MCMC - Tuning und Konvergenzdiagnostik

Volker Schmid

29. Mai 2017

Beispiel MCMC bei Poisson-Lognormal

```
library(coda)
```

```
##
## Attaching package: 'coda'

## The following object is masked from 'package:rstan':
##
##     traceplot
```

```
library(bayeskurs)
```

```
## Bayes Kurs 0.3.2

##
## Attaching package: 'bayeskurs'

## The following object is masked _by_ '.GlobalEnv':
##
```

Konvergenz

MCMC-Algorithmen erzeugen eine Markovkette aus Ziehungen aus der Posteriori-Verteilung. Probleme:

- Der Algorithmus muss konvergieren, damit er aus der stationären Verteilung zieht
- Ziehungen sind automatisch abhängig. Damit ist der Algoritmus ineffizienter als unabhängiges Ziehen
- Die Effizienz hängt stark vom genauen Algorithmus ab, z.B. Metropolis-Hastings oder Gibbs-Sampler, Wahl der Vorschlagsdichte, etc..

MCMC mit coda I

```
summary(mcmc.simple)
```

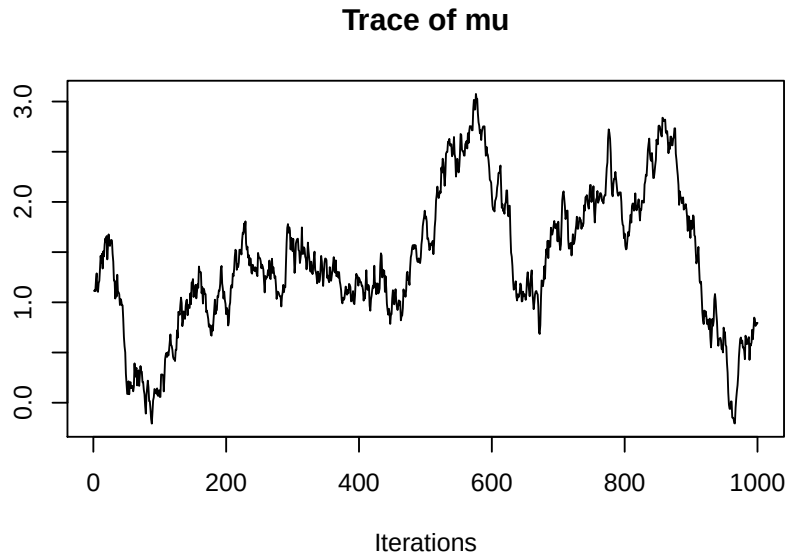
```
##
## Iterations = 1:1000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variab
##    plus standard error of the mean:
##
##           Mean     SD Naive SE Time-series SE
## mu       1.415 0.6830  0.02160        0.29944
## lambda 10.430 0.6565  0.02076        0.04004
##
```

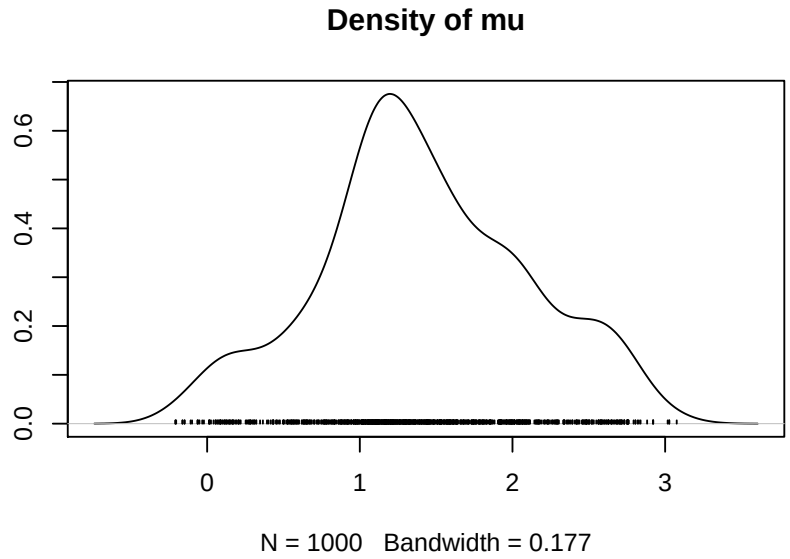
MCMC mit coda II

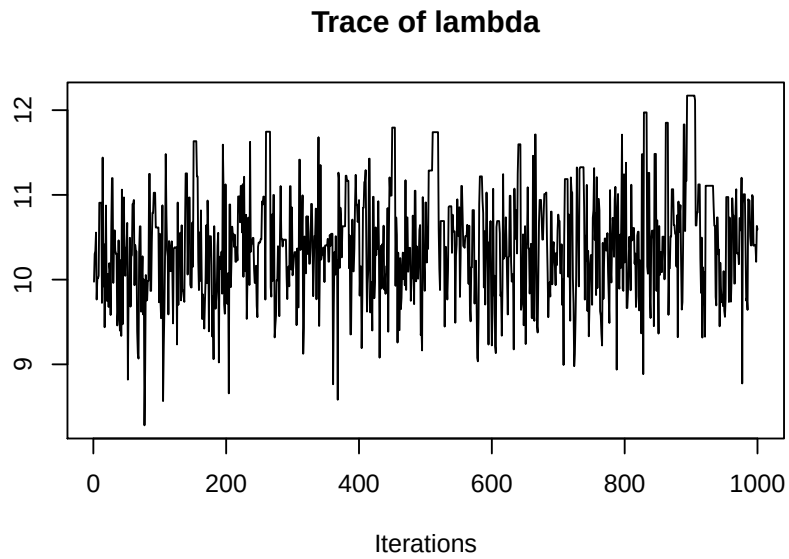
```
## 2. Quantiles for each variable:
##
##           2.5%   25%    50%    75%  97.5%
## mu     0.07019 1.023  1.341  1.913  2.733
## lambda 9.20793 9.981 10.404 10.868 11.794
```

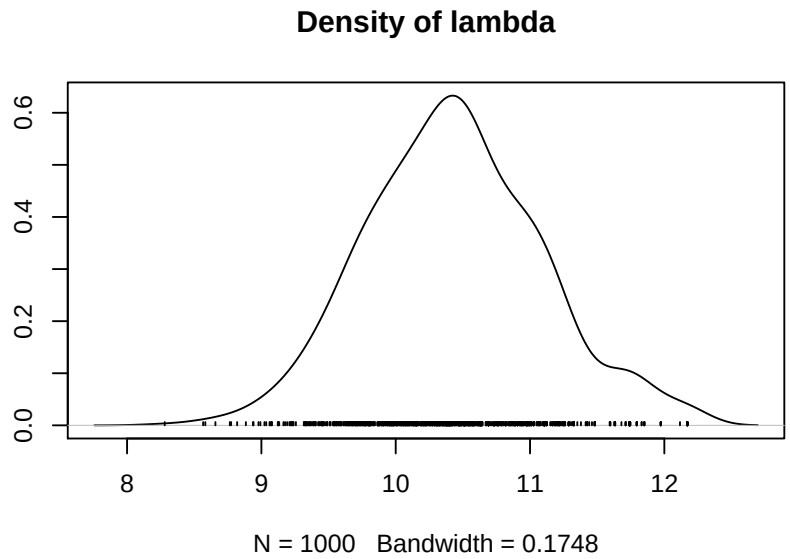
MCMC mit coda

```
plot(mcmc.simple)
```



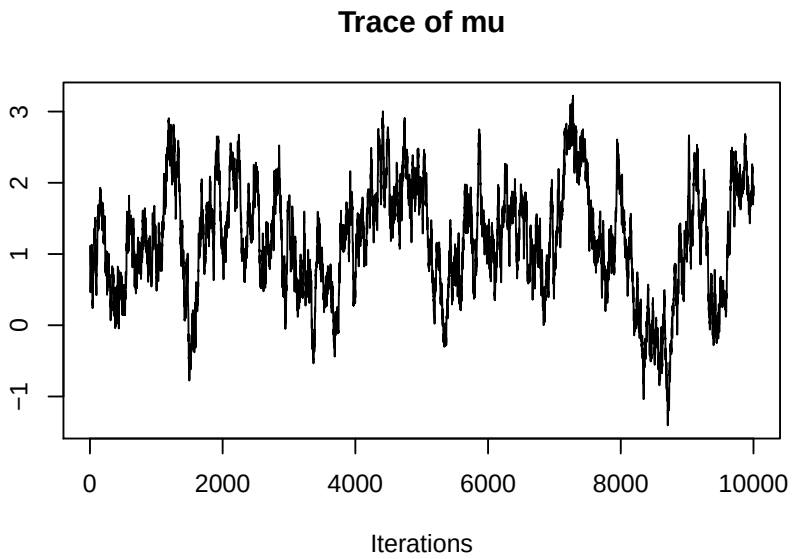


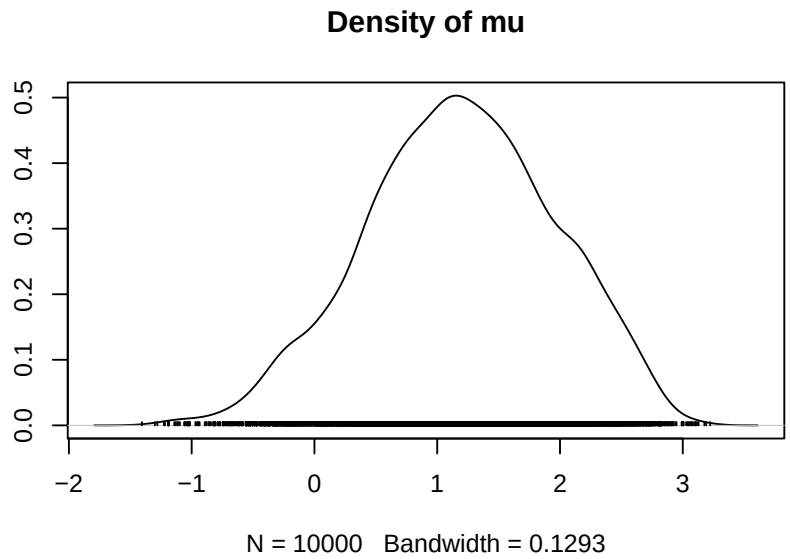


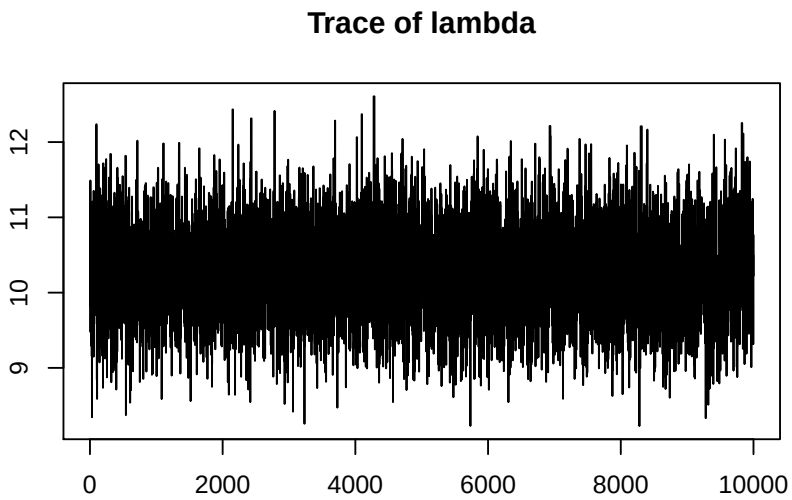


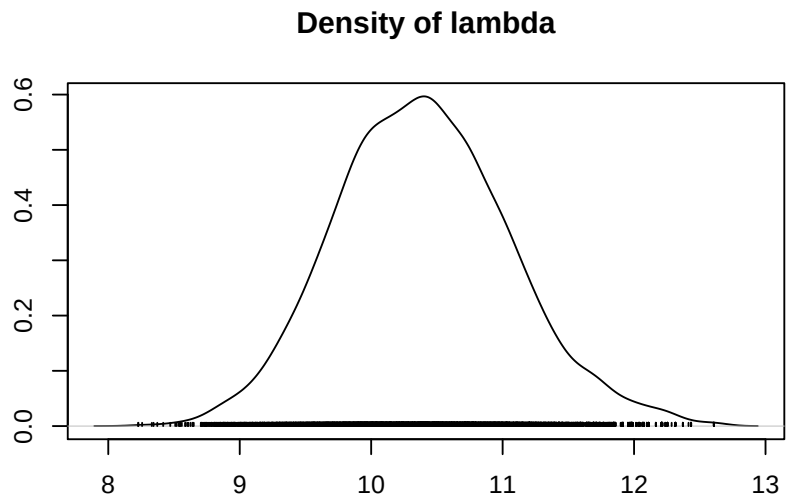
Längere Ketten

```
mcmc.laenger<-poisson.lognormal.mcmc(sumx,n, I=10000)
plot(mcmc.laenger)
```









Tuning des Random Walk Proposals

Random Walk Proposal:

$$\theta^* = \theta^{(i-1)} + u$$

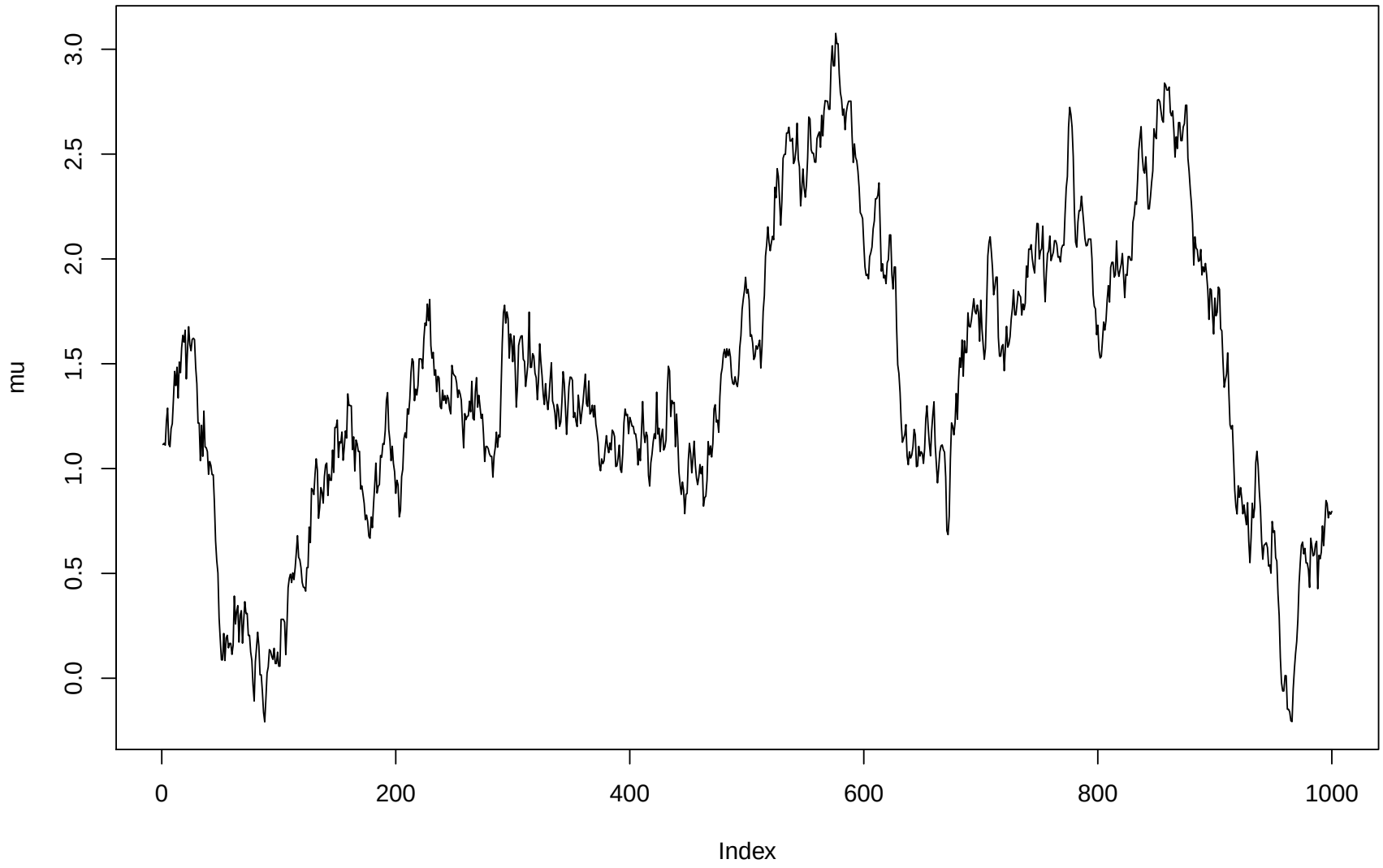
oft mit $u \sim N(0, c)$. Wie wählt man c?

Akzeptanzrate = Anteil akzeptierter Vorschläge * Niedrige Akzeptanzrate: Kette bleibt oft im selben Zustand $\rightarrow$ schlecht * Zu hohe Akzeptanzrate: Kette bewegt sich (eventuell) nur langsam $\rightarrow$ schlecht

Tuning: Finde optimalen Wert für c. Für Random Walk Proposal gelten Akzeptanzraten zwischen ca. 30% und 60% als optimal (?)

Beispiel zu hohe Akzeptranzrate

```
plot(as.vector(mcmc.simple[,1]),type="l",ylab="mu")
```



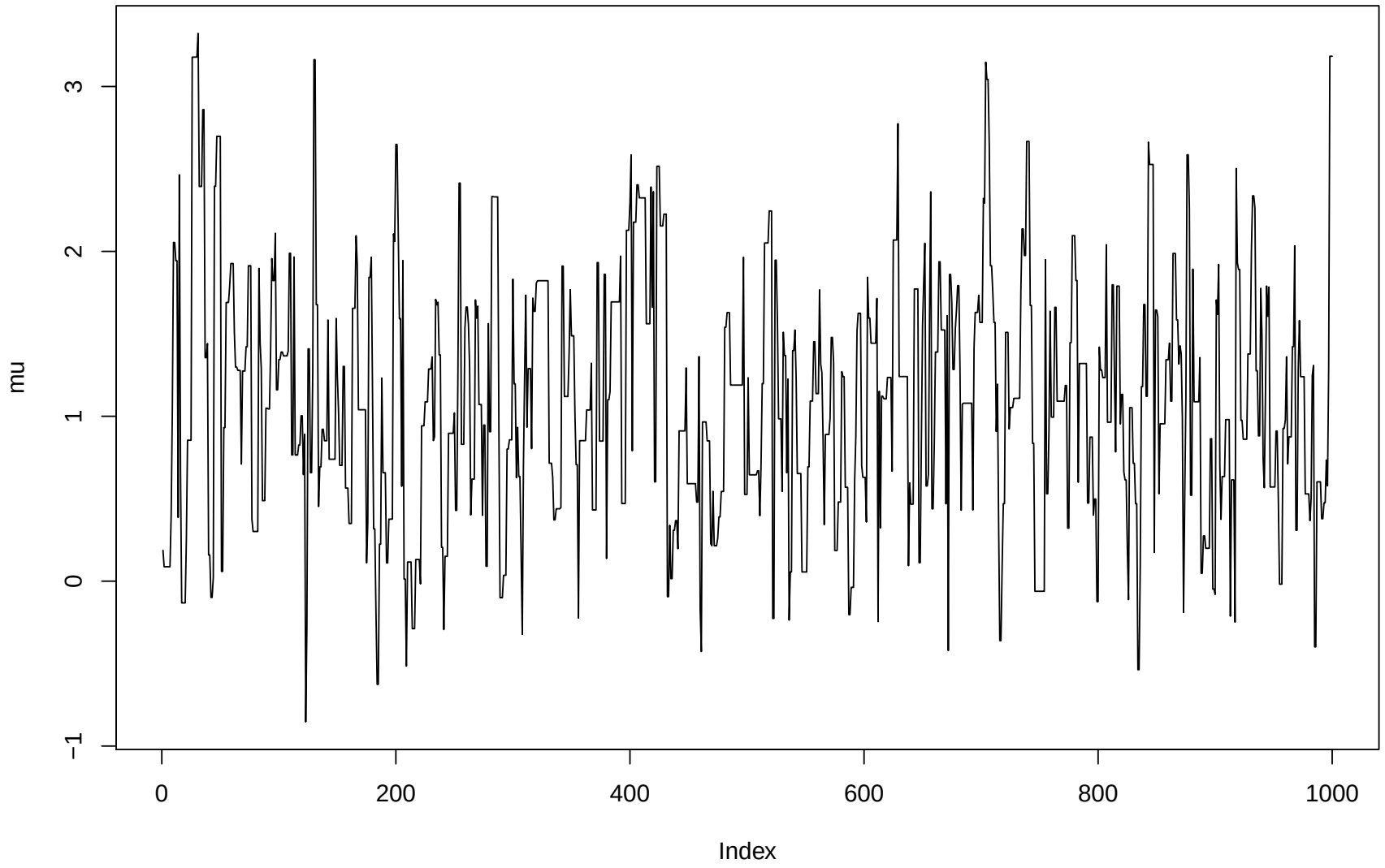
Poisson-Lognormal mit Tuning

```
mcmc.tuning<-poisson.lognormal.mcmc(sumx,n,do.tuning=TRUE)
```

```
## Akzeptanzrate 0.96

## Akzeptanzrate 0.86

## Akzeptanzrate 0.8

## Akzeptanzrate 0.7
##
## Akzeptanzrate 0.7

## Akzeptanzrate 0.66

## Akzeptanzrate 0.46
```

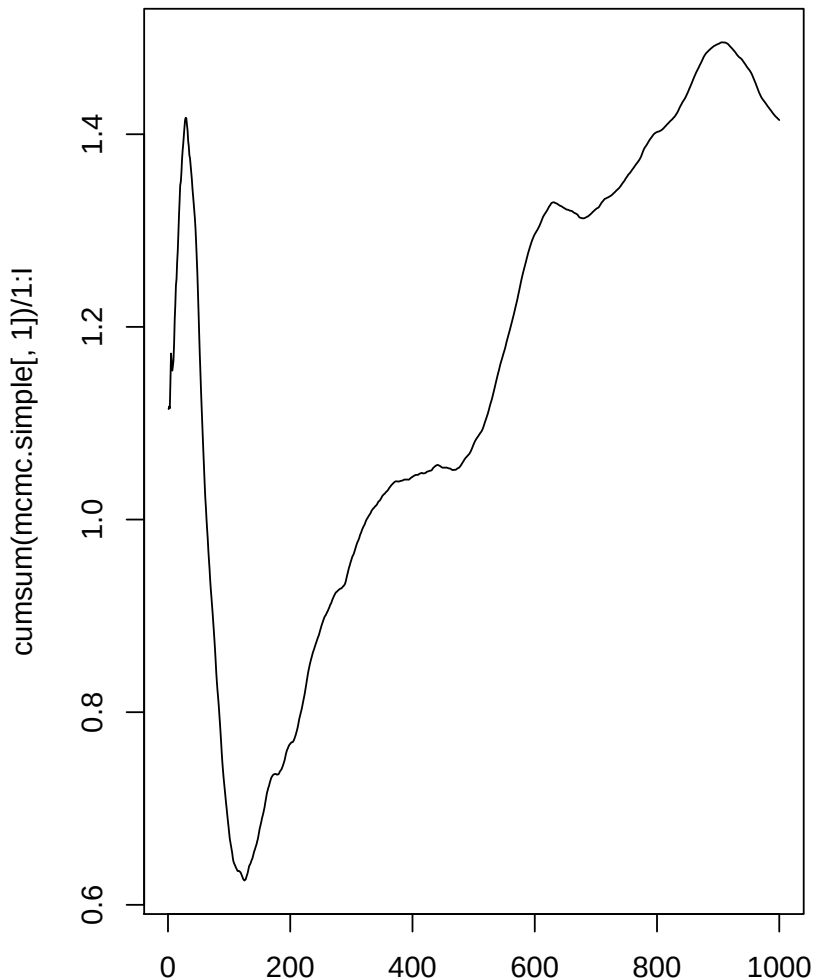
Poisson-Lognormal mit Tuning

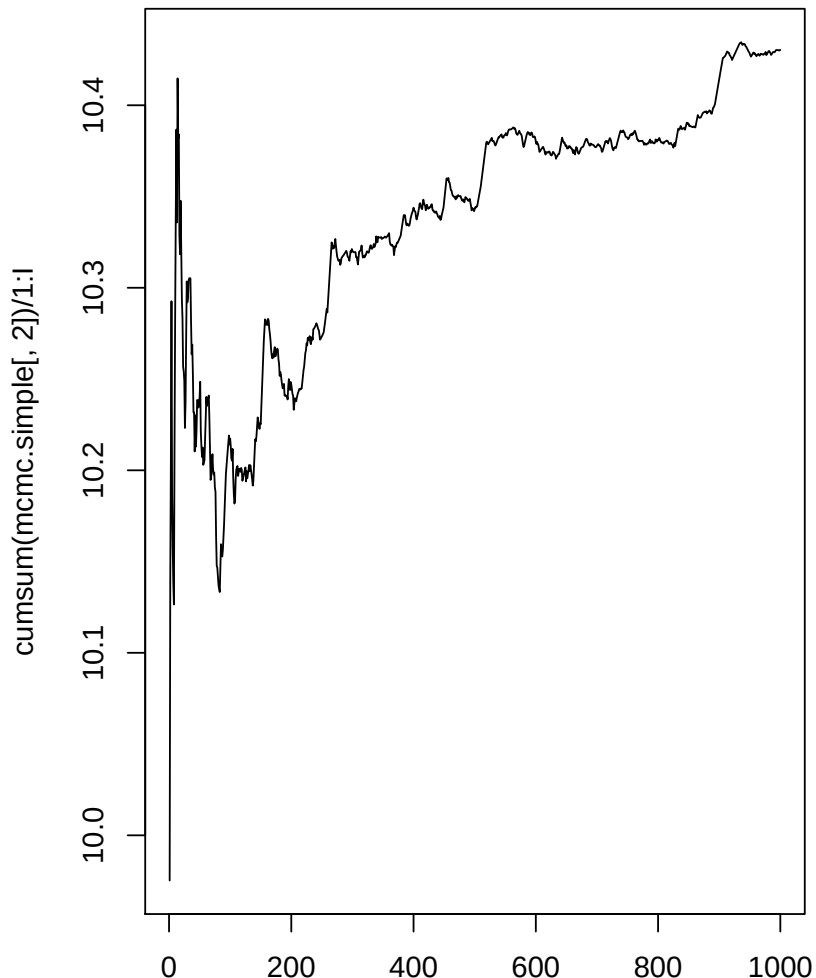
```
plot(as.vector(mcmc.tuning[,1]),type="l",ylab="mu")
```



Running mean plots

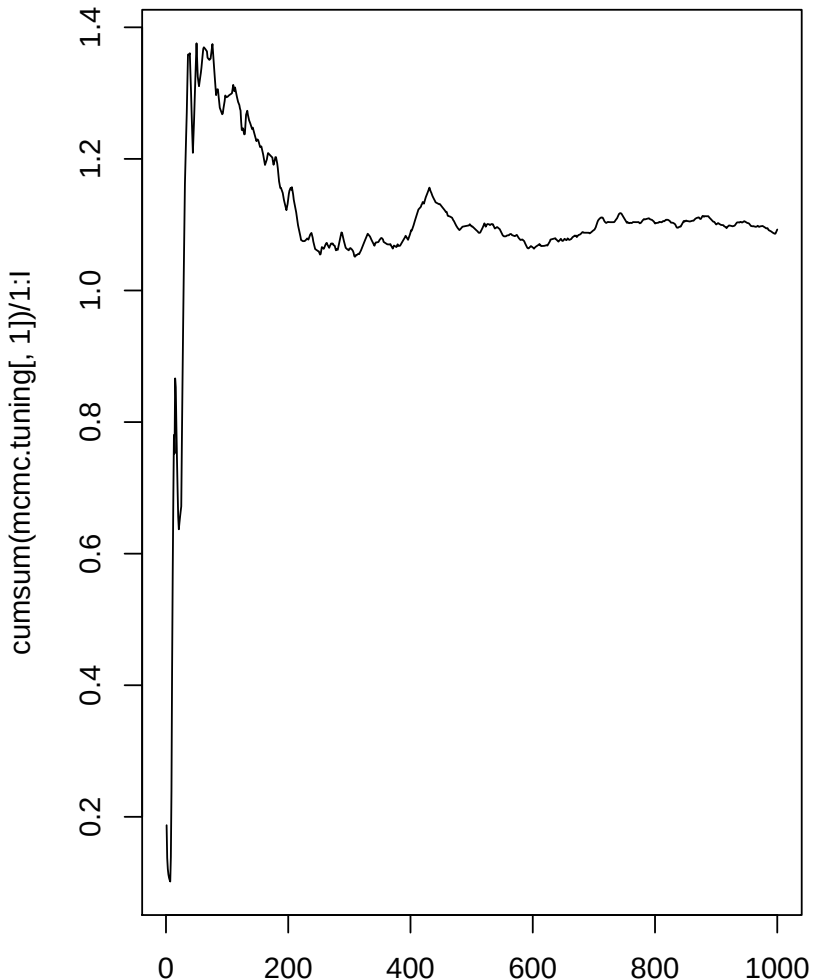
```
par(mfrow=c(1,2))
I=1000
plot(cumsum(mcmc.simple[,1])/1:I,type="l")
plot(cumsum(mcmc.simple[,2])/1:I,type="l")
```

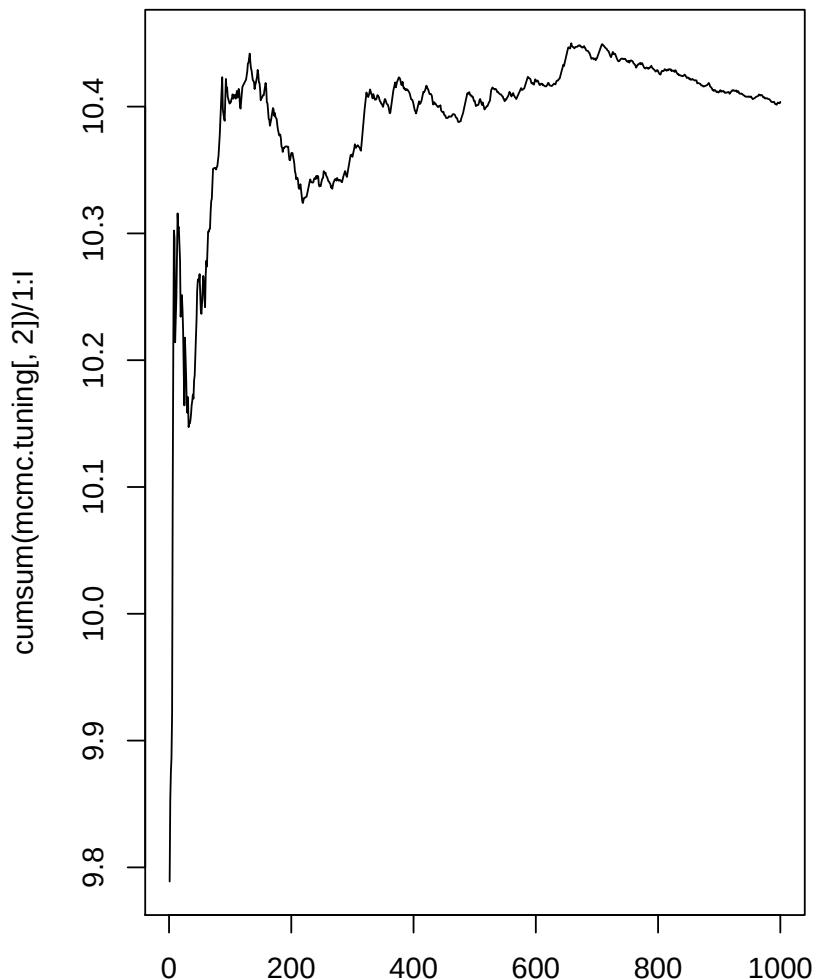




Running mean plots

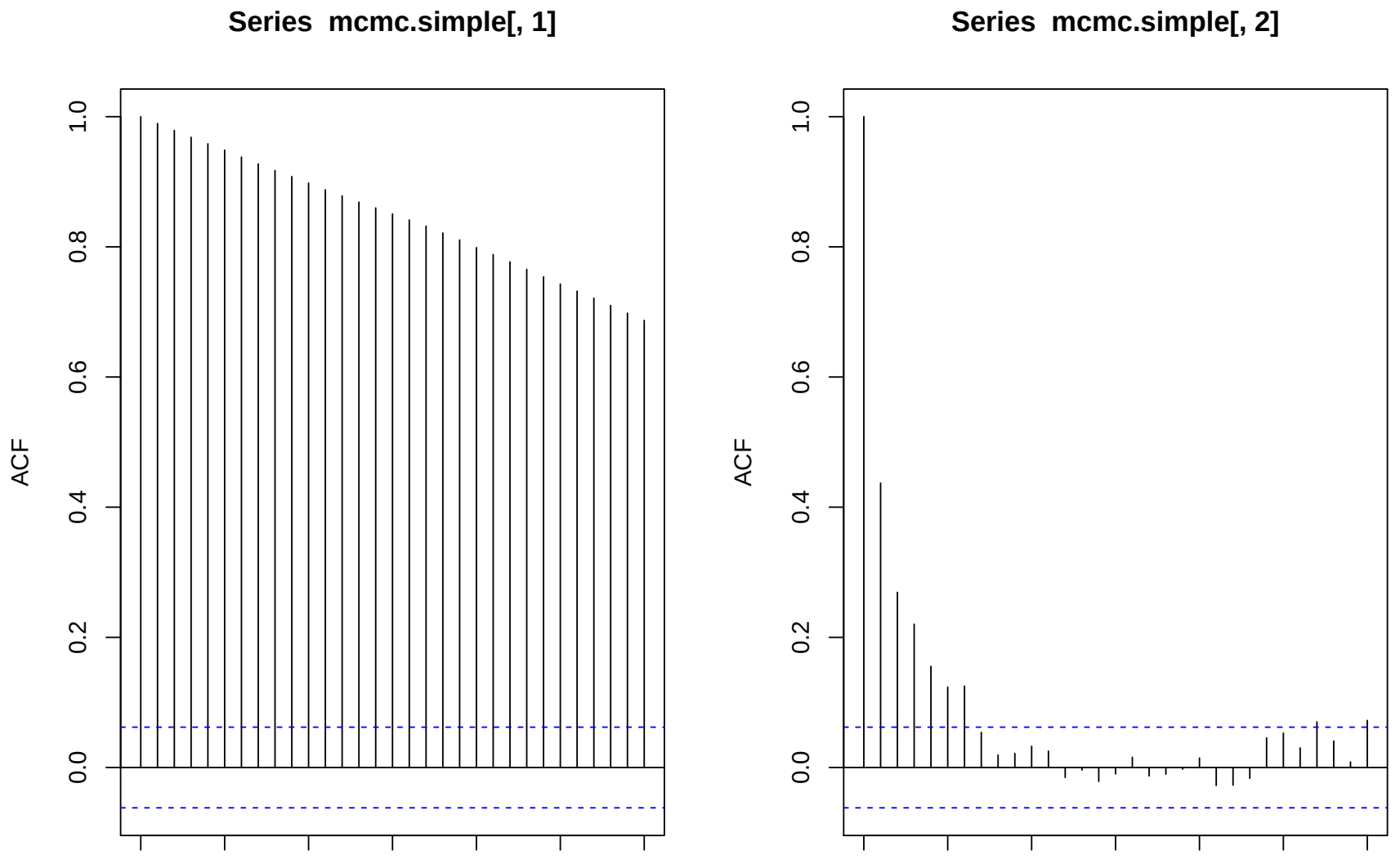
```
par(mfrow=c(1,2))
I=1000
plot(cumsum(mcmc.tuning[,1])/1:I,type="l",)
plot(cumsum(mcmc.tuning[,2])/1:I,type="l")
```





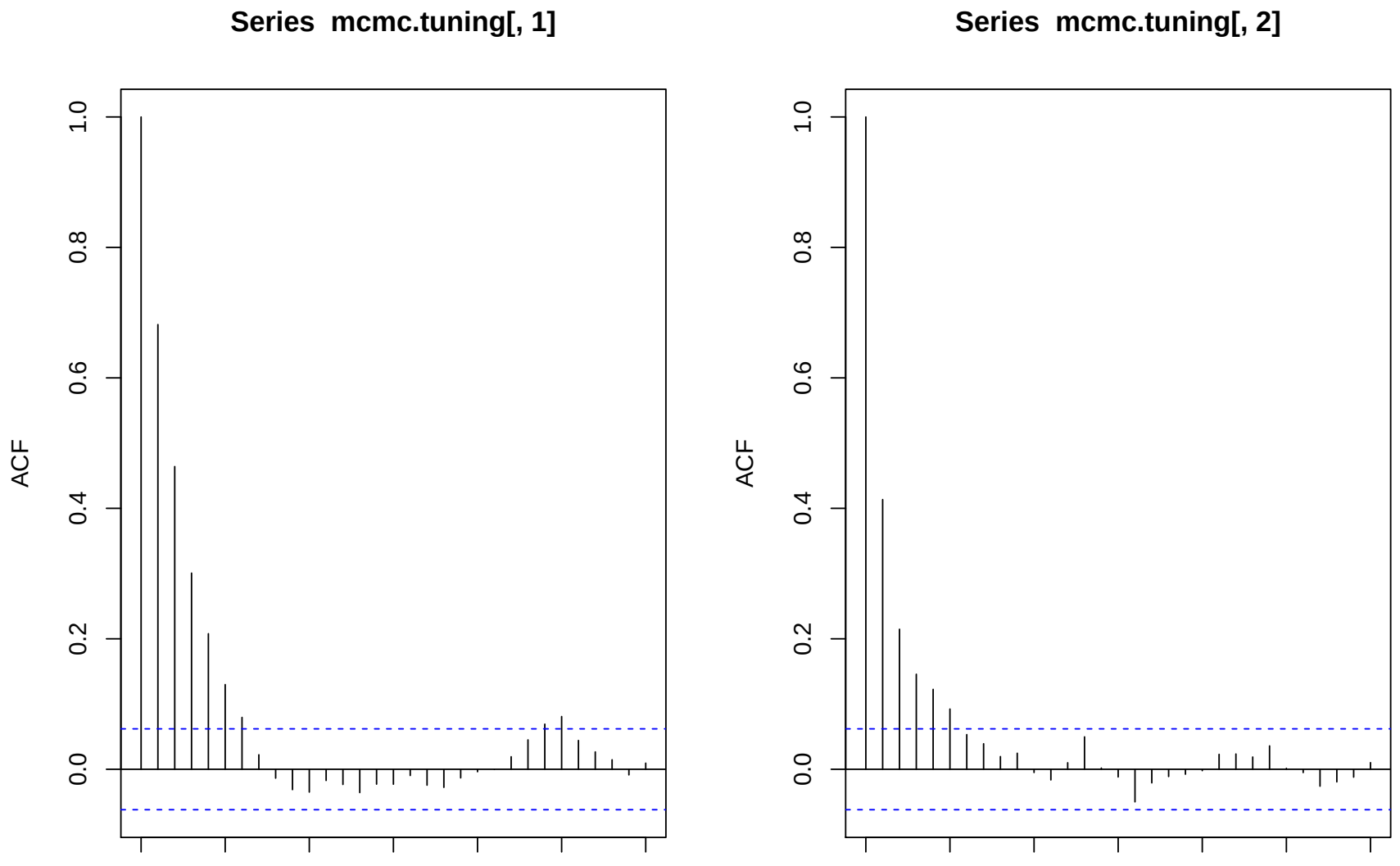
Auto correlation function

```
par(mfrow=c(1,2))
acf(mcmc.simple[,1])
acf(mcmc.simple[,2])
```



Auto correlation function

```
par(mfrow=c(1,2))
acf(mcmc.tuning[,1])
acf(mcmc.tuning[,2])
```



Konvergenzdiagnostik

Gelman-Rubin-Diagnostik

- Idee: vergleiche die Varianz von mehreren parallel gelaufenen Ketten
- Bei Konvergenz sollte die Varianz in den Ketten der Varianz zwischen den Ketten entsprechen
- Within Chain Variance (unterschätzt die wahre Varianz, wenn Ketten noch nicht konvergiert)

$$W = \frac{1}{m}\sum_{j=1}^{m} s_j^2, s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\theta_{ij} - \bar{\theta}_j)^2$$

- Between Chain Variance

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

Gelman-Rubin-Diagnostik

- Geschätzte Varianz

$$\hat{Var}(\theta) = (1 - \frac{1}{n}W + \frac{1}{n}B)$$

- Potential scale reduction factor

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

- Ist $\hat{R}$ zu groß (> 1.1?), sollten die Ketten länger laufen.

Gelman-Rubin-Diagnostik

```
require(coda)
mc.list<-parallel::mclapply(rep(sumx, 5),
            poisson.lognormal.mcmc,n=n)
print(gelman.diag(mc.list))
```

```
## Potential scale reduction factors:
##
##        Point est. Upper C.I.
## mu           1.23       1.53
## lambda       1.01       1.03
##
## Multivariate psrf
##
## 1.24
```

Gelman-Rubin-Diagnostik

```
require(coda)
mc.list<-parallel::mclapply(rep(sumx, 5),
            poisson.lognormal.mcmc,n=n,I=10000)
print(gelman.diag(mc.list))
```

```
## Potential scale reduction factors:
##
##        Point est. Upper C.I.
## mu           1.01       1.03
## lambda       1.00       1.01
##
## Multivariate psrf
##
## 1.01
```

Gelman-Rubin-Diagnostik

- Gelman-Rubins R muss für jeden Parameter geschätzt werden
- Ziehungen aller Ketten werden dann zusammengeworfen
- Alternativ auch Aufteilung einer Kette möglich

Geweke-Diagnostik

- Idee: Teste, ob zwei Teile einer Kette aus der selben Verteilung stammen (Teste Differenz der Mittelwerte)
- In der Regel erste 10% und letzte 50% der Kette

```
print(geweke.diag(mcmc.simple))
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##     mu lambda
## -1.567 -3.476
```

Geweke-Diagnostik

```
print(geweke.diag(mcmc.tuning))
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##         mu     lambda
## 0.8079148 0.0004382
```

Raftery und Lewis Diagnostik

- Wir interessieren uns für ein Quantil q.
- Wieviele Iterationen N und welchen burn-in M brauchen wir, um mit Wahrscheinlichkeit s innerhalb der Toleranz r das Quantil q zu schätzen?
- Nach einer Pilotkette lassen wir die längere Kette entsprechend laufen.

Raftery und Lewis Diagnostik

```
print(raftery.diag(mcmc.simple,
                   q = 0.5, r = 0.05, s = 0.9))
```

```
##
## Quantile (q) = 0.5
## Accuracy (r) = +/- 0.05
## Probability (s) = 0.9
##
##          Burn-in  Total Lower bound  Dependence
##          (M)      (N)   (Nmin)       factor (I)
##  mu      160      13765 271          50.80
##  lambda 6         510   271           1.88
```

Heidelberg und Welch Diagnostik

- Teste, ob die Kette aus einer stationären Verteilung kommt
- Zuerst: Cramer-von Mises-Test mit Niveau α auf ganzer Kette
- Falls Nullhypothese abgelehnt, verwirf erste 10%, 20%, . . . , bis zu 50%
- Falls Nullhypothese angenommen: Half-width-test
- Berechne $(1 - \alpha)$-Kredibilitätsintervall (KI)
- Teststatistik: Hälfte der Breite des KI durch Mittelwert

Heidelberg und Welch Diagnostik

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

```
##
##        Stationarity start     p-value
##        test          iteration
## mu     passed          1       0.431
## lambda passed          101     0.107
##
##        Halfwidth Mean  Halfwidth
##        test
## mu     failed     1.41 0.5869
## lambda passed    10.45 0.0835
```

Heidelberg und Welch Diagnostik

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

```
##
##        Stationarity start     p-value
##        test          iteration
## mu     passed        1         0.855
## lambda passed        1         0.462
##
##        Halfwidth Mean  Halfwidth
##        test
## mu     passed     1.09 0.105
## lambda passed    10.40 0.069
```