Aufgabe 1

Im folgenden benutzen wir den Datensatz `cats` aus dem `MASS`-Paket.

```
data(cats, package="MASS")
summary(cats)
```

Eine Beschreibung des Datensatzes erhalten Sie mit dem Befehl

```
?MASS::cats
```

Wir interessieren uns für folgende Modelle:

1. $Hwt_i = \alpha + \beta \cdot Bwt_i + \gamma \cdot Sex_i + \epsilon_i$
2. $Hwt_i = \alpha + \beta \cdot \sqrt{Bwt_i} + \gamma \cdot Sex_i + \epsilon_i$
3. $Hwt_i = \alpha + \beta \cdot Bwt_i + \gamma \cdot Sex_i + \delta \cdot Bwt_i \cdot Sex_i + \epsilon_i$

(a) Welche Priori-Annahmen sind sinnvoll?

(b) Schätzen Sie die Modelle unter geeigneten Priori-Annahmen.

(c) Stellen Sie mittels Konvergenzdiagnostik sicher, dass die Ziehungen wirklich aus der Posteriori kommen.

(d) Führen Sie eine Sensitivitätsanalyse durch, in dem Sie die Priori-Annahmen verändern. Verändern sich die Ergebnisse?

(e) Vergleichen Sie die Modelle mittels DIC, welches Modell passt am Besten zu den Daten?

```
library(MASS)
data(cats)
help(cats)
```

```
## starting httpd help server ... done
```

Daten als Liste für JAGS vorbereiten:

```
daten<-as.list(data.frame(hwt=cats$Hwt, bwt=cats$Bwt, sex=ifelse(cats$Sex=="F",0,1)))
daten$n=length(cats$Sex)
```

Die drei Modelle im JAGS-Format. Idee der Prioris: leicht informativ, um Konvergenz zu sichern. Kovariableneffekte normal (wie lineares Modell) mit geringer Priori-Präzision (hohe Varianz). Präszision des Fehler Gamma-Verteilt (wie bei Normalverteilung) mit hoher Priori-Varianz (hier: 1000). Der Priori-Erwartungswert der Fehler-Varianz dabei relativ hoch (1000), damit Daten durch Modell und nicht nur durch Fehler erklärt werden.

```
model_1 <- "model{
    for (i in 1:n) {
        hwt[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta*bwt[i] + gamma*sex[i]
    }
    ## Prioris
    tau ~ dgamma(1, .001)
```

```
    sigma2 <- 1/tau
    alpha ~ dnorm(0, .001)
    beta ~ dnorm(0,  .001)
    gamma ~ dnorm(0, .001)

}"

model_2 <- "model{
    for (i in 1:n) {
        hwt[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta*bwt[i] + gamma*sex[i]
    }
     ## Prioris
    tau ~ dgamma(1, .001)
    sigma2 <- 1/tau
    alpha ~ dnorm(0, .001)
    beta ~ dnorm(0,  .001)
    gamma ~ dnorm(0, .001)
}"

model_3 <- "model{
    for (i in 1:n) {
        hwt[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta*bwt[i] + gamma*sex[i] + delta*sex[i]*bwt[i]
    }
    ## Prioris
    tau ~ dgamma(1, 1)
    sigma2 <- 1/tau
    alpha ~ dnorm(0, .1)
    beta ~ dnorm(0, .1)
    gamma ~ dnorm(0, .1)
    delta ~ dnorm(0, .1)
}"
```

Alle drei Modelle kompilieren und burn-in von 10000. Wir benutzen hier zwei MCMC-Ketten; wir erhalten damit zwei Trace Plots und können die Konvergenz besser einschätzen.

```
library(rjags)
```

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

```
## Linked to JAGS 4.3.1
```

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

```
model_1 <- jags.model(file = textConnection(model_1),
                      data = daten,
                      n.chains = 2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 144
##    Unobserved stochastic nodes: 4
##    Total graph size: 493
##
## Initializing model
```

```
model_2 <- jags.model(file = textConnection(model_2),
                      data = daten,
                      n.chains = 2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 144
##    Unobserved stochastic nodes: 4
##    Total graph size: 493
##
## Initializing model
```

```
model_3 <- jags.model(file = textConnection(model_3),
                      data = daten,
                      n.chains = 2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 144
##    Unobserved stochastic nodes: 5
##    Total graph size: 524
##
## Initializing model
```

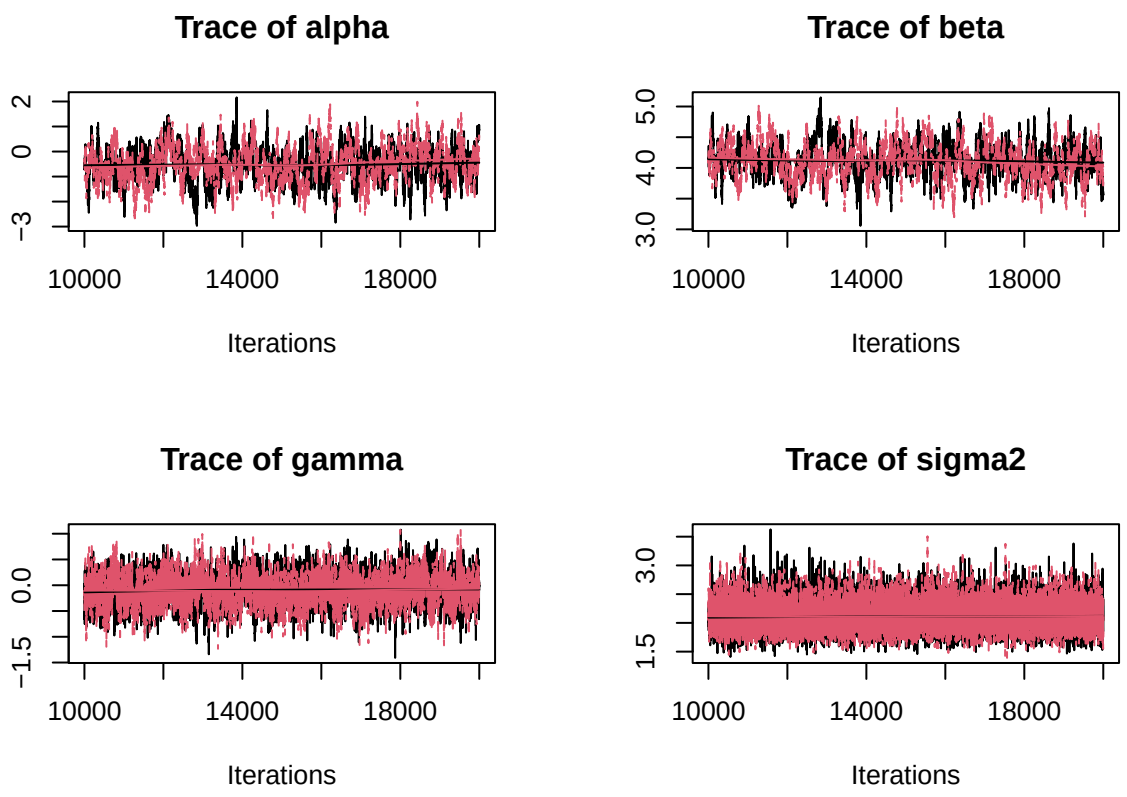
```
update(model_1, 10000)
update(model_2, 10000)
update(model_3, 10000)
```

Ziehen aus den drei Modellen.

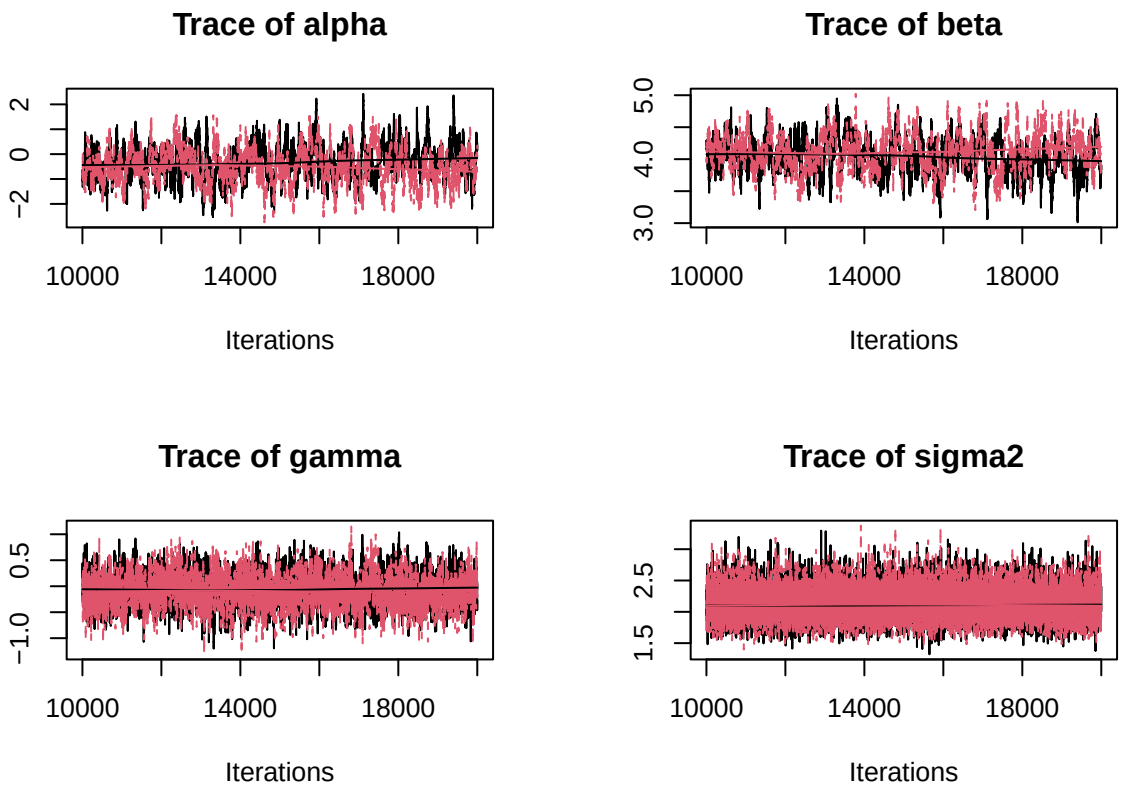
```
library(coda)
samples_1 = coda.samples(model_1, c("alpha", "beta", "gamma", "sigma2"), 10000, 1)
samples_2 = coda.samples(model_2, c("alpha", "beta", "gamma", "sigma2"), 10000, 1)
samples_3 = coda.samples(model_3, c("alpha", "beta", "gamma", "sigma2", "delta"), 10000, 1)
```

Visuelle Inspektion der Ziehungen mit Trace Plots.

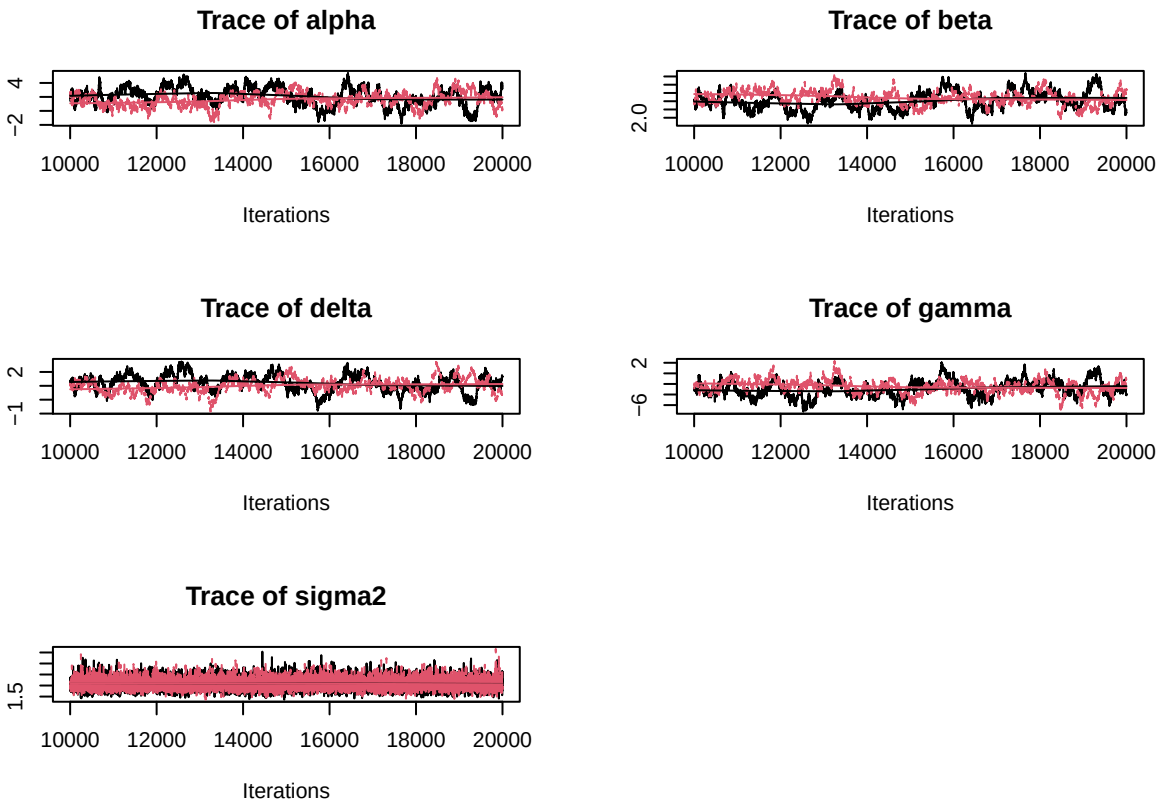
```
plot(samples_1, density=FALSE)
```




```
plot(samples_2, density=FALSE)
```



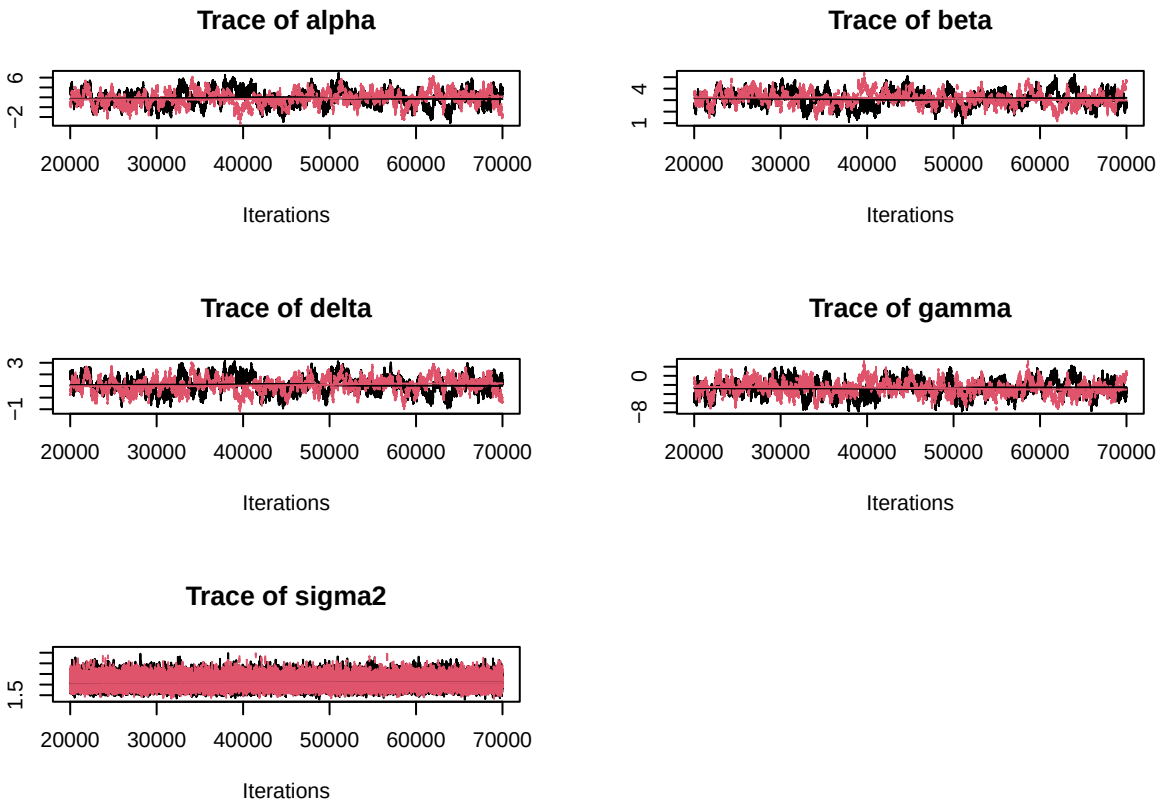

```
plot(samples_3, density=FALSE)
```



Die Trace Plots sind bei Modell 3 noch etwas unstabil, daher lieber mehr Iterationen.

Fehlt noch: Konvergenzdiagnostik

```
samples_3 = coda.samples(model_3, c("alpha", "beta", "gamma", "sigma2", "delta"), 50000, 1)
plot(samples_3, density=FALSE)
```



Posteriori-Schätzer:

```
summary(samples_1)
```

```
##
## Iterations = 10001:20000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean     SD Naive SE Time-series SE
## alpha  -0.50868 0.7150 0.005056       0.041456
## beta    4.11345 0.2899 0.002050       0.018192
## gamma  -0.09723 0.3069 0.002170       0.007534
## sigma2  2.12471 0.2571 0.001818       0.001876
##
## 2. Quantiles for each variable:
##
##           2.5%     25%      50%      75%  97.5%
```

```
## alpha  -1.9123 -0.9947 -0.50632 -0.02263 0.8847
## beta    3.5431  3.9187  4.11044  4.31248 4.6809
## gamma  -0.6926 -0.3075 -0.09749  0.11269 0.5032
## sigma2  1.6826  1.9430  2.10579  2.28391 2.6871
```

Geschlecht hat keinen relevanten Einfluß, Kredibilitätsintervall von γ enthält die Null. Körpergewicht hat positiven linearen Einfluß auf das Gewicht des Herzens.

```
summary(samples_2)
```

```
##
## Iterations = 10001:20000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean     SD Naive SE Time-series SE
## alpha  -0.42265 0.7135 0.005045       0.041659
## beta    4.07845 0.2894 0.002046       0.017966
## gamma  -0.08107 0.3054 0.002160       0.008346
## sigma2  2.12231 0.2536 0.001793       0.001793
##
## 2. Quantiles for each variable:
##
##           2.5%     25%      50%    75%  97.5%
## alpha  -1.8121 -0.9052 -0.42764 0.0637 0.9964
## beta    3.5079  3.8802  4.07968 4.2735 4.6445
## gamma  -0.6739 -0.2862 -0.08385 0.1223 0.5230
## sigma2  1.6821  1.9453  2.10282 2.2787 2.6768
```

Geschlecht hat keinen relevanten Einfluß, Kredibilitätsintervall von γ enthält die Null. Wurzel des Körpergewichts hat positiven linearen Einfluß auf das Gewicht des Herzens. Varianz des Fehlers praktisch identisch zu Modell 1.

```
summary(samples_3)
```

```
##
## Iterations = 20001:70000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
```

```
##    plus standard error of the mean:
##
##          Mean     SD  Naive SE Time-series SE
## alpha   1.781 1.5330 0.0048478       0.116888
## beta    3.137 0.6477 0.0020483       0.048872
## delta   1.086 0.6846 0.0021648       0.049101
## gamma  -2.698 1.6843 0.0053262       0.114200
## sigma2  2.094 0.2526 0.0007989       0.001017
##
## 2. Quantiles for each variable:
##
##           2.5%     25%    50%    75%  97.5%
## alpha  -1.1359  0.7027  1.782  2.868 4.6975
## beta    1.9077  2.6799  3.136  3.593 4.3692
## delta  -0.2239  0.6040  1.091  1.566 2.4090
## gamma  -5.9646 -3.8723 -2.709 -1.515 0.5391
## sigma2  1.6597  1.9160  2.074  2.250 2.6438
```

Wie in Modell 1: Körpergewicht hat positiven linearen Einfluß auf das Gewicht des Herzens. Das Geschlecht hat hier deutlichen Einfluß auf das Gewicht des Herzens, ist bei männlichen Katzen des Körpergewichts (Interaktion) um 2.758 Gramm niedriger (Posteriori-Erwartungswert) als bei weiblichen. Linearer Einfluß des Körpergewicht ist bei männlichen Katzen jedoch nochmals deutlich höher.

Sensitivitätsanalyse am Beispiel Modell 1

1b: Mehr Information auf Präzision des Fehlers, Erwartungswert niedriger:

```
model_1b <- "model{
    for (i in 1:n) {
        hwt[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta*bwt[i] + gamma*sex[i]
    }
    ## Prioris
    tau ~ dgamma(1, 1)
    sigma2 <- 1/tau
    alpha ~ dnorm(0, .001)
    beta ~ dnorm(0,  .001)
    gamma ~ dnorm(0, .001)
}"
```

1c: Flachere Prioris (weniger Information)

```
model_1c <- "model{
    for (i in 1:n) {
        hwt[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta*bwt[i] + gamma*sex[i]
    }
```

```
    ## Prioris
    tau ~ dgamma(.0001, .000001)
    sigma2 <- 1/tau
    alpha ~ dnorm(0, .00001)
    beta ~ dnorm(0, .00001)
    gamma ~ dnorm(0, .00001)
}"
```

1d: Informativere Prioris

```
model_1d <- "model{
    for (i in 1:n) {
        hwt[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta*bwt[i] + gamma*sex[i]
    }
    ## Prioris
    tau ~ dgamma(10, 10)
    sigma2 <- 1/tau
    alpha ~ dnorm(0, 1)
    beta ~ dnorm(0,  1)
    gamma ~ dnorm(0, 1)
}"

model_1b <- jags.model(file = textConnection(model_1b), data = daten)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 144
##    Unobserved stochastic nodes: 4
##    Total graph size: 493
##
## Initializing model
```

```
update(model_1b, 10000)
model_1c <- jags.model(file = textConnection(model_1c), data = daten)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 144
##    Unobserved stochastic nodes: 4
##    Total graph size: 495
##
## Initializing model
```

```
update(model_1c, 10000)
model_1d <- jags.model(file = textConnection(model_1d), data = daten)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 144
##    Unobserved stochastic nodes: 4
##    Total graph size: 493
##
## Initializing model
```

```
update(model_1d, 10000)

samples_1b = coda.samples(model_1b, c("alpha", "beta", "gamma", "sigma2"), 20000, 1)
samples_1c = coda.samples(model_1c, c("alpha", "beta", "gamma", "sigma2"), 20000, 1)
samples_1d = coda.samples(model_1d, c("alpha", "beta", "gamma", "sigma2"), 20000, 1)

summary(samples_1)
```

```
##
## Iterations = 10001:20000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean     SD Naive SE Time-series SE
## alpha  -0.50868 0.7150 0.005056       0.041456
## beta    4.11345 0.2899 0.002050       0.018192
## gamma  -0.09723 0.3069 0.002170       0.007534
## sigma2  2.12471 0.2571 0.001818       0.001876
##
## 2. Quantiles for each variable:
##
##           2.5%     25%      50%      75%  97.5%
## alpha  -1.9123 -0.9947 -0.50632 -0.02263 0.8847
## beta    3.5431  3.9187  4.11044  4.31248 4.6809
## gamma  -0.6926 -0.3075 -0.09749  0.11269 0.5032
## sigma2  1.6826  1.9430  2.10579  2.28391 2.6871
```

```
summary(samples_1b)
```

```
##
## Iterations = 10001:30000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean     SD Naive SE Time-series SE
## alpha  -0.39839 0.7229 0.005112       0.043605
## beta    4.06951 0.2943 0.002081       0.018254
## gamma  -0.08078 0.3101 0.002193       0.008274
## sigma2  2.13742 0.2585 0.001828       0.002018
##
## 2. Quantiles for each variable:
##
##           2.5%     25%      50%     75%  97.5%
## alpha  -1.7829 -0.8907 -0.41173 0.08655 1.0445
## beta    3.4880  3.8742  4.07022 4.26771 4.6344
## gamma  -0.6873 -0.2881 -0.08361 0.12659 0.5388
## sigma2  1.6886  1.9553  2.11619 2.29587 2.7102
```

```
summary(samples_1c)
```

```
##
## Iterations = 10001:30000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean     SD Naive SE Time-series SE
## alpha  -0.45273 0.7410 0.005240       0.046505
## beta    4.09142 0.3010 0.002128       0.019564
## gamma  -0.08885 0.3103 0.002194       0.008543
## sigma2  2.15409 0.2614 0.001848       0.001912
##
## 2. Quantiles for each variable:
##
##           2.5%     25%     50%     75%  97.5%
## alpha  -1.9624 -0.9522 -0.4347 0.05139 0.9503
```

```
## beta    3.5206  3.8817  4.0861 4.29489 4.6985
## gamma  -0.6982 -0.2970 -0.0897 0.12072 0.5255
## sigma2  1.7003  1.9701  2.1364 2.31521 2.7325
```

```
summary(samples_1d)
```

```
##
## Iterations = 10001:30000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean     SD Naive SE Time-series SE
## alpha  0.17120 0.5876 0.004155       0.028786
## beta   3.82548 0.2430 0.001719       0.012763
## gamma  0.05545 0.2842 0.002010       0.006891
## sigma2 2.01289 0.2272 0.001606       0.001764
##
## 2. Quantiles for each variable:
##
##           2.5%     25%     50%    75%  97.5%
## alpha  -1.0148 -0.2302 0.18693 0.5742 1.2989
## beta    3.3583  3.6610 3.81862 3.9910 4.3109
## gamma  -0.5067 -0.1338 0.05499 0.2470 0.6064
## sigma2  1.6159  1.8540 1.99620 2.1533 2.5079
```

Auf die Interpretation der Ergebnisse haben die veränderten Prioris keine Auswirkung.

Ansonsten sind die Auswirkungen wie zu erwarten: Niederiger Priori-Erwartungswert der Fehlerpräzision ändert auch den Posteriori-Erwartungswert der Fehlerpräzision. Allgemein mehr Priori-Information ändert die Posteriori-Standardabweichungen der Parameter, und damit die Unsicherheit über die Parameter. Insgesamt wären hier weniger informative Prioris vorzuziehen (also Modell 1c).

Modellvergleich

Allgemein: Modell 3 zeigt hier relevanten Einfluß des Geschlechts, ist deshalb gegenüber Modell 1 vorzuziehen.

Quantifizieren lässt sich der Modell-Vergleich z.B. mit DIC:

```
dic.samples(model=model_1, n.iter=20000)
```

```
## Mean deviance:  518
## penalty 3.932
## Penalized deviance: 522
```

```
dic.samples(model=model_2, n.iter=20000)
```

```
## Mean deviance:  518
## penalty 3.975
## Penalized deviance: 521.9
```

```
dic.samples(model=model_3, n.iter=20000)
```

```
## Mean deviance:  515
## penalty 4.466
## Penalized deviance: 519.5
```

Die "mean deviance" ist bei Modell 3 am geringsten, damit ist hier die Anpassung am besten – Modell 1 und 2 sind beide gleich. Das die Anpassung besser ist, ist auch dadurch begründet, dass die Anzahl der Parameter bei Modell 3 größer ist. Hierfür gibt es eine "penalty", die in etwa einer (geschätzen) Anzahl freier Parameter im Modell entspricht. Auch die Summe "Penalized deviance" (= mean deviance + penalty) ist bei Modell 3 am geringsten, damit ist Modell 3 den anderen beiden vorzuziehen.