> ########## ########## Simulation Blocker Beispiel mit Fixed Effects mit NIMBLE ########## ##########

> ########## Verwendung nimbleModel

> ########## Die Working Directory muss auf Ihre Bedürfnisse angepasst werden.

>

>

>

>

> # Teil 1 Creating a model and Simulation ------------------------------------------------------------------

>

>

>

> ##### Clear data

> rm(list=ls())

>

>

>

> ##### Sichergehen richtiger Working directory

> setwd("C:/Users/IvanB/Desktop/Masterarbeit/Statistische Programme und Gibbs Sampler/NIMBLE/Nachrechnen TSD2/Blocker")

>

>

>

> ##### load libraries

> library(car)

> library(nimble)

> library(coda)

> #library(igraph)

>

>

>

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

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

> #head(data) # Shows the first six entries

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

> #head(data2) # Shows the first six entries

>

>

>

> ##### Values for simulation, prepare dat for NIMBLE (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

>

>

>

> ##### Zuordnen der Argumente für NIBMLE

>

>

> ### Zuordnen Konstanten

> Nimble\_constants = list(ns=ns, nt=nt, na=na, t=t)

> #Nimble\_constants

>

>

> ### Zuordnen data

> Nimble\_data = list(r=r, n=n)

> #Nimble\_data

>

>

> ### Zuordnen Inits

> 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),

+ # zusätzlich noch einzufügen:

+ A=0)

>

>

>

> ### Create Model Code

> Code\_Model<- nimbleCode( {

+ for(i in 1:ns){ # LOOP THROUGH STUDIES

+ 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] + d[t[i,k]] - d[t[i,1]] # 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

+ }

+ totresdev <- sum(resdev[1:22]) #Total Residual Deviance

+ d[1]<-0 # treatment effect is zero for reference treatment

+ for (k in 2:nt){ d[k] ~ dnorm(0,.0001) } # vague priors for treatment effects

+

+ # 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 1:nt) { logit(T[k]) <- A + d[k] }

+ A ~ dnorm(-2.2, 3.3)

+ })

>

>

> # nimbleModel prozessiert BUGS-Modellcode und optionale Konstanten, Daten und Initialwerte. Liefert ein NIMBLE-Modell zurück.

> # dieser Schritt ist bei den hier getesteten BUGS Modellen nicht nötig

> Model\_Nimble <- 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...

model building finished.

>

> Model\_Nimble$initializeInfo()

All model variables are initialized.

>

>

>

> ##### Simulation

> mcmc.out <- 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("rhat",'dev', "totresdev", "T", "d"))

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...

checking model calculations...

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...

|-------------|-------------|-------------|-------------|

|-------------------------------------------------------|

> #help(buildMCMC)

>

>

>

>

>

> # Teil 2: Anzeigen Ergebnisse der Simulation ------------------------------

>

>

>

>

> #### Zusammenfassung posterioreer Werte

> mcmc.out[["summary"]][["all.chains"]]

Mean Median St.Dev. 95%CI\_low 95%CI\_upp

T[1] 0.11018893 0.09933529 0.05438688 3.632502e-02 0.2461734

T[2] 0.08774594 0.07848473 0.04504982 2.801341e-02 0.2017606

d[1] 0.00000000 0.00000000 0.00000000 0.000000e+00 0.0000000

d[2] -0.26068308 -0.26029670 0.04918766 -3.562744e-01 -0.1653182

dev[1, 1] 0.57608553 0.27031745 0.79628552 4.944460e-04 2.8307226

dev[2, 1] 1.09991934 0.58728925 1.38911701 1.398626e-03 4.9526990

dev[3, 1] 0.78138437 0.36908772 1.07647422 8.020610e-04 3.8799675

dev[4, 1] 0.62576944 0.29605185 0.87185086 6.197179e-04 3.1554430

dev[5, 1] 1.14545889 0.64469482 1.39506969 1.403633e-03 5.0630738

dev[6, 1] 0.70959509 0.31099369 1.02119509 5.939523e-04 3.6440874

dev[7, 1] 2.10196791 1.46800060 2.10955526 7.011110e-03 7.5163706

dev[8, 1] 0.86437134 0.45617602 1.08301125 1.068774e-03 3.7947101

dev[9, 1] 0.76508197 0.35597583 1.05065293 9.128923e-04 3.7840030

dev[10, 1] 0.82215318 0.37952165 1.13231128 7.810625e-04 4.0239592

dev[11, 1] 0.52555867 0.22690537 0.75673650 4.062692e-04 2.6899364

dev[12, 1] 0.97597631 0.49607930 1.26139070 1.223869e-03 4.4482384

dev[13, 1] 0.89681317 0.43398454 1.21385688 1.107147e-03 4.3624183

dev[14, 1] 3.83194901 3.29974276 2.76565053 1.345032e-01 10.4500366

dev[15, 1] 0.58398450 0.26274360 0.81926651 4.478508e-04 2.9914218

dev[16, 1] 0.65951879 0.30750319 0.92118277 6.428441e-04 3.2973368

dev[17, 1] 1.13953318 0.79453301 1.16038585 3.028262e-03 4.1906185

dev[18, 1] 0.98653145 0.52387856 1.24178595 1.246025e-03 4.4186256

dev[19, 1] 0.89738003 0.48399419 1.09821684 8.878065e-04 3.9540637

dev[20, 1] 0.56486539 0.26264764 0.77671730 4.430798e-04 2.7171709

dev[21, 1] 1.36660364 0.81470517 1.57541299 1.792908e-03 5.6077209

dev[22, 1] 1.35014329 0.81438430 1.55313012 2.512501e-03 5.5337133

dev[1, 2] 0.58102339 0.27350676 0.79265809 5.891409e-04 2.8414278

dev[2, 2] 0.97046328 0.55868978 1.15101506 1.385583e-03 4.0542278

dev[3, 2] 0.48948832 0.22689593 0.67338587 5.361567e-04 2.3567107

dev[4, 2] 0.53395859 0.23385616 0.77487366 4.766568e-04 2.7350805

dev[5, 2] 1.26414301 0.82291517 1.37554044 2.511738e-03 4.9722193

dev[6, 2] 0.54910470 0.25768664 0.76332710 5.562957e-04 2.7367546

dev[7, 2] 2.39053302 1.88235817 2.10703352 1.850632e-02 7.7320523

dev[8, 2] 0.97488139 0.50019026 1.24851317 9.234817e-04 4.5354718

dev[9, 2] 0.62585617 0.29429108 0.85449051 5.983840e-04 3.0179096

dev[10, 2] 0.75232321 0.35664979 1.02966384 8.228484e-04 3.6195897

dev[11, 2] 0.64478655 0.29651745 0.90105288 6.999680e-04 3.1849115

dev[12, 2] 0.99707309 0.55627663 1.20683990 1.405712e-03 4.2603660

dev[13, 2] 0.74443099 0.39223138 0.93259790 1.050247e-03 3.3772220

dev[14, 2] 4.25386954 3.73835475 2.87310646 2.888904e-01 11.2022825

dev[15, 2] 0.50861368 0.23010991 0.71550993 4.745827e-04 2.5463055

dev[16, 2] 0.61422400 0.29685553 0.83274972 6.016478e-04 2.9789344

dev[17, 2] 1.13252387 0.56746858 1.47431888 1.277224e-03 5.3063971

dev[18, 2] 1.17732079 0.71289926 1.36481918 1.781313e-03 4.9595852

dev[19, 2] 1.14481034 0.65375258 1.39038744 1.993099e-03 4.9763300

dev[20, 2] 0.48554162 0.22473522 0.68156574 4.761131e-04 2.4431690

dev[21, 2] 1.34909398 0.88010785 1.44240008 3.138385e-03 5.1822576

dev[22, 2] 1.34887916 0.89207097 1.43512619 3.349261e-03 5.2576455

rhat[1, 1] 3.39894824 3.23728911 1.31893424 1.300904e+00 6.3338758

rhat[2, 1] 11.82609662 11.64830053 2.47292267 7.461078e+00 17.0913809

rhat[3, 1] 10.05358540 9.86723776 2.36349762 5.905569e+00 15.0925680

rhat[4, 1] 127.80153512 127.63465487 8.53586473 1.114403e+02 144.8753428

rhat[5, 1] 31.18022105 31.03626678 4.04688358 2.368095e+01 39.6879567

rhat[6, 1] 5.26949460 5.15318030 1.56109334 2.552974e+00 8.6609756

rhat[7, 1] 138.83245727 138.69853748 8.59413508 1.227841e+02 156.0528211

rhat[8, 1] 52.27040390 52.16225119 4.80055832 4.325417e+01 61.7874670

rhat[9, 1] 34.72581142 34.61780564 4.14278565 2.698250e+01 43.2330173

rhat[10, 1] 182.43482237 182.38104119 10.10498924 1.631003e+02 202.5586906

rhat[11, 1] 53.18768631 53.02883669 4.94395699 4.390008e+01 63.2295336

rhat[12, 1] 51.25240404 51.05920541 4.83027054 4.224417e+01 61.0316341

rhat[13, 1] 14.10321308 13.96131159 2.73672558 9.149933e+00 19.8720908

rhat[14, 1] 58.00344502 57.86650021 5.63749707 4.726472e+01 69.2893375

rhat[15, 1] 30.25136141 30.11400340 3.62589405 2.343275e+01 37.8046931

rhat[16, 1] 39.85466588 39.71757779 4.29570836 3.172487e+01 48.6267969

rhat[17, 1] 15.20889399 15.13428649 2.24478857 1.101967e+01 19.8302773

rhat[18, 1] 7.92862420 7.73289360 2.07583397 4.335255e+00 12.4327127

rhat[19, 1] 4.45475073 4.27962517 1.45471793 2.071176e+00 7.7284846

rhat[20, 1] 40.57644117 40.44473806 4.29063931 3.254649e+01 49.2281981

rhat[21, 1] 37.85995400 37.75236743 4.31427685 2.985839e+01 46.6098441

rhat[22, 1] 34.12140972 33.93616053 4.29829882 2.634974e+01 43.3180549

rhat[1, 2] 2.61216843 2.48089886 1.03621595 9.804030e-01 4.9662393

rhat[2, 2] 9.17911362 9.03507904 1.96541495 5.727021e+00 13.3417056

rhat[3, 2] 5.90364102 5.78966440 1.43114687 3.427772e+00 8.9684323

rhat[4, 2] 101.28865089 101.14788189 7.04015439 8.776130e+01 115.5010770

rhat[5, 2] 23.84382224 23.66855760 3.17363342 1.806019e+01 30.5239400

rhat[6, 2] 4.72649686 4.61462597 1.43312759 2.252478e+00 7.9062119

rhat[7, 2] 111.46021971 111.35383997 7.25109555 9.777174e+01 126.0840236

rhat[8, 2] 55.46627372 55.32288029 5.21112460 4.553250e+01 65.9338403

rhat[9, 2] 27.15939845 27.05545560 3.36342376 2.093445e+01 34.0724946

rhat[10, 2] 143.36144459 143.19284262 8.55460434 1.268660e+02 160.4890615

rhat[11, 2] 62.68981334 62.45003488 5.92768241 5.178083e+01 74.8405255

rhat[12, 2] 40.87168671 40.75421066 4.05282685 3.336517e+01 49.1113942

rhat[13, 2] 10.91968192 10.77912212 2.15481261 7.063790e+00 15.5065936

rhat[14, 2] 44.10556340 43.96890430 4.42886321 3.577647e+01 53.1754911

rhat[15, 2] 25.65181780 25.52590384 3.25726046 1.961149e+01 32.3306057

rhat[16, 2] 31.20098816 31.05895263 3.52954695 2.466640e+01 38.5075961

rhat[17, 2] 24.83022639 24.69870909 3.72958918 1.789490e+01 32.5390671

rhat[18, 2] 6.06828148 5.91438750 1.61520615 3.286275e+00 9.5547604

rhat[19, 2] 4.49457493 4.31468621 1.47889804 2.077761e+00 7.7848885

rhat[20, 2] 31.33107004 31.24739864 3.49303496 2.478036e+01 38.3460522

rhat[21, 2] 32.11524591 31.93323216 3.77588537 2.503573e+01 39.9043339

rhat[22, 2] 26.84545591 26.62947407 3.45035154 2.063855e+01 34.2540582

totresdev 46.80358719 46.12146309 6.79454016 3.542817e+01 61.9064039

>

>

>

> #### Berechnung der CrI

>

> ## Berechnung CrI von T[1]

> T1\_1 <- quantile(mcmc.out$samples[["chain1"]][,"T[1]"] , c(0.025, 0.975))

> T1\_2 <- quantile(mcmc.out$samples[["chain2"]][,"T[1]"] , c(0.025, 0.975))

> T1\_3 <- quantile(mcmc.out$samples[["chain3"]][,"T[1]"] , c(0.025, 0.975))

>

> # CrI von T[1]

> (T1\_1 + T1\_2 + T1\_3)/3

2.5% 97.5%

0.03623797 0.24572485

>

>

> ## Berechnung CrI von T[2]

> T2\_1 <- quantile(mcmc.out$samples[["chain1"]][,"T[2]"] , c(0.025, 0.975))

> T2\_2 <- quantile(mcmc.out$samples[["chain2"]][,"T[2]"] , c(0.025, 0.975))

> T2\_3 <- quantile(mcmc.out$samples[["chain3"]][,"T[2]"] , c(0.025, 0.975))

>

> # CrI von T[2]

> (T2\_1 + T2\_2 + T2\_3)/3

2.5% 97.5%

0.02800949 0.20183345

>

>

> ## Berechnung CrI von d[2]

> d2\_1 <- quantile(mcmc.out$samples[["chain1"]][,"d[2]"] , c(0.025, 0.975))

> d2\_2 <- quantile(mcmc.out$samples[["chain2"]][,"d[2]"] , c(0.025, 0.975))

> d2\_3 <- quantile(mcmc.out$samples[["chain3"]][,"d[2]"] , c(0.025, 0.975))

>

> # CrI von d[2]

> (d2\_1 + d2\_2 + d2\_3)/3

2.5% 97.5%

-0.3562135 -0.1648239

>

>

>

>

>

> # Teil 3: Leverage Plot und nachträgliche Berechnung von pD -----------------

>

>

>

>

> out\_lePlo <- capture.output( mcmc.out$summary)

> cat("Hilf\_lePlo", out\_lePlo, file="Hilf2.txt", sep="\n", append=TRUE)

>

>

> Hilf\_data = read.table("Hilf2.txt", sep = "", header=F, skip=295, nrows=88)

> #Hilf\_data

>

>

>

> #### Berechnung w\_ik

>

> Hilf\_dev <- Hilf\_data[1:44,3]

> #Hilf\_dev

> 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

Hilf\_dev\_k1 Hilf\_dev\_k2 total w\_ik\_neg w\_ik

[1,] 0.5629973 0.5671882 0.5650928 -0.7517265 0.7517265

[2,] 1.0571300 0.9369906 0.9970603 -0.9985291 0.9985291

[3,] 0.7988474 0.4948522 0.6468498 -0.8042697 0.8042697

[4,] 0.6229676 0.5434619 0.5832148 -0.7636850 0.7636850

[5,] 1.1182374 1.2868184 1.2025279 -1.0965983 1.0965983

[6,] 0.6980478 0.5555913 0.6268196 -0.7917194 0.7917194

[7,] 2.1286033 2.3025744 2.2155888 -1.4884854 1.4884854

[8,] 0.8888509 0.9722691 0.9305600 -0.9646554 0.9646554

[9,] 0.7460399 0.6399451 0.6929925 -0.8324617 0.8324617

[10,] 0.8295918 0.7378540 0.7837229 -0.8852813 0.8852813

[11,] 0.5230622 0.6061656 0.5646139 -0.7514079 0.7514079

[12,] 0.9737305 1.0298571 1.0017938 -1.0008965 1.0008965

[13,] 0.9125317 0.7338516 0.8231917 -0.9072991 0.9072991

[14,] 3.7997680 4.2931090 4.0464385 -2.0115761 2.0115761

[15,] 0.5665541 0.4878831 0.5272186 -0.7260982 0.7260982

[16,] 0.6642520 0.6037301 0.6339911 -0.7962355 0.7962355

[17,] 1.1384909 1.1388446 1.1386678 -1.0670838 1.0670838

[18,] 0.9706464 1.1799571 1.0753018 -1.0369676 1.0369676

[19,] 0.9123877 1.1173352 1.0148615 -1.0074033 1.0074033

[20,] 0.5495752 0.4661360 0.5078556 -0.7126399 0.7126399

[21,] 1.3563631 1.3521765 1.3542698 -1.1637310 1.1637310

[22,] 1.3497230 1.3446559 1.3471895 -1.1606849 1.1606849

>

> # 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] 24.6341

>

>

>

> #### Berechnung leverage\_ik

>

> Hilf\_rhat <- Hilf\_data[45:88,3]

> #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] 0.5092425 0.6273493 0.7008575 0.6191187 0.4968497 0.5955668 0.6421239 0.4703897 0.5980591 0.6470458 0.4814427 0.5430389 0.6462739

[14] 0.4659986 0.5464842 0.5498134 0.3141347 0.4431432 0.3376158 0.5373189 0.6184362 0.6443490

>

>

> 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] 0.5080753 0.3367167 0.3375872 0.5357424 0.5285265 0.4220070 0.4603255 0.5887366 0.4283222 0.5207330 0.5868047 0.5297276 0.3727305

[14] 0.5963530 0.4658279 0.4883420 0.7016639 0.5825854 0.6697985 0.4511088 0.4112479 0.3815920

>

>

>

> #### Erzeugen leverage plot

> 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 fixed 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, grid=TRUE )

> 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)

>

>

>

> #### Plot of model

> #directed acyclic graph

> #durch igraph

> Model\_Nimble$plotGraph() # Anweisung geht nicht bei nimbleMCMC

>

>

>

>

>

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