SMC: Exercises set III

Niharika

June 18, 2018

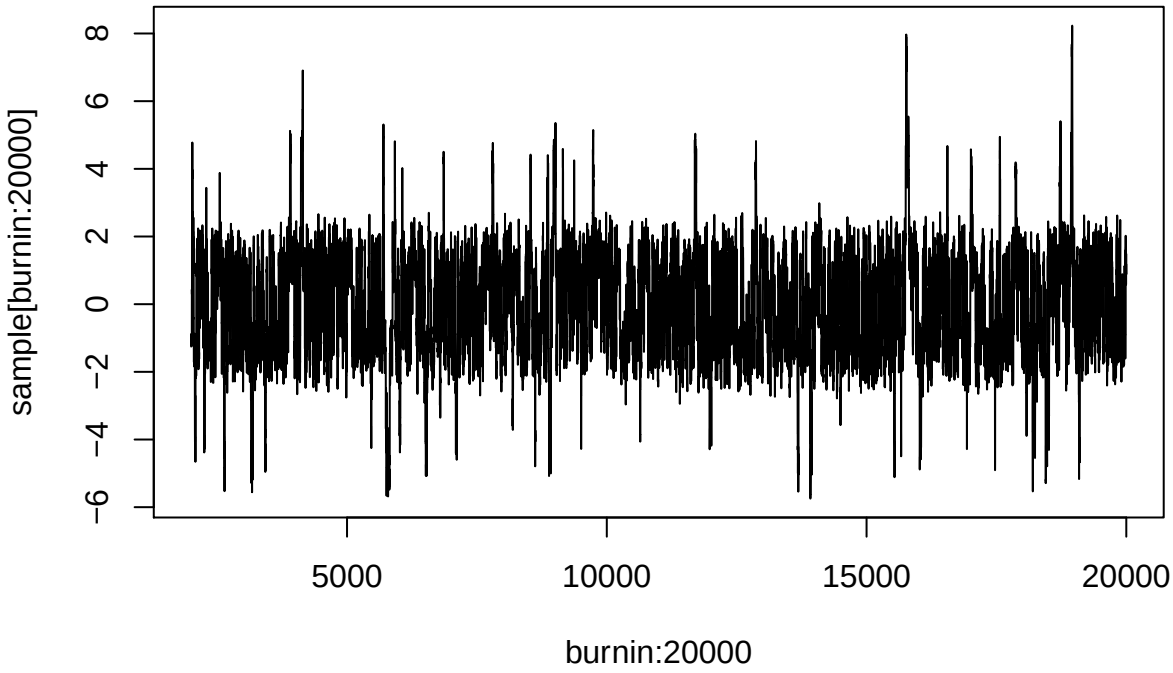
This document provides solution for the Exercises set III given at http://www.it.uu.se/research/systems_and_control/education/2017/smc/homework/SMC2017_exercises3.pdf.

III.1 Metropolis-Hastings

Sample from the following distribution, using Guassian random walk as proposal distribution.

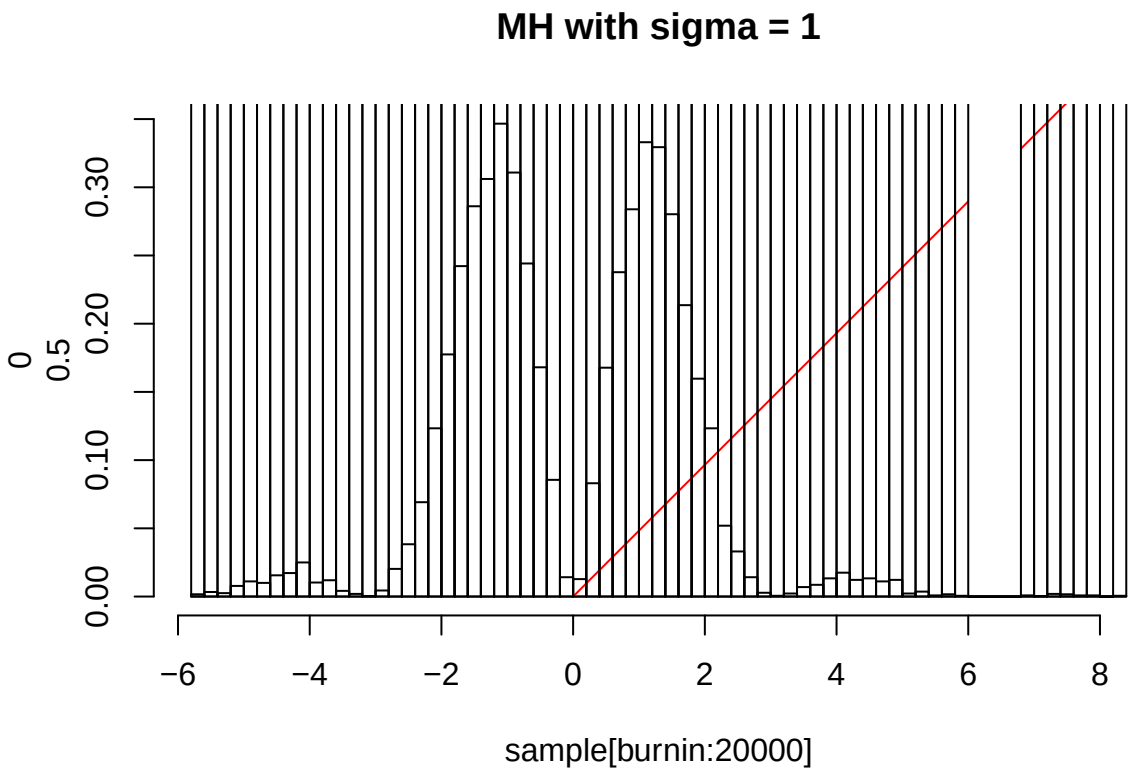
$$\pi(x) \propto sin^2(x)exp(-|x|)$$

```
targetFunc <- function(x)
{
  return(sin(x)^2 * exp(- abs(x)))
}

sampleFromProposal = function(m, sigma)
{
  return(rnorm(1, m, sd = sigma))
}

MH <- function(iter, sigma)
{
  x = rep(0, iter)
  # Initial state
  x[1] <- 1
  for(i in 2:iter){
    # Sample a new value from proposal
    x_temp = sampleFromProposal(m = x[i-1], sigma)
    u <- runif(1)
    # Compute acceptance probability
    acc_prob <- targetFunc(x_temp)/targetFunc(x[i-1])
    acc_prob <- min(1, acc_prob)
    if(u < acc_prob){
      x[i] <- x_temp
    }
    else{
      x[i] <- x[i-1]
    }
  }
  return(x)
}

burnin <- 2000
sample <- MH(20000,1)
plot(burnin:20000, sample[burnin:20000], type = "l")
```




```
h <- hist(sample[burnin:20000], breaks = 50, freq = FALSE,
     main = "MH with sigma = 1", ylab = c(0,0.5))
# overlay the actual function
x_lab <- seq(-7, 7, length = 1000)
lines(h, x_lab, 5/4*targetFunc(x_lab), type = "l", col="red")
```

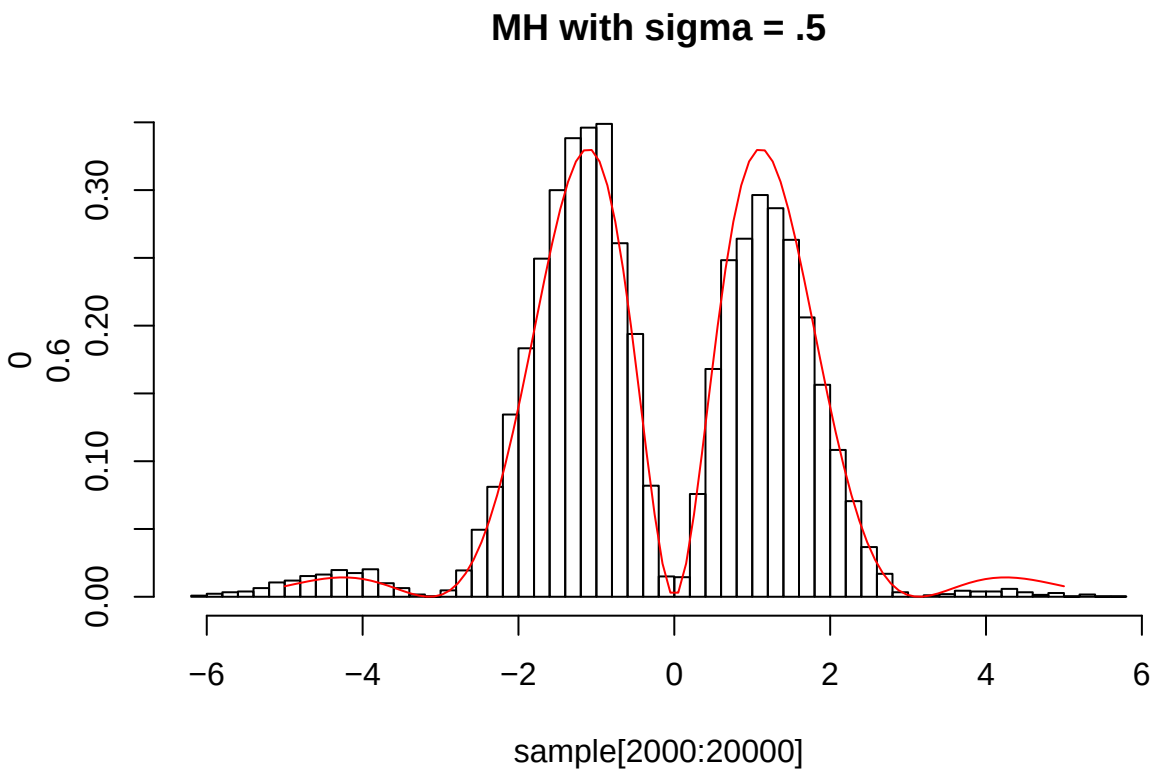
```
## Warning in if (freq) x$counts else x$density: the condition has length > 1
## and only the first element will be used
```



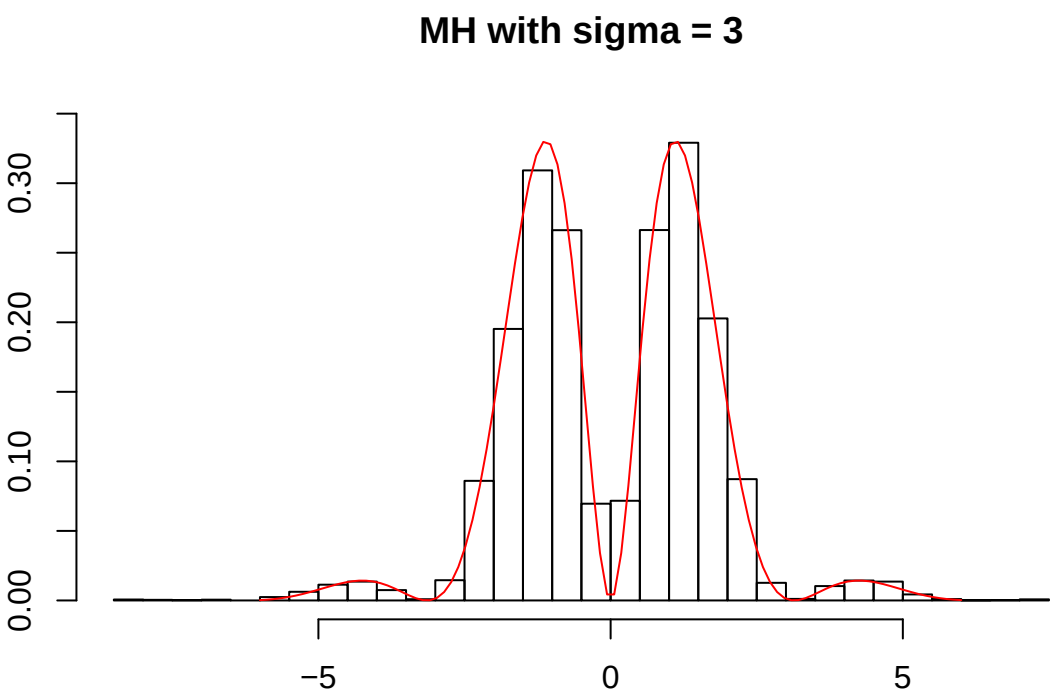
Plotting MH output with sigma = .5 and sigma = 3

```
sample <- MH(20000, .5)
hist(sample[2000:20000], breaks = 50, freq = FALSE,
     main = "MH with sigma = .5", ylab = c(0,0.6))
x_lab <- seq(-5, 5, length = 100)
```

```
lines(x_lab, 5/4*targetFunc(x_lab), type = "l", col="red")
```




```
sample <- MH(20000, 3)
x_lab <- seq(-6, 6, length = 100)
h <- hist(sample[2000:20000], breaks = 50, freq = FALSE,
     main = "MH with sigma = 3", ylim = c(0,0.34), xlab = "", ylab = "")
lines(x_lab, 5/4*targetFunc(x_lab), type = "l", col="red")
```

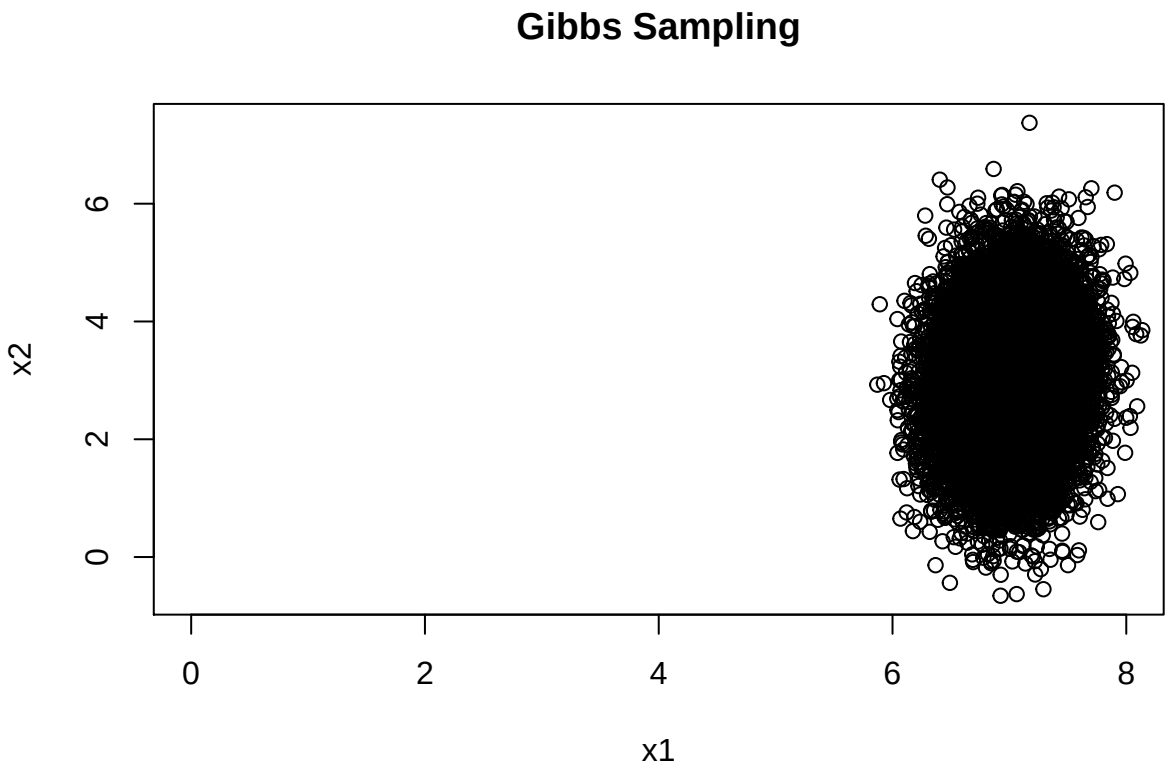


Gibbs Sampling

Consider the following target distribution

$$\pi(x) = N\left(\begin{pmatrix} 7 \\ 3 \end{pmatrix}, \begin{pmatrix} 0.3 & 0.1 \\ 0.1 & 1 \end{pmatrix}\right)$$

```r
GibbsSampler <- function(iter, sigma)
{
  m <- c(7,1)
  Sigma <- matrix(c(0.3, 0.1, .1, 1), 2, 2, byrow = TRUE)
  x <- matrix(0, iter, 2)
  x[1,] <- c(0,0)

  for(i in 2:iter){
    x[i,1] <- rnorm(1, 7 + 0.1 * (x[i-1, 2] - 3), 0.29)
    x[i,2] <- rnorm(1, 3 + (1/3) * (x[i, 1] - 7), 29/30)
  }
  return(x)
}

sample <- GibbsSampler(20000,1)
plot(sample[2000:20000,1], sample[2000:20000,2], type = "p",
     main = "Gibbs Sampling", xlab = "x1", ylab = "x2", xlim = c(0,8))
```



III.3 Resampling

```r
N <- 100
x <- rnorm(N, 1, 2)
w <- runif(N, 0, 1)
w <- w/sum(w)

mhat = sum(w * x)

multiResmpling <- function(w)
{
  s <- multinomial.resample(w)
  return(s)
```

```
}

systematicResmpling <- function(w)
{
  s <- systematic.resample(w)
  return(s)
}

stratifiedResmpling <- function(w)
{
  s <- stratified.resample(w)
  return(s)
}

M <- 10000
diff_m = rep(0, M)
diff_s = rep(0, M)
diff_t = rep(0, M)

for (k in 1:M) {
  s <- multiResmpling(w)
  m_hat_m <- mean(x[s])
  diff_m[k] <- m_hat_m - mhat

  s <- systematicResmpling(w)
  m_hat_s <- mean(x[s])
  diff_s[k] <- m_hat_s - mhat

  s <- stratifiedResmpling(w)
  m_hat_t <- mean(x[s])
  diff_t[k] <- m_hat_t - mhat
}

var(diff_m)
```

```
## [1] 0.02217374
```

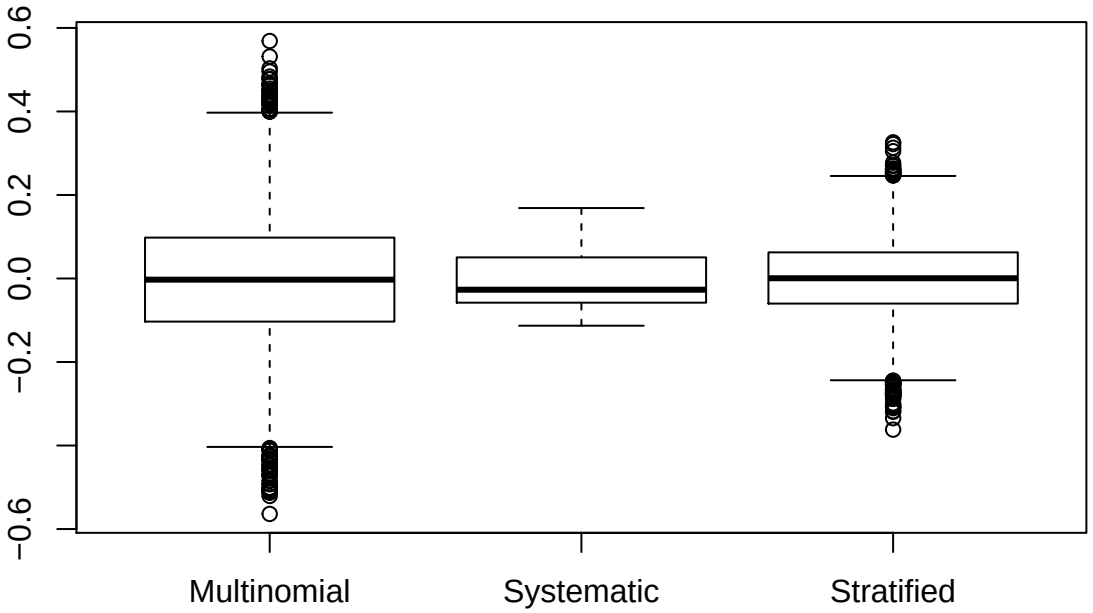
```
var(diff_s)
```

```
## [1] 0.005237104
```

```
var(diff_t)
```

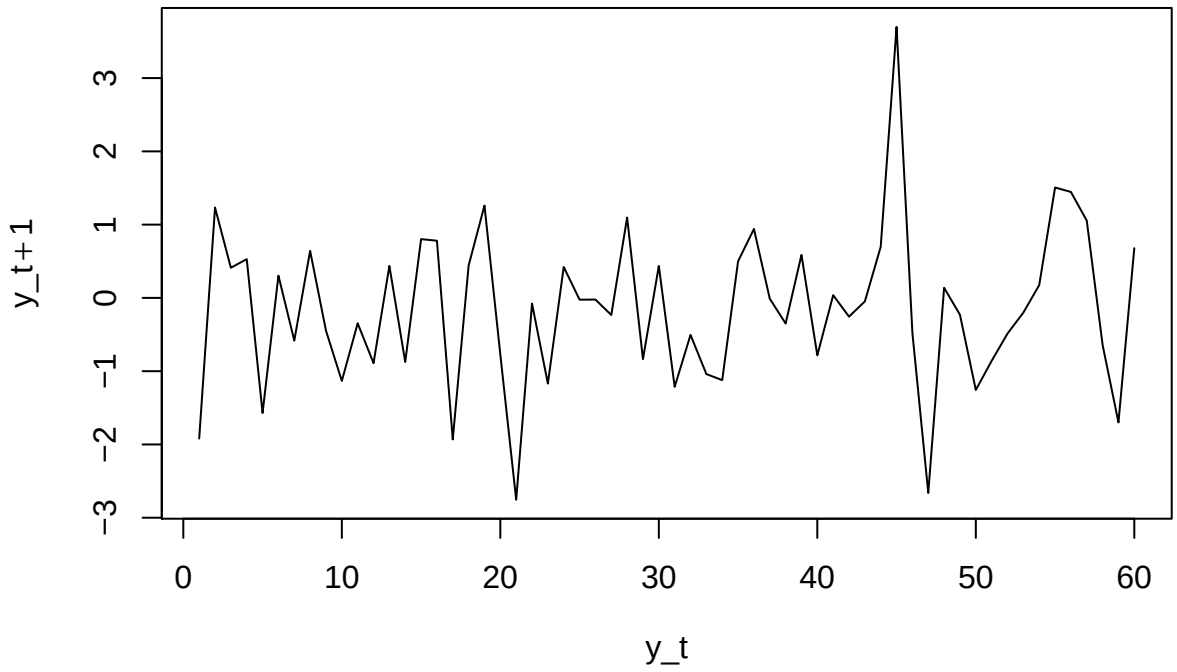
```
## [1] 0.008422676
```

```
df <- cbind(diff_m, diff_s, diff_t)
colnames(df) <- c("Multinomial", "Systematic", "Stratified")
boxplot(df)
```



III.4 Path-space view

```
T <- 60
N <- 100
#Read the observations and plot them
y <- read.csv("seOMXlogreturns2012to2014.csv", header = FALSE)
y <- y[,1]
plot(c(1:T), y[1:T], type = "l", xlab = expression(y_t), ylab = expression(y_t+1))
```

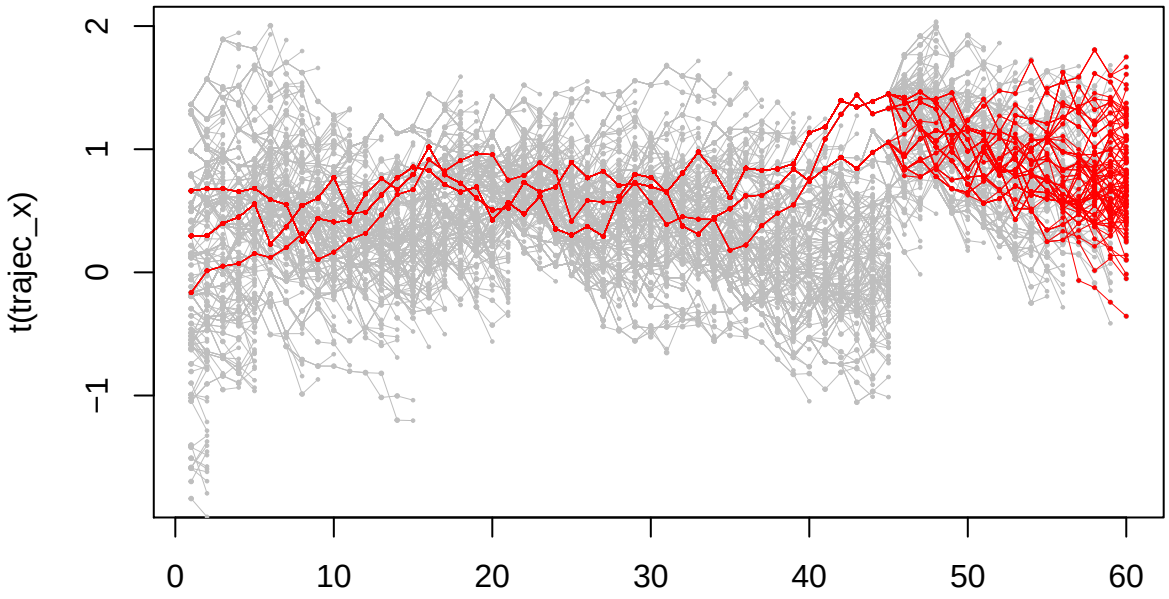



```
########## Boot strap particle filter##############
# given parameters
#Bootstrap particle filter for volatility model
volatilityModelBPF <- function(param, x0, y, N = 100, resamplingMethod = "multi",
                               essMax = 0) {
  T <- length(y)
  #Initialize the parameters
  A <- param[1]
  Q <- param[2]
  beta <- param[3]
```

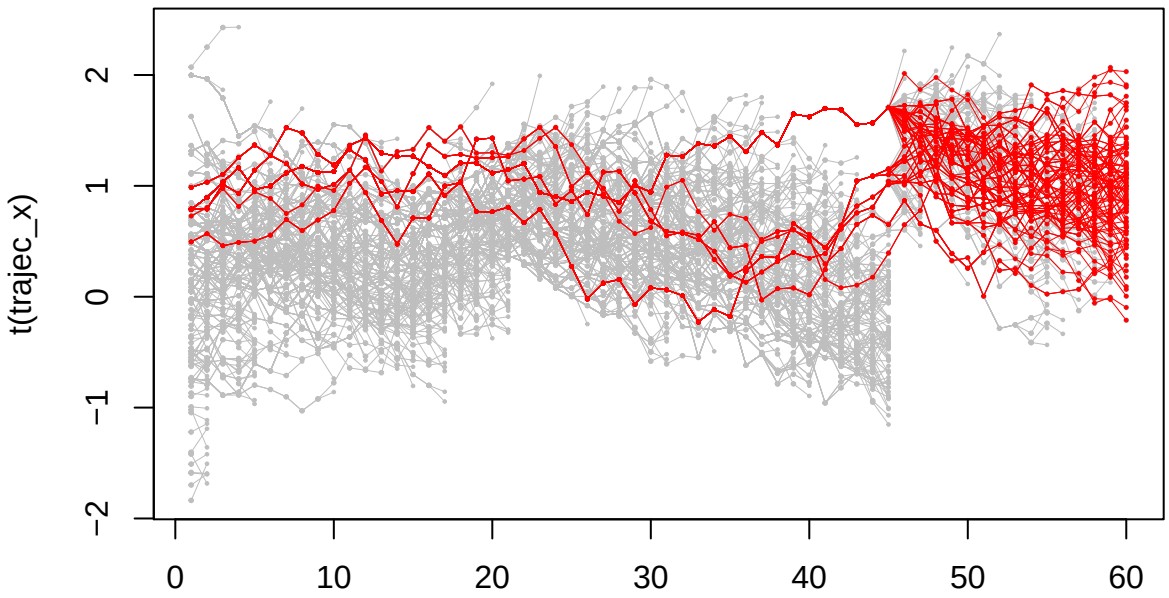
```
# Initialize variables
particles <- matrix(0, nrow = N, ncol = T)
ancestorIndices <- matrix(0, nrow = N, ncol = T)
normalisedWeights <- matrix(0, nrow = N, ncol = T)
xHatFiltered <-rep(0, T)
PHatFiltered <- rep(0, T)
logLikelihood <- 0

particles[ ,1] <- x0
xHatFiltered[1] <- mean(x0)
normalisedWeights[, 1] = 1 / N
PHatFiltered[1] <- var(x0)
ancestorIndices[, 1] = 1:N

for (t in 2:T) {
  # Resample
  ess = 1 / sum(normalisedWeights^2)
  # Resample
  if(essMax == 0 || ess < essMax){
    if(resamplingMethod == "systematic")
    {
      newAncestors <- systematic.resample(normalisedWeights[, t - 1])
    }
    else if(resamplingMethod == "stratified")
    {
      newAncestors <- stratified.resample(normalisedWeights[, t - 1])
    }
    else # by default it is multinomial
    {
      newAncestors <- multinomial.resample(normalisedWeights[, t - 1])
    }
  }
  else{
    newAncestors <- ancestorIndices[, t-1]
  }
  ancestorIndices[, t] <- newAncestors

  # Propagate
  particles[, t] <- A * particles[newAncestors, t-1] +
    sqrt(Q) * rnorm(N)

  #Likelihood
  weights <- dnorm(y[t], mean = 0, sd = sqrt(beta) * exp(particles[, t]/2),
                         log = TRUE)

  max_weight <- max(weights)
  weights <- exp(weights - max_weight)
  normalisedWeights[, t] <- weights / sum(weights)

  # Estimate the state
  xHatFiltered[t] <- mean(particles[, t])
  PHatFiltered[t] <- var(particles[, t])

  # accumulate the log-likelihood
  logLikelihood = logLikelihood + max_weight +
    log(sum(weights)) - log(N)

}
```

```
  return(list(xHatFiltered = xHatFiltered,
              PHatFiltered = PHatFiltered,
              logLikelihood = logLikelihood,
              particles = particles,
              ancestorIndices = ancestorIndices,
              normalisedWeights = normalisedWeights))

}

plotTrajectory <- function(particles, anc)
{
  N = dim(particles)[1]
  T = dim(particles)[2]

  # all trajectories
  trajec_x=matrix(0,N,T)
  for(i in 1:59)
  {
    trajec_x[,i] = particles[anc[,i+1],i]
  }
  trajec_x[, 60] = particles[, 60]

  # Survived trajectories
  trajec_s=matrix(0,N,T)
  trajec_s[, 60] = particles[, 60]
  # Generate the trajectories from ancestor indices
  ind = anc[,T]
  for(t in (T-1):1)
  {
    trajec_s[,t] = particles[ind,t];
    ind = anc[ind,t]
  }

  matplot(t(trajec_x), col=c("gray"), cex = .25, pch = 19, type = "p")

  # dead trajectories
  for (i in 1:(T-2)) {
    K <- T-i
    ind = setdiff(c(1:N), anc[, K+1])
    if(length(ind) > 0)
    {
      trajec_d=matrix(0, length(ind), K)
      for(t in K:1)
      {
        trajec_d[,t] = particles[ind,t]
        ind = anc[ind,t]
      }

      matlines(t(trajec_d), col=c("grey"), lwd=.25, lty=1, pch = 19, type = "o", cex = .25)
    }
  }

  matlines(t(trajec_s), col=c("red"),lwd=.25, lty=1, pch = 19, type = "o", cex = .25)
  #matpoints(t(trajec_s), col=c("red"), cex = .25, pch = 19)

  #matplot(t(trajec_x), col=c("grey"), lwd=.25, lty=1, type = "l")


}
```

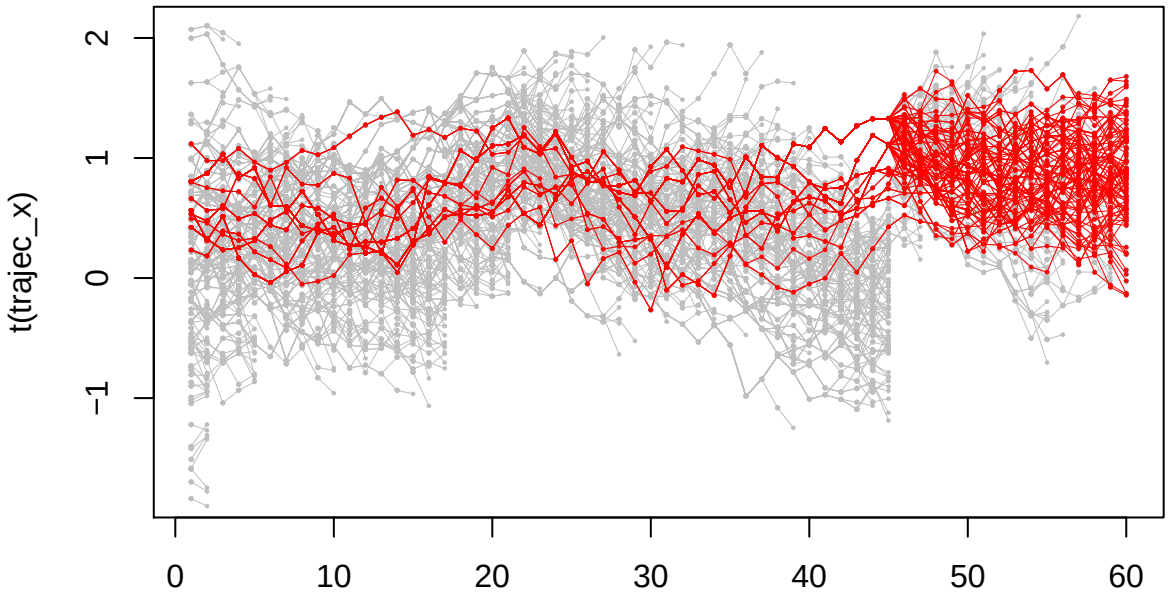
```r
N <- 100
param <- c(0.98, 0.16^2, 0.70^2)
x0 <- 0.8 * rnorm(N)

# Plot trajectories using multinomial sampling
bpf <- volatilityModelBPF(param, x0, y[1:T], N = N, resamplingMethod = "multi")
particles <- bpf$particles
anc <- bpf$ancestorIndices
plotTrajectory(particles, anc)
```



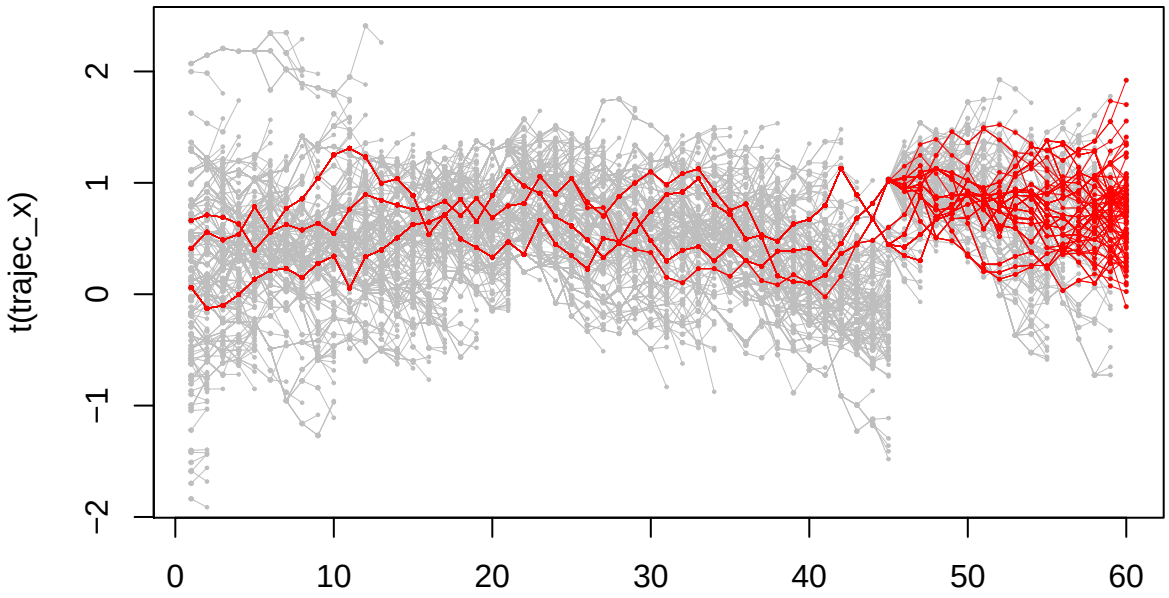

```r
# Plot trajectories using stratified sampling
bpf <- volatilityModelBPF(param, x0, y[1:T], N = N, resamplingMethod = "stratified")
particles <- bpf$particles
anc <- bpf$ancestorIndices
plotTrajectory(particles, anc)
```



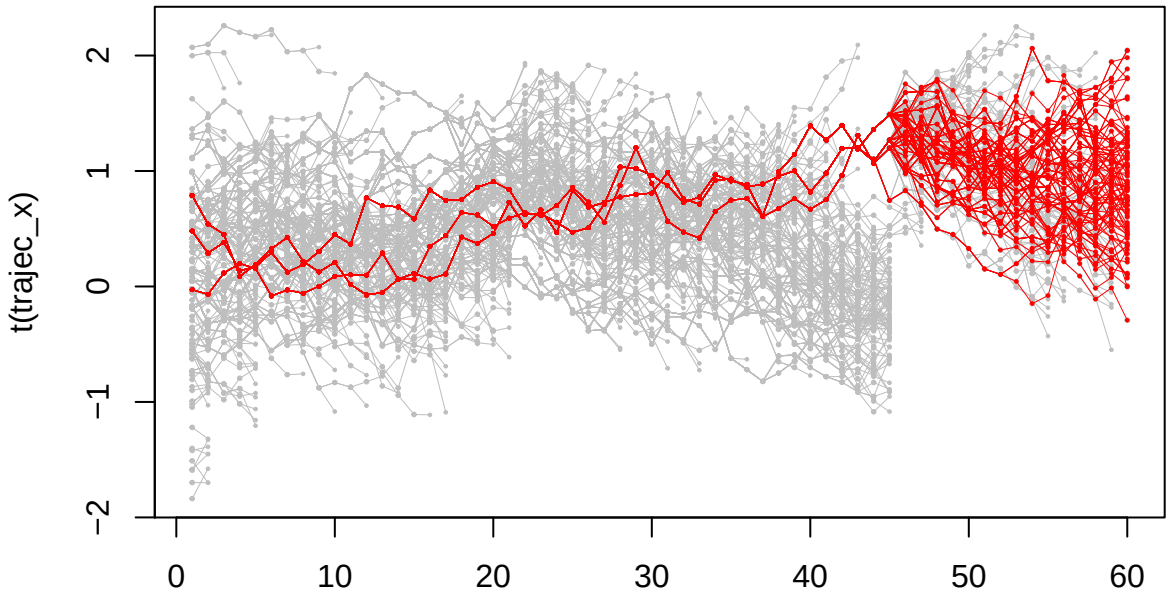

```r
# Plot trajectories using systematic sampling
bpf <- volatilityModelBPF(param, x0, y[1:T], N = N, resamplingMethod = "systematic")
particles <- bpf$particles
anc <- bpf$ancestorIndices
plotTrajectory(particles, anc)
```




```
# Plot trajectories using multinomial sampling with ESS = N/2
bpf <- volatilityModelBPF(param, x0, y[1:T], N = N, resamplingMethod = "multi",
                          essMax = N/2)
particles <- bpf$particles
anc <- bpf$ancestorIndices
plotTrajectory(particles, anc)
```




```
# Plot trajectories using stratified sampling with ESS = N/2
bpf <- volatilityModelBPF(param, x0, y[1:T], N = N, resamplingMethod = "stratified",
                          essMax = N/2)
particles <- bpf$particles
anc <- bpf$ancestorIndices
plotTrajectory(particles, anc)
```




```
# Plot trajectories using systematic sampling with ESS = N/2
bpf <- volatilityModelBPF(param, x0, y[1:T], N = N, resamplingMethod = "systematic",
                          essMax = N/2)
particles <- bpf$particles
anc <- bpf$ancestorIndices
plotTrajectory(particles, anc)
```