Linear Regression with Metropolis-Hastings and Stan

We compare a simple Metropolis-Hastings algorithm with Stan for a real simple linear regression problem.

a) Loading of data. You can load the data via:

```r
df = read.csv("https://raw.githubusercontent.com/tensorchiefs/data/main/data/sbp.csv")
```

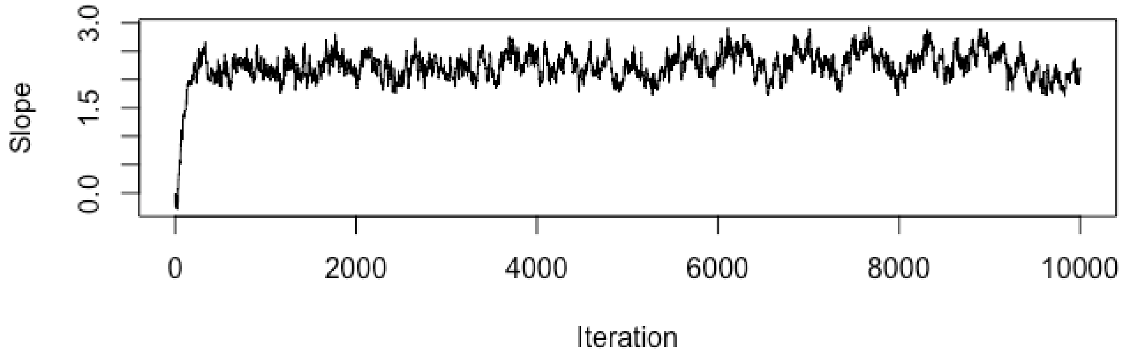
b) The code for the Metropolis-Hastings algorithm can be found at https://github.com/oduerr/da/blob/maste
you can use via

```r
#source('https://raw.githubusercontent.com/oduerr/da/master/code/MCMC_MH.R') #Load the Met
source('~/Documents/GitHub/da/code/MCMC_MH.R') #Load the Metropolis-Hastings code
proposal_sd <- c(0.1, 5, 5)
data = list(N=33, x=df$x, y=df$y, K=1)
n_iter = 10000
initial_values <- c(0, 0, 1)
# Run the Metropolis-Hastings algorithm
samples_hm <- metropolis_hastings(log_posterior, initial_values, data, n_iter, proposal_sd
```

What does the code do, what is the meaning of the variables `proposal_sd`,`n_iter`, `initial_values` and `samples`? Make a trace plot of the slope from the samples.

```r
#The code runs a Metropolis-Hastings algorithm to sample from the posterior distribution of
```

```r
plot(samples_hm[,1], type = "l", xlab = "Iteration", ylab = "Slope")
```



c) Formulate the same problem in Stan, have a lock at the code to come up with the complete code including the priors. Sample one chain from the posterior.

```
data{
  int<lower=0> N;
  vector[N] y;
  vector[N] x;
}

parameters{
  real a;
  real b;
  real<lower=0> sigma;
}

model{
  //y ~ normal(mu, sigma);
  y ~ normal(a * x + b, sigma);
  a ~ normal(0, 10);
  b ~ normal(0, 10);
  sigma ~ uniform(0,100);
}
```

```
library(cmdstanr)
```

```
This is cmdstanr version 0.8.0
```

```
- CmdStanR documentation and vignettes: mc-stan.org/cmdstanr


- CmdStan path: /Users/oli/.cmdstan/cmdstan-2.34.1


- CmdStan version: 2.34.1
```

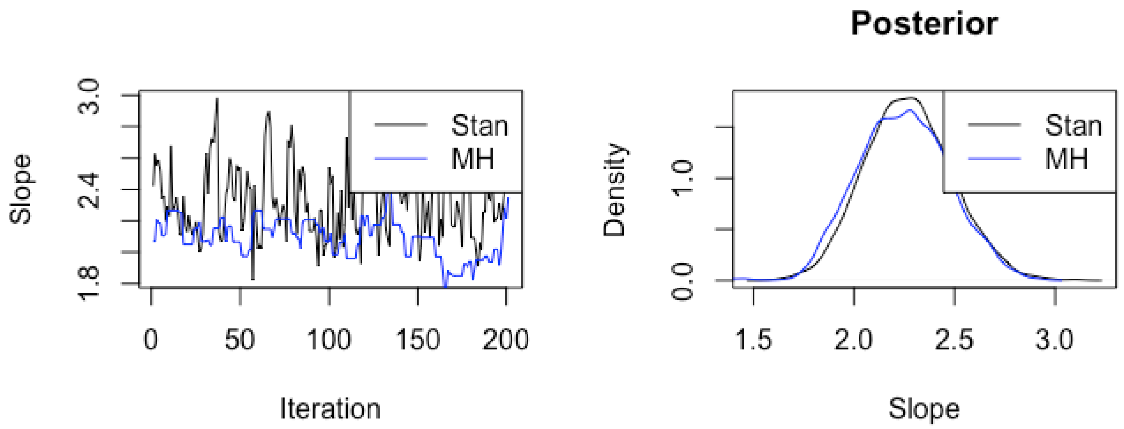
```r
#Reading the model definition
m_rcmdstan <- cmdstan_model('~/Documents/GitHub/da/lab/lr_1_MH_vs_Stan/linear_regression.stan
samples_stan = m_rcmdstan$sample(data=data, chains=1, iter_sampling = 10000)
```

```
Running MCMC with 1 chain...

Chain 1 Iteration:     1 / 11000 [  0%]  (Warmup)
Chain 1 Iteration:   100 / 11000 [  0%]  (Warmup)
Chain 1 Iteration:   200 / 11000 [  1%]  (Warmup)
Chain 1 Iteration:   300 / 11000 [  2%]  (Warmup)
Chain 1 Iteration:   400 / 11000 [  3%]  (Warmup)
Chain 1 Iteration:   500 / 11000 [  4%]  (Warmup)
Chain 1 Iteration:   600 / 11000 [  5%]  (Warmup)
Chain 1 Iteration:   700 / 11000 [  6%]  (Warmup)
Chain 1 Iteration:   800 / 11000 [  7%]  (Warmup)
Chain 1 Iteration:   900 / 11000 [  8%]  (Warmup)
Chain 1 Iteration:  1000 / 11000 [  9%]  (Warmup)
Chain 1 Iteration:  1001 / 11000 [  9%]  (Sampling)
Chain 1 Iteration:  1100 / 11000 [ 10%]  (Sampling)
Chain 1 Iteration:  1200 / 11000 [ 10%]  (Sampling)
Chain 1 Iteration:  1300 / 11000 [ 11%]  (Sampling)
Chain 1 Iteration:  1400 / 11000 [ 12%]  (Sampling)
Chain 1 Iteration:  1500 / 11000 [ 13%]  (Sampling)
Chain 1 Iteration:  1600 / 11000 [ 14%]  (Sampling)
Chain 1 Iteration:  1700 / 11000 [ 15%]  (Sampling)
Chain 1 Iteration:  1800 / 11000 [ 16%]  (Sampling)
Chain 1 Iteration:  1900 / 11000 [ 17%]  (Sampling)
Chain 1 Iteration:  2000 / 11000 [ 18%]  (Sampling)
Chain 1 Iteration:  2100 / 11000 [ 19%]  (Sampling)
Chain 1 Iteration:  2200 / 11000 [ 20%]  (Sampling)
Chain 1 Iteration:  2300 / 11000 [ 20%]  (Sampling)
Chain 1 Iteration:  2400 / 11000 [ 21%]  (Sampling)
Chain 1 Iteration:  2500 / 11000 [ 22%]  (Sampling)
Chain 1 Iteration:  2600 / 11000 [ 23%]  (Sampling)
Chain 1 Iteration:  2700 / 11000 [ 24%]  (Sampling)
Chain 1 Iteration:  2800 / 11000 [ 25%]  (Sampling)
Chain 1 Iteration:  2900 / 11000 [ 26%]  (Sampling)
Chain 1 Iteration:  3000 / 11000 [ 27%]  (Sampling)
Chain 1 Iteration:  3100 / 11000 [ 28%]  (Sampling)
Chain 1 Iteration:  3200 / 11000 [ 29%]  (Sampling)
Chain 1 Iteration:  3300 / 11000 [ 30%]  (Sampling)
Chain 1 Iteration:  3400 / 11000 [ 30%]  (Sampling)
Chain 1 Iteration:  3500 / 11000 [ 31%]  (Sampling)
Chain 1 Iteration:  3600 / 11000 [ 32%]  (Sampling)
Chain 1 Iteration:  3700 / 11000 [ 33%]  (Sampling)
Chain 1 Iteration:  3800 / 11000 [ 34%]  (Sampling)
Chain 1 Iteration:  3900 / 11000 [ 35%]  (Sampling)
Chain 1 Iteration:  4000 / 11000 [ 36%]  (Sampling)
Chain 1 Iteration:  4100 / 11000 [ 37%]  (Sampling)
Chain 1 Iteration:  4200 / 11000 [ 38%]  (Sampling)
Chain 1 Iteration:  4300 / 11000 [ 39%]  (Sampling)
Chain 1 Iteration:  4400 / 11000 [ 40%]  (Sampling)
Chain 1 Iteration:  4500 / 11000 [ 40%]  (Sampling)
Chain 1 Iteration:  4600 / 11000 [ 41%]  (Sampling)
Chain 1 Iteration:  4700 / 11000 [ 42%]  (Sampling)
Chain 1 Iteration:  4800 / 11000 [ 43%]  (Sampling)
Chain 1 Iteration:  4900 / 11000 [ 44%]  (Sampling)
Chain 1 Iteration:  5000 / 11000 [ 45%]  (Sampling)
Chain 1 Iteration:  5100 / 11000 [ 46%]  (Sampling)
Chain 1 Iteration:  5200 / 11000 [ 47%]  (Sampling)
Chain 1 Iteration:  5300 / 11000 [ 48%]  (Sampling)
Chain 1 Iteration:  5400 / 11000 [ 49%]  (Sampling)
Chain 1 Iteration:  5500 / 11000 [ 50%]  (Sampling)
Chain 1 Iteration:  5600 / 11000 [ 50%]  (Sampling)
Chain 1 Iteration:  5700 / 11000 [ 51%]  (Sampling)
Chain 1 Iteration:  5800 / 11000 [ 52%]  (Sampling)
Chain 1 Iteration:  5900 / 11000 [ 53%]  (Sampling)
Chain 1 Iteration:  6000 / 11000 [ 54%]  (Sampling)
Chain 1 Iteration:  6100 / 11000 [ 55%]  (Sampling)
Chain 1 Iteration:  6200 / 11000 [ 56%]  (Sampling)
Chain 1 Iteration:  6300 / 11000 [ 57%]  (Sampling)
Chain 1 Iteration:  6400 / 11000 [ 58%]  (Sampling)
Chain 1 Iteration:  6500 / 11000 [ 59%]  (Sampling)
Chain 1 Iteration:  6600 / 11000 [ 60%]  (Sampling)
Chain 1 Iteration:  6700 / 11000 [ 60%]  (Sampling)
Chain 1 Iteration:  6800 / 11000 [ 61%]  (Sampling)
Chain 1 Iteration:  6900 / 11000 [ 62%]  (Sampling)
Chain 1 Iteration:  7000 / 11000 [ 63%]  (Sampling)
Chain 1 Iteration:  7100 / 11000 [ 64%]  (Sampling)
Chain 1 Iteration:  7200 / 11000 [ 65%]  (Sampling)
Chain 1 Iteration:  7300 / 11000 [ 66%]  (Sampling)
Chain 1 Iteration:  7400 / 11000 [ 67%]  (Sampling)
Chain 1 Iteration:  7500 / 11000 [ 68%]  (Sampling)
Chain 1 Iteration:  7600 / 11000 [ 69%]  (Sampling)
Chain 1 Iteration:  7700 / 11000 [ 70%]  (Sampling)
Chain 1 Iteration:  7800 / 11000 [ 70%]  (Sampling)
Chain 1 Iteration:  7900 / 11000 [ 71%]  (Sampling)
Chain 1 Iteration:  8000 / 11000 [ 72%]  (Sampling)
Chain 1 Iteration:  8100 / 11000 [ 73%]  (Sampling)
Chain 1 Iteration:  8200 / 11000 [ 74%]  (Sampling)
Chain 1 Iteration:  8300 / 11000 [ 75%]  (Sampling)
Chain 1 Iteration:  8400 / 11000 [ 76%]  (Sampling)
Chain 1 Iteration:  8500 / 11000 [ 77%]  (Sampling)
Chain 1 Iteration:  8600 / 11000 [ 78%]  (Sampling)
Chain 1 Iteration:  8700 / 11000 [ 79%]  (Sampling)
Chain 1 Iteration:  8800 / 11000 [ 80%]  (Sampling)
Chain 1 Iteration:  8900 / 11000 [ 80%]  (Sampling)
Chain 1 Iteration:  9000 / 11000 [ 81%]  (Sampling)
Chain 1 Iteration:  9100 / 11000 [ 82%]  (Sampling)
Chain 1 Iteration:  9200 / 11000 [ 83%]  (Sampling)
Chain 1 Iteration:  9300 / 11000 [ 84%]  (Sampling)
Chain 1 Iteration:  9400 / 11000 [ 85%]  (Sampling)
Chain 1 Iteration:  9500 / 11000 [ 86%]  (Sampling)
Chain 1 Iteration:  9600 / 11000 [ 87%]  (Sampling)
Chain 1 Iteration:  9700 / 11000 [ 88%]  (Sampling)
Chain 1 Iteration:  9800 / 11000 [ 89%]  (Sampling)
Chain 1 Iteration:  9900 / 11000 [ 90%]  (Sampling)
Chain 1 Iteration: 10000 / 11000 [ 90%]  (Sampling)
Chain 1 Iteration: 10100 / 11000 [ 91%]  (Sampling)
Chain 1 Iteration: 10200 / 11000 [ 92%]  (Sampling)
Chain 1 Iteration: 10300 / 11000 [ 93%]  (Sampling)
Chain 1 Iteration: 10400 / 11000 [ 94%]  (Sampling)
Chain 1 Iteration: 10500 / 11000 [ 95%]  (Sampling)
Chain 1 Iteration: 10600 / 11000 [ 96%]  (Sampling)
Chain 1 Iteration: 10700 / 11000 [ 97%]  (Sampling)
Chain 1 Iteration: 10800 / 11000 [ 98%]  (Sampling)
Chain 1 Iteration: 10900 / 11000 [ 99%]  (Sampling)
Chain 1 Iteration: 11000 / 11000 [100%]  (Sampling)
Chain 1 finished in 0.2 seconds.
```

```
samples_stan = samples_stan$draws(format = 'df')
```

Compare the trace plots of the slope from the Metropolis-Hastings algorithm and Stan. Also compare the posterior plots. Which of the two algorithms has a larger n_eff?

```
  par(mfrow=c(1,2))
  plot(samples_stan$a[1000:1200], type = "l", xlab = "Iteration", ylab = "Slope")
  lines(samples_hm[1000:1200,1], col='blue')
  legend("topright", legend=c("Stan", "MH"), col=c("black", "blue"), lty=1)

  plot(density(samples_stan$a), main = "Posterior", xlab = "Slope", ylab = "Density")
  lines(density(samples_hm[,1]), col='blue')
  legend("topright", legend=c("Stan", "MH"), col=c("black", "blue"), lty=1)
```


Slope
Stan
MH
Iteration
Posterior
Density
Stan
MH
Slope