> ########## ########## Simulation Blocker Beispiel mit Random 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),

+ sd = c( 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),

+ # zusätzlich noch einzufügen:

+ A=0,

+ delta=matrix(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

+ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), ncol=2))

>

>

>

> ### Create Model Code

> 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])) #Deviance contribution

+ + (n[i,k]-r[i,k]) \* (log(n[i,k]-r[i,k]) - log(n[i,k]-rhat[i,k])))

+ }

+ 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; Modifikation

+ }

+ }

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

+

+ 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 âstandardâ 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 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... 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.

>

> Model\_Nimble$initializeInfo()

Missing values (NAs) or non-finite values were found in model variables: lifted\_d1\_over\_sqrt\_oPtaud\_oBi\_comma\_k\_cB\_cP\_L12, md, taud, sw. This is not an error, but some or all variables may need to be initialized for certain algorithms to operate properly. For more information on model initialization, see help(modelInitialization).

>

>

>

> ##### 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", "sd"))

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...

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] 0.11037922 0.09951332 0.05454649 3.637580e-02 0.2462285

T[2] 0.08922794 0.07964635 0.04599611 2.832450e-02 0.2034609

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

d[2] -0.24499456 -0.24646456 0.06563467 -3.717845e-01 -0.1120949

dev[1, 1] 0.57136645 0.26297872 0.80548480 5.353780e-04 2.7880985

dev[2, 1] 1.06681610 0.53493532 1.38011880 1.101096e-03 4.9782226

dev[3, 1] 0.83760418 0.38988943 1.16859583 9.397848e-04 4.2341669

dev[4, 1] 0.80513916 0.34911344 1.19757835 6.934616e-04 4.1716744

dev[5, 1] 0.97263822 0.47706155 1.30758872 1.341126e-03 4.6817800

dev[6, 1] 0.74465858 0.34905498 1.01956888 7.395878e-04 3.5529968

dev[7, 1] 1.33224020 0.66719160 1.73245586 1.660600e-03 6.2339933

dev[8, 1] 0.80826916 0.38769609 1.13161526 8.091684e-04 4.0777013

dev[9, 1] 0.81656493 0.37633755 1.12972025 7.995424e-04 3.9844277

dev[10, 1] 0.88867132 0.40476417 1.24669532 9.203943e-04 4.5486768

dev[11, 1] 0.65540455 0.28153833 0.97719812 6.507686e-04 3.5238277

dev[12, 1] 0.88131317 0.41793076 1.20541156 7.716426e-04 4.3119055

dev[13, 1] 0.90655322 0.43099240 1.24247127 8.707556e-04 4.3251177

dev[14, 1] 2.20768066 1.47389874 2.33474222 4.325095e-03 8.3516082

dev[15, 1] 0.66147159 0.31294045 0.90041246 7.060420e-04 3.2678893

dev[16, 1] 0.70103781 0.32466206 0.97382742 7.897610e-04 3.4190863

dev[17, 1] 0.96548216 0.55084638 1.16335150 1.132437e-03 4.2053877

dev[18, 1] 0.88970320 0.45269662 1.15213843 8.569422e-04 4.2168863

dev[19, 1] 0.86080289 0.45266370 1.11095571 1.092678e-03 3.9886113

dev[20, 1] 0.67544646 0.30211645 0.97391556 6.130173e-04 3.4026463

dev[21, 1] 1.19570601 0.63509452 1.51469012 1.304126e-03 5.3057489

dev[22, 1] 1.20046329 0.60183167 1.54820332 1.152992e-03 5.5042155

dev[1, 2] 0.59624600 0.28465195 0.81552510 7.722011e-04 2.9333906

dev[2, 2] 0.93263235 0.50711071 1.15323751 1.212033e-03 4.1279549

dev[3, 2] 0.53621947 0.23953356 0.77040621 5.423122e-04 2.7121957

dev[4, 2] 0.75448636 0.34523767 1.06903713 7.880081e-04 3.7607167

dev[5, 2] 1.06960406 0.58222502 1.32425695 1.590193e-03 4.7577605

dev[6, 2] 0.57732389 0.26308789 0.81290395 6.220023e-04 2.8938490

dev[7, 2] 1.36540435 0.73433627 1.69991722 1.573603e-03 6.0386602

dev[8, 2] 0.90977039 0.42716388 1.23608172 9.171834e-04 4.4181142

dev[9, 2] 0.69552808 0.31803312 0.98401742 7.383233e-04 3.4439807

dev[10, 2] 0.84322907 0.38498163 1.18283751 9.006795e-04 4.2099865

dev[11, 2] 0.74605595 0.34153333 1.05314027 9.510717e-04 3.7250912

dev[12, 2] 0.91019419 0.44305902 1.20183419 1.198146e-03 4.2951487

dev[13, 2] 0.75731793 0.37160967 1.00584999 7.898722e-04 3.5835734

dev[14, 2] 2.43682056 1.70075442 2.46105920 6.184637e-03 8.8348411

dev[15, 2] 0.60282290 0.27425353 0.85044714 5.706028e-04 3.0032097

dev[16, 2] 0.68068537 0.31177305 0.96163380 5.891365e-04 3.3917357

dev[17, 2] 1.05963740 0.50951714 1.40821407 1.271566e-03 5.0758700

dev[18, 2] 1.11366020 0.63082727 1.34881090 1.527737e-03 4.8693294

dev[19, 2] 1.11111875 0.60474091 1.37642834 1.385477e-03 4.8604268

dev[20, 2] 0.60168371 0.27132200 0.86608278 5.673635e-04 3.0969877

dev[21, 2] 1.15286379 0.63454290 1.41518697 1.580283e-03 5.1238961

dev[22, 2] 1.16470803 0.66830576 1.39387394 1.555684e-03 5.0065076

rhat[1, 1] 3.33851289 3.18317322 1.31474373 1.269092e+00 6.2832213

rhat[2, 1] 11.97432862 11.82863017 2.54071817 7.446166e+00 17.3411157

rhat[3, 1] 10.03644422 9.83685178 2.45341680 5.766459e+00 15.3893982

rhat[4, 1] 126.89141748 126.64169386 9.66866690 1.087083e+02 146.6207428

rhat[5, 1] 30.11380465 29.94655846 4.30003299 2.220822e+01 39.0821648

rhat[6, 1] 5.26605796 5.11922635 1.61319731 2.596830e+00 8.8254271

rhat[7, 1] 144.76237873 144.56128430 10.34865414 1.252975e+02 165.8951314

rhat[8, 1] 50.59431715 50.49411632 5.55420598 4.002996e+01 62.0714562

rhat[9, 1] 35.07396234 34.84185700 4.52444856 2.685427e+01 44.5266579

rhat[10, 1] 185.02525675 184.81916822 11.74409035 1.627496e+02 208.9503740

rhat[11, 1] 52.51993830 52.39499648 5.59248282 4.204658e+01 64.2328536

rhat[12, 1] 49.79318524 49.72581576 5.36991527 3.969465e+01 60.7235560

rhat[13, 1] 14.27954840 14.09995287 2.89829364 9.190568e+00 20.4499224

rhat[14, 1] 53.41248989 53.26894926 6.47751284 4.116369e+01 66.4125397

rhat[15, 1] 30.25164064 30.09513385 3.87048380 2.315178e+01 38.1203125

rhat[16, 1] 39.26895024 39.16440704 4.60493112 3.050602e+01 48.5800654

rhat[17, 1] 14.56653822 14.45916616 2.51104097 9.949038e+00 19.8459284

rhat[18, 1] 7.70395542 7.52460020 2.04718480 4.165906e+00 12.2556797

rhat[19, 1] 4.36131866 4.19987148 1.48454889 1.985969e+00 7.7467707

rhat[20, 1] 40.19571179 40.11099584 4.68886411 3.119789e+01 49.6300723

rhat[21, 1] 38.98135765 38.70562936 4.87295714 3.017870e+01 49.0048739

rhat[22, 1] 35.20880635 35.00081754 4.87090443 2.642205e+01 45.1639814

rhat[1, 2] 2.63260461 2.48547033 1.07063596 9.637489e-01 5.1051610

rhat[2, 2] 9.02717982 8.86811147 2.03305594 5.479451e+00 13.4134392

rhat[3, 2] 5.88497267 5.75726866 1.54144305 3.290846e+00 9.2896625

rhat[4, 2] 102.01947076 101.76469129 8.47799453 8.614513e+01 119.1954165

rhat[5, 2] 24.95243419 24.68295679 3.79368337 1.826763e+01 33.2276471

rhat[6, 2] 4.71065462 4.57023502 1.49549495 2.247371e+00 8.0503896

rhat[7, 2] 105.30453324 105.24793130 8.80039803 8.817038e+01 122.5742541

rhat[8, 2] 57.30069770 57.03970587 6.21843148 4.597677e+01 70.2037779

rhat[9, 2] 26.83865142 26.70538723 3.76869061 1.992066e+01 34.5770015

rhat[10, 2] 140.94696215 140.83174519 10.15175657 1.215500e+02 161.2507010

rhat[11, 2] 63.50070903 63.17145426 6.58988965 5.128709e+01 77.0484768

rhat[12, 2] 42.20960720 41.97956956 4.83329679 3.342575e+01 52.1836137

rhat[13, 2] 10.76265150 10.60561890 2.31550164 6.695418e+00 15.7376256

rhat[14, 2] 48.53001765 48.04990045 6.15065964 3.784031e+01 61.8939524

rhat[15, 2] 25.67414304 25.55803640 3.55560868 1.915895e+01 33.0593151

rhat[16, 2] 31.86746210 31.68505544 4.07687727 2.452335e+01 40.3786522

rhat[17, 2] 25.41392046 25.21294498 4.01983932 1.812268e+01 33.8464908

rhat[18, 2] 6.24370042 6.08266772 1.72804049 3.317296e+00 10.0758573

rhat[19, 2] 4.62728048 4.43517234 1.58928918 2.107870e+00 8.2381561

rhat[20, 2] 31.63394029 31.50299125 3.97315288 2.424476e+01 39.8379972

rhat[21, 2] 30.98275813 30.86440222 4.27985307 2.290544e+01 39.8072928

rhat[22, 2] 25.83338337 25.71698323 3.93339058 1.844924e+01 33.9108991

sd 0.14956655 0.14153351 0.07749420 3.301264e-02 0.3255120

totresdev 41.26304608 40.82515979 7.99802767 2.678426e+01 58.3263395

>

>

>

> #### 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%

0.03638067 0.24540365

>

>

> ## 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%

0.0284283 0.2043957

>

>

> ## 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%

-0.3716472 -0.1119456

>

>

> ## Berechnung CrI von sd

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

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

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

>

> # CrI von sd

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

2.5% 97.5%

0.03357339 0.32609913

>

>

>

>

>

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

>

>

>

>

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

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

>

>

> Hilf\_data = read.table("Hilf.txt", sep = "", header=F, skip=298, 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,] 0.5713664 0.5962460 0.5838062 -0.7640721 0.7640721

[2,] 1.0668161 0.9326323 0.9997242 -0.9998621 0.9998621

[3,] 0.8376042 0.5362195 0.6869118 -0.8288014 0.8288014

[4,] 0.8051392 0.7544864 0.7798128 -0.8830701 0.8830701

[5,] 0.9726382 1.0696041 1.0211211 -1.0105054 1.0105054

[6,] 0.7446586 0.5773239 0.6609912 -0.8130137 0.8130137

[7,] 1.3322402 1.3654043 1.3488223 -1.1613881 1.1613881

[8,] 0.8082692 0.9097704 0.8590198 -0.9268332 0.9268332

[9,] 0.8165649 0.6955281 0.7560465 -0.8695093 0.8695093

[10,] 0.8886713 0.8432291 0.8659502 -0.9305644 0.9305644

[11,] 0.6554045 0.7460560 0.7007302 -0.8370963 0.8370963

[12,] 0.8813132 0.9101942 0.8957537 -0.9464426 0.9464426

[13,] 0.9065532 0.7573179 0.8319356 -0.9121050 0.9121050

[14,] 2.2076807 2.4368206 2.3222506 -1.5238932 1.5238932

[15,] 0.6614716 0.6028229 0.6321472 -0.7950769 0.7950769

[16,] 0.7010378 0.6806854 0.6908616 -0.8311808 0.8311808

[17,] 0.9654822 1.0596374 1.0125598 -1.0062603 1.0062603

[18,] 0.8897032 1.1136602 1.0016817 -1.0008405 1.0008405

[19,] 0.8608029 1.1111187 0.9859608 -0.9929556 0.9929556

[20,] 0.6754465 0.6016837 0.6385651 -0.7991027 0.7991027

[21,] 1.1957060 1.1528638 1.1742849 -1.0836443 1.0836443

[22,] 1.2004633 1.1647080 1.1825857 -1.0874675 1.0874675

>

> # 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] 19.34817

>

>

>

> #### 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] 0.5326166 0.7022717 0.7366888 0.8050378 0.6100944 0.6352267 0.9101129 0.6569302 0.6976214 0.8359988 0.6497307 0.6856817 0.6964822

[14] 0.7220713 0.6383023 0.6503375 0.4232630 0.4617155 0.3688933 0.6742767 0.7450698 0.7836031

>

>

> 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] 0.5433755 0.3999925 0.3835648 0.7544824 0.6834530 0.4550126 0.7832211 0.7718653 0.5531600 0.7762892 0.7418324 0.6942475 0.4403839

[14] 0.9488114 0.5814291 0.6335649 0.7752197 0.6389675 0.7279204 0.5967080 0.5735248 0.5430274

>

>

>

> #### 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 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, 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 ########## ########## ##########