

Hamiltonian MC und STAN

Volker Schmid

12. Juni 2017

Hamiltonsche Mechanik

Stochastische (Radon-Nikodým-)Dichte f entspricht physikalischer (Gibbs-)Energie ϕ :

$$p(x) \propto \exp(-\phi(x))$$

Hamiltonsche Mechanik: Die Bewegung eines Objekts durch einen Raum lässt sich durch Differentialgleichungen mit Ort x (potentielle Energie $U(x)$) und Impuls v (kinetische Energie $K(v)$) beschreiben

$$E(x, v) = U(x) + K(v)$$

Hamiltonian MC (Hybrid MC)

- ▶ Benutze Hilfsvariable v
- ▶ Zusätzliche zufällige Komponente (ansonsten bleibt $E(x, v)$ konstant): Ziehe zufällig aus der Priori von v (Normalverteilung)

Vorstellung: Bewegung eines Pucks auf der Parameter-Oberfläche, der immer wieder in zufällige Richtungen geschubst wird.

Für MCMC müssen Hamilton-Gleichungen zeitdiskretisiert werden, mit Schrittgröße ϵ (einfach: Euler-Methode, besser: Leapfrog).

Euler/Leapfrog

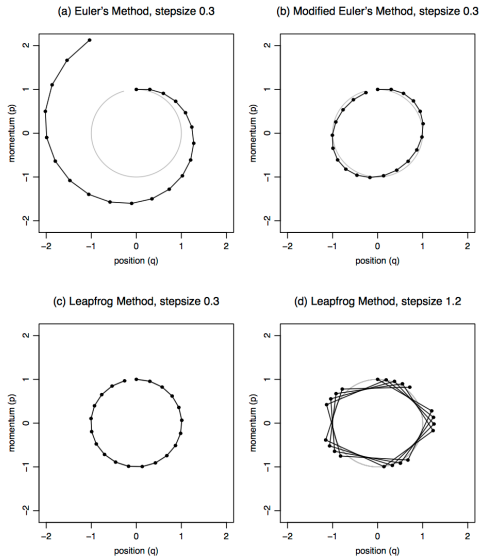


Figure 1: 20 Iterationen Euler vs. Leapfrog

Vor- und Nachteile von HMC

- ▶ Funktioniert nur bei differenzierbaren Funktionen
- ▶ Einzelne Iteration ist sehr viel aufwendiger als bei M-H oder Gibbs
- ▶ I.d.R. geringere Abhängigkeit, großer Abstand zwischen Ziehungen
- ▶ Hohe Akzeptanzrate
- ▶ Probleme bei isolierten lokalen Minima (bzw. Maxima)
- ▶ Tuning ist notwendig (ϵ)

Literatur: Radford M. Neal: MCMC using Hamiltonian dynamics, in: Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. Handbook of Markov Chain Monte Carlo. Chapman & Hall / CRC Press, 2011

STAN (hier: RStan)

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

Beispiel Lineare Regression

$$y_i \sim N(x_i' \beta, \sigma^2)$$

$$\beta_j \sim N(0, 1000)$$

$$\sigma^2 \sim \chi^2(2)$$

STAN Modell

```
data {  
  int<lower=0> N;  
  int<lower=0> M;  
  matrix[N, M] x;  
  vector[N] y;  
}  
parameters {  
  vector[M] beta;  
  real<lower=0> sigma;  
}  
model {  
  y ~ normal(x * beta, sigma);  
  for(i in 1:M)  
    beta[i] ~ normal(0, 1000);  
  sigma ~ chi_square(2);  
}
```

STAN Beispiel (1)

```
# Simulate some data  
N <- 1000  
M <- 3  
x <- cbind(rep(1,N), rpois(N,5), runif(N,2,3))  
truebeta <- c(-14, 0, 5)  
truesigma <- 1  
y <- rnorm(N, x%*%truebeta, truesigma)
```


STAN Beispiel (2)

```
stan_data <- list(N=N, M=M, y=y, x=x)
limo <- stan(file = 'limo.stan',
  model_name='LinearesModell', data=stan_data,
  iter=1000, chains=4)
```

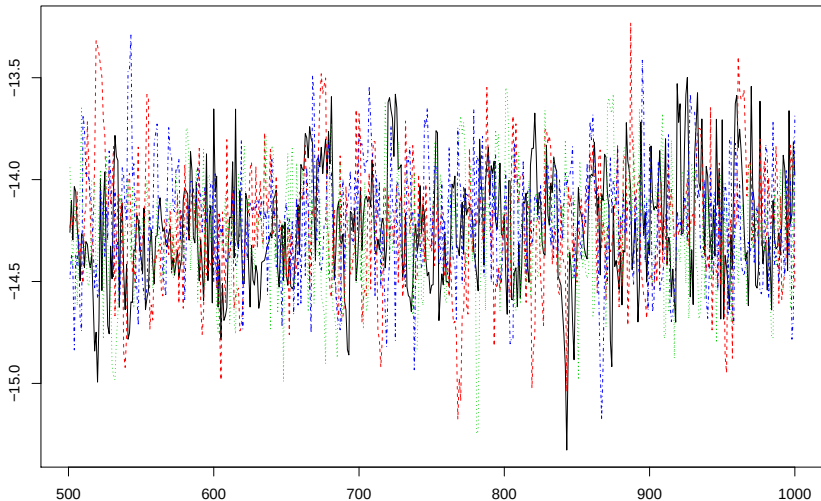
STAN Beispiel Ergebnisse

```
print(limo)
```

```
## Inference for Stan model: LinearesModell.  
## 4 chains, each with iter=1000; warmup=500; thin=1;  
## post-warmup draws per chain=500, total post-warmup draws  
##  
##           mean se_mean   sd    2.5%    25%    50%  
## beta[1]  -14.23    0.01 0.29   -14.78   -14.43   -14.24  
## beta[2]   -0.01    0.00 0.01    -0.04    -0.02    -0.01  
## beta[3]    5.13    0.00 0.11     4.90     5.06     5.13  
## sigma     1.00    0.00 0.02     0.96     0.99     1.00  
## lp__      -500.24    0.05 1.39   -503.67  -500.88  -499.94  
##           Rhat  
## beta[1]    1  
## beta[2]    1  
## beta[3]    1  
## sigma      1  
## lp         1
```

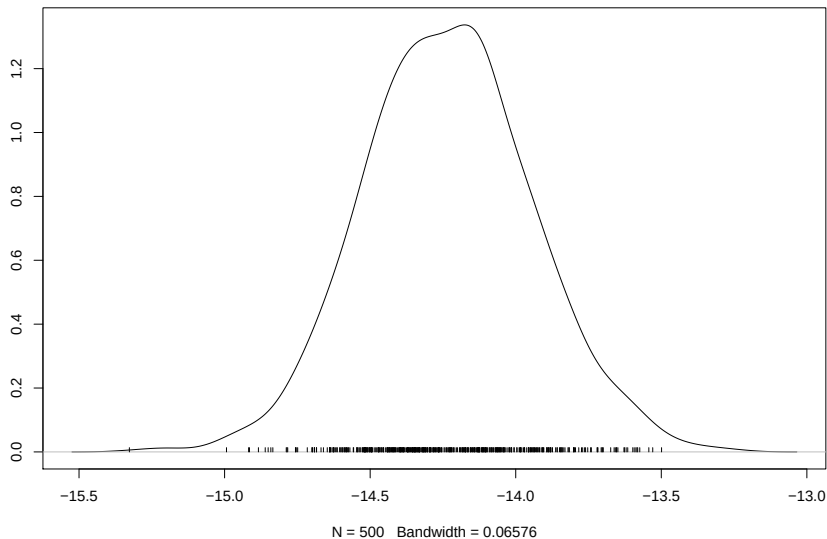
STAN Beispiel Ergebnisse

```
beta<-As.mcmc.list(limo,pars="beta")  
beta0<-As.mcmc.list(limo,pars="beta[1]")  
coda::traceplot(beta0)
```



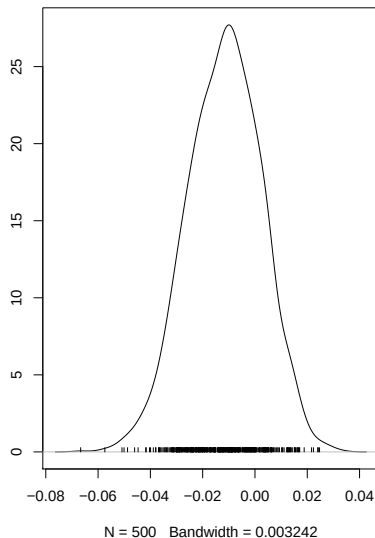
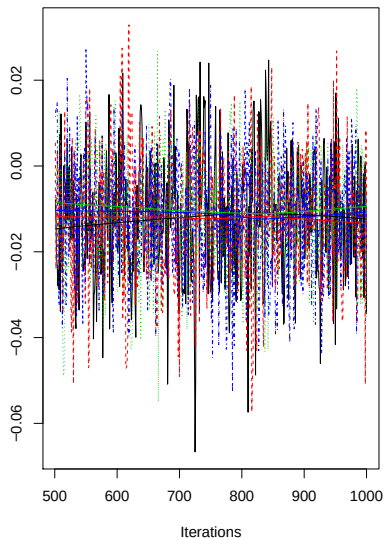
STAN Beispiel Ergebnisse

```
coda::densplot(beta0)
```



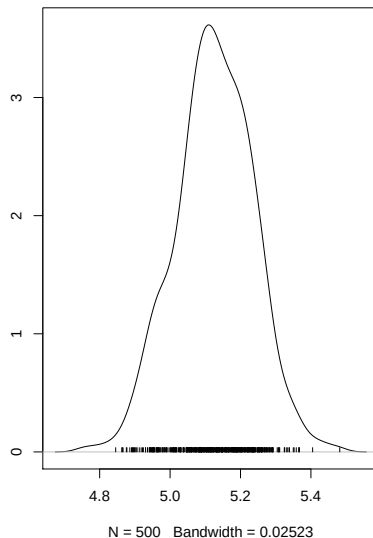
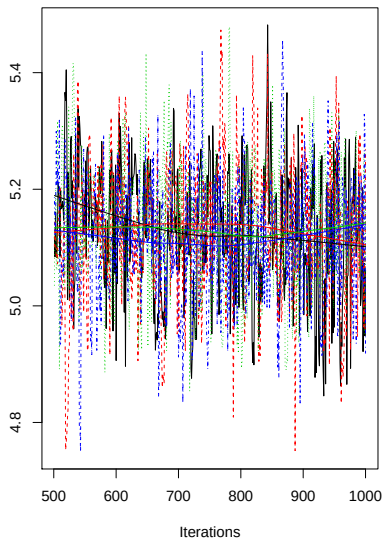
STAN Beispiel Ergebnisse

```
beta1<-As.mcmc.list(limo,pars="beta[2] ")  
plot(beta1)
```



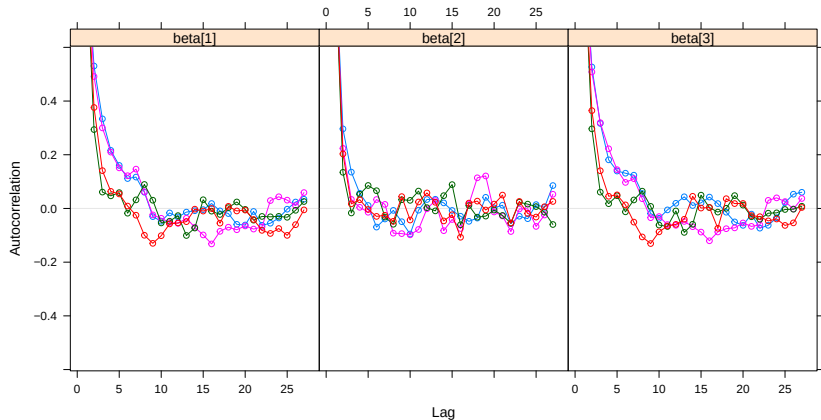
STAN Beispiel Ergebnisse

```
beta2<-As.mcmc.list(limo,pars="beta[3] ")  
plot(beta2)
```



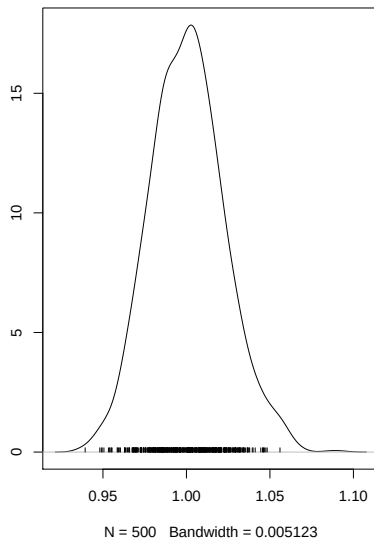
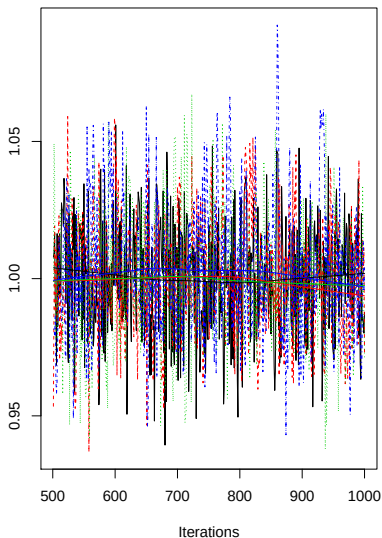
STAN Beispiel Ergebnisse

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



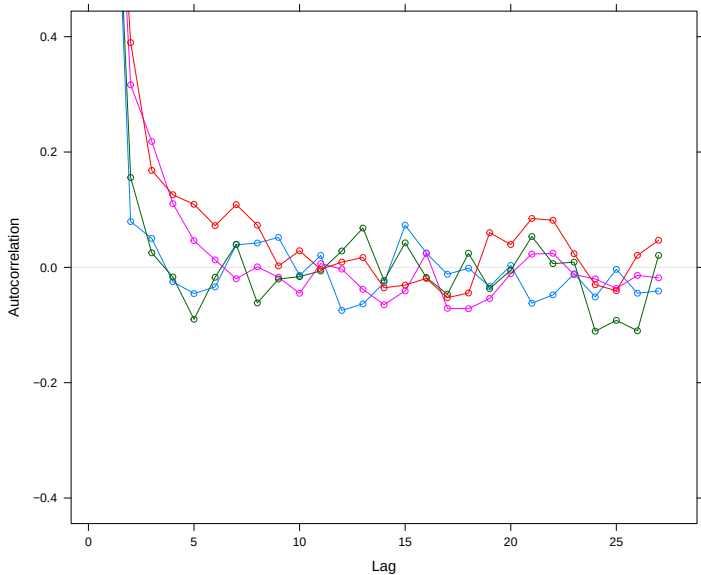
STAN Beispiel Ergebnisse

```
sigma<-As.mcmc.list(limo,pars="sigma")  
plot(sigma)
```



STAN Beispiel Ergebnisse

```
coda::acfplot(sigma)
```



STAN Beispiel Ergebnisse

```
coda::gelman.diag(As.mcmc.list(limo))
```

```
## Potential scale reduction factors:
```

```
##
```

```
##          Point est. Upper C.I.
```

```
## beta[1]          1.00          1.01
```

```
## beta[2]          1.00          1.01
```

```
## beta[3]          1.00          1.01
```

```
## sigma           1.01          1.02
```

```
## lp__            1.00          1.01
```

```
##
```

```
## Multivariate psrf
```

```
##
```

```
## 1.01
```