> ########## ########## Simulation Blocker Beispiel mit Random Effects mit dem R2jags package

> ########## Die Working Directory muss auf Ihre Bedürfnisse angepasst werden "samps\_coda" aktiviert werden.

>

>

> # Teil Simulation mit JAGS ------------------------------------------------------------------

>

>

> ##### Clear data

> rm(list=ls())

>

>

> ##### load libraries

> library(rjags) # R2jags benötigt rjags

Lade nötiges Paket: coda

Linked to JAGS 4.3.0

Loaded modules: basemod,bugs

> library(R2jags)

Attache Paket: ‘R2jags’

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

traceplot

> library(random)

> #load.module("glm")

> load.module("lecuyer")

module lecuyer loaded

> #load.module("dic")

>

>

> ##### Sichergehen richtiger Working directory

> setwd("C:/Users/IvanB/Desktop/Masterarbeit/Statistische Programme und Gibbs Sampler/Programm JAGS/Nachrechnen TSD2-Dokument/Nachrechnen mit R2jags/Blocker")

>

>

> ##### 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

>

>

> ##### Parameter to monitor/save

> params <- c("d[2]", "T[1]", "T[2]", "sd" , "totresdev" )

>

>

> ##### 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)

>

>

> ##### define JAGS model within R

> cat("model{ # \*\*\* PROGRAM STARTS

+ 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[]) #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)

+ } ",

+ file="Blocker\_Random.txt")

>

>

> ##### Set up the JAGS model and settings

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

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

module glm loaded

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%

> # zusätzlich noch mehrere Argumente standardmäßig dabei, v.a. interessant: DIC, jags.module

>

>

> #### optional, falls nicht konvergiert:

> #jags.m.upd <- autojags(jags.m)

>

>

>

> # Ausgabe posteriore Werte, Berechnung Median und Berechnung DIC --------------------------

>

> print(jags.m)

Inference for Bugs model at "Blocker\_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

T[1] 0.112 0.056 0.037 0.072 0.100 0.137 0.255 1.001 3000

T[2] 0.090 0.047 0.029 0.057 0.079 0.112 0.211 1.001 3000

d[2] -0.248 0.065 -0.373 -0.293 -0.249 -0.204 -0.115 1.001 3000

sd 0.138 0.082 0.006 0.076 0.133 0.190 0.316 1.024 320

totresdev 41.615 8.023 27.349 35.998 41.088 46.788 58.945 1.005 460

deviance 259.046 8.023 244.780 253.429 258.518 264.219 276.376 1.005 470

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 = 32.1 and DIC = 291.1

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

>

> #### Median

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

$T

[1] 0.09972195 0.07946544

$d

[1] -0.2492721

$deviance

[1] 258.5182

$sd

[1] 0.1325897

$totresdev

[1] 41.08764

>

>

>

> # Berechnung Daten für leverage plot --------------------------------------

>

> #### erneute Simulation für die Erzeugung von dev und rhat

> params\_lev\_plot <- c("dev", "rhat" )

> jags.m\_levPlot <- jags(data=dat, inits=all.inits, parameters.to.save=params\_lev\_plot, n.chains = 3, n.iter=10000, n.burnin=0,

+ model.file="Blocker\_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%

>

>

> #jags.m\_levPlot[["BUGSoutput"]][["summary"]]

> out\_lePlo <- capture.output( jags.m\_levPlot[["BUGSoutput"]][["summary"]])

> cat("Hilf\_lePlo", out\_lePlo, file="Hilf.txt", sep="\n", append=TRUE)

>

> Hilf\_data = read.table("Hilf.txt", sep = "", header=F, skip=2, nrows=89)

> #Hilf\_data

>

>

> #### Berechnung w\_ik

>

> Hilf\_dev <- Hilf\_data[1:44,2]

> Hilf\_dev

[1] 0.5829955 1.0867171 0.7906725 0.8087222 1.0131691 0.7207412 1.3576668 0.8430031 0.7947328 0.8942742 0.6487065 0.8639446 0.9179019

[14] 2.3639031 0.6595514 0.7058580 1.0094181 0.9109699 0.8837859 0.6640569 1.2454450 1.2001015 0.5677097 0.9095700 0.5194309 0.7363245

[27] 1.1221345 0.5726955 1.4430317 0.9199727 0.6847574 0.8278519 0.7296970 0.9141888 0.7019012 2.5286923 0.5960270 0.6870914 1.0781582

[40] 1.1057728 1.0910483 0.6127757 1.0919903 1.1576753

> Hilf\_dev\_k1 <- Hilf\_dev[1:22]

> #Hilf\_dev\_k1

> Hilf\_dev\_k2 <- Hilf\_dev[23:44]

> #Hilf\_dev\_k2

> Hilf\_dev\_II <- cbind(Hilf\_dev\_k1, Hilf\_dev\_k2)

> #Hilf\_dev\_II

> Hilf\_dev\_III <- cbind(Hilf\_dev\_II, total = rowMeans(Hilf\_dev\_II))

> #Hilf\_dev\_III

> w\_ik <- sqrt(Hilf\_dev\_III[,3])

> w\_ik\_neg <- -w\_ik

>

> fertige\_Daten\_w\_ik <- cbind(Hilf\_dev\_III, w\_ik\_neg, w\_ik)

> #fertige\_Daten\_w\_ik

>

> # manuelle Berechnung von pD

> #dev ist Std-Abweichung jedes einzelnen Werts

> # insg 22 Wertepaare

> Var\_manuell <- sum(Hilf\_dev)^2/44

> pD\_manuell <- Var\_manuell/2

> pD\_manuell

[1] 19.63222

>

> #### Berechnung leverage\_ik

>

> Hilf\_rhat <- Hilf\_data[46:89,2]

> #Hilf\_rhat

> Hilf\_rhat\_k1 <- Hilf\_rhat[1:22]

> #Hilf\_rhat\_k1

> Hilf\_rhat\_k2 <- Hilf\_rhat[23:44]

> #Hilf\_rhat\_k2

> dev\_tilde\_erst\_k1 <- data[,1]\*log(data[,1]/Hilf\_rhat\_k1)

> #dev\_tilde\_erst\_k1

> dev\_tilde\_zweit\_k1 <- (data[,3]-data[,1])\*log((data[,3]-data[,1])/(data[,3]-Hilf\_rhat\_k1))

> #dev\_tilde\_zweit\_k1

> dev\_tilde\_gesamt\_k1 <- 2\*(dev\_tilde\_erst\_k1+dev\_tilde\_zweit\_k1)

> #dev\_tilde\_gesamt\_k1

> leverage\_k1 <- fertige\_Daten\_w\_ik[,1] - dev\_tilde\_gesamt\_k1

>

> leverage\_k1

[1] 0.5299988 0.7050115 0.6932975 0.8086201 0.6486638 0.6228980 0.8591418 0.6716305 0.6876449 0.8325688 0.6423352 0.6606504 0.6848803

[14] 0.6908089 0.6384951 0.6500069 0.4259139 0.4696381 0.3701701 0.6602392 0.7114843 0.7570898

>

> dev\_tilde\_erst\_k2 <- data[,2]\*log(data[,2]/Hilf\_rhat\_k2)

> #dev\_tilde\_erst\_k2

> dev\_tilde\_erst\_k2 <- data[,2]\*log(data[,2]/Hilf\_rhat\_k2)

> #dev\_tilde\_erst\_k2

> dev\_tilde\_zweit\_k2 <- (data[,4]-data[,2])\*log((data[,4]-data[,2])/(data[,4]-Hilf\_rhat\_k2))

> #dev\_tilde\_zweit\_k2

> dev\_tilde\_gesamt\_k2 <- 2\*(dev\_tilde\_erst\_k2+dev\_tilde\_zweit\_k2)

> #dev\_tilde\_gesamt\_k2

> leverage\_k2 <- fertige\_Daten\_w\_ik[,2] - dev\_tilde\_gesamt\_k2

>

> leverage\_k2

[1] 0.5269115 0.4087351 0.3645833 0.7356503 0.6824313 0.4398609 0.8078573 0.7492016 0.5352714 0.7554300 0.7205097 0.6992940 0.4149144

[14] 0.9149172 0.5757078 0.6268163 0.7985261 0.6458935 0.7097582 0.6053958 0.5559085 0.5106389

>

>

> #### Erzeugen leverage plot

> library(car)

Lade nötiges Paket: carData

> scatterplot(c(fertige\_Daten\_w\_ik[,"w\_ik\_neg"], fertige\_Daten\_w\_ik[,"w\_ik"]), c(leverage\_k1, leverage\_k2), main="Leverage plot for the random effects model", xlim=c(-3,3), ylim=c(0,4.5), xlab=expression('w'[ik]), ylab=expression('leverage'[ik]), regLine =F, smooth=F, boxplots=F )

> curve(-x^2+1, from=-3, to=3, col="red", lty="solid", add=T)

> curve(-x^2+2, from=-3, to=3, col="green", lty="dashed", add=T)

> curve(-x^2+3, from=-3, to=3, col="blueviolet", lty="dotted", add=T)

> curve(-x^2+4, from=-3, to=3, col="blue", lty="dotdash", add=T)

>

>

>

> # zusätzliche Diagnostik, bei Bedarf aktivieren --------------------------------------------------------------------

>

>

> #pdf("Blocker\_Random\_trace.pdf")

> #plot(jags.m)

> #traceplot(jags.m)

> #dev.off()

>

>

> # Generate MCMC object for analysis

> #Anm: es scheint, dass die Zeile "jags.m.mcmc <- as.mcmc(jags.m) " manuell ausgeführt werden muss

> #jags.m.mcmc <- as.mcmc(jags.m)

> #jags.m.mcmc

>

> #pdf("jags.m.mcmc.autocorr.pdf") # Autocorrelation plot

> #autocorr.plot(jags.m.mcmc)

> #dev.off()

>

>

> # Other diagnostics using CODA:

> #gelman.plot(jags.m.mcmc)

> #geweke.diag(jags.m.mcmc)

> #geweke.plot(jags.m.mcmc)

> #raftery.diag(jags.m.mcmc)

> #heidel.diag(jags.m.mcmc)

>

>

>

>

>

> ########## ########## ########## Simulation beendet ########## ########## ##########