

Einführung in die Bayes Statistik- Slides Begleitung zum Online Kurs

Regression und Anwendung

Martje Rave

Sommersemester 2025

ACHTUNG!

Diese Folien sind **NUR** Kurs begleitend und können noch viele Fehler beinhalten. Für die Klausurvorbereitung benutzt bitte den Online-Kurs und die Übungsblätter.
Bitte gebt mir Bescheid, wenn ihr Fehler findet.

- MCMC-Algorithmen erzeugen Markovketten aus der Posterior-Verteilung.
- Probleme:
 - Der Algorithmus muss konvergieren (stationäre Verteilung).
 - Ziehungen sind abhängig, was die Effizienz reduziert.
 - Effizienz hängt vom gewählten Algorithmus ab (z.B. Metropolis-Hastings vs. Gibbs).

Beispiel: Poisson-Lognormal Modell

Daten: Anzahl von Kaiserschnitten in 24 Krankenhäusern.

Modell:

$$y_j \sim \text{Po}(\lambda_j) \quad (1)$$

$$\log(\lambda_j) \sim \mathcal{N}(\mu, 1) \quad (2)$$

$$\mu \sim \mathcal{N}(0, 1000) \quad (3)$$

- Unbekannte Parameter: μ und τ
- Ziehung via Metropolis-Hastings:
 - Für μ : Random-Walk-Proposal
 - Für τ : Ad-hoc Normalapproximation

R Code: Funktion zur MCMC-Simulation (Teil 1)

```
poisson.lognormal.mcmc <- function(sumx, n, tau = 1000, l = 1000, ..  
  mu <- rep(NA, l)  
  lambda <- rep(NA, l)  
  mu[1] <- 1  
  lambda[1] <- sumx / n
```

```
log.fc.mu <- function(mu, lambda, sumx, tau) {  
  return(n * log(lambda) - 0.5 * mu^2 * 0.5 * mu^2 / tau)  
}
```

```
log.fc.lambda <- function(lambda, mu, sumx, n) {  
  return((sumx - n) * log(lambda) - n * lambda - 0.5 * log(lambda)^2)  
}
```

R Code: MCMC-Ziehung (Teil 2)

```
akzeptanz <- 0
i <- 1
while (i < l) {
  # Ziehe mu via Random Walk
  mustern <- rnorm(1, mu[i], rw.sd)
  ...
  if (runif(1) < alpha) {
    mu[i+1] <- mustern
  }

  # Ziehe lambda via ad hoc Normalapproximation
  lambdastern <- rnorm(...)
  ...
  if (runif(1) < alpha) {
    lambda[i+1] <- lambdastern
  }
}
```

```
if (i == tuning && do.tuning) {  
  ...  
  message(paste("Akzeptanzrate Tuning:", round(ak.rate * 100, 1)))  
  message(paste("Neuer Varianzparameter:", round(rw.sd, 3)))  
}  
lambda <- lambda[-1]  
mu <- mu[-1]  
ak.rate <- akzeptanz / i  
return(coda::mcmc(cbind(mu, lambda)))
```

```
n <- dim(data)[1]
sumx <- sum(data$n)
mcmc.simple <- poisson.lognormal.mcmc(sumx, n)
```

coda stellt verschiedene Funktionen zur Analyse von MCMC-Ziehungen bereit.

In unserem Beispiel:

- Ziehungen wurden als `coda::mcmc`-Objekt gespeichert.
- Analyse erfolgt über `summary()` und `plot()`.

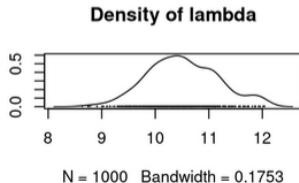
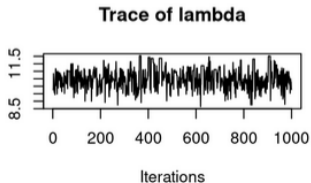
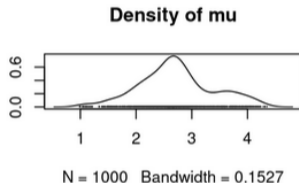
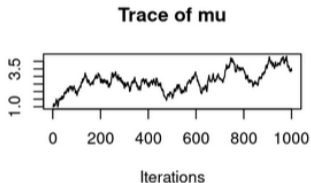
Zusammenfassung der MCMC-Ziehungen

`summary(mcmc.simple)`

- Iterationen: 1:1000
- Kette: 1
- Empirisches Mittel:
 - μ : 2.72 (SD = 0.6674, Time-series SE = 0.2455)
 - λ : 10.52 (SD = 0.6585, Time-series SE = 0.0411)
- Quantile:
 - μ : [2.25, 2.38, 2.67, 3.07, 4.08]
 - λ : [9.31, 10.06, 10.27, 10.99, 11.87]

Trace- und Dichteplots

`plot(mcmc.simple)`



- **lambda:** gut gemischt – deckt schnell den Posteriorbereich ab.
- **mu:** schlechte Mischung – Kette wandert langsam durch den Wertebereich.

Fazit: Der Algorithmus ist für μ ineffizient und braucht evtl. eine bessere Proposalverteilung.

- Für μ verwenden wir einen **Random-Walk-Proposal**:

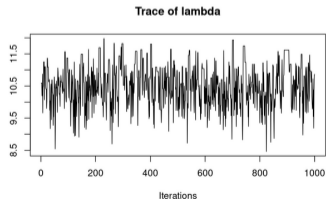
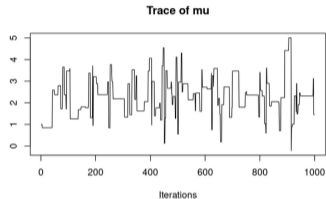
$$\mu^* = \mu^{(i-1)} + u, \quad u \sim \mathcal{N}(0, c)$$

- Die Varianz c beeinflusst das Mischverhalten des Samplers.

- Maß für die Effizienz des Proposals: Anteil akzeptierter Vorschläge.
- **Zu niedrig:** Kette bleibt oft im selben Zustand \Rightarrow schlecht.
- **Zu hoch:** Kette bewegt sich kaum \Rightarrow schlecht.
- **Zielbereich (Random-Walk):** Akzeptanzrate zwischen 30–60%.

Beispiel: Zu niedrige Akzeptanzrate

- Beispiel mit **nur 12%** Akzeptanzrate (zu niedrig).
- Ursache: Zu hohe Varianz im Proposal.



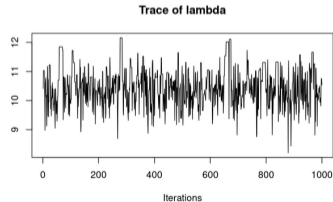
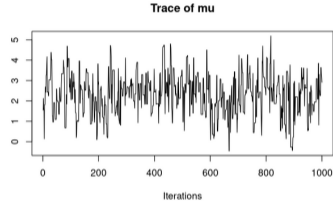
Wie wird getunt?

- Algorithmus läuft für z.B. 50 Iterationen.
- Akzeptanzrate wird berechnet.
- **Zu hoch:** Random-Walk-Varianz erhöhen.
- **Zu niedrig:** Random-Walk-Varianz senken.
- Neue Iteration starten mit aktualisierter Varianz.

- Auch andere Proposal-Verteilungen können getunt werden.
- Tuning und Burn-in überlagern sich:
 - Startet Algorithmus erneut, behält aber aktuelle Werte als Startwerte.
- Ziel: Effizientere Exploration des Posteriorraums.

Traceplots nach Tuning

- λ Die Kette zeigt nun ein gutes Mischungsverhalten ("good mixing").



- Zeigt Konvergenz des Mittelwerts bis zur jeweiligen Iteration.
- Nach Gesetz der großen Zahlen konvergiert der Mittelwert gegen Erwartungswert.
- Visualisierung manchmal trügerisch (z.B. Nachkommastellen).

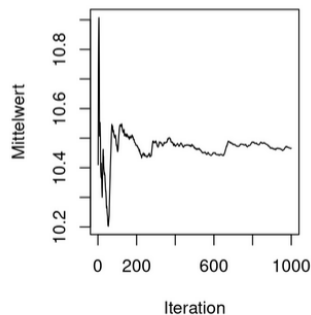
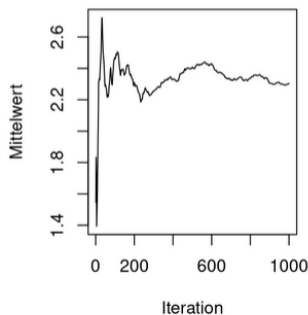
Running Mean: R-Code und Plot

```
par(mfrow=c(1,2))
```

```
l <- 1000
```

```
plot(cumsum(mcmc.tuning[,1])/1:l, type="l", ylab="Mittelwert")
```

```
plot(cumsum(mcmc.tuning[,2])/1:l, type="l", ylab="Mittelwert")
```

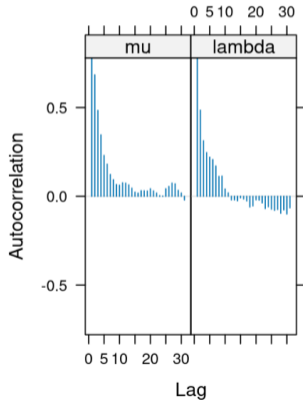


Autokorrelationsfunktion (ACF)

- Misst die Abhängigkeit aufeinanderfolgender Ziehungen.
- Niedrige Autokorrelation \Rightarrow effizientere Kette.
- `coda::acfplot()` zeigt das grafisch.

Autokorrelationsplot: Vorher/Nachher

```
acfplot(mcmc.simple)  
acfplot(mcmc.tuning)
```



- Visuelle Inspektion kann subjektiv sein.
- Es gibt objektive Methoden, z.B. die Gelman-Rubin-Diagnostik.
- Ziel: Beurteilung, ob Ketten ausreichend konvergiert sind.

Vergleicht die Varianz innerhalb und zwischen mehreren Ketten:

$$W = \frac{1}{m} \sum_{j=1}^m s_j^2$$

$$B = \frac{n}{m-1} \sum_{j=1}^m (\bar{\theta}_j - \bar{\theta})^2$$

$$\widehat{\text{Var}}(\theta) = \left(1 - \frac{1}{n}\right) W + \frac{1}{n} B$$

$$\hat{R} = \sqrt{\frac{\widehat{\text{Var}}(\theta)}{W}}$$

Konvergenz: $\hat{R} \approx 1$, z.B. $\hat{R} < 1.1$

Anwendung: Mehrere Ketten

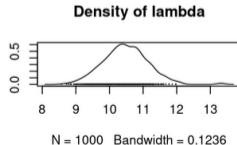
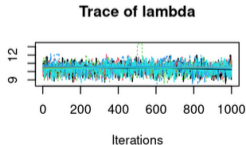
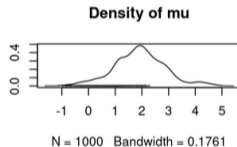
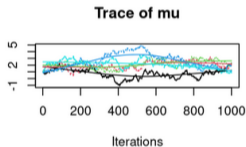
```
mc.list <- parallel::mclapply(rep(sumx, 5), poisson.lognormal.mcmc,  
mc.list <- coda::as.mcmc.list(mc.list)
```

Ziel: Prüfe, ob alle Ketten aus der gleichen Verteilung stammen.

Methode: `gelman.diag(mc.list)`

Visuelle Konvergenzbeurteilung

- Traceplot zeigt mehrere farbige Ketten.
- Idealerweise schwanken sie um denselben Mittelwert.
- Bei μ unterscheiden sich die Ketten deutlich \Rightarrow keine Konvergenz.
- λ sieht besser aus – aber Vorsicht: alle Parameter müssen konvergieren!



Gelman-Rubin-Diagnose: Ohne Tuning

```
gelman.diag(mc.list)
```

```
## mu      1.81   3.84
```

```
## lambda 1.82   1.83
```

```
## Multivariate psrf: 1.79
```

- \hat{R} für beide Parameter und multivariat deutlich > 1.1 .
- \Rightarrow Schlechte Konvergenz – längere Ketten oder besserer Algorithmus nötig.

Mehr Iterationen

```
mc.list <- mclapply(rep(sumx, 5),  
                    poisson.lognormal.mcmc, n=n, l=20000)  
gelman.diag(mc.list)
```

```
## mu      1.02   1.06  
## lambda 1.00   1.00  
## Multivariate psrf: 1.03
```

Fazit: Mehr Iterationen führen zu deutlicher Verbesserung der Konvergenz.

Tuning + kurze Kette (1000 Ziehungen)

```
mc.list <- mclapply(rep(sumx, 5),  
                    poisson.lognormal.mcmc, n=n,  
                    do.tuning=TRUE)  
gelman.diag(mc.list)  
  
## mu      1.01  1.03  
## lambda 1.01  1.02  
## Multivariate psrf: 1.02
```

Fazit: Schon 1000 Ziehungen + gutes Tuning liefern akzeptable \hat{R} -Werte.

- Idee: Teste, ob erste 10% und letzte 50% der Kette aus der gleichen Verteilung stammen.
- `geweke.diag()` berechnet Z-Werte pro Parameter.
- `pnorm()` kann zur Interpretation verwendet werden (Nah an 0 oder 1 \Rightarrow Ablehnung).

```
geweke.diag(mcmc.simple)  
pnorm(geweke.diag(mcmc.simple)$z)
```

```
geweke.diag(mcmc.tuning)  
pnorm(geweke.diag(mcmc.tuning)$z)
```

- **Ohne Tuning:** z.B. p-Wert für $\mu = 0.048 \Rightarrow$ Kette nicht stabil.
- **Mit Tuning:** p-Werte 0.5 \Rightarrow Kein Hinweis auf Instabilität.

- Prüft Stationarität (Cramer-von-Mises-Test) und Konvergenz der Kette.
- Zwei Tests:
 - Stationarität
 - Half-width-Test für Mittelwertabschätzung
- Kette wird ggf. schrittweise gekürzt.

```
heidel.diag(mcmc.simple)
```

```
## mu: failed Half-width  
## lambda: passed
```

```
heidel.diag(mcmc.tuning)
```

```
## mu, lambda: passed
```

- **Ohne Tuning:** Für μ reicht die Kette nicht zur Mittelwertschätzung.
- **Mit Tuning:** Beide Parameter erfüllen Stationaritäts- und Konvergenztest.

- Schätzt, wie viele Iterationen und welchen Burn-in man braucht, um Quantile mit gegebener Toleranz zu schätzen.
- Benötigt gewünschte Genauigkeit r , Quantil q und Sicherheit s .
- Liefert Burn-in, Kettenlänge und Abhängigkeit.

```
raftery.diag(mcmc.simple, q=0.05, r=0.02, s=0.9)
```

```
## mu: Burn-in = 45, N = 361
```

```
## lambda: Burn-in = 3, N = 322
```

```
raftery.diag(mcmc.tuning)
```

```
## mu: Burn-in = 5, N = 361
```

```
## lambda: Burn-in = 3, N = 322
```

Fazit:

- Algorithmus mit Tuning benötigt weniger Iterationen für gleiche Genauigkeit.
- Für höhere Genauigkeit muss Pilotkette eventuell verlängert werden.

- HMC ist eine moderne Variante der Monte-Carlo-Methoden.
- Nutzt Ideen aus der klassischen Mechanik: Energie, Impuls, Bewegungsgleichungen.
- Ziel: effizienteres Sampling mit geringerer Autokorrelation.

- Potenzielle Energie: $U(z)$, kinetische Energie: $K(v)$
- Gesamtenergie: $E(z, v) = U(z) + K(v)$
- Bewegung durch Raum: definiert durch Energiegradient
- In HMC bewegt sich ein Punkt entlang dieser Fläche

- Zusätzliches Hilfsvariable \vec{v} nötig (aus $\mathcal{N}(0, I)$)
- Bewegung mit Leapfrog-Verfahren simuliert
- HMC akzeptiert neue Punkte mit hoher Wahrscheinlichkeit
- Vorteil: große Sprünge, geringere Autokorrelation

Vorteile:

- Effizient bei starker Abhängigkeit
- Gute Abdeckung des Posteriorraums
- Niedrige Autokorrelation

Nachteile:

- Nur für differenzierbare Dichten
- Komplexeres Tuning (Schrittweite ϵ , Anzahl Schritte L)
- Aufwändiger als Gibbs

```
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

- Modellbeschreibung in eigener Datei (ähnlich BUGS)
- Unterstützt Vektoren, Matrizen, Trunkierung, Priors etc.

Modell: Krankenhausschnitt-Daten

```
data {  
  int<lower=0> N;  
  matrix[N, M] X;  
  vector[N] y;  
}  
parameters {  
  vector[M] beta;  
  real<lower=0> sigma;  
}  
model {  
  y ~ normal(X * beta, sigma);  
  beta ~ normal(0, 1000);  
  sigma ~ chi_square(2);  
}
```

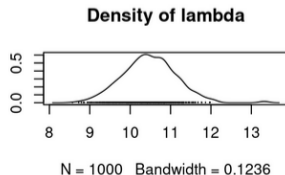
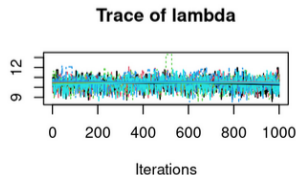
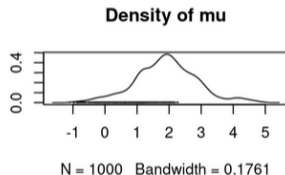
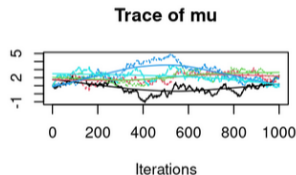
```
linreg <- stan(model_code = linmodel.stan ,  
              data = standata ,  
              model_name = "Schweiz",  
              iter = 2000, warmup = 1000)
```

- 4 Ketten à 2000 Iterationen
- Davon je 1000 Warmup
- Automatisch parallele Ausführung

Parameter-Schätzung und Plots

`plot(linreg)`

- Zeigt Median, 80%- und 95%-Intervalle
- Schnelle Übersicht über Schätzung und Unsicherheit



Mehr Details mit `print()`

```
print(linreg)
```

- Posterior-Statistik pro Parameter: Mittelwert, SD, Quantile, n_{eff} , \hat{R}
- Wichtig: $\hat{R} \approx 1.00 \Rightarrow$ Konvergenz

Einzelne Parameter visualisieren

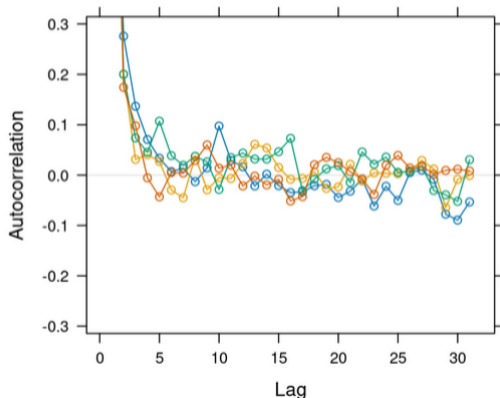
```
beta <- As.mcmc.list(linreg, pars = "beta[2]")  
plot(beta)
```

- Trace + Dichte für z.B. β_2 (Agriculture)
- Unterschiedliche Farben für verschiedene Ketten

Autokorrelationsfunktion

```
coda::acfplot(beta)
```

- Sehr schwache Korrelation in HMC
- Autokorrelationsplot bestätigt gute Mischung



Beispiel:

- Zahl getöteter oder schwer verletzter Autofahrer in England (1969–1984)
- Modell:

$$\sqrt{y_t} \sim \mathcal{N}(\mu_t, \sigma^2), \quad \mu_t = \alpha + \beta x_t + \gamma_t + \delta_t$$

- Parameter $\theta = (\alpha, \beta, \gamma_t, \delta_t)$ erhalten Normalverteilungspriors.
- Gibbs-Sampling für semi-konjugierte Priorverteilungen.

BayesX: Structured Additive Regression

- MCMC oder Empirical Bayes für additive Modelle
- Modelle: GAM, GAMM, Geoadditive Modelle, etc.
- Flexible Modellierung mit strukturierter Glättung (RW1, Saison, Splines)
- R-Anbindung: R2BayesX

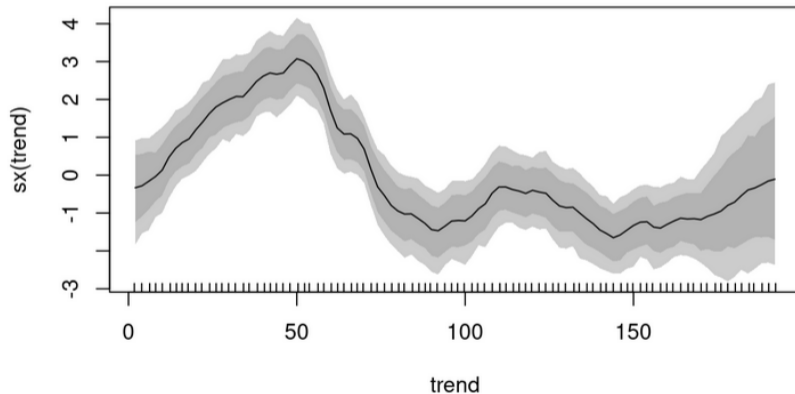
```
formula.bx <- sqrt(y) ~ belt +  
  sx(trend, bs = "rw1", a = 1, b = 0.0005) +  
  sx(seasonal, bs = "season", a = 1, b = 0.1)
```

- *trend*: Monatsindex (RW1)
- *seasonal*: saisonale Struktur
- *belt*: Kovariable
- Modelliert: \sqrt{y}

```
mcmc.bx <- bayesx(formula = formula.bx, data = Drivers ,  
  control = bayesx.control(family = "gaussian",  
    method = "MCMC", hyp.prior = c(4,4),  
    iterations = 10000, burnin = 5000))
```

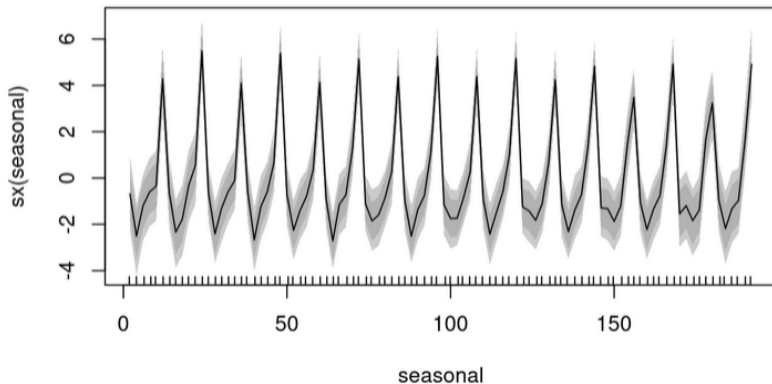
- Methode: MCMC mit 10.000 Iterationen, 5.000 Burn-in
- Priors für Glättungshyperparameter: $a = b = 4$

```
plot(mcmc.bx$effects$sx("trend"))
```



Effekte visualisieren

```
plot(mcmc.bx$effects$sx("seasonal"))
```



```
summary(mcmc.bx)
```

Beispielausgabe:

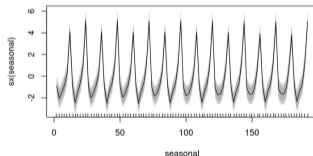
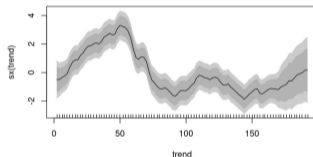
- Fixed Effects: Intercept, Belt
- Smooth Terms: Trend, Season
- Schätzfehler, Quantile, Varianzkomponenten

- Alternative Methode: Empirischer Bayes / REML
- Keine Hyperprior-Verteilung nötig
- Gleiches Modell, andere Schätzmethode:

```
bayesx(..., control = bayesx.control(method = "REML"))
```

Vergleich: REML vs MCMC

- REML liefert Punktschätzer für Glättung (keine Posteriorverteilung)
- Intervallbreite der Effekte in REML oft kleiner
- MCMC liefert Volldistributionen und Unsicherheiten



Warum sind die Intervallschätzer beim Empirischen Bayes-Ansatz kleiner?

- Empirischer Bayes ignoriert Unsicherheit über den Präzisionsparameter τ .
- Im „vollen“ Bayes-Ansatz (z.B. MCMC) wird auch die Unsicherheit über τ berücksichtigt.
- Daraus ergibt sich eine breitere Posteriorverteilung für θ .

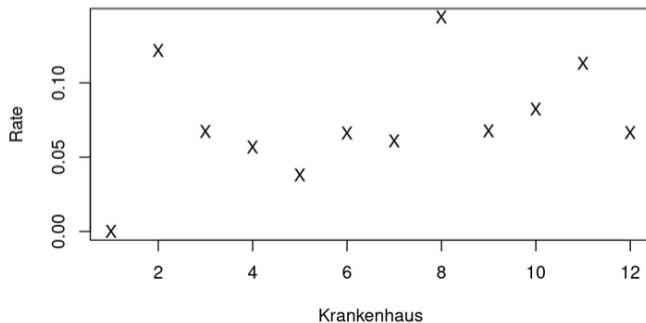
Ziel: Marginale Posteriorverteilung von θ :

$$p(\theta \mid y) = \int p(\theta, \tau \mid y) d\tau = \int p(\theta \mid \tau, y) p(\tau \mid y) d\tau$$

- Diese Mischung verschiedener Modelle nennt man **Model Averaging**.
- Statt ein einzelnes Modell zu wählen, gewichten wir alle möglichen τ nach ihrer Posteriorverteilung.

Daten:

- Todesfälle nach Herzoperationen bei Babys
- Beobachtungen von 12 Krankenhäusern
- Totesrate ist krankenhausspezifisch
- Individuelle Raten stammen aus gemeinsamer Verteilung



Datenmodell:

$$y_i \sim \text{Bin}(n_i, \pi_i), \quad \pi_i = \frac{\exp(\theta_i)}{1 + \exp(\theta_i)}$$

Priors:

$$\theta_i \mid \tau \sim \mathcal{N}(\mu, \tau^{-1}), \quad \tau \sim \text{Gamma}(a, b)$$

- μ : Effekt aller Krankenhäuser
- τ : Präzision der Krankenhauseffekte

- Hierarchisches Modell mit 3 Ebenen:

$$y \mid \theta \sim F(\theta), \quad \theta \sim \mathcal{N}(\mu, \tau^{-1}), \quad \tau \sim \text{Gamma}(a, b)$$

- Marginale Posterior:

$$p(\theta \mid y) = \int p(\theta \mid \tau, y) p(\tau \mid y) d\tau$$

- Ziel: Approximation dieses Integrals mit Laplace

Ziel:

$$\int \exp(-nh(z)) dz$$

- Näherung durch Maximum von $h(z)$ und Taylorentwicklung

- **Definition:**

$$\int \exp(-nh(z)) dz \approx \sqrt{\frac{2\pi}{nh''(\hat{z})}} \exp(-nh(\hat{z}))$$

- Gute Näherung für unimodale, glatte Funktionen

Laplace-Approximation des Posterior-Erwartungswerts

- Ziel: $E(g(\theta) \mid x) = \frac{\int g(\theta) \exp(-nq(\theta)) d\theta}{\int \exp(-nh(\theta)) d\theta}$
- Mit $\hat{\theta}$, $\tilde{\theta}$ als Maximumstellen:

$$E(g(\theta) \mid x) \approx \sqrt{\frac{h''(\hat{\theta})}{q''(\tilde{\theta})}} \exp(-n[q(\tilde{\theta}) - h(\hat{\theta})])$$

- Fehlerordnung: $\mathcal{O}(1/n^2)$

Vergleich für verschiedene Stichprobengrößen n :

- Größeres $n \Rightarrow$ bessere Näherung
- Fehler reduziert sich mit wachsendem n
- Gilt auch für Binomial-Modell \rightarrow Poisson-Näherung

Integrated Nested Laplace Approximation (INLA)

- Ziel: Posterior-Approximation für hierarchische Modelle
- Idee:
 - Approximiere $p(\tau | y)$ über Gitterpunkte
 - Nutze Laplace-Näherung für $p(\theta | \tau, y)$
 - Kombiniere zu $p(\theta | y) = \sum_{\tau} p(\theta | \tau, y)p(\tau | y)$
- Sehr schnell im Vergleich zu MCMC bei vielen Modellen

- R-Paket INLA
- Modellstruktur wie bei BayesX:
 - hospital = Faktor für Krankenhäuser
 - IID zufällige Effekte
 - Gamma-Hyperpriors auf Präzision
- `family = "binomial", Ntrials = n`

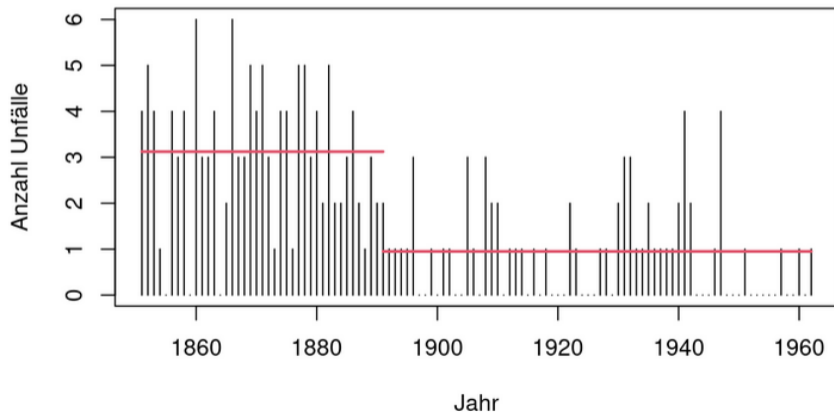
- Posterior für:
 - Intercept
 - Krankenhauspezifische Effekte (Boxplots)
 - Präzision (Gammaform)
- Ausgabe enthält Posterior-Median, Quantile, Log-Likelihood etc.

- Änderung eines Modellparameters zu einem bestimmten Zeitpunkt
- Typisch bei Zeitreihen
- Zeit kann diskret oder stetig sein

- Aktienkurse
- Schlafdaten
- Krankheiten mit Schüben
- Stärke von Sportmannschaften
- Krankheitsausbrüche (z.B. Epidemien)

Beispiel: Unfälle in Kohlebergwerken

- Daten: jährliche Anzahl von Unfällen (1851–1962)
- Deutlicher Rückgang ab ca. 1900



Bruchpunktmodell:

$$Y_i \sim \begin{cases} \text{Poisson}(\lambda_1), & i = 1, \dots, \theta \\ \text{Poisson}(\lambda_2), & i = \theta + 1, \dots, 112 \end{cases}$$

- θ : Bruchzeitpunkt
- λ_1, λ_2 : Unfallraten vor und nach dem Bruch

- $\lambda_i \sim \text{Gamma}(3, \alpha)$, für $i = 1, 2$
- $\alpha \sim \text{Gamma}(10, 10)$
- $\theta \sim \text{Uniform}(1, \dots, 112)$

$$p(\lambda_1, \lambda_2, \alpha, \theta \mid y) \propto \text{Likelihood} \times \text{Priors}$$

- Likelihood via Poisson
- Posterior nur bis auf Normierung bekannt
- Gibbs-Sampling möglich mit Full Conditionals

Für λ_1 :

$$\lambda_1 \mid \dots \sim \text{Gamma} \left(3 + \sum_{i=1}^{\theta} y_i, \theta + \alpha \right)$$

Für λ_2 :

$$\lambda_2 \mid \dots \sim \text{Gamma} \left(3 + \sum_{i=\theta+1}^{112} y_i, 112 - \theta + \alpha \right)$$

Für α :

$$\alpha \mid \dots \sim \text{Gamma}(16, 10 + \lambda_1 + \lambda_2)$$

Für θ : diskrete Posteriorverteilung

$$p(\theta \mid \dots) \propto \left(\frac{\lambda_1}{\lambda_2}\right)^{\sum_{i=1}^{\theta} y_i} \cdot \exp(\theta(\lambda_2 - \lambda_1))$$

- Bruchpunktmodelle sind nützlich zur Modellierung struktureller Änderungen
- Volle Bayessche Inferenz erlaubt Unsicherheitsquantifizierung
- Gibbs-Sampling effizient durch Full-Conditionals
- Erweiterbar auf mehrere Brüche oder andere Verteilungen