hw4

Jiahao Tian

2023-03-13

Problem: Consider Bayesian inference for data arising from counting processes. Specifically, given a set of covariates $X_i \in R^p$, and an associated set of regression coefficients $\beta \in R^p$, assume:

$$
\begin{aligned}
Y_i|\lambda_i &\sim Poi(e^{\lambda_i}) \\
\lambda_i|\beta,\tau &\sim N(X_i'\beta,\tau) \\
\beta &\sim N(0,(X'X)^{-1}g) \\
\tau &\sim IG(1,1)
\end{aligned}
$$

a) Describe an MCMC strategy aimed at obtaining samples from the posterior distribution: $p(\beta,\tau|Y)$.

- By Gibbs sampling, we first need to find out the full conditional posterior distribution for λ, β, τ.
- Full conditional distributions have the same starting point: the full joint posterior distribution. Thus, the process of finding full conditional distributions is the same as finding the posterior distribution of each parameter.
- From the joint posterior distribution $p(\beta,\tau|Y)$, we can get:

a. Full conditional posterior distribution for β

$$
\begin{aligned}
p(\beta|\lambda_i,\tau) &= p(\lambda_i|\beta)P(\beta) \\
&\propto \prod \exp(-\frac{(\lambda_i - X_i\beta)^2}{2\tau})\exp(-\frac{1}{2g}\beta^{'}(X'X)\beta) \\
&= \exp(-\frac{1}{2\tau}(\beta^{'}X^{'}X\beta - 2\beta X^{'}\lambda_i) - \frac{1}{2g}\beta^{'}(X'X)\beta) \\
&= \exp(-\frac{1}{2}(\frac{\beta^{'}X^{'}X\beta - 2\beta X^{'}\lambda_i}{\tau} - \beta^{'}(X'X)g^{-1}\beta) \\
&= \exp(-\frac{1}{2}(\frac{\beta^{'}(X^{'}X + (X'X)g^{-1})\beta}{\tau} - 2\beta X^{'}\lambda_i)) \\
&= \exp(\frac{1}{2}[\beta' X' X(\frac{1}{g} + \frac{1}{\tau})\beta - 2\frac{1}{g}\beta' X'\lambda]) \\
which\ follows &\ \sim N(\frac{X^{'}\lambda_i\tau}{(X'X)\tau + (X'X)g}, \frac{g\tau}{(X'X)\tau + (X'X)g})
\end{aligned}
$$

b. Full conditional posterior distribution for τ

$$
\begin{aligned}
p(\tau|\lambda_i,\beta) &= p(\lambda_i|\tau)P(\tau) \\
&\propto \prod \exp(-\frac{(\lambda_i - X_i\beta)^2}{2\tau})\ \tau^{-\frac{n}{2}}\tau^{-1-1}\exp(-\frac{1}{\tau}) \\
&= \tau^{-1-1}\tau^{-\frac{n}{2}}\exp(-\frac{1}{2\tau}(\lambda_i - X_i\beta)'(\lambda_i - X_i\beta) - \frac{1}{\tau}) \\
&= \tau^{-1-1}\tau^{-\frac{n}{2}}\exp(-\frac{1}{\tau}(\frac{(\lambda_i - X_i\beta)'(\lambda_i - X_i\beta)}{2} + 1) \\
&which\ follows\ \ \sim IG\ (\ 1+\frac{n}{2},\ 1+\frac{1}{2}((\lambda_i - X_i\beta)'(\lambda_i - X_i\beta))
\end{aligned}
$$

c. Full conditional posterior distribution for λ

$$
\begin{aligned}
p(\lambda_i \mid \beta,\tau,Y_i) &= p(Y_i \mid \lambda_i)p(\lambda_i \mid \beta,\tau) \\
&= \prod_{i=1}^{n} \frac{exp(e^{\beta})e^{\beta Y_i}}{Y!} \cdot \frac{1}{\sqrt{2\pi\tau}} exp(-\frac{(\lambda_i - X_i\beta)^2}{2\tau}) \\
&because\ it\ is\ poisson\ distribution\ e^{\lambda_i}\ needs\ to\ be\ greater\ then\ 0 \\
&\propto I_{e^{\lambda_i}\geq 0} \cdot exp(-\frac{(\lambda_i - X_i\beta)^2}{2\tau}) exp(\lambda_i Y_i) exp(-e^{\lambda_i})
\end{aligned}
$$

- Then we can take turns sampling these distributions like so:

a. Using $\lambda_{i-1},\beta_{i-1}$, draw τ_i from $p(\tau|\beta=\beta_{i-1},\lambda=\lambda_{i-1})$.
b. Using λ_{i-1},τ_{i-1}, draw β_i from $p(\beta|\tau=\tau_{i-1},\lambda=\lambda_{i-1})$.
c. Using β_{i-1},τ_{i-1}, draw λ_i from $p(\lambda_i|\tau=\tau_{i-1},\beta=\beta_{i-1})$.
d. Together, steps 1, 2 and 3 complete one cycle of the Gibbs sampler and produce the draw for $(\beta_i,\tau_i,\lambda_i)$ in one iteration of a MCMC sampler.

b) Describe an HMC strategy aimed at obtaining samples from the posterior distribution: $p(\beta,\tau|Y)$.

- We first need to find out $\nabla U(x),\ U(x) = -log\pi(x)$.
- a. for β.

$$
\begin{aligned}
p(\beta|\lambda_i,\tau) &= p(\lambda_i|\beta)P(\beta) \\
&\propto \exp(-\frac{1}{2}(\frac{\beta^{'}(X^{'}X+(X'X)g^{-1})\beta}{\tau} - 2\beta X^{'}\lambda_i)) \\
&which\ follows\ \ \sim N(\frac{X^{'}\lambda_i\tau}{(X'X)\tau+(X'X)g}, \frac{g\tau}{(X'X)\tau+(X'X)g})
\end{aligned}
$$

$$
\nabla(-log\ p(\beta|\lambda_i,\tau)) \propto \beta(\frac{(X^{'}X+(X'X)g^{-1})}{\tau} - X'\lambda_i)
$$

- b. for τ.

$$
\begin{aligned}
p(\tau|\lambda_i,\beta) = p(\lambda_i|\tau)P(\tau) \\
\propto \tau^{-1-1}\tau^{-\frac{n}{2}}\exp(-\frac{1}{\tau}(\frac{(\lambda_i - X_i\beta)'(\lambda_i - X_i\beta)}{2} + 1) \\
which\ follows\ \ \sim IG\ (\ 1+\frac{n}{2},\ 1+\frac{1}{2}((\lambda_i - X_i\beta)'(\lambda_i - X_i\beta)) \\
\nabla(-log\ p(\tau|\lambda_i,\beta)) \propto \frac{\frac{n}{2}+2}{\tau} - \frac{(\lambda_i - X_i\beta)'(\lambda_i - X_i\beta)}{2\tau^2} - \frac{1}{\tau^2}
\end{aligned}
$$

- c. for λ.

$$
\begin{aligned}
p(\lambda_i \mid \beta,\tau,Y_i) = p(Y_i \mid \lambda_i)p(\lambda_i \mid \beta,\tau) \\
\propto exp(e^{\lambda_i} + \lambda_i Y_i - \frac{(\lambda_i - X_i\beta)^2}{2\tau}) \\
\nabla(-log\ p(\lambda_i|\tau,\beta,Y_i)) \propto \frac{2(\lambda_i - X_i\beta)}{2\tau} - e^{\lambda_i} + Y_i
\end{aligned}
$$

- After finds out $\nabla U(x)$ for each parameters, we can use leapfrog then implement HMC sampler in MCMC.

c) Test algorithms in (a) and (b) on the following data:

```
y = c(18,17,15,20,10,20,25,13,12)
x1 = gl(3,1,9)
x2 = gl(3,3)
dat = data.frame(y, x1, x2)

y = dat$y
X = cbind(rep(1, length(y)), dat$x1, dat$x2)
```

From part a):

```
update_beta = function (g, X, lamb, tau) {
  xtx = solve(t(X) %*% X)
  bmean = g / (g + tau) * xtx %*% t(X) %*% lamb
  bvar = tau * g / (g + tau) * xtx
  rmvnorm(1, mean = bmean, sigma = bvar)
}
```

```
update_tau = function (n, X, beta, lamb) {
  xb = X %*% t(beta)
  shape_1 = n / 2 + 1
  rate_1 = 1 + 1 / 2 * t(lamb - xb) %*% (lamb - xb)
  1 / rgamma(1, shape = shape_1, rate = rate_1 / 2)
}
```

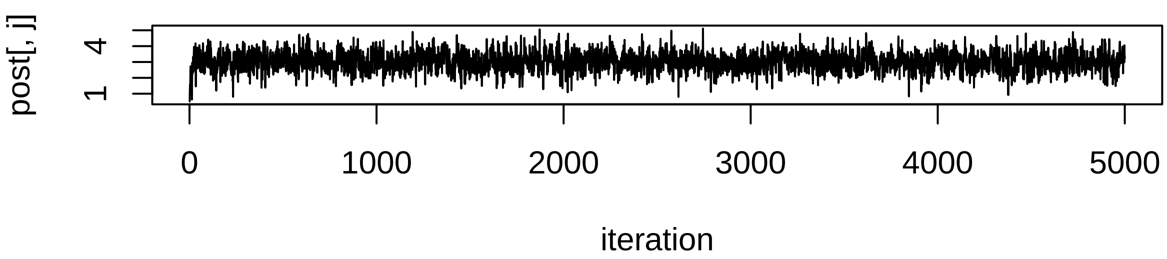
```r
## log lambda
update_lg_lamb = function(lambi, Xi, yi, beta, tau) {
  return(-1 / (2 * tau) * (lambi - Xi %*% t(beta))^2 + lambi * yi - exp(lambi))
}
```

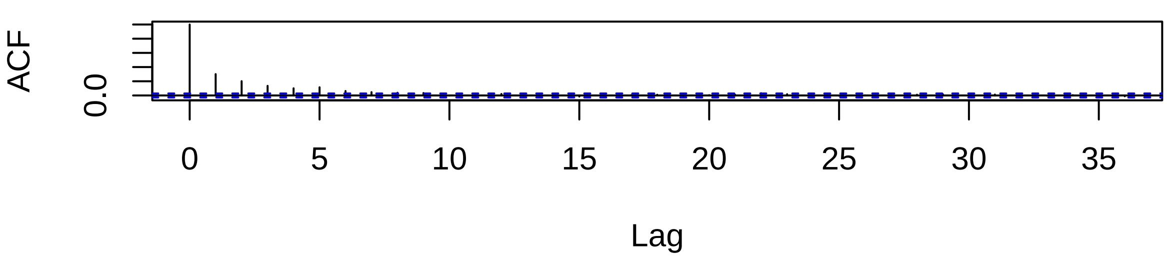
```r
gibs = function (n_iter, burn, X, y, cand_sd) {

  n = length(y)
  p = ncol(X)
  g = n

  ## initialize
  lamb_now = rep(1, n)
  beta_now = rep(1, p)
  tau_now = 1

  ## save chain
  beta_out = matrix(NA, nrow = n_iter, ncol = p)
  lamb_out = matrix(NA, nrow = n_iter, ncol = n)
  tau_out = rep(NA, n_iter)


  ## Gibbs sampler
  for(i in 1:n_iter) {

    # beta
    beta_now = update_beta(g = g, X = X, lamb = lamb_now, tau = tau_now)

    # tau
    tau_now = update_tau(n = n, X = X, beta = beta_now, lamb = lamb_now)

    # Random-Walk Metropolis-Hastings for lambda
    for(idx in 1:n) {

      ## step 1, initialize
      lambi = lamb_now[idx]
      Xi = X[idx, ]
      yi = y[idx]

      ## step 2, iterate
      lamb_cand = rnorm(1, lambi, cand_sd) # draw a candidate
      lamb_0 = update_lg_lamb(lambi = lambi,
                              Xi = Xi,
                              yi = yi,
                              beta = beta_now,
                              tau = tau_now)
      # evaluate with the candidate
      lamb_1 = update_lg_lamb(lambi = lamb_cand,
                              Xi = Xi,
                              yi = yi,
                              beta = beta_now,
                              tau = tau_now)
      ratio = lamb_1 - lamb_0 # log of acceptance ratio
```

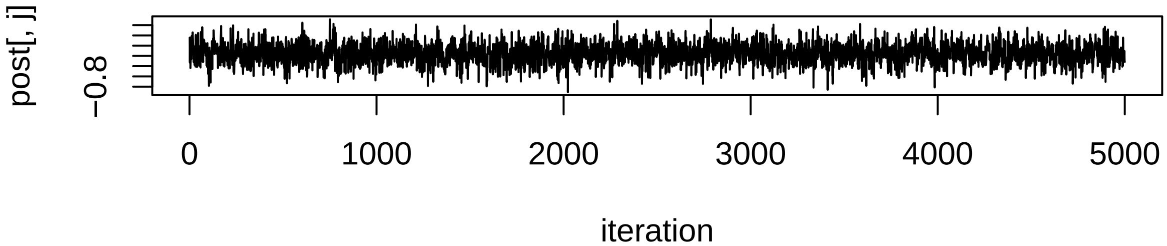
```
      if (log(runif(1)) < ratio) { # accept the candidate
        lamb_now[idx] = lamb_cand
      }
      }
    ## collect results
    beta_out[i,] = beta_now
    lamb_out[i,] = lamb_now
    tau_out[i] = tau_now

  }

  cbind(beta = beta_out, tau = tau_out, lamb = lamb_out)

}
```

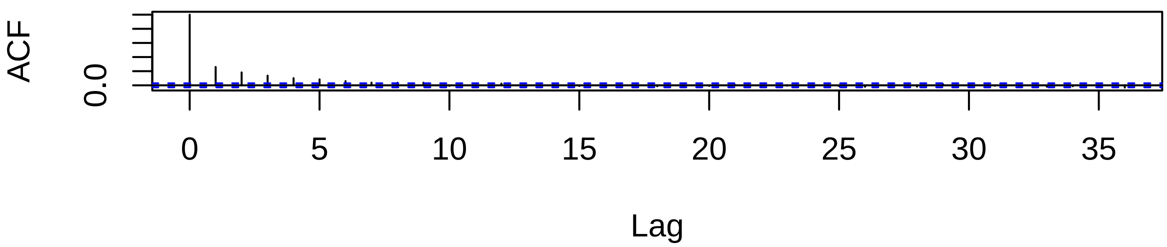
```
set.seed(11111)

post = gibs(n_iter = 5000, burn = 1000, X = X, y = y, cand_sd = 0.5)
```

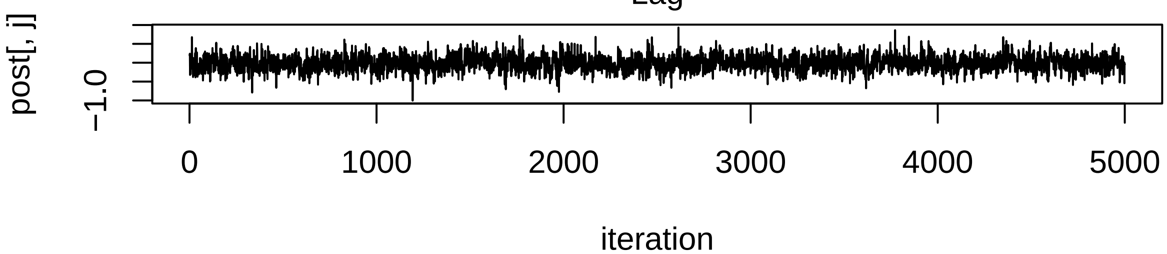
```
par(mfrow = c(2,1))
for(j in 1:13){
matplot(c(1:5000), post[,j], type = "l", xlab = "iteration")
acf(post[, j], main = colnames(post)[j]) }
```

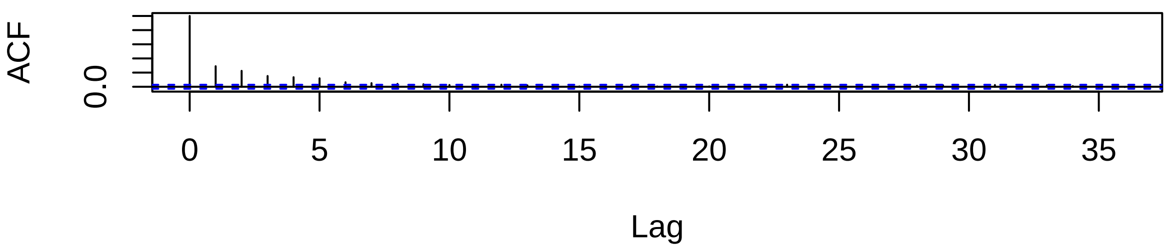


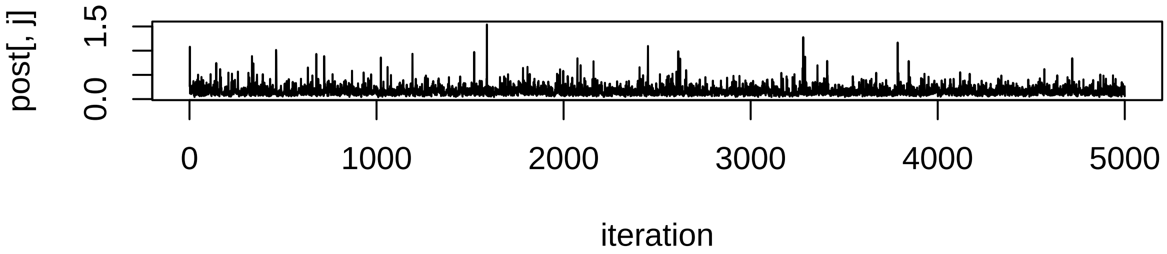



post[, j]
−0.8
0
1000
2000
3000
4000
5000
iteration


ACF
0.0
0
5
10
15
20
25
30
35
Lag

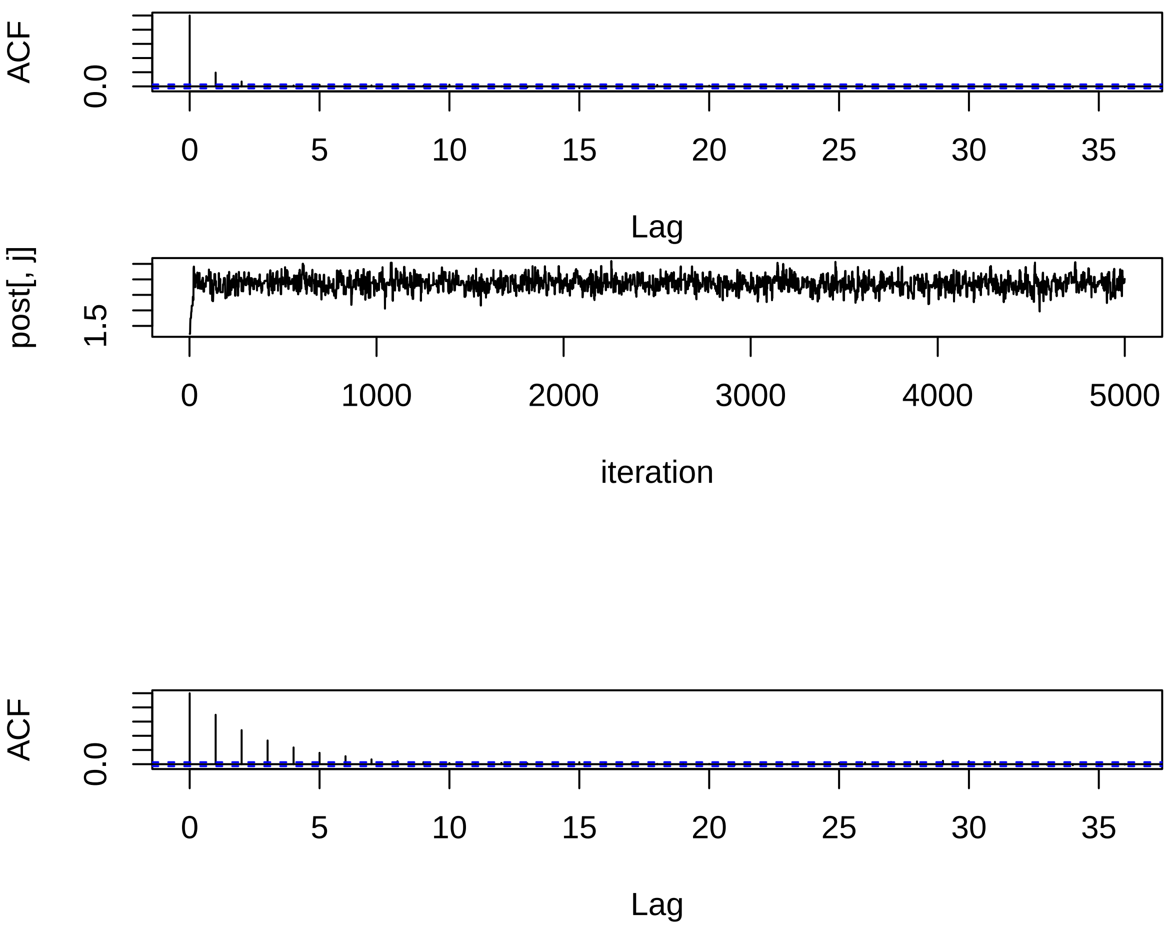

post[, j]
−1.0
0
1000
2000
3000
4000
5000
iteration


ACF
0.0
0
5
10
15
20
25
30
35
Lag

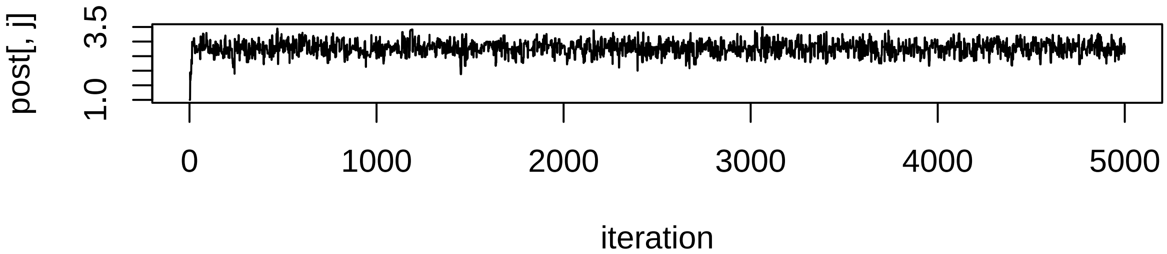


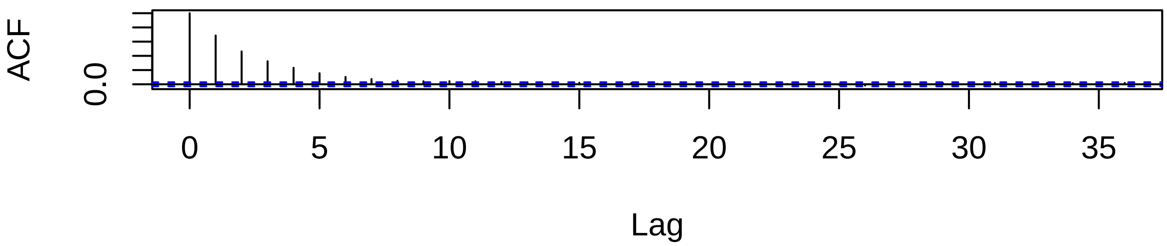
post[, j]
0.0
1.5
0
1000
2000
3000
4000
5000
iteration

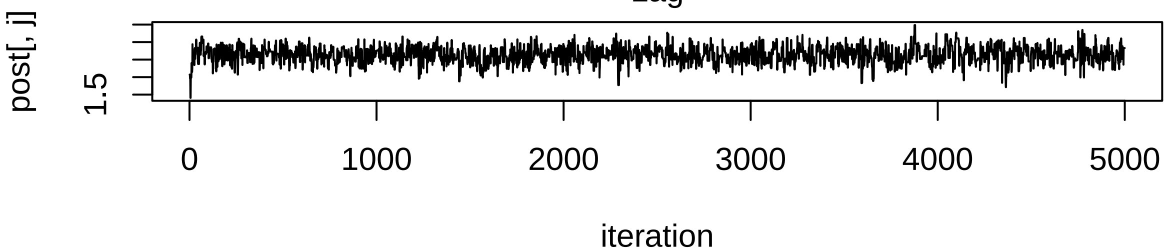
tau

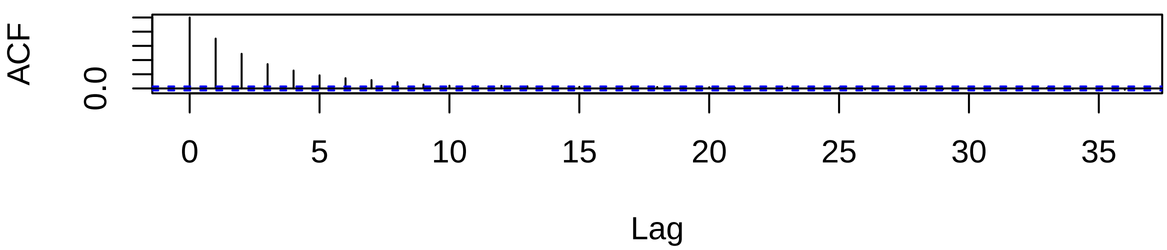


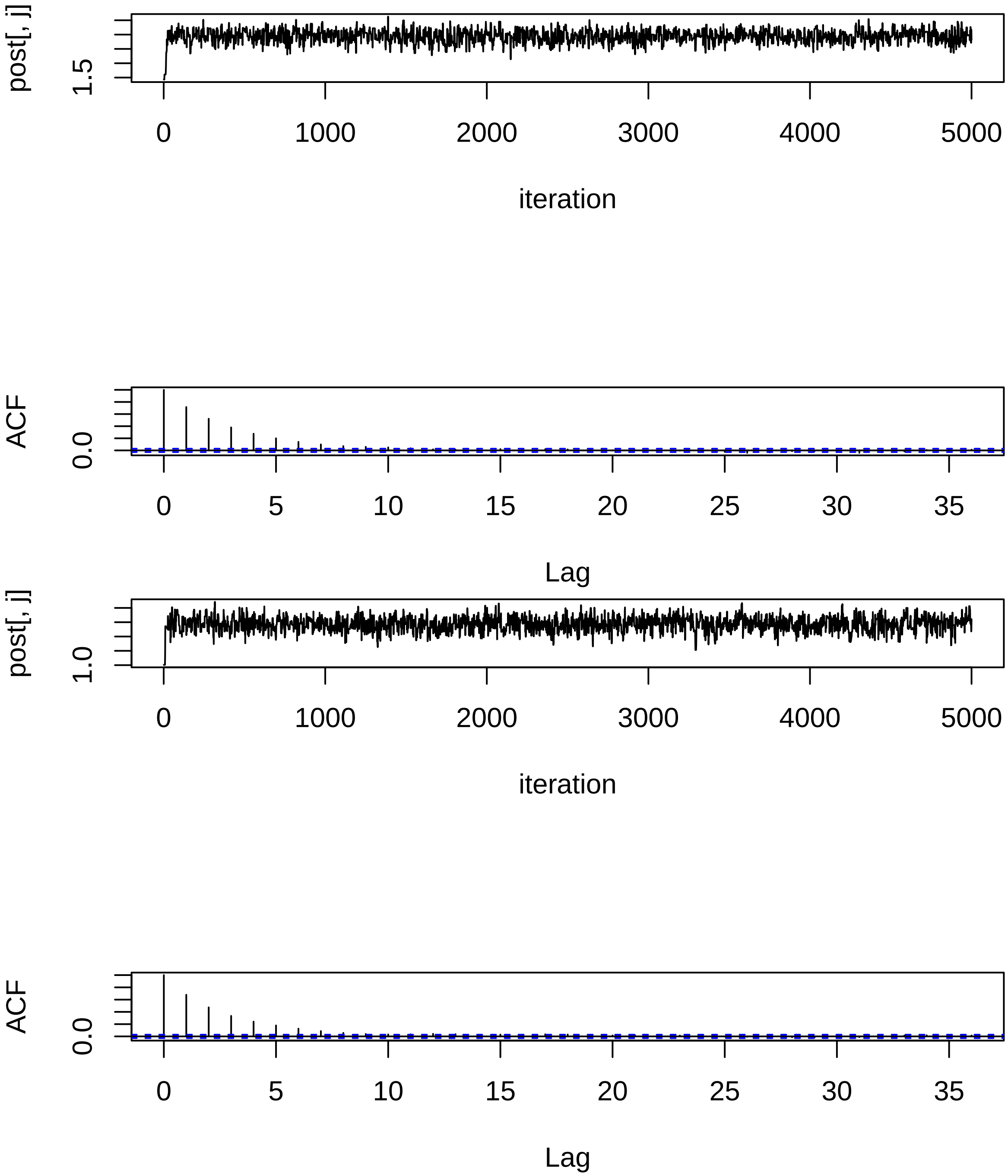
ACF
0.0
0
5
10
15
20
25
30
35
Lag
post[, j]
1.5
0
1000
2000
3000
4000
5000
iteration
ACF
0.0
0
5
10
15
20
25
30
35
Lag

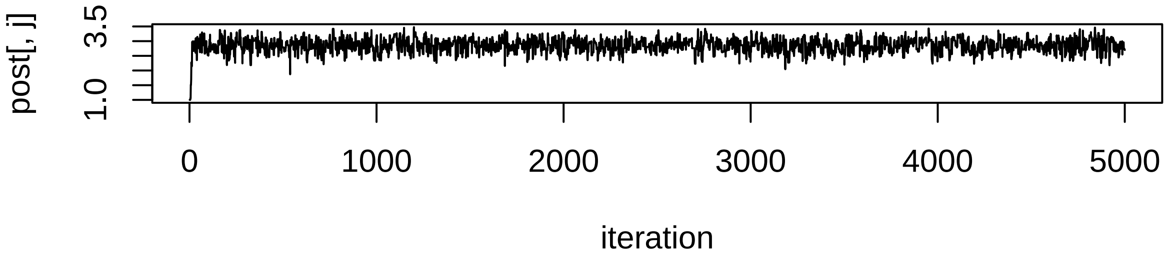

post[, j]
1.0
3.5
0
1000
2000
3000
4000
5000
iteration

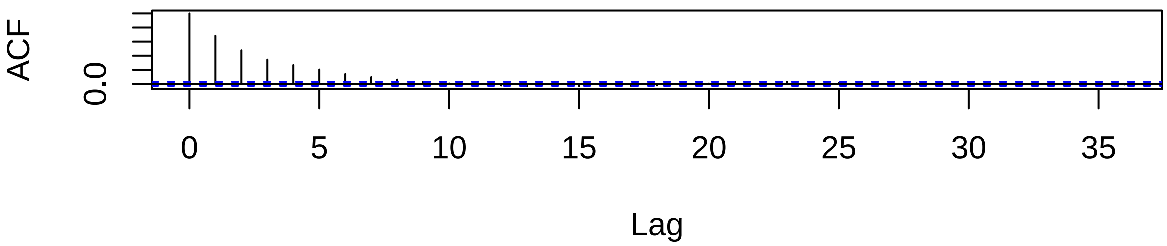

ACF
0.0
0
5
10
15
20
25
30
35
Lag

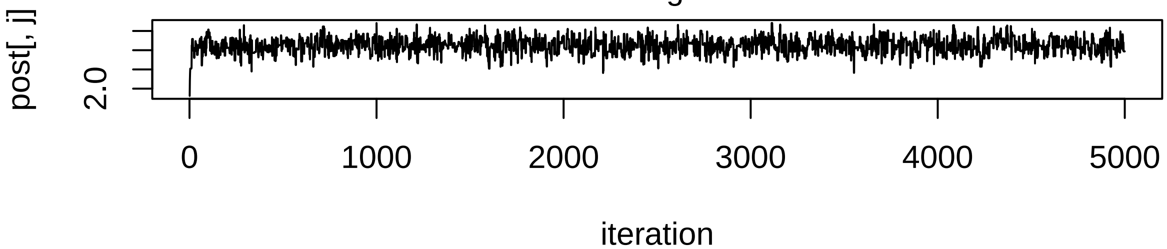

post[, j]
1.5
0
1000
2000
3000
4000
5000
iteration

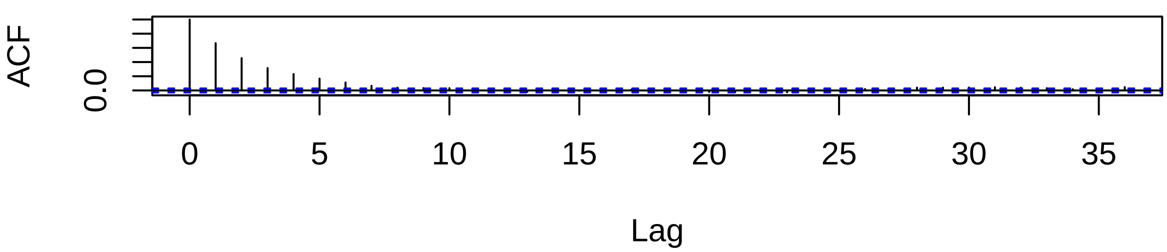

ACF
0.0
0
5
10
15
20
25
30
35
Lag

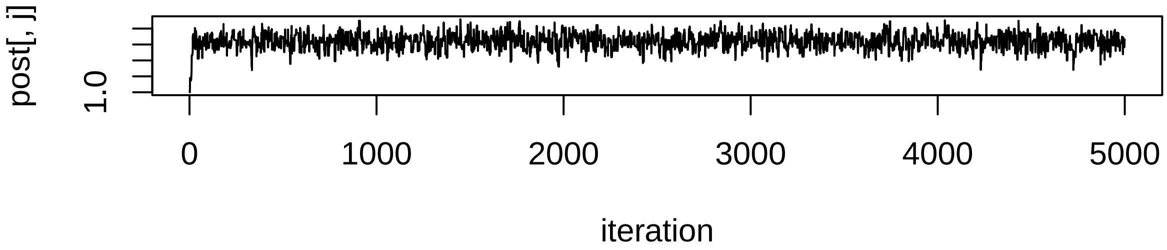

post[, j]
1.5
0
1000
2000
3000
4000
5000
iteration
ACF
0.0
0
5
10
15
20
25
30
35
Lag
post[, j]
1.0
0
1000
2000
3000
4000
5000
iteration
ACF
0.0
0
5
10
15
20
25
30
35
Lag

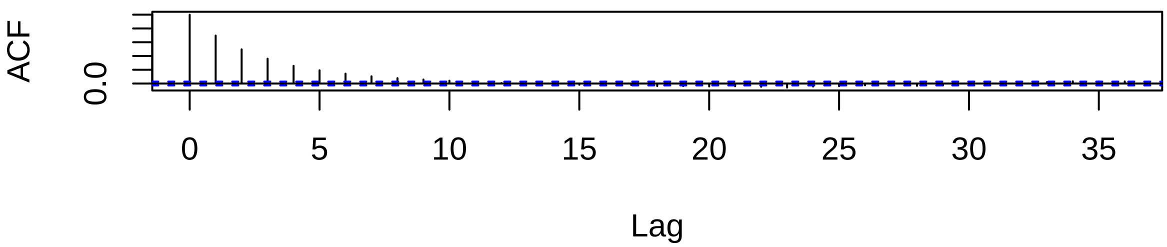

post[, j]
1.0
3.5
0
1000
2000
3000
4000
5000
iteration

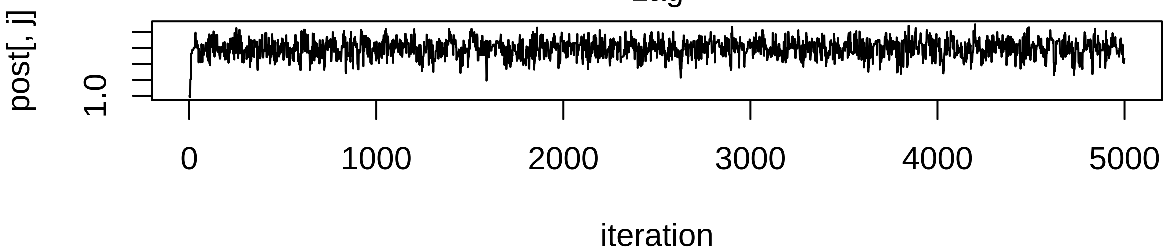

ACF
0.0
0
5
10
15
20
25
30
35
Lag

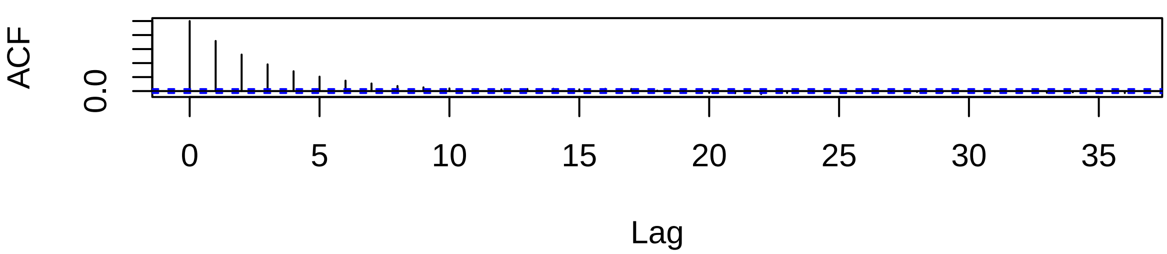

post[, j]
2.0
0
1000
2000
3000
4000
5000
iteration


ACF
0.0
0
5
10
15
20
25
30
35
Lag









From part b):

```
gradient = function(lamb, tau, beta) {
  nu = 1 / tau
  mu = X %*% beta
  g1 = (lamb - mu) * nu - y + exp(lamb)
return(as.vector(g1))
}
```

- Here use Leapfrog path

```
leap = function(x0, d0, epsilon, nn, M, tau, beta) {
Mi = M
diag(Mi) = 1 / diag(2 * M)
p = length(x0)
xx = dd = matrix(0, nrow = p, ncol =nn)
xx[, 1] = x0
dd[, 1] = d0
for(i in 1:(nn - 1)) {
dd[, (i + 1)] = dd[, i] - 0.5 * epsilon *gradient(xx[, i], tau, beta)
xx[, (i + 1)] = xx[, i] + epsilon * gradient(dd[, (i + 1)] %*% Mi, tau, beta)
dd[, (i + 1)] = dd[, (i + 1)] - 0.5 * epsilon * gradient(xx[, (i + 1)], tau, beta)
}
return(list(xx = xx, dd = dd))
}
```

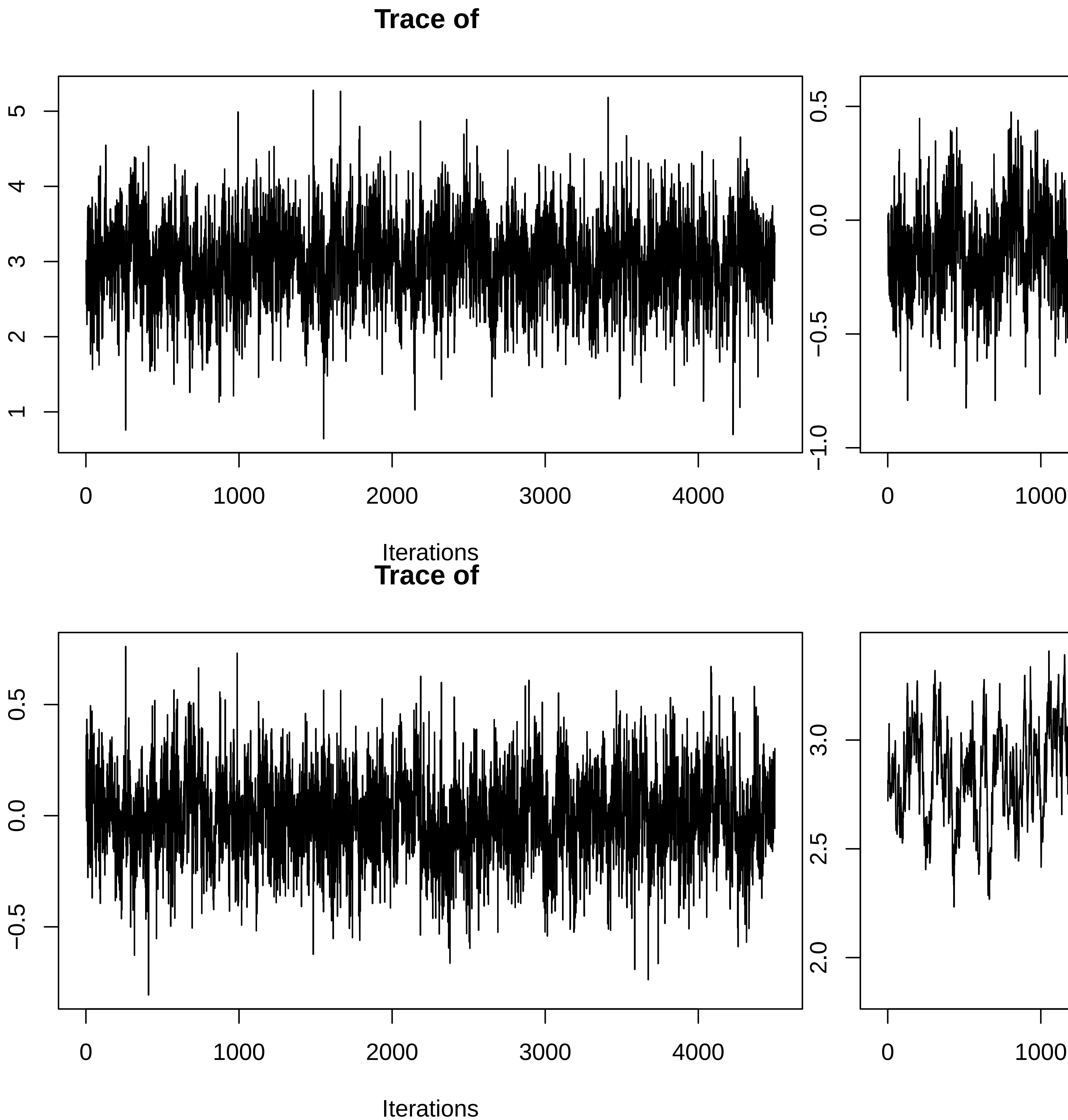
- HMC sampler

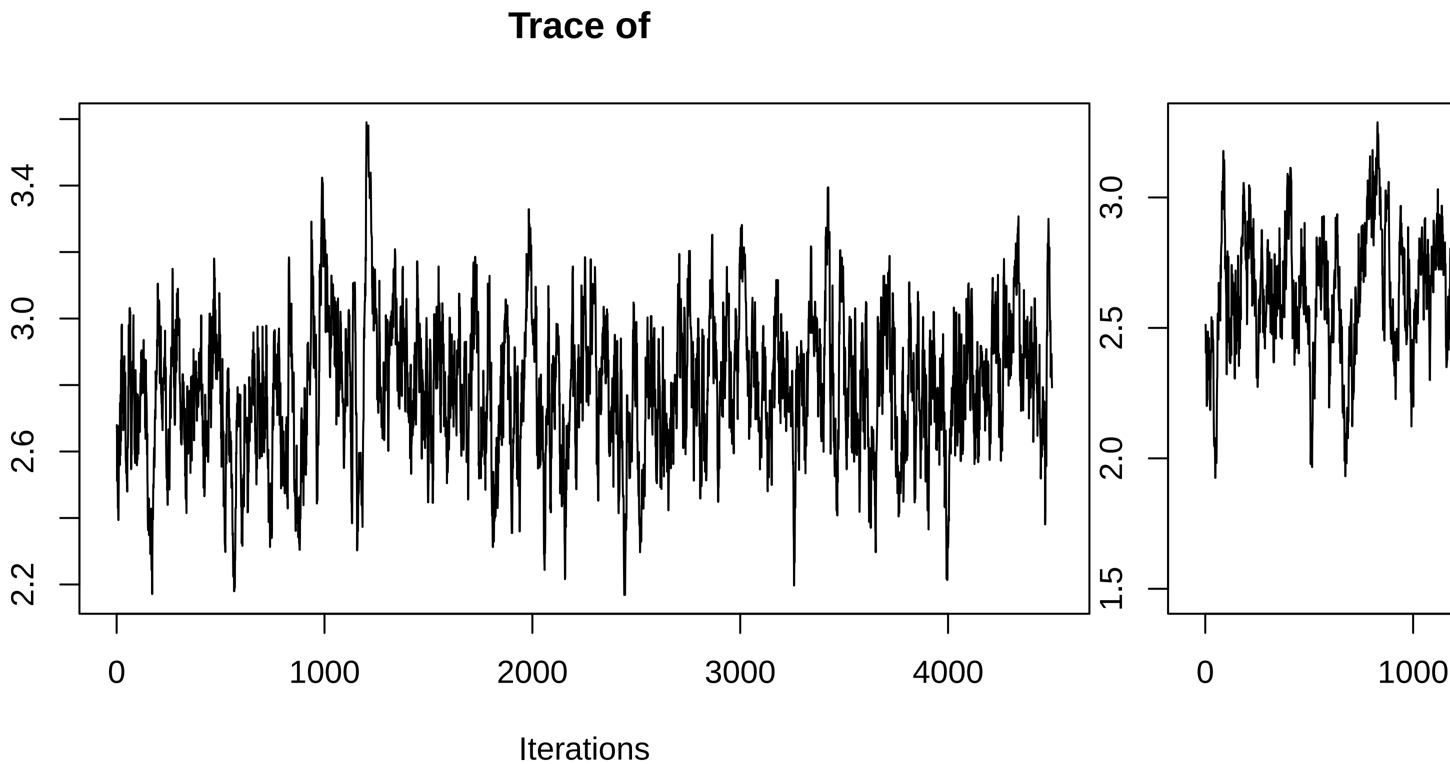
```
hmc = function(y, X, n_iter, burn, eta, L, eps) {

  ## initialize
  n = length(y)
  p = ncol(X)
  XT = t(X)
  XTX = XT %*% X
  XTXI = ginv(XTX)
  g = n
  mdelta = rep(0, n)
  M = eta * diag(n)

  ## save chain
  lamb_now = rnorm(n, log(y + 0.01), 0.1)
  beta_now = XTXI %*% XT %*% lambda
  tau_now = sum(lambda - X %*% beta)^2/ (n - p)

  beta_out = matrix(NA, nrow = n_iter, ncol = p)
  lamb_out = matrix(NA, nrow = n_iter, ncol = n)
  tau_out = rep(NA, n_iter)


  ## Sampler
  for(i in 1:n_iter){
    m0 = X %*% beta_now
    delta = as.vector(rmvnorm(1, mdelta, M))
    p0 = dmvnorm(delta, mdelta, M, log = TRUE)
    u0 = sum(-(lamb_now - m0)^2 / (2 * tau_now) + lamb_now * y - exp(lamb_now))

    e1 = rgamma(1, (eps * 200 + 1), rate = 200)
    L1 = max(2, rpois(1, L + 1))
    new = leap(x0 = lamb_now,
               d0 = delta,
               epsilon = e1,
```

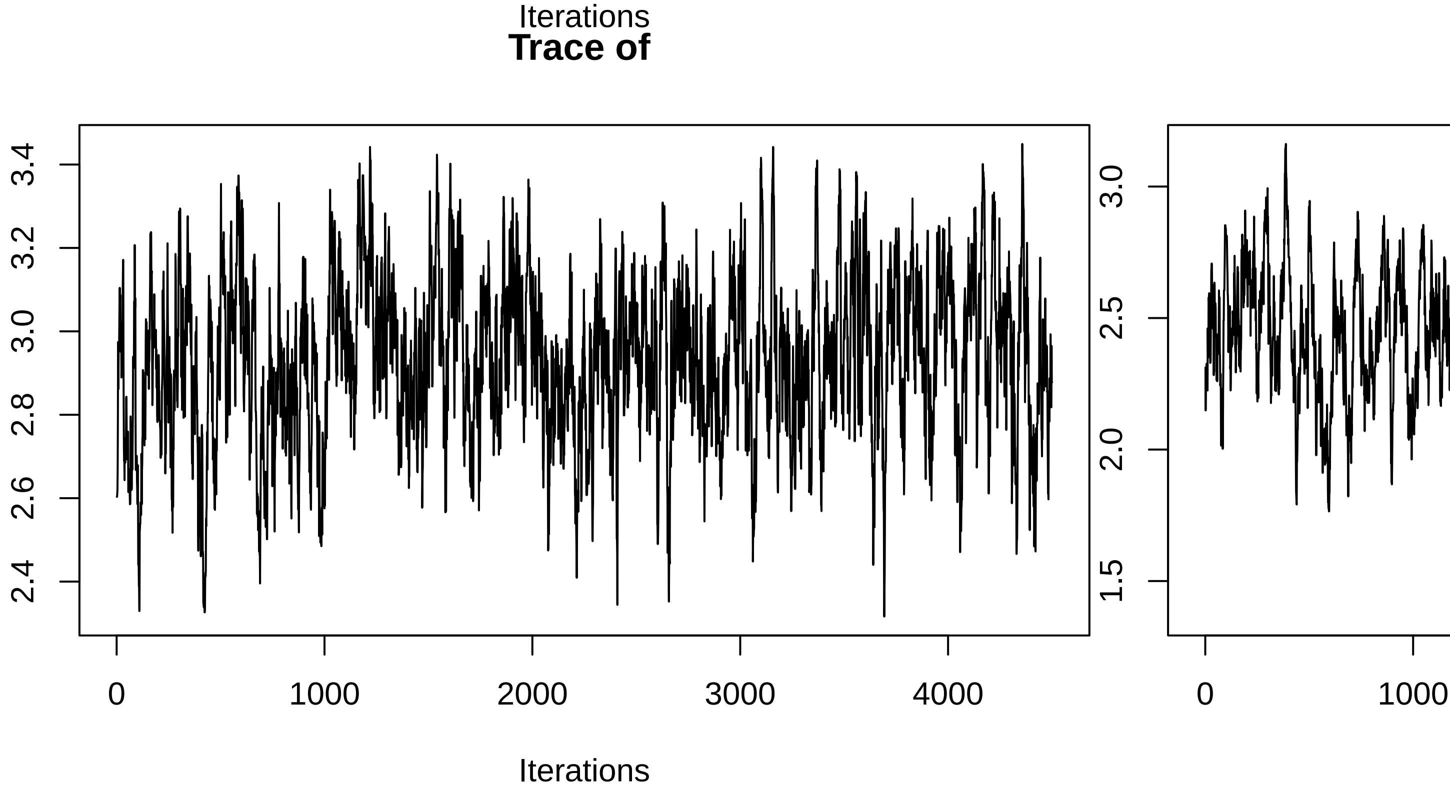
```
              nn = L1,
              M,
              tau_now,
              beta_now)

    l1 = as.vector(new$xx[ ,L1])
    delta1 = as.vector(new$dd[ ,L1])

    p1 = dmvnorm(delta1, mdelta, M, log = TRUE)
    u1 = sum(-(l1 - m0)^2 / (2 * tau) + l1 * y - exp(l1))

    ratio = u1 - u0 + p1 - p0

    if(log(runif(1)) < ratio) {
      lamb_now = l1
      delta = delta1
      }

    # beta
    beta_now = update_beta(g = g, X = X, lamb = lamb_now, tau = tau_now)

    # tau
    tau_now = update_tau(n = n, X = X, beta = beta_now, lamb = lamb_now)

    if(i > burn){
      i1 = i - burn
      lamb_out[i1,] = lamb_now
      beta_out[i1,] = beta_now
      tau[i1] = tau_now
      }

    }
  cbind(beta = beta_out, lambda = lamb_out, tau = tau_out)
 }
```

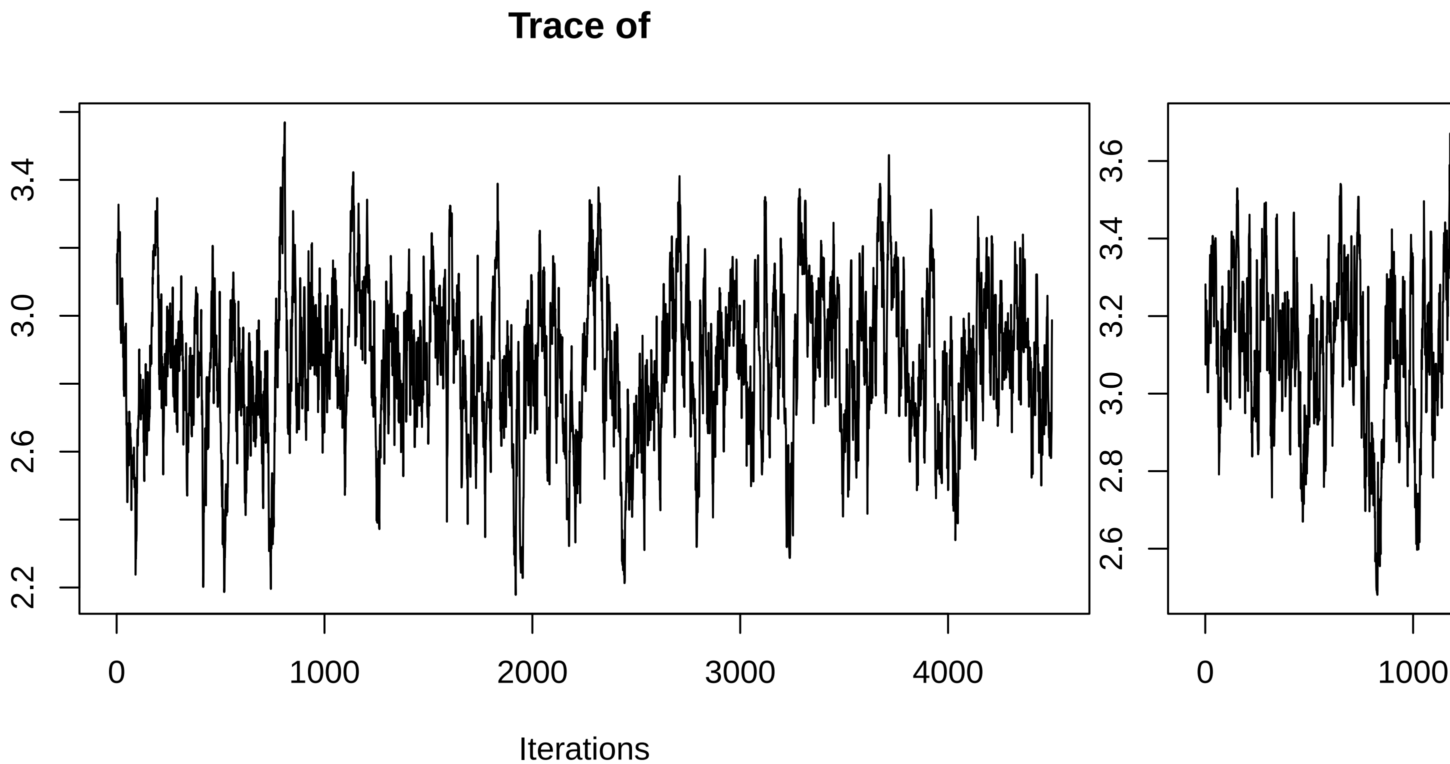
```
set.seed(888)
post1 = hmc(y, X, n_iter = 5000, burn = 1000, eta = 1, L =2, eps = 0.01)
```

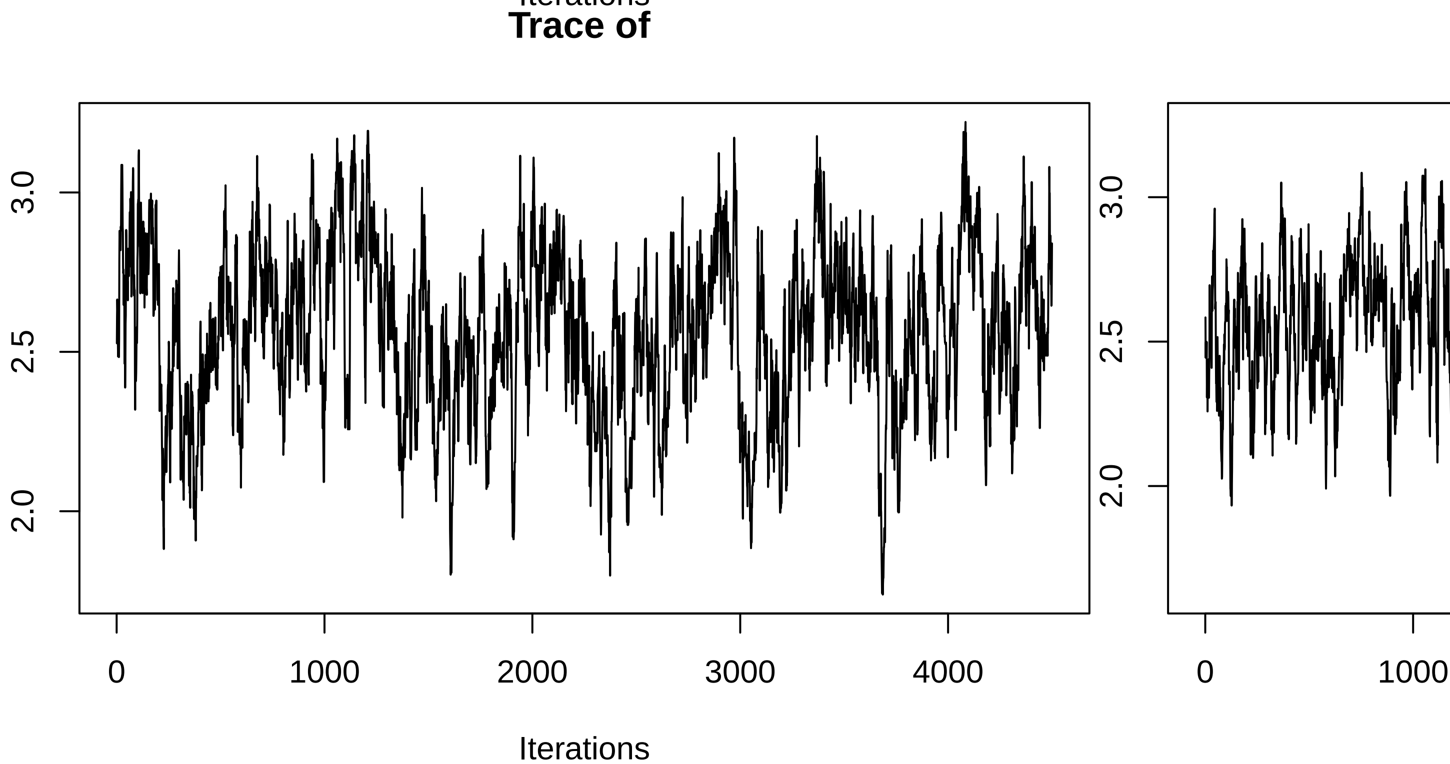
```
traceplot(as.mcmc(post1))
```

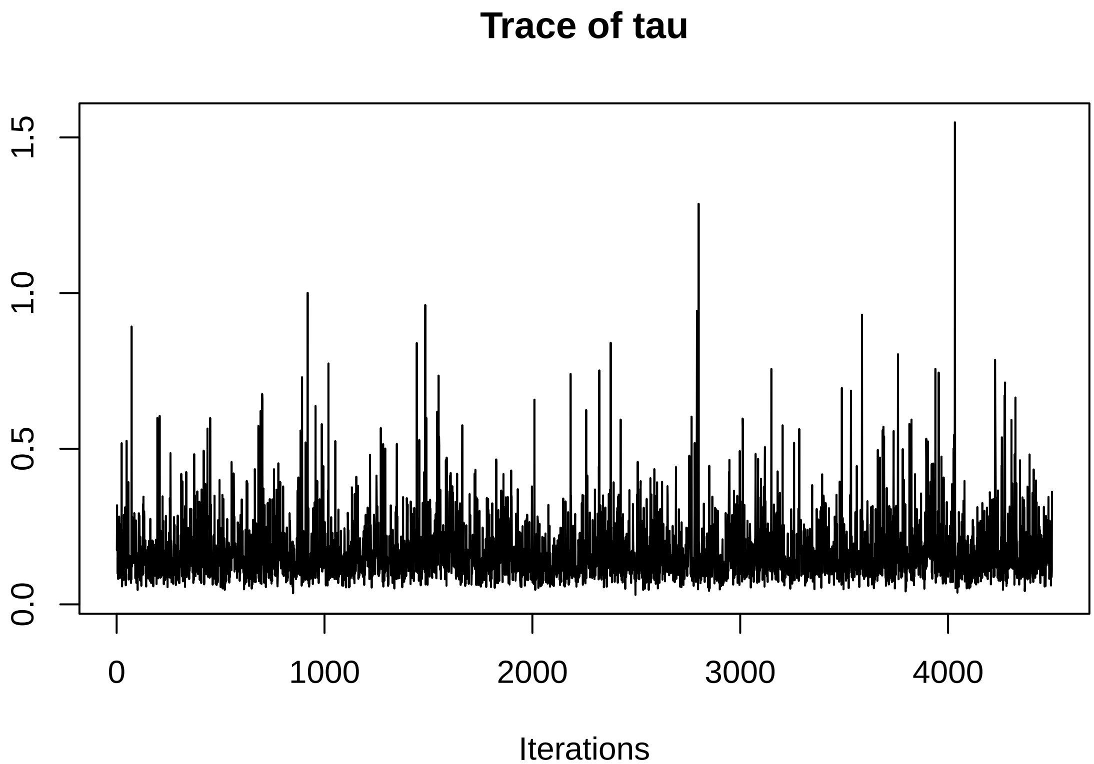

Trace of
Iterations
Trace of
Iterations
0
1000
2000
3000
4000


Trace of
3.4
3.0
2.6
2.2
0
1000
2000
3000
4000
Iterations
3.0
2.5
2.0
1.5
0
1000


Trace of
3.4
3.2
3.0
2.8
2.6
2.4
0
1000
2000
3000
4000
Iterations
3.0
2.5
2.0
1.5
0
1000


Trace of
3.4
3.0
2.6
2.2
3.6
3.4
3.2
3.0
2.8
2.6
0
1000
2000
3000
4000
0
1000
Iterations


Trace of
3.0
2.5
2.0
3.0
2.5
2.0
0
1000
2000
3000
4000
0
1000
Iterations



- All algorithms converged with 5000 simulations and 1,000 burn-in iterations based on the ACF and trace plots. On run-time usage, very similar in both.
- As we can tell from the plots, HMC method improved the parameters convergence and mixing.