Series 6

1. **Hamiltonian MC using RStan**

   The goal of this exercise is to use the R package RStan to simulate from the posterior of a Bayesian model. Assume that X is distributed as

   $$X \sim N(\theta, \sigma^2).$$

   For the parameters θ and σ^2, use the following priors

   $$\theta \sim N(\xi, \kappa^2), \quad 1/\sigma^2 \sim Gamma(\gamma, \lambda).$$

   a) Simulate data according to

   $$X \sim N(-2, 3^2),$$

   and use the R package RStan to sample from the posterior distribution. For the hyper-parameters, use

   $$\xi = 0, \kappa = 10000, \gamma = 1, \lambda = 1.$$

   b) Inspect trace plots, autocorrelation as well as the bivariate posterior distribution.

   Hints:

   - Install RStan by following the instructions on https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started
   - Create a .stan file to define your model. Include a "transformed parameters" block in order to define the inverse gamma prior on the variance σ^2.
   - See the R example from the lecture for an example of the use of RStan. For more information, see the stan "user's guide and reference manual" available on http://mc-stan.org/users/documentation/

Solution

a) We frist create the following .stan file.

```
data {
int<lower=0> n; // number of data points
real x[n]; // data
}
parameters {
real theta;
real<lower=0> sigma2i;
}
```

transformed parameters {
real<lower=0> sigma;
sigma=pow(sigma2i,-0.5);
}
model {
theta ∼ normal(0, 10000);
sigma2i ∼ gamma(1,1);
x ∼ normal(theta, sigma);
}

We then use the code below to first simulate data and then to sample from the posterior using stan.

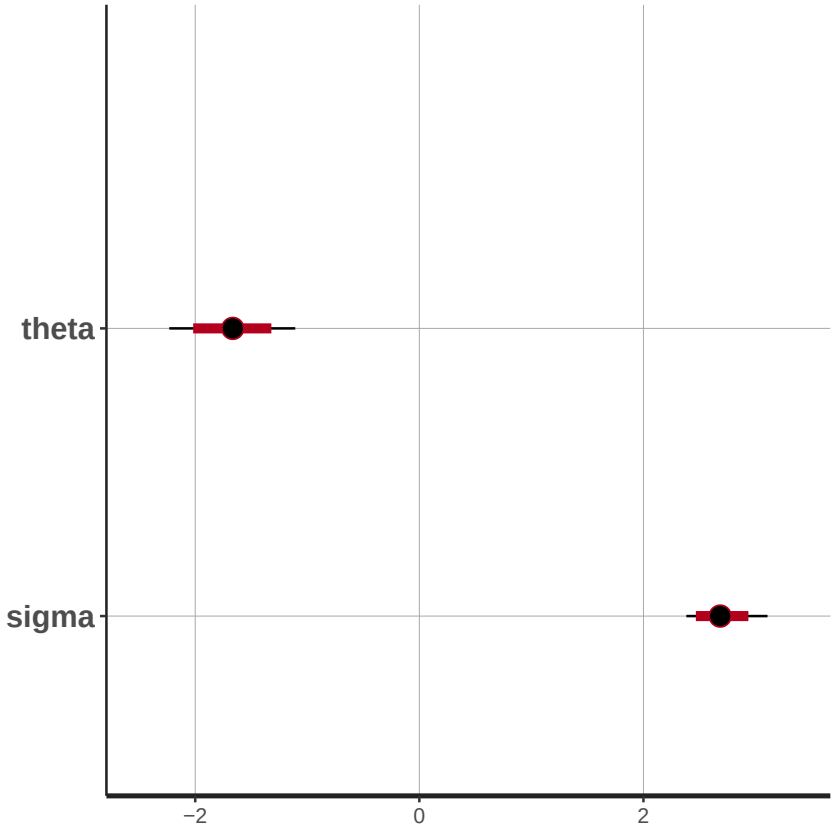
b) See the code and plots below.

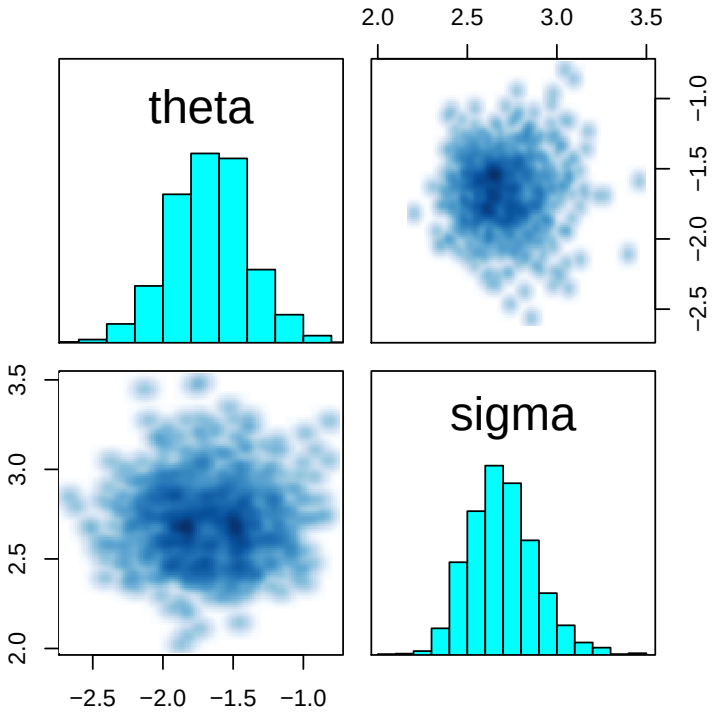
```
library("rstan")
set.seed(1)
x <- rnorm(100,mean=-2,sd=3) #generating data
n <- length(x)
```

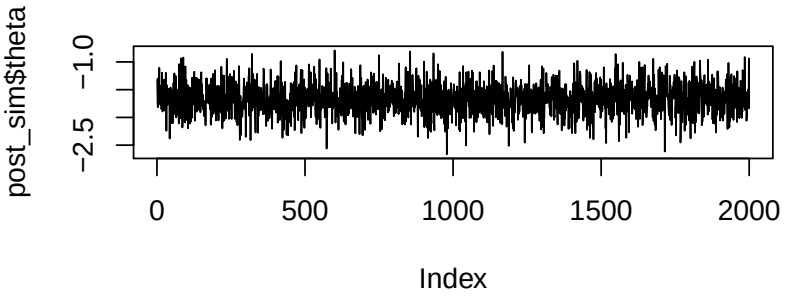
```
post <- stan(file="bayes_normal.stan",data=c("x","n"),
             iter=2500, chains=1,warmup=500)
```

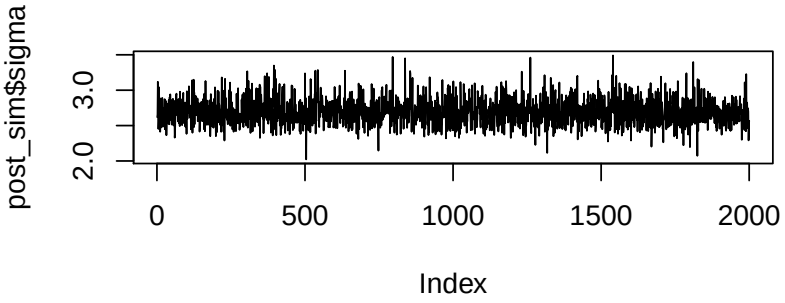
```
##Show summary measures
print(post)

## Inference for Stan model: bayes_normal.
## 1 chains, each with iter=2500; warmup=500; thin=1;
## post-warmup draws per chain=2000, total post-warmup draws=2000.
##
##            mean se_mean   sd    2.5%     25%     50%
## theta     -1.67    0.01 0.28   -2.23   -1.85   -1.67
## sigma2i    0.14    0.00 0.02    0.10    0.13    0.14
## sigma      2.70    0.01 0.19    2.38    2.57    2.68
## lp__    -151.75    0.04 1.09 -154.74 -152.14 -151.40
##             75%   97.5% n_eff Rhat
## theta     -1.48   -1.11  2013    1
## sigma2i    0.15    0.18  1383    1
## sigma      2.81    3.11  1324    1
## lp__    -151.00 -150.75   704    1
##
## Samples were drawn using NUTS(diag_e) at Tue Dec 10 08:55:40 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

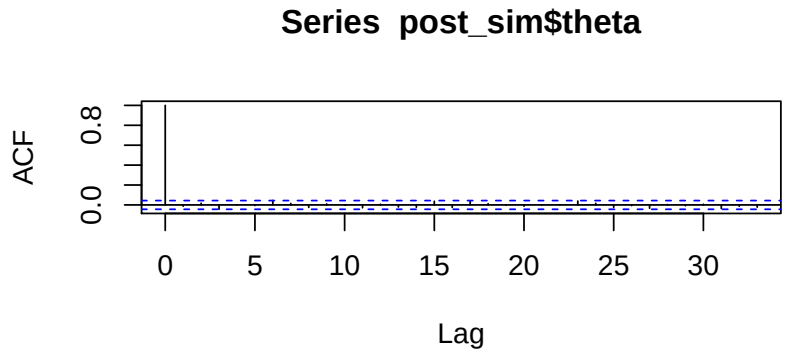
```
##Show univariate posterio densities
plot(post, pars = c("theta", "sigma"))

## ci_level:  0.8 (80% intervals)
## outer_level:  0.95 (95% intervals)

##Inspect bivariate posterior of theta and sigma
pairs(post, pars = c("theta", "sigma"))

##Manually extract simulated values in order to monitor convergence,
##inspect posterior distributions, and run calculations with samples from posterior
post_sim <- extract(post)
##Show trace plots
par(mfrow=c(2,1))
plot(post_sim$theta,type="l")
plot(post_sim$sigma,type="l")
##Show acf
acf(post_sim$theta)
acf(post_sim$sigma)
```

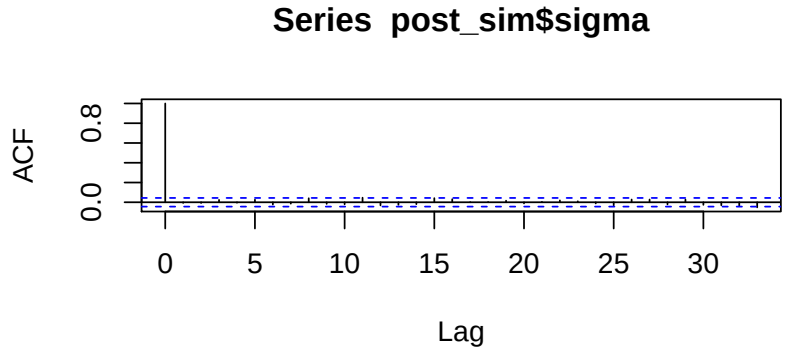

theta
sigma
−2
0
2


theta
sigma
2.0
2.5
3.0
3.5
−1.0
−1.5
−2.0
−2.5


post_sim$theta
−2.5
−1.0
0
500
1000
1500
2000
Index


post_sim$sigma
2.0
3.0
0
500
1000
1500
2000
Index





2. **Own implementation of Hamiltonian MC**

   The goal of this exercise is to implement a Hamiltonian Monte Carlo algorithm. We use the schools data and the same model as shown in the RStan example during the lecture. I.e, we use the following model

   $$y_j \sim N(\theta_j, \sigma_j^2)$$

   and

   $$\theta_j \sim N(\mu, \tau^2),$$

   where y_j are the observed school effects and σ_j^2 are the known standard error estimates. The posterior can be written as

   $$p(\theta, \mu, \tau) \propto \prod_{j=1}^{J} p(y_j|\theta_j, \sigma_j^2) p(\theta_j|\mu, \tau^2),$$

   where we have assumed that $p(\mu, \tau) \propto 1$ and where both $p()$'s are normal densities.

a) Construct a HMC algorithm using the leap-frog method in order to sample from the posterior.

b) Try different values of T, m_i, and the step size ε in order that the algorithm mixes well (in particular, low autocorrelation of the parameter samples). Inspect trace plots, autocorrelation as well as the scatter plots of the posterior distribution.

Hints:

- You can use the following code skeleton.

```
##Read data
schools <- read.csv("schools.csv", header=TRUE)
J <- nrow(schools)
y <- schools$estimate
sigma <- schools$sd

##Log posterior density
log_post <- function(th, y, sigma){
  J <- length(th) - 2
  theta <- th[1:J]
  mu <- th[J+1]
  tau <- th[J+2]
  if (is.nan(tau) | tau<=0)
    return(-Inf)
  else{
    log_prior <- XXXXX
    log_likelihood <- XXXXX
    return(XXXXX)
  }
}
##Gradient of log posterior density
grad_log_post <- function(param, y, sigma){
  J <- length(param) - 2
  theta <- param[1:J]
  mu <- param[J+1]
  tau <- param[J+2]
  if (tau<=0)
    return(c(rep(0,J),0,0))
  else {##calculate gradient
    d_theta <- XXXXX
    d_mu <- XXXXX
    d_tau <- XXXXX
    return(XXXXX)
  }
}
```

```
##Function that does on simulation step and returns the next value of the chain
hmc_iteration <- function(param, y, sigma, epsilon, L, M) {
  M_inv <- 1/M
  d <- length(param)
  u <- rnorm(d, 0, sqrt(M))
  param_old <- param
  log_H_old <- XXXXX
  u <- u + 0.5*epsilon*XXXXX
  for (l in 1:L){
    param <- param + epsilon*XXXXX
    if(l==L) u <- u + 0.5*XXXXX else u <- u + epsilon*XXXXX
  }
  log_H_prop <- XXXXX
  r <- exp(XXXXX)
  if (is.nan(r)) r <- 0
  ##acceptance probability
  p_jump <- XXXXX
  XXXXX
  return(XXXXX)
}
```

- See also the R example from the lecture.

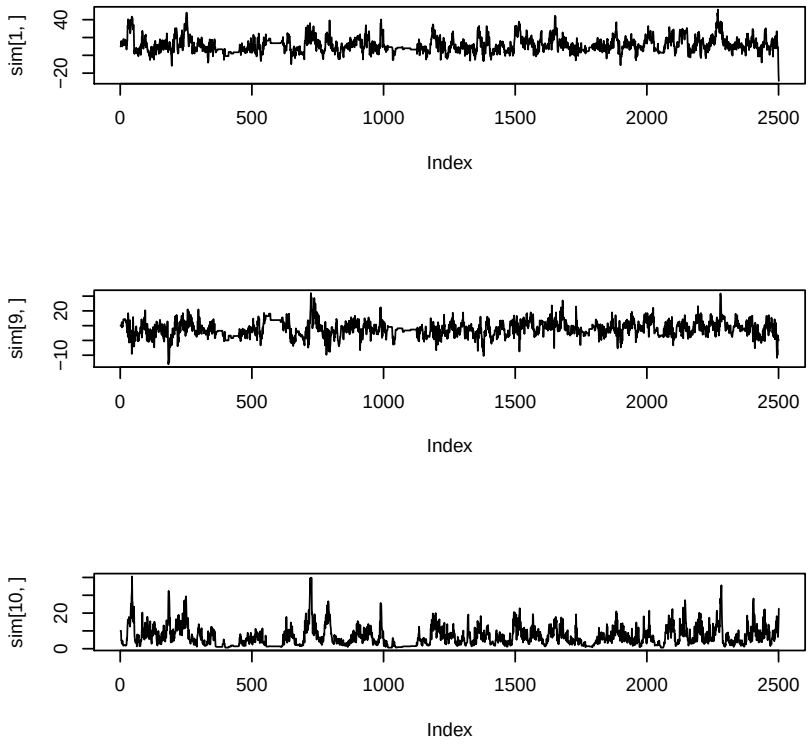
Solution

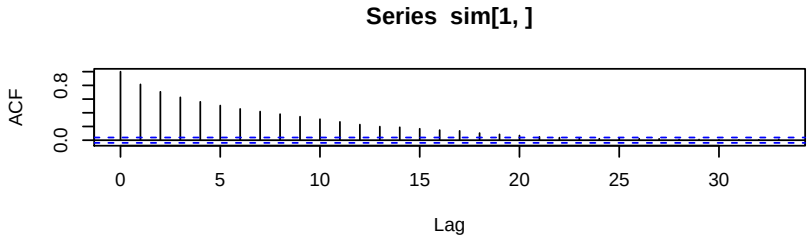
a), b) We define three functions: One that computes the logarithm of the posterior density (log_post), one that computes the gradient of this (grad_log_post), and one that does on HMC sampling step (hmc_iteration). We then run the algorithm and simulate 2500 samples. See the code below for more details.

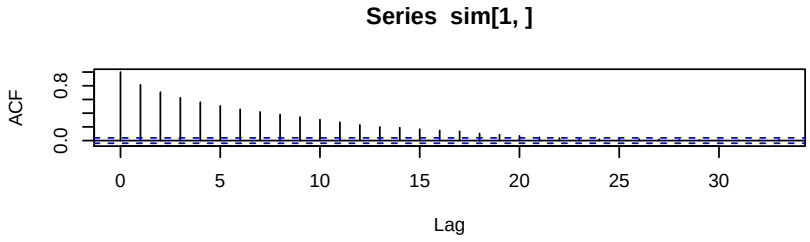
```
##Read data
schools <- read.csv("schools.csv", header=TRUE)
J <- nrow(schools)
y <- schools$estimate
sigma <- schools$sd

##Log posterior density
log_post <- function(th, y, sigma){
  J <- length(th) - 2
  theta <- th[1:J]
  mu <- th[J+1]
  tau <- th[J+2]
  if (is.nan(tau) | tau<=0)
    return(-Inf)
  else{
```

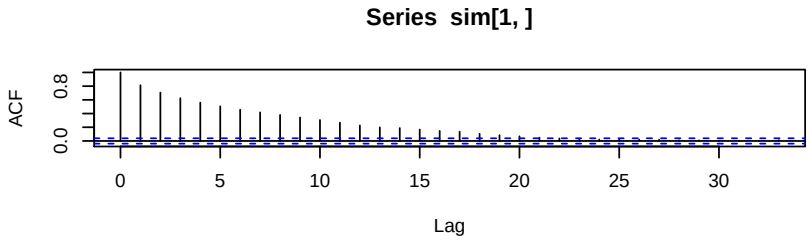
```
    log_prior <- sum(dnorm(theta, mu, tau, log=TRUE))
    log_likelihood <- sum(dnorm(y, theta, sigma, log=TRUE))
    return(log_prior + log_likelihood)
  }
}
##Gradient of log posterior density
grad_log_post <- function(param, y, sigma){
  J <- length(param) - 2
  theta <- param[1:J]
  mu <- param[J+1]
  tau <- param[J+2]
  if (tau<=0)
    return(c(rep(0,J),0,0))
  else {##calculate gradient
    d_theta <- - (theta-y)/sigma^2 - (theta-mu)/tau^2
    d_mu <- -sum(mu-theta)/tau^2
    d_tau <- -J/tau + sum((mu-theta)^2)/tau^3
    return(c(d_theta, d_mu, d_tau))
  }
}
##Function that does on simulation step and returns the next value of the chain
hmc_iteration <- function(param, y, sigma, epsilon, L, M) {
  M_inv <- 1/M
  d <- length(param)
  u <- rnorm(d, 0, sqrt(M))
  param_old <- param
  log_H_old <- log_post(param,y,sigma) - 0.5*sum(M_inv*u^2)
  u <- u + 0.5*epsilon*grad_log_post(param, y, sigma)
  for (l in 1:L){
    param <- param + epsilon*M_inv*u
    u <- u + (if (l==L) 0.5 else 1)*epsilon*grad_log_post(param,y,sigma)
  }
  log_H_prop <- log_post(param,y,sigma) - 0.5*sum(M_inv*u^2)
  r <- exp(log_H_prop - log_H_old)
  if (is.nan(r)) r <- 0
  ##acceptance probability
  p_jump <- min(r,1)
  th_new <- if (runif(1) < p_jump) param else param_old
  return(th_new)
}
##Specify tuning parameters
epsilon <- 0.1
L=20
M <- rep(0.1, J+2)##masses
iter=2500
```

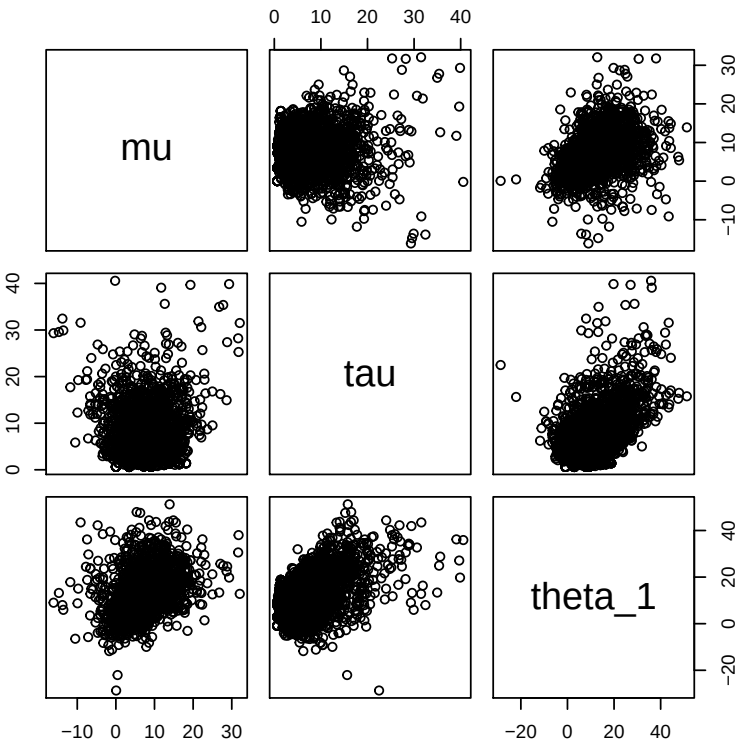
```
##Parameters
sim=matrix(NA,ncol=(iter+1),nrow=(J+2))
##Starting values
sim[,1]=c(rep(10,J),10,10)
rownames(sim)=c(paste("theta_",1:8,sep=""),"mu","tau")
##Run HMC algorithm
set.seed(1)
for (t in 1:iter) sim[,t+1] <- hmc_iteration(param=sim[,t], y, sigma, epsilon, L, M)

##Trace plots
par(mfrow=c(3,1))
plot(sim[1,],type="l")
plot(sim[9,],type="l")
plot(sim[10,],type="l")
##Autocorrelation function
acf(sim[1,])
acf(sim[1,])
acf(sim[1,])
##Scatter plots
pairs(t(sim[c(9,10,1),]))
```


sim[1,]
−20
40
0
500
1000
1500
2000
2500
Index
sim[9,]
−10
20
0
500
1000
1500
2000
2500
Index
sim[10,]
0
20
0
500
1000
1500
2000
2500
Index


Series sim[1,]
ACF
0.0
0.8
0
5
10
15
20
25
30
Lag


Series sim[1,]
ACF
0.0
0.8
0
5
10
15
20
25
30
Lag


Series sim[1,]
ACF
0.0
0.8
0
5
10
15
20
25
30
Lag


mu
tau
theta_1
0 10 20 30 40
−10 0 10 20 30
0 10 20 30 40
−10 0 10 20 30
−20 0 20 40
−20 0 20 40