name = "ProcessedModel", constants = Nimble\_constants,  + data = Nimble\_data, inits = Nimble\_inits)  defining model...  building model...  setting data and initial values...  running calculate on model (any error reports that follow may simply reflect missing values in model variables) ...  checking model sizes and dimensions... This model is not fully initialized. This is not an error. To see which variables are not initialized, use model$initializeInfo(). For more information on model initialization, see help(modelInitialization).  model building finished.  > mcmc.out.nimbleModel <- nimbleMCMC(code = Code\_Model, constants = Nimble\_constants,  + data = Nimble\_data, inits = Nimble\_inits,  + nchains = 3, niter = 20000, nburnin = 10000,  + summary = TRUE, WAIC = F,  + monitors = c("mu", "d", "sd", "A"))  defining model...  building model...  setting data and initial values...  running calculate on model (any error reports that follow may simply reflect missing values in model variables) ...  checking model sizes and dimensions... This model is not fully initialized. This is not an error. To see which variables are not initialized, use model$initializeInfo(). For more information on model initialization, see help(modelInitialization).  checking model calculations...  NAs were detected in model variables: sd, logProb\_sd, tau, taud, lifted\_d1\_over\_sqrt\_oPtaud\_oBi\_comma\_k\_cB\_cP\_L12, delta, logProb\_delta, p, w, logProb\_r, rhat, dev, resdev, totresdev.  model building finished.  compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilation details.  compilation finished.  running chain 1...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  running chain 2...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  running chain 3...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  > toc()  Compiling and Simulation/Sampling: 72.39 sec elapsed  > tic("generating summary")  > mcmc.out.nimbleModel[["summary"]][["all.chains"]]  Mean Median St.Dev. 95%CI\_low 95%CI\_upp  A -2.2053574 -2.2064393 0.55331946 -3.287729763 -1.1169168  d[1] 0.0000000 0.0000000 0.00000000 0.000000000 0.0000000  d[2] -0.2484924 -0.2523728 0.06581974 -0.372553550 -0.1154655  mu[1] -2.4347464 -2.4043682 0.44892676 -3.394368098 -1.6328728  mu[2] -2.1930023 -2.1850921 0.23741014 -2.683951805 -1.7416426  mu[3] -2.1335068 -2.1246432 0.27454242 -2.696452123 -1.6156203  mu[4] -2.3962608 -2.3959351 0.08140664 -2.558859050 -2.2380321  mu[5] -2.4183742 -2.4143948 0.15589087 -2.732250340 -2.1226908  mu[6] -2.2205804 -2.2084132 0.34432177 -2.939939743 -1.5894198  mu[7] -1.7125721 -1.7144054 0.08498507 -1.876583654 -1.5426494  mu[8] -2.1144944 -2.1105586 0.12297235 -2.373170016 -1.8858768  mu[9] -1.9609630 -1.9569549 0.14553913 -2.254586226 -1.6853928  mu[10] -2.2426285 -2.2421788 0.07001647 -2.381559452 -2.1052524  mu[11] -2.3169380 -2.3149790 0.11641569 -2.551002576 -2.0941654  mu[12] -1.4662924 -1.4620369 0.12935708 -1.723632684 -1.2176585  mu[13] -2.9960697 -2.9896825 0.21898907 -3.438421287 -2.5781569  mu[14] -2.7383930 -2.7334280 0.12986132 -3.011867114 -2.4972762  mu[15] -1.3517485 -1.3477029 0.16011731 -1.672171624 -1.0477890  mu[16] -1.4905004 -1.4881260 0.14459657 -1.786487938 -1.2155364  mu[17] -2.0109093 -2.0024539 0.19920631 -2.428055881 -1.6449752  mu[18] -2.9735477 -2.9605462 0.28859853 -3.568089325 -2.4383277  mu[19] -3.4440685 -3.4254534 0.35705763 -4.201224645 -2.7977560  mu[20] -1.4856641 -1.4860805 0.14207935 -1.768445881 -1.2087566  mu[21] -2.1306840 -2.1296138 0.14085903 -2.411852035 -1.8565664  mu[22] -2.9135038 -2.9129492 0.14343630 -3.198153639 -2.6293308  sd 0.1324494 0.1248523 0.08631658 0.008704454 0.3205438  > toc()  generating summary: 0.02 sec elapsed  > toc()  Total nimbleModel: 72.42 sec elapsed  > mcmc.out.nimbleModel\_II <- nimbleMCMC(code = Code\_Model, constants = Nimble\_constants,  + data = Nimble\_data, inits = Nimble\_inits,  + nchains = 3, niter = 20000, nburnin = 10000,  + summary = F, WAIC = F, samplesAsCodaMCMC = T,  + monitors = c("mu", "d", "sd", "A"))  defining model...  building model...  setting data and initial values...  running calculate on model (any error reports that follow may simply reflect missing values in model variables) ...  checking model sizes and dimensions... This model is not fully initialized. This is not an error. To see which variables are not initialized, use model$initializeInfo(). For more information on model initialization, see help(modelInitialization).  checking model calculations...  NAs were detected in model variables: sd, logProb\_sd, tau, taud, lifted\_d1\_over\_sqrt\_oPtaud\_oBi\_comma\_k\_cB\_cP\_L12, delta, logProb\_delta, p, w, logProb\_r, rhat, dev, resdev, totresdev.  model building finished.  compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilation details.  compilation finished.  running chain 1...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  running chain 2...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  running chain 3...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  >  >  >  >  > ##### ##### readBUGSModel package ##### #####  >  >  > tic("Total readBUGSModel")  > tic("Compiling and Simulation/Sampling")  > readBUGS\_Model <- readBUGSmodel(model='Blocker\_Random\_Model\_Nimble.bug', data = 'Blocker\_Data\_Nimble.R',  + inits = 'Blocker\_Inits\_Nimble.R' )  defining model...  Detected r as data within 'constants'.  Adding r as data for building model.  building model...  setting data and initial values...  running calculate on model (any error reports that follow may simply reflect missing values in model variables) ...  checking model sizes and dimensions... This model is not fully initialized. This is not an error. To see which variables are not initialized, use model$initializeInfo(). For more information on model initialization, see help(modelInitialization).  model building finished.  > mcmc.out.readBUGSModel <- nimbleMCMC(code = readBUGS\_Model,  + nchains = 3, niter = 20000, nburnin = 10000,  + summary = TRUE, WAIC = F,  + monitors = c("mu", "d", "sd", "A"))  compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilation details.  compilation finished.  running chain 1...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  running chain 2...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  running chain 3...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  > toc()  Compiling and Simulation/Sampling: 62.73 sec elapsed  > tic("generating summary")  > mcmc.out.readBUGSModel[["summary"]][["all.chains"]]  Mean Median St.Dev. 95%CI\_low 95%CI\_upp  A -2.2002049 -2.2032824 0.54742259 -3.27741400 -1.1253620  d[1] 0.0000000 0.0000000 0.00000000 0.00000000 0.0000000  d[2] -0.2462332 -0.2479233 0.06790585 -0.37519047 -0.1080069  mu[1] -2.4513840 -2.4225643 0.45444070 -3.43241916 -1.6558143  mu[2] -2.1823704 -2.1776528 0.24458588 -2.67689314 -1.7212254  mu[3] -2.1382106 -2.1295316 0.27405090 -2.70827739 -1.6279829  mu[4] -2.3992282 -2.3983591 0.08417966 -2.56611099 -2.2373617  mu[5] -2.4205602 -2.4129140 0.15905785 -2.74251070 -2.1224287  mu[6] -2.2285685 -2.2164325 0.34470163 -2.95522893 -1.5840567  mu[7] -1.7040604 -1.7056774 0.08430213 -1.86286618 -1.5380438  mu[8] -2.1171983 -2.1139502 0.12424914 -2.36988131 -1.8828372  mu[9] -1.9556540 -1.9538800 0.15116031 -2.25228946 -1.6615677  mu[10] -2.2390509 -2.2392623 0.06949046 -2.37548093 -2.1032901  mu[11] -2.3159981 -2.3143627 0.11986445 -2.55928561 -2.0871953  mu[12] -1.4757275 -1.4724716 0.13242385 -1.74150963 -1.2226008  mu[13] -2.9946355 -2.9900953 0.21602840 -3.43755959 -2.5880576  mu[14] -2.7547980 -2.7493112 0.13311270 -3.03571091 -2.5118893  mu[15] -1.3502961 -1.3496940 0.16563055 -1.68181058 -1.0306767  mu[16] -1.4955435 -1.4931342 0.14817277 -1.78901646 -1.2130789  mu[17] -2.0145463 -2.0057449 0.20046786 -2.42236995 -1.6363286  mu[18] -2.9789646 -2.9663110 0.29578061 -3.60079422 -2.4321948  mu[19] -3.4463760 -3.4324209 0.35599817 -4.19632525 -2.7849340  mu[20] -1.4937097 -1.4914048 0.14552754 -1.78423145 -1.2118298  mu[21] -2.1216719 -2.1197333 0.14235413 -2.40446598 -1.8511962  mu[22] -2.9087682 -2.9081826 0.14958450 -3.20615247 -2.6229180  sd 0.1510076 0.1413248 0.08172981 0.02542484 0.3324879  > toc()  generating summary: 0.02 sec elapsed  > toc()  Total readBUGSModel: 62.75 sec elapsed  > mcmc.out.readBUGSModel\_II <- nimbleMCMC(code = readBUGS\_Model,  + nchains = 3, niter = 20000, nburnin = 10000,  + summary = F, samplesAsCodaMCMC = T, WAIC = F,  + monitors = c("mu", "d", "sd", "A"))  compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++ compilation details.  compilation finished.  running chain 1...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  running chain 2...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  running chain 3...  |-------------|-------------|-------------|-------------|  |-------------------------------------------------------|  >  >  >  >  >  > # Part 4: Testing further diagnostica with the coda package ---------------  >  >  >  >  > ##### ##### Testing of the effective Size ##### #####  >  >  > ## of WinBUGS  > effectiveSize(test\_WinBUGS$sims.matrix)  mu[1] mu[2] mu[3] mu[4] mu[5] mu[6] mu[7] mu[8] mu[9] mu[10] mu[11] mu[12] mu[13] mu[14]  1002.0000 986.0226 1002.0000 1002.0000 1002.0000 1002.0000 1097.6112 1103.5022 1002.0000 1002.0000 1002.0000 1002.0000 1002.0000 1002.0000  mu[15] mu[16] mu[17] mu[18] mu[19] mu[20] mu[21] mu[22] d[2] sd A deviance  873.4399 1211.0671 1002.0000 1002.0000 1002.0000 1002.0000 1002.0000 1002.0000 1002.0000 1247.7062 1321.5293 1002.0000  >  >  > ## of JAGS  > # jagsUI  > effectiveSize(jags.m.jagsUI)  A d[1] d[2] deviance mu[1] mu[10] mu[11] mu[12] mu[13] mu[14] mu[15] mu[16] mu[17] mu[18]  3000.0000 0.0000 1490.2365 2016.7827 3285.8150 2531.3452 3000.0000 3000.0000 3000.0000 1504.1134 3000.0000 3000.0000 2609.1048 3000.0000  mu[19] mu[2] mu[20] mu[21] mu[22] mu[3] mu[4] mu[5] mu[6] mu[7] mu[8] mu[9] sd  3000.0000 3000.0000 3000.0000 2769.0339 3000.0000 3000.0000 3000.0000 2661.3659 3323.1077 1354.1399 3000.0000 3000.0000 340.1969  > # R2jags  > effectiveSize(jags.m.R2jags)  A d[1] d[2] deviance mu[1] mu[10] mu[11] mu[12] mu[13] mu[14] mu[15] mu[16] mu[17] mu[18]  3000.0000 0.0000 1510.2779 1522.2109 3000.0000 3000.0000 3000.0000 1883.9479 3000.0000 1328.3735 3000.0000 3000.0000 3316.3022 3000.0000  mu[19] mu[2] mu[20] mu[21] mu[22] mu[3] mu[4] mu[5] mu[6] mu[7] mu[8] mu[9] sd  3000.0000 3178.2544 3000.0000 3000.0000 3000.0000 3000.0000 2782.6933 3174.6696 3000.0000 2150.4139 2693.7747 3000.0000 478.7636  > # rjags  > effectiveSize(samps\_coda)  A d[1] d[2] mu[1] mu[2] mu[3] mu[4] mu[5] mu[6] mu[7] mu[8] mu[9] mu[10] mu[11]  87583.847 0.000 5662.327 51120.845 45883.217 46188.226 24543.274 25035.650 45620.843 10596.286 17715.435 34200.314 23681.710 25311.956  mu[12] mu[13] mu[14] mu[15] mu[16] mu[17] mu[18] mu[19] mu[20] mu[21] mu[22] sd  21691.033 43446.863 7122.078 38177.891 30835.139 19721.787 42480.098 44743.425 34219.628 26699.120 29431.514 1307.807  > # runjags  > effectiveSize(jags.m.runjags)  mu[1] mu[2] mu[3] mu[4] mu[5] mu[6] mu[7] mu[8] mu[9] mu[10] mu[11] mu[12]  43690.7333 36986.3704 48499.0459 19850.1981 17782.4507 45934.5075 6275.4269 12959.8959 29999.9685 15014.7369 17921.9142 15009.8874  mu[13] mu[14] mu[15] mu[16] mu[17] mu[18] mu[19] mu[20] mu[21] mu[22] d[1] d[2]  36448.7424 4356.0249 35364.4895 25416.0853 14590.1364 35674.3221 35790.9434 26318.9553 19007.3035 22090.3522 0.0000 4123.6849  sd A  865.1717 90235.8369  Warning message:  In as.mcmc.runjags(x) : Combining the 3 mcmc chains together  >  >  > ## of NIMBLE  > # nimbleModel  > EES1 <- effectiveSize(mcmc.out.nimbleModel[["samples"]][["chain1"]])  > EES2 <- effectiveSize(mcmc.out.nimbleModel[["samples"]][["chain2"]])  > EES3 <- effectiveSize(mcmc.out.nimbleModel[["samples"]][["chain3"]])  > e <- (EES1+EES2+EES3)/3  > e  A d[1] d[2] mu[1] mu[2] mu[3] mu[4] mu[5] mu[6] mu[7] mu[8]  10000.00000 0.00000 276.98318 1943.73206 1884.88030 2083.45264 1136.98505 1257.97151 2076.40684 623.88232 954.13123  mu[9] mu[10] mu[11] mu[12] mu[13] mu[14] mu[15] mu[16] mu[17] mu[18] mu[19]  1574.01433 1070.55780 1039.94249 1205.14458 1975.13926 436.74226 1637.22136 1429.87867 1016.27674 1954.14863 2073.21674  mu[20] mu[21] mu[22] sd  1557.60177 1174.45065 1495.29295 54.58571  > # readBUGSModel  > EES1\_II <- effectiveSize(mcmc.out.readBUGSModel[["samples"]][["chain1"]])  > EES2\_II <- effectiveSize(mcmc.out.readBUGSModel[["samples"]][["chain2"]])  > EES3\_II <- effectiveSize(mcmc.out.readBUGSModel[["samples"]][["chain3"]])  > f <- (EES1\_II+EES2\_II+EES3\_II)/3  > f  A d[1] d[2] mu[1] mu[2] mu[3] mu[4] mu[5] mu[6] mu[7] mu[8] mu[9]  9770.52852 0.00000 389.11392 2048.61781 2001.72016 1938.17265 1176.15380 1251.16873 1928.09467 720.74967 891.14748 1454.24462  mu[10] mu[11] mu[12] mu[13] mu[14] mu[15] mu[16] mu[17] mu[18] mu[19] mu[20] mu[21]  1079.71414 1050.94833 1167.69264 1712.77156 498.39468 1523.56626 1411.51863 945.30504 1939.54205 1930.85591 1452.47641 1297.79648  mu[22] sd  1377.14999 84.18761  >  >  >  >  >  > # Part 5: Testing trace- and density plot with the plot function ---------------------------------------------------------  >  >  >  > ## the plots are generated with the plot()-function if possible  > ## else they are generated by a separate function  >  > ## the files are saved in external pdf files  >  >  > par(ask=F)  >  > ## of WinBUGS  > pdf(file="Traceplots\_WinBUGS.pdf")  > plot(test\_WinBUGS)  > R2jags::traceplot(test\_WinBUGS, ask = F)  > plot(density(test\_WinBUGS$sims.matrix)) # better only direct via programm  > dev.off()  RStudioGD  2  >  >  > ## of JAGS  > # jagsUI  > pdf(file="Traceplots\_jagsUI.pdf")  > plot(jags.m.jagsUI)  > R2jags::traceplot(jags.m.jagsUI, ask = F)  > densplot()  Error in as.matrix(x) : argument "x" is missing, with no default  > dev.off()  RStudioGD  2  >  > # R2jags  > pdf(file="Traceplots\_R2jags.pdf")  > plot(jags.m.R2jags)  > R2jags::traceplot(jags.m.R2jags, ask = F)  > dev.off()  RStudioGD  2  >  > # rjags  > pdf(file="Traceplots\_rjags.pdf")  > plot(samps\_coda)  > dev.off()  RStudioGD  2  >  > # runjags  > pdf(file="Traceplots\_runjags.pdf")  > plot(jags.m.runjags)  Generating plots...  > R2jags::traceplot(jags.m.runjags$mcmc, ask = F)  > dev.off()  RStudioGD  2  >  >  > ## of nimble  > plot(mcmc.out.nimbleModel\_II)  > plot(mcmc.out.readBUGSModel\_II)  >  >  >  >  >  > # Part 6: Testing autocorrelation plot with the plot function ---------------------------------------------------------  >  >  >  >  > par(ask=F)  >  > ## of WinBUGS  > pdf(file="autocorrplot\_WinBUGS.pdf")  > autocorr.plot(test\_WinBUGS$sims.matrix, ask = F)  > dev.off()  RStudioGD  2  >  >  > ## of JAGS  > # jagsUI  > pdf(file="autocorrplot\_jagsUI.pdf")  > autocorr.plot(jags.m.jagsUI, ask = F)  > dev.off()  RStudioGD  2  >  > # R2jags  > pdf(file="autocorrplot\_R2jags.pdf")  > autocorr.plot(jags.m.R2jags, ask = F)  > dev.off()  RStudioGD  2  >  > # rjags  > pdf(file="autocorrplot\_rjags.pdf")  > autocorr.plot(samps\_coda, ask = F)  > dev.off()  RStudioGD  2  >  > # runjags  > pdf(file="autocorrplot\_runjags.pdf")  > autocorr.plot(jags.m.runjags, ask = F)  Warning message:  In as.mcmc.runjags(x) : Combining the 3 mcmc chains together  > dev.off()  RStudioGD  2  >  >  > ## of nimble  > # nimblemodel  > pdf(file="autocorrplot\_nimblemodel.pdf")  > autocorr.plot(mcmc.out.nimbleModel\_II, ask = F)  > dev.off()  RStudioGD  2  > # readBUGSmodel  > pdf(file="autocorrplot\_nimblemodel.pdf")  > autocorr.plot(mcmc.out.readBUGSModel\_II, ask = F)  > dev.off()  RStudioGD  2  >  >  >  >  >  > # Part 7: Comparison of Rhat ----------------------------------------------  >  >  >  > ### use of rhat function of mcmcr  >  >  > ## of WinBUGS  > rhat(as.mcmc(test\_WinBUGS$sims.matrix), by = "term")  $mu  [1] 1.000 1.000 1.004 1.000 1.001 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.001 1.000  $d  [1] 1 1  $sd  [1] 1.001  $A  [1] 1  $deviance  [1] 1  >  >  > ## of JAGS  > # jagsUI  > rhat(as.mcmc(jags.m.jagsUI$BUGSoutput$sims.matrix), by = "term" )  $A  [1] 1  $d  [1] 1 1  $deviance  [1] 1  $mu  [1] 1.000 1.000 1.000 1.001 1.003 1.000 1.000 1.000 1.001 1.000 1.001 1.000 1.000 1.000 1.000 1.001 1.000 1.000 1.000 1.000 1.000 1.000  $sd  [1] 1.001  >  > # R2jags  > rhat(as.mcmc(jags.m.R2jags$BUGSoutput$sims.matrix), by = "term" )  $A  [1] 1  $d  [1] 1 1  $deviance  [1] 1  $mu  [1] 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.001  $sd  [1] 1  >  > # rjags  > # no function available  > rhat(samps\_coda, by = "term", as\_df = FALSE)  $A  [1] 1  $d  [1] 1 1  $mu  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  $sd  [1] 1.001  >  > # runjags  > # psrf = rhat  > rhat(jags.m.runjags$mcmc, by = "term")  $A  [1] 1  $d  [1] 1.000 1.001  $mu  [1] 1.000 1.000 1.000 1.000 1.000 1.000 1.001 1.000 1.000 1.000 1.000 1.000 1.000 1.001 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000  $sd  [1] 1.006  >  >  > ## of nimble  > # nimblemodel  > rhat(mcmc.out.nimbleModel\_II, by = "term", as\_df = FALSE)  $A  [1] 1  $d  [1] 1.000 1.003  $mu  [1] 1.000 1.001 1.000 1.000 1.002 1.000 1.005 1.003 1.001 1.002 1.001 1.002 1.000 1.009 1.000 1.001 1.002 1.001 1.000 1.000 1.002 1.001  $sd  [1] 1.036  >  > # readBUGSmodel  > rhat(mcmc.out.readBUGSModel\_II, by = "term", as\_df = FALSE)  $A  [1] 1  $d  [1] 1.000 1.006  $mu  [1] 1.000 1.000 1.000 1.001 1.001 1.000 1.002 1.003 1.001 1.002 1.001 1.001 1.001 1.006 1.001 1.000 1.001 1.000 1.001 1.002 1.000 1.000  $sd  [1] 1.013  >  >  >  >  >  > # Part 8: Testing with functions of the sbfnk/fitR package ---------------------------------------------------  >  >  > ## sbfnk/fitR: Tool box for fitting dynamic infectious disease models to time series.  >  >  >  >  > ##### ##### burnAndThin() function ##### #####  >  >  > ## this test compares to "mixing" diagnostics and should show the effect of increasing of thinning,  > ## e.g. to reduce the amount of memory and storage space in long chains.  > ## Afterwards, the autocorrelation plots will compared to the unthinned.  >  >  > # BaT = Burn and Thin  >  >  > par(ask=F)  >  >  > ## of WinBUGS  > BaT\_BUGS <- burnAndThin(test\_WinBUGS$sims.matrix)  > pdf(file="BaT\_autocorrplot\_WinBUGS.pdf")  > autocorr.plot(BaT\_BUGS, ask = F)  > dev.off()  RStudioGD  2  >  >  > ## of JAGS  > # jagsUI  > BaT\_jagsUI <- burnAndThin(jags.m.jagsUI$BUGSoutput$sims.matrix)  > pdf(file="BaT\_autocorrplot\_jagsUI.pdf")  > autocorr.plot(BaT\_jagsUI, ask = F)  > dev.off()  RStudioGD  2  >  > # R2jags  > BaT\_R2jags <- burnAndThin(jags.m.R2jags$BUGSoutput$sims.matrix)  > pdf(file="BaT\_autocorrplot\_R2jags.pdf")  > autocorr.plot(BaT\_R2jags, ask = F)  > dev.off()  RStudioGD  2  >  > # rjags  > BaT\_rjags <- burnAndThin(samps\_coda)  > pdf(file="BaT\_autocorrplot\_rjags.pdf")  > autocorr.plot(BaT\_rjags, ask = F)  > dev.off()  RStudioGD  2  >  > # runjags  > BaT\_runjags <- burnAndThin(jags.m.runjags$mcmc)  > pdf(file="BaT\_autocorrplot\_runjags.pdf")  > autocorr.plot(BaT\_runjags, ask = F)  > dev.off()  RStudioGD  2  >  >  > ## of nimble  > # nimblemodel  > BaT\_nimblemodel <- burnAndThin(mcmc.out.nimbleModel\_II)  > pdf(file="BaT\_autocorrplot\_nimblemodel.pdf")  > autocorr.plot(BaT\_nimblemodel, ask = F)  > dev.off()  RStudioGD  2  >  >  > # readBUGSmodel  > BaT\_readBUGSmodel <- burnAndThin(mcmc.out.readBUGSModel\_II)  > pdf(file="BaT\_autocorrplot\_nimblemodel.pdf")  > autocorr.plot(BaT\_readBUGSmodel, ask = F)  > dev.off()  RStudioGD  2  >  >  >  >  >  > ##### ##### effective sample size (ESS) function ##### #####  >  >  >  > ## this is a way to estimate for burn-in cut-off.  > ## EEs is the number of independent samples equivalent to our number of autocorrelated samples  > ## s.: http://sbfnk.github.io/mfiidd/mcmc\_diagnostics.html#1\_objectives  > ## Aim: which package achieve earlier the state for sampling  > ## -> reduction of expenses  >  > ## Note: plots need many space,  > ## only ESS from the mcmcr package will be better  >  > #### ESS  > ## of WinBUGS  > ess(as.mcmc(test\_WinBUGS$sims.matrix), by = "term")  $mu  [1] 1002 815 1002 1002 1002 713 1002 969 1002 825 1002 1002 1002 993 847 1002 1002 998 1002 1002 935 996  $d  [1] 1002 1002  $sd  [1] 1002  $A  [1] 1002  $deviance  [1] 936  >  >  > ## of JAGS  > # jagsUI  > ess(as.mcmc(jags.m.jagsUI), by = "term")  $A  [1] 2958  $d  [1] 3000 1584  $deviance  [1] 1935  $mu  [1] 2967 2913 2883 2796 2679 3000 1536 2760 2745 2658 2745 2919 2934 1560 2688 2928 2484 2946 2958 2814 2421 2826  $sd  [1] 378  >  > # R2jags  > ess(as.mcmc(jags.m.R2jags), by = "term")  $A  [1] 2928  $d  [1] 3000 1422  $deviance  [1] 1659  $mu  [1] 2961 2892 2937 2925 3000 2874 2112 2478 2985 2712 2994 1917 2964 1587 2949 2742 2931 2919 2940 2949 2865 2871  $sd  [1] 510  >  > # rjags  > ess(samps\_coda, by = "term")  $A  [1] 90000  $d  [1] 90000 4410  $mu  [1] 48420 41400 45990 22140 19530 45900 6390 12960 33300 22770 25200 16110 42390 4860 36450 25920 14850 43200 43830 34110 20880 24030  $sd  [1] 990  >  > # runjags  > ess(jags.m.runjags$mcmc, by = "term")  $A  [1] 89910  $d  [1] 90000 3510  $mu  [1] 48240 37980 47700 18540 17640 48780 4590 11520 32040 14670 17190 15660 39510 3870 34380 26190 13410 37800 42840 29610 18540 22770  $sd  [1] 900  >  >  > ## of nimble  > # nimblemodel  > ess(mcmc.out.nimbleModel\_II, by = "term")  $A  [1] 29970  $d  [1] 30000 1110  $mu  [1] 6150 5700 5970 3300 3870 5760 1650 2580 4320 2460 3120 3090 4590 1080 4200 3810 2340 4920 5640 3780 3390 3690  $sd  [1] 240  >  > # readBUGSmodel  > ess(mcmc.out.readBUGSModel\_II, by = "term")  $A  [1] 28650  $d  [1] 30000 900  $mu  [1] 6210 5280 5820 3210 3420 5790 1080 2160 3810 2670 2610 3150 5280 720 4350 4050 3450 5550 5610 3840 2820 4230  $sd  [1] 150  >  >  >  >  >  > # Part 9: Convergence diagnostic with the Gelman-Rubin Diagnostic ------------------------------------------  >  >  > ## with the use of the BayesianTools package  > ## Note: gelmanDiagnostics() need know an object of mcmcSampler or mcmcSamplerList  > ## so the functions of the coda package are used  > ## Note: Rhat = ‘potential scale reduction factor’ from the gelman.diag()  > ## however the gelman.diag() generate an upper C.I. for Rhat and can be used for Verification  > ## so this Part can be continued if necessary  >  >  >  >  >  > ########## ########## ########## Test finished ########## ########## ########## |
|  |
| |  | | --- | | > | |