STAT-S676

Assignment 2

John Koo

```r
library(ggplot2)
import::from(magrittr, `%>%`, set_colnames)
theme_set(theme_bw())
```

See source code here: https://github.com/johneverettkoo/stats-hw

Problem 1

Most of this code is from Dr. Womack's file lm_mcmc.cpp (from Canvas).

```r
cpp.file <- '~/dev/stats-hw/stat-s676/package/src/lm_mcmc_horseshoe.cpp'
writeLines(readLines(cpp.file))
```

```
// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::depends(RcppGSL)]]

#include<iostream>
#include<armadillo>
#include<RcppArmadillo.h>
#include<gsl/gsl_rng.h>
#include<gsl/gsl_randist.h>


void pseudo_inverse_and_rank
(
  const arma::mat &W,
  arma::mat& p_inv,
  arma::mat& sqrt_pinv_mat,
  unsigned int& rank
)
{
  unsigned int dims = W.n_cols;
  arma::vec EVAL(dims);
  arma::mat EVEC(dims,dims);
  double tol = static_cast<double>(dims)*EVAL.max()*arma::datum::eps;
  arma::eig_sym(EVAL,EVEC,W);
  arma::uvec inds1 = arma::find(EVAL>tol);
  arma::uvec inds0 = arma::find(EVAL<=tol);
  EVAL.elem(inds0).fill(0);
  arma::vec EE=arma::ones<arma::vec>(inds1.n_elem);
  EVAL.elem(inds1)=EE/EVAL.elem(inds1);
  rank = inds1.n_elem;
  sqrt_pinv_mat = EVEC*arma::diagmat(arma::sqrt(EVAL));
  p_inv = sqrt_pinv_mat*sqrt_pinv_mat.t();
}
```

```
arma::vec rand_norm(const gsl_rng * r,unsigned int d){
  arma::vec out(d);
  for(unsigned int i=0;i<d;++i){
    out(i)=gsl_ran_ugaussian_ratio_method(r);
  }
  return(out);
}

// http://gallery.rcpp.org/articles/simulate-multivariate-normal/
arma::mat mvrnormArma(int n, arma::vec mu, arma::mat sigma) {
  int ncols = sigma.n_cols;
  arma::mat Y = arma::randn(n, ncols);
  return arma::repmat(mu, 1, n).t() + Y * arma::chol(sigma);
}

//[[Rcpp::export]]
Rcpp::List lm_mcmc_horseshoe
(
  const arma::vec &y,
  const arma::mat &X_covariates,
  const unsigned int& N_sims,
  const unsigned long int& seed
)
{
  unsigned int p = X_covariates.n_cols;
  unsigned int n = X_covariates.n_rows;
  double p_double = static_cast<double>(p);
  double n_double = static_cast<double>(n);
  double gamma_shape = (n_double + p_double + 2.0) * 0.5;
  double gamma_rate;

  arma::mat X(n,p+1);
  X.col(0) = arma::ones<arma::vec>(n);
  X.cols(1,p) = X_covariates;

  // precompute stuff
  arma::mat XtX = X.t()*X;
  arma::vec Xty = X.t()*y;
  double yty = arma::as_scalar(y.t()*y);
  arma::mat XtX_pinv;
  arma::mat XtX_pinv_sqrt;
  unsigned int rank;
  pseudo_inverse_and_rank(XtX,XtX_pinv,XtX_pinv_sqrt,rank);

  // init
  arma::vec beta = XtX_pinv*Xty;
  // arma::vec beta_mean = beta*n_double/(1+n_double);
  double sigma_sq = yty - arma::as_scalar(Xty.t()*beta);
  arma::vec tau = arma::ones<arma::vec>(p+1);
  arma::vec lambda = arma::ones<arma::vec>(p+1);
  double phi = 1.0;
  double eta = 1.0;

  // outputs
```

```
  arma::vec sigma_sq_out(N_sims);
  arma::mat beta_out(N_sims, p+1);
  arma::mat tau_out(N_sims, p+1);

  // set up rng
  const gsl_rng_type * T;
  gsl_rng * r;
  gsl_rng_env_setup();
  T = gsl_rng_mt19937;
  r = gsl_rng_alloc (T);
  gsl_rng_set(r,seed);

  for(unsigned int i = 0; i < N_sims; ++i) {
    // draw beta
    beta = mvrnormArma(
      1,
      arma::vectorise(arma::inv(XtX + arma::diagmat(tau)) * Xty),
      sigma_sq * arma::inv(XtX + arma::diagmat(tau))
    ).t();
    beta_out.row(i) = beta.t();

    // draw sigma_sq
    gamma_rate = arma::as_scalar(
      0.5 * dot(y - X * beta, y - X * beta) +
      dot(tau % beta, beta)
    ) + 1.0;
    sigma_sq = 1.0 / R::rgamma(gamma_shape, 1.0 / gamma_rate);
    sigma_sq_out(i) = sigma_sq;

    // draw tau and lambda
    for (unsigned int j = 0; j < p+1; ++j) {
      tau(j) = arma::as_scalar(
        R::rexp(1.0) /
          arma::as_scalar(beta(j) * beta(j) / 2.0 / sigma_sq + lambda(j) / 2.0)
      );
      lambda(j) = R::rexp(1.0) / arma::as_scalar(tau(j) / 2.0 + phi / 2.0);
    }
    tau_out.row(i) = tau.t();

    // draw phi
    phi = R::rgamma((p_double + 1.0) / 2.0,
                    1.0 / (arma::sum(lambda) / 2.0 + eta / 2.0));

    // draw eta
    eta = R::rexp(phi / 2.0 + 0.5);
  }

  gsl_rng_free(r);

  return(Rcpp::List::create(Rcpp::Named("beta")=beta_out,
                            Rcpp::Named("sigma_sq")=sigma_sq_out,
                            Rcpp::Named("tau")=tau_out));
}
```

Next, let's compile and install, and then see if it works by comparing it to R's built-in `lm` function:

```r
Rcpp::compileAttributes('~/dev/stats-hw/stat-s676/package')
devtools::install('~/dev/stats-hw/stat-s676/package')

library(Stat676pack)

y <- iris$Petal.Width
X <- as.matrix(dplyr::select(iris, Petal.Length, Sepal.Width, Sepal.Length))
summary(lm(y ~ X))
```

```
Call:
lm(formula = y ~ X)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.60959 -0.10134 -0.01089  0.09825  0.60685 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)   -0.24031    0.17837  -1.347     0.18    
XPetal.Length  0.52408    0.02449  21.399  < 2e-16 ***
XSepal.Width   0.22283    0.04894   4.553 1.10e-05 ***
XSepal.Length -0.20727    0.04751  -4.363 2.41e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.192 on 146 degrees of freedom
Multiple R-squared:  0.9379,	Adjusted R-squared:  0.9366 
F-statistic: 734.4 on 3 and 146 DF,  p-value: < 2.2e-16
```

```r
out <- lm_mcmc_horseshoe(y, X, 10000, 676)
summary(as.data.frame(out$beta[9001:1000, ]))
```

```
       V1                  V2               V3                V4          
 Min.   :-0.939445   Min.   :0.4101   Min.   :-0.0348   Min.   :-0.40323  
 1st Qu.:-0.245581   1st Qu.:0.4959   1st Qu.: 0.1467   1st Qu.:-0.23876  
 Median :-0.099362   Median :0.5164   Median : 0.1869   Median :-0.20168  
 Mean   :-0.135756   Mean   :0.5161   Mean   : 0.1866   Mean   :-0.20090  
 3rd Qu.:-0.003894   3rd Qu.:0.5365   3rd Qu.: 0.2274   3rd Qu.:-0.16407  
 Max.   : 0.473285   Max.   :0.6289   Max.   : 0.4418   Max.   : 0.02277  
```

```r
sigma(lm(y ~ X))
```

```
[1] 0.1919671
```

```r
summary(out$sigma[901:1000])
```

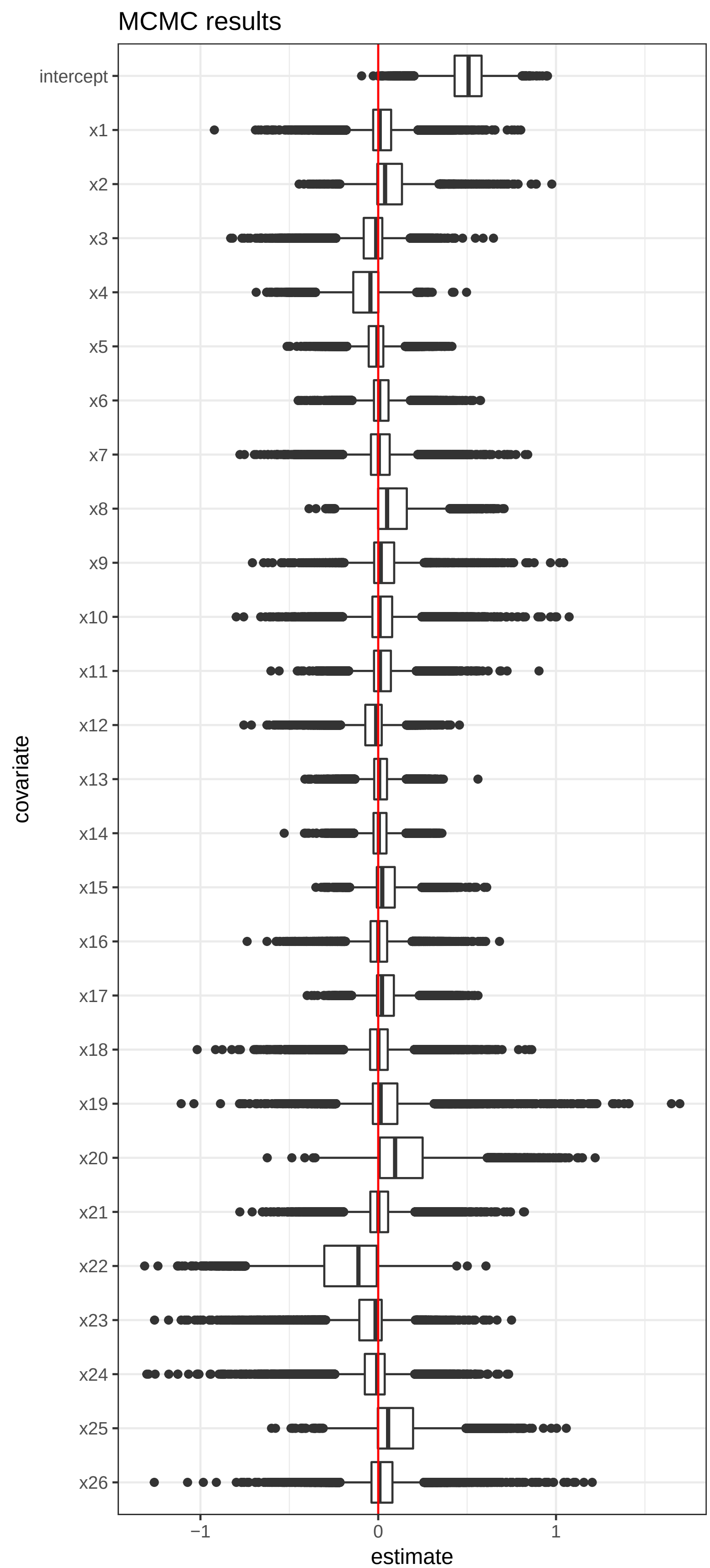
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
0.03835 0.04856 0.05150 0.05278 0.05562 0.07167 
```

Problem 2

```r
# load the data
load('~/dev/stats-hw/stat-s676/pyrimidine.RData')
X <- unname(as.matrix(dplyr::select(D, -y)))
y <- D$y

# build MCMC model
out <- lm_mcmc_horseshoe(y, X, 10000, 314159)
```

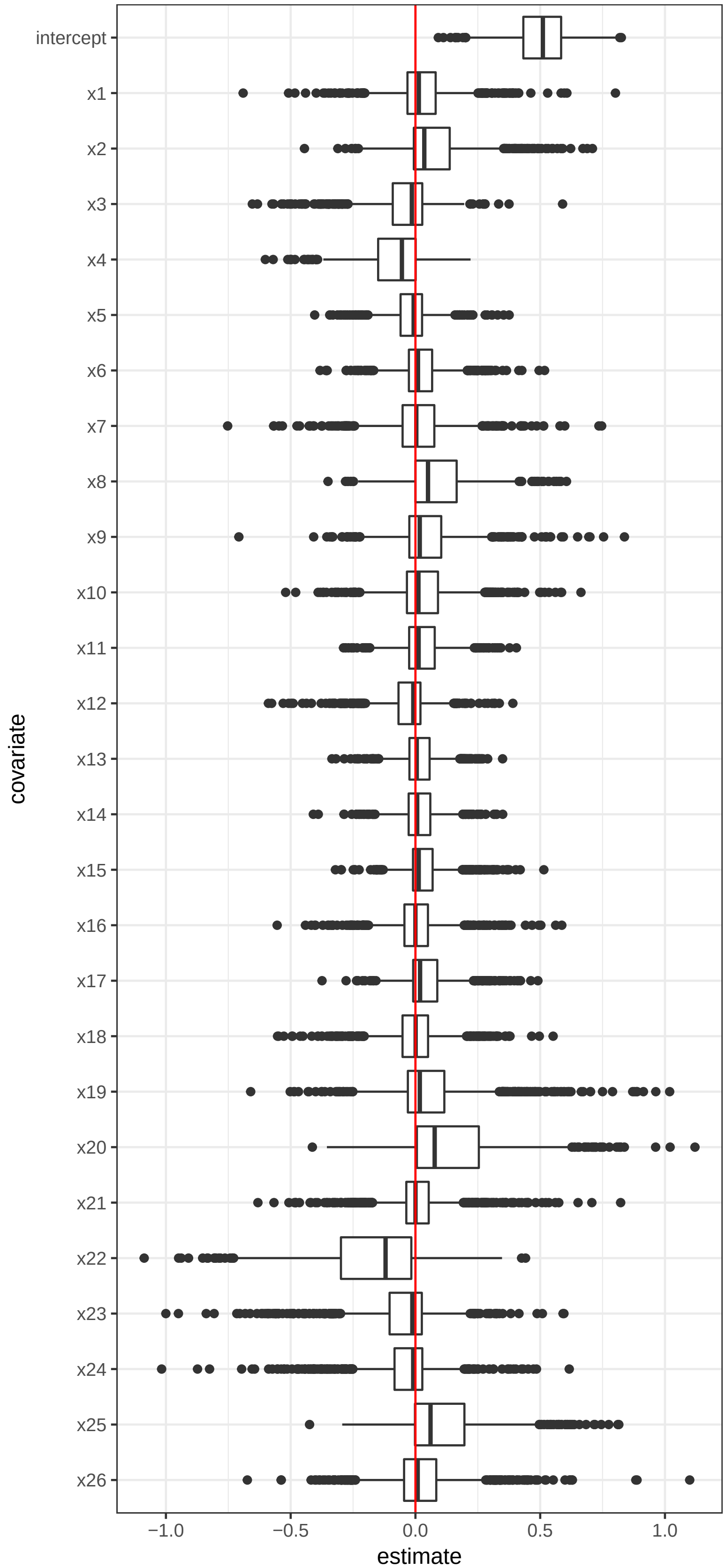
We can see if any of them are significantly not 0.

```r
beta.df <- as.data.frame(out$beta) %>%
  set_colnames(c('intercept', colnames(D)[-ncol(D)]))
tau.df <- as.data.frame(out$tau) %>%
  set_colnames(c('intercept', colnames(D)[-ncol(D)]))
sigma_sq <- out$sigma_sq

beta.df %>%
  # to long df
  tidyr::gather('covariate', 'estimate', 1:27) %>%
  # make sure the labels are plotted in order
  dplyr::mutate(covariate = factor(covariate,
                                   levels = c(paste0('x', seq(26, 1)),
                                              'intercept'))) %>%
  ggplot() +
  geom_boxplot(aes(x = covariate, group = covariate, y = estimate)) +
  geom_hline(yintercept = 0, colour = 'red') +
  coord_flip() +
  labs(title = 'MCMC results')
```


MCMC results
intercept
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
x16
x17
x18
x19
x20
x21
x22
x23
x24
x25
x26
covariate
−1
0
1
estimate

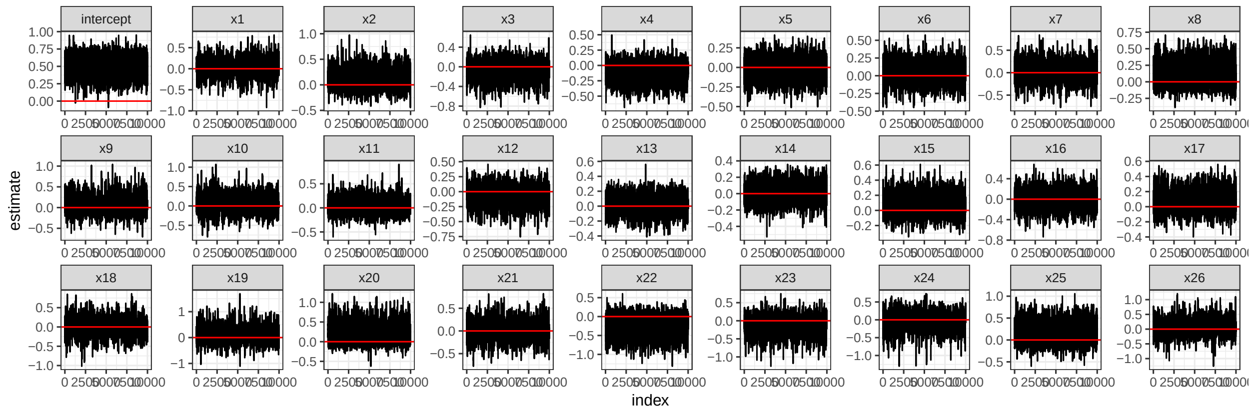
```r
beta.df[9001:10000, ] %>%  # just the last 1000
  tidyr::gather('covariate', 'estimate', 1:27) %>%
  dplyr::mutate(covariate = factor(covariate,
                                   levels = c(paste0('x', seq(26, 1)),
                                              'intercept'))) %>%
  ggplot() +
  geom_boxplot(aes(x = covariate, group = covariate, y = estimate)) +
  geom_hline(yintercept = 0, colour = 'red') +
  coord_flip() +
  labs(title = 'MCMC results (last 1000 results)')
```

MCMC results (last 1000 results)

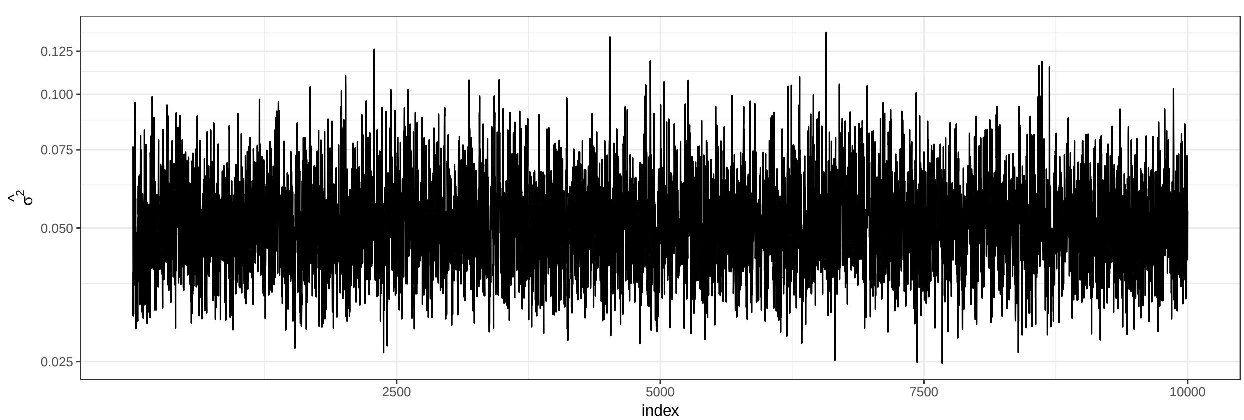


And we can see if each covariate found some steady-state-ish solution.

```
beta.df %>%
  tidyr::gather('covariate', 'estimate', 1:27) %>%
  dplyr::mutate(covariate = factor(covariate,
                                   levels = c('intercept',
                                              paste0('x', seq(26)))),
                index = rep(seq(10000), 27)) %>%
  ggplot() +
  geom_line(aes(x = index, y = estimate)) +
  geom_hline(yintercept = 0, colour = 'red') +
  facet_wrap(~ covariate, scales = 'free', nrow = 3)
```



```
ggplot() +
  geom_line(aes(x = seq(length(sigma_sq)), y = sigma_sq)) +
  labs(x = 'index', y = expression(hat(sigma^2))) +
  coord_trans(y = 'log10')
```

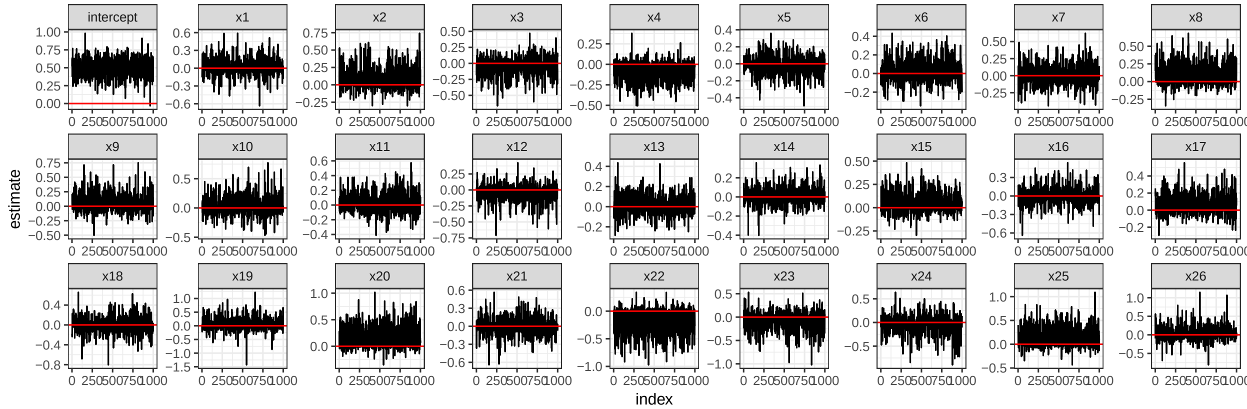


It might be better to implement burn-in and thinning to account for the time it takes to converge as well as ACF:

```
#' @title Horseshoe Gibbs sampler for linear models
#' @description This is just a wrapper for the C++ code.
#' @param y (numeric) An n-dimensional vector of responses
#' @param X (numeric) An n by p dimensional matrix of inputs
#' @param n.iter (numeric) The number of iterations to output
```

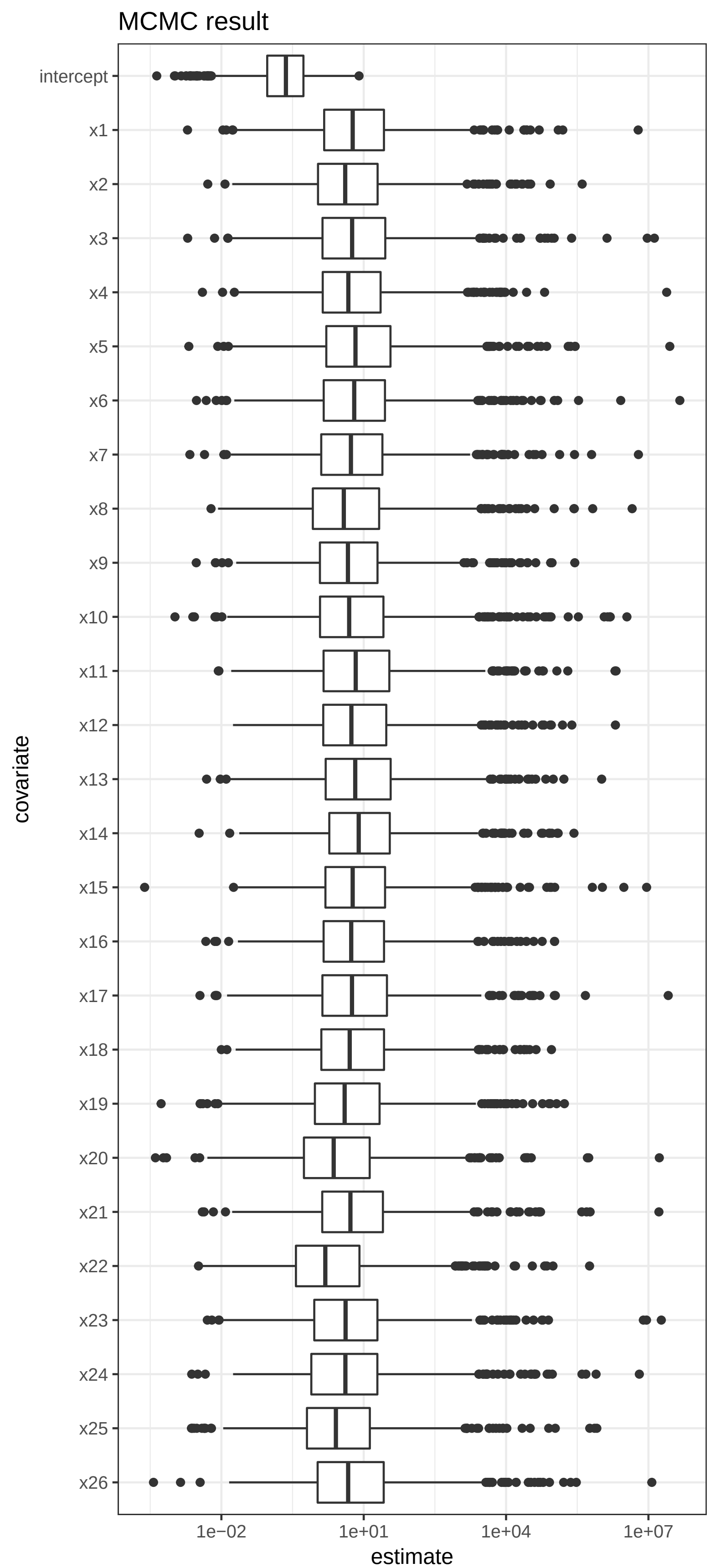
```r
#' @param burn (numeric) Number of iterations for burn-in
#' @param thin (numeric) Thinning rate
#' @param seed (numeric) RNG seed (defaults to system time)
#' @return (list) The estimates for the covariates, sigma^2, and tau
#' @export gibbs.horseshoe
gibbs.horseshoe <- function(y, X,
                            n.iter = 1e3, burn = 1e2, thin = 1e2,
                            seed = NULL) {
  # rng stuff
  if (is.null(seed)) seed <- as.numeric(Sys.time())

  # pass to C++ code
  out <- lm_mcmc_horseshoe(y, X, burn + n.iter * thin, seed)

  # take out burn-in
  beta. <- out$beta[-seq(burn), ]
  sigma_sq <- out$sigma_sq[-seq(burn)]
  tau <- out$tau[-seq(burn), ]

  # thin
  beta. <- beta.[seq(to = n.iter * thin, by = thin), ]
  sigma_sq <- sigma_sq[seq(to = n.iter * thin, by = thin)]
  tau <- tau[seq(to = n.iter * thin, by = thin), ]

  return(list(beta = beta.,
              sigma_sq = sigma_sq,
              tau = tau))
}

out <- Stat676pack::gibbs.horseshoe(y, X, n.iter = 1e3)

beta.df <- as.data.frame(out$beta) %>%
  set_colnames(c('intercept', colnames(D)[-ncol(D)]))
tau.df <- as.data.frame(out$tau) %>%
  set_colnames(c('intercept', colnames(D)[-ncol(D)]))
sigma_sq <- out$sigma_sq
```

```r
beta.df %>%
  tidyr::gather('covariate', 'estimate', 1:27) %>%
  dplyr::mutate(covariate = factor(covariate,
                                   levels = c('intercept',
                                              paste0('x', seq(26)))),
                index = rep(seq(1e3), 27)) %>%
  ggplot() +
  geom_line(aes(x = index, y = estimate)) +
  geom_hline(yintercept = 0, colour = 'red') +
  facet_wrap(~ covariate, scales = 'free', nrow = 3)
```



We can also see how the τs are distributed:

```
tau.df %>%
  tidyr::gather('covariate', 'estimate', 1:27) %>%
  dplyr::mutate(covariate = factor(covariate,
                                    levels = c(paste0('x', seq(26, 1)),
                                               'intercept'))) %>%
  ggplot() +
  geom_boxplot(aes(x = covariate, group = covariate, y = estimate)) +
  coord_flip() +
  labs(title = 'MCMC result') +
  scale_y_log10()
```


MCMC result
intercept
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
x16
x17
x18
x19
x20
x21
x22
x23
x24
x25
x26
covariate
1e−02
1e+01
1e+04
1e+07
estimate

```r
tau.df %>%
  tidyr::gather('covariate', 'estimate', 1:27) %>%
  dplyr::mutate(covariate = factor(covariate,
                                   levels = c('intercept',
                                              paste0('x', seq(26))))) %>%
  dplyr::group_by(covariate) %>%
  dplyr::summarise_all(median) %>%
  dplyr::ungroup() %>%
  ggplot() +
  geom_point(aes(x = covariate, y = estimate)) +
  geom_path(aes(x = covariate, y = estimate)) +
  labs(y = expression(tau))
```