SMC: Exercises set IV

Niharika

Sep 22, 2018

This document provides solution for the Exercises set IV given at http://www.it.uu.se/research/systems_and_control/education/2017/smc/homework/SMC2017_exercises4.pdf.

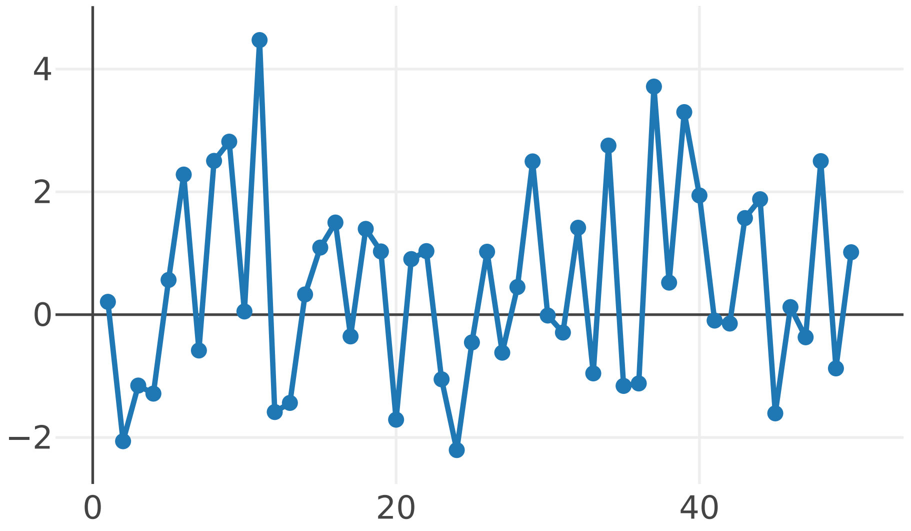
IV.1 Particle Metropolis-Hastings

Consider the following state-space model:

$$x_t = cos(\theta * x_{t-1}) + v_t, \quad v_t \sim N(0,1)$$
$$y_t = x_t + e_t, \quad e_t \sim N(0,1)$$
$$x_0 \sim N(0,1)$$

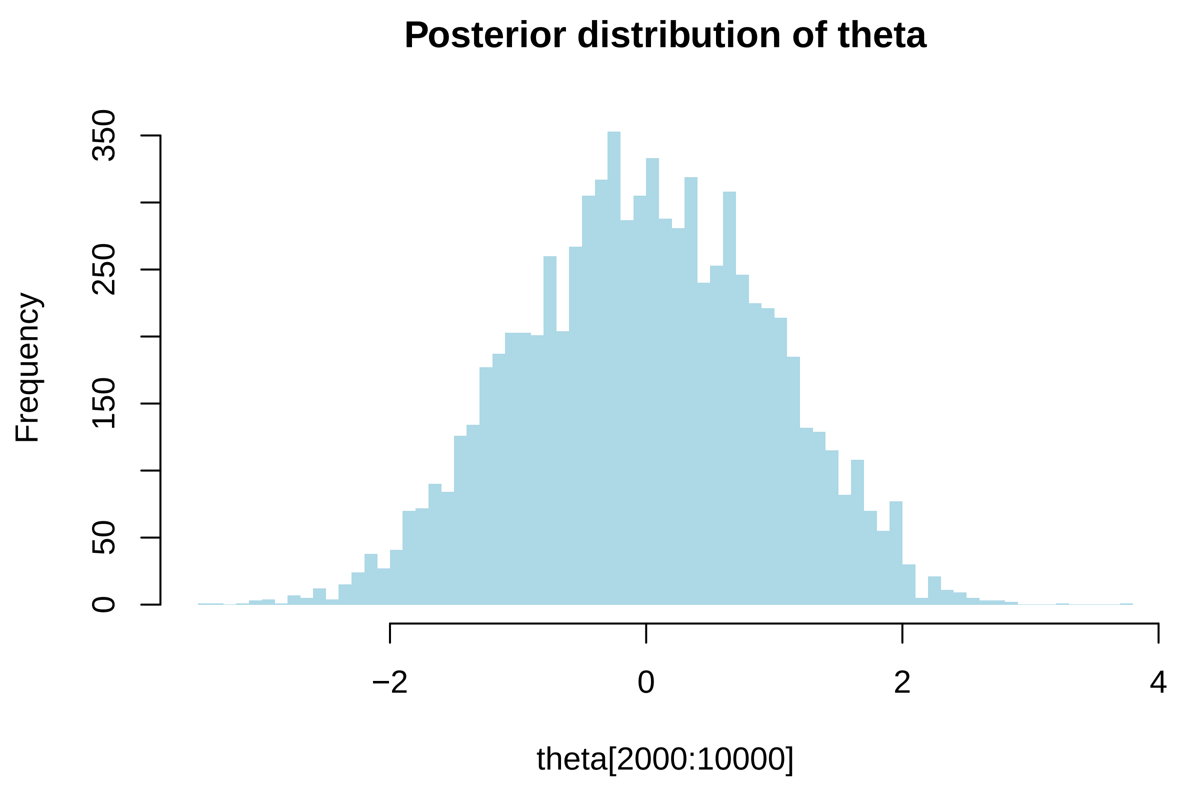
Generate y assuming $\theta = 1$.

```r
T <- 50
N <- 100

# simulate data
simulateData <- function(param, T)
{
  theta <- param[1]
  sigma_v <- param[2]
  sigma_e <- param[3]

  x <- rep(0, T)
  y <- rep(0, T)

  x0 <- 0 # state 0
  x[1] <- cos(theta*x0) + rnorm(1, 0, sigma_v)
  y[1] <- x[1] +rnorm(1, 0 ,sigma_e)

  for(t in 2:T)
  {
    x[t] <- cos(theta*x[t-1]) + rnorm(1, 0, sigma_v)
    y[t] <- x[t] +rnorm(1, 0, sigma_e)
  }

  return(list(x=x, y=y))
}

param <- c(1, 1, 1)
simulatedData <- simulateData(param, T)

plot_ly(x = c(1:T), y = simulatedData$y, name = 'simulated states', type = 'scatter', mode = 'lines+markers')
```



```
bootstrapParticleFilter <- function(param, y, N) {
  T <- length(y)
  theta <- param[1]
  sigma_v <- param[2]
  sigma_e <- param[3]

  # Initialize variables
  particles <- matrix(0, nrow = N, ncol = T)
  weights <- matrix(1, nrow = N, ncol = T)
  normalisedWeights <- matrix(0, nrow = N, ncol = T)
  xHat <-rep(0, T)
  logLikelihood <- 0

  #state at t=0
  x0 <- rnorm(N, 0, 1)

  particles[ ,1] <- x0
  normalisedWeights[, 1] = 1 / N

  for (t in 2:T) {
    # Resample
    newAncestors <- sample(N, replace = TRUE, prob = normalisedWeights[, t - 1])

    # Propagate
    particles[, t] <- cos(theta * particles[newAncestors, t-1]) +
                      rnorm(N, 0, 1)

    #Likelihood
    weights[, t] <- dnorm(y[t], mean = particles[, t], sd = 1, log = TRUE)

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

    # Estimate the state
    xHat[t] <- mean(particles[, t])

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

  }
```

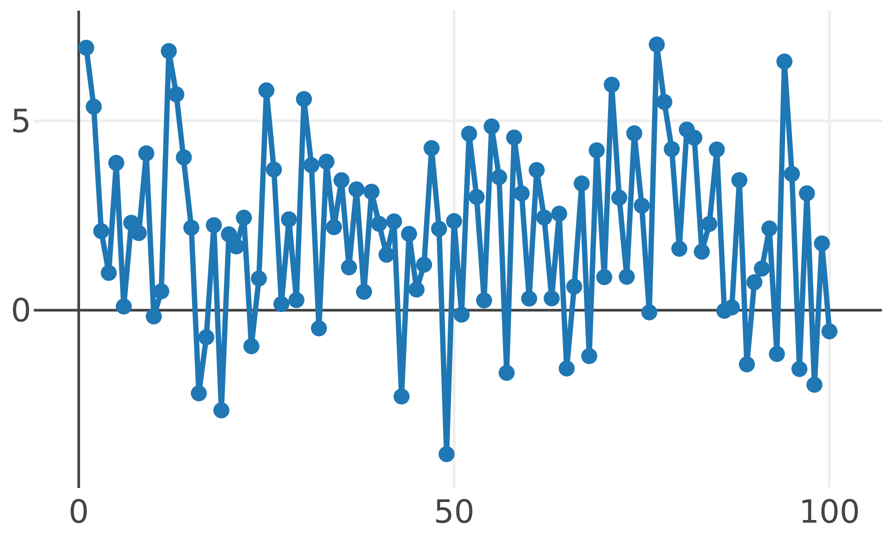
```r
  list(xHat = xHat,
       logLikelihood = logLikelihood,
       particles = particles,
       weights = normalisedWeights)

}

particleMH <- function(y, N, iter)
{
  theta <- rep(0, iter)
  theta_proposed <- rep(0, iter)
  logLikelihood <- rep(0, iter)
  logLikelihood_proposed <- rep(0, iter)

  # initialize
  theta[1] <- rnorm(1, 0 ,1)
  param <- c(theta[1], 1, 1)
  result <- bootstrapParticleFilter(y, param, N)
  logLikelihood[1]<- result$logLikelihood

  for (k in 2:iter) {
    # Propose a new parameter
    theta_proposed[k] <- theta[k - 1] + rnorm(1)

    param <- c(theta_proposed[k], 1, 1)
    result <- bootstrapParticleFilter(y, param, N)
    logLikelihood_proposed[k] <- result$logLikelihood


    # Compute the acceptance probability

    numerator = logLikelihood_proposed[k] + dnorm(theta_proposed[k], log = TRUE)
    denominator = logLikelihood[k-1] + dnorm(theta[k-1], log = TRUE)

    # Compute acceptance probability
    acc_prob = exp(numerator - denominator)
    acc_prob = min(1, acc_prob)

    u <- runif(1)
    if (u < acc_prob) {
      # Accept the parameter
      theta[k] <- theta_proposed[k]
      logLikelihood[k] <- logLikelihood_proposed[k]
    } else {
      # Reject the parameter
      theta[k] <- theta[k - 1]
      logLikelihood[k] <- logLikelihood[k - 1]
    }
  }
  return(theta)
}

theta <- particleMH(y = simulatedData$y, N, 10000)
hist(theta[2000:10000], breaks = 100, lty=0,
     main = "Posterior distribution of theta", col = "lightblue")
```



IV.2 Conditional particle

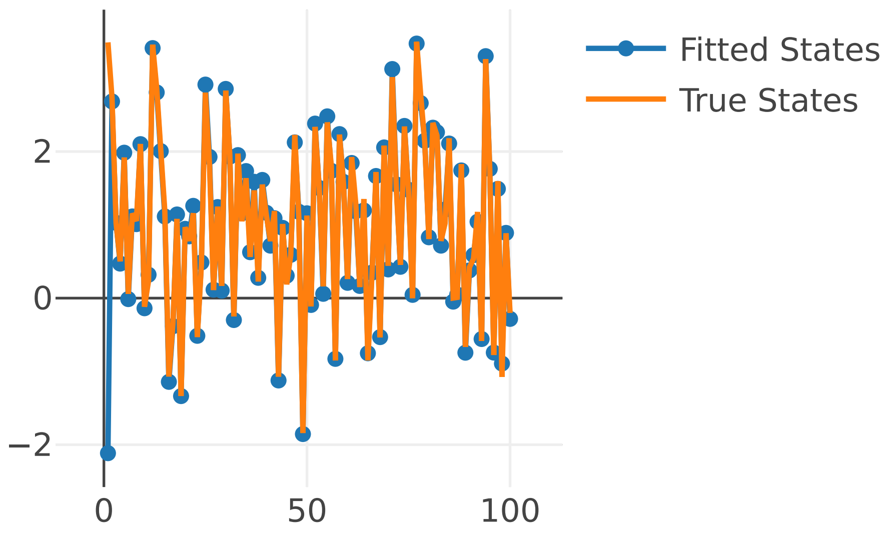
Fully adapted particle filter is transformed into conditional particle fiter.

```
T <- 100
N <- 100
# simulate data
simulateData <- function(param, T)
{
  sigma_v <- param[1]
  sigma_e <- param[2]

  x <- rep(0, T)
  y <- rep(0, T)

  #initial state
  x0 <- 0
  x[1] <- 2*cos(x0)^2 + rnorm(1, 0, sigma_v)
  y[1] <- 2 * x[1] + rnorm(1, 0 ,sigma_e)

  for(t in 2:T)
  {
    x[t] <- 2*cos(x[t-1])^2 + rnorm(1, 0, sigma_v)
    y[t] <- 2*x[t] +rnorm(1, 0, sigma_e)
  }

  return(list(x=x, y=y))
}

param <- c(1, .1)
simulatedData <- simulateData(param, T)
plot_ly(x = c(1:T), y = simulatedData$y, name = 'Observed data', type = 'scatter', mode = 'lines+markers')
```



```
########## Conditions Fully Adapted Particle Filter#############

conditionalAPF <- function(x_ref, y, N)
{
  T <- length(y)

  # Initialize variables
  particles <- matrix(0, nrow = N, ncol = T)
  weights <- matrix(1, nrow = N, ncol = T)
  normalisedWeights <- matrix(0, nrow = N, ncol = T)
  xHat <-rep(0, T)
  logLikelihood <- 0

  #state at t=0
  x0 <- rnorm(N, 0, 1)
  # Replace last state with state from reference trajectory
  x0[1] = x_ref[1]

  particles[ ,1] <- x0
  normalisedWeights[, 1] = 1 / N

  for (t in 2:T) {
    # compute resampling weights
    weights[, t] <- dnorm(y[t], mean = 2*cos(particles[, t-1])^2,
                          sd = sqrt(4.01), log = TRUE)
    max_weight <- max(weights[, t])
    weights[, t] <- exp(weights[, t] - max_weight)
    normalisedWeights[, t] <- weights[, t] / sum(weights[, t])

    newAncestors <- sample(N, replace = TRUE, prob = normalisedWeights[, t])

    # Propagate
    particles[, t] <-   0.1/sqrt(4.01)*rnorm(N, 0, 1) +
                       (2/4.01) * y[t]  + .01/4.01 *
                       cos(particles[newAncestors, t-1])^2

    # Replace last sample with reference trajectory
    newAncestors[N] = N
    particles[N, t] = x_ref[t]
  }

  b =  sample(1:N, 1)
  return(particles[b, ])
```

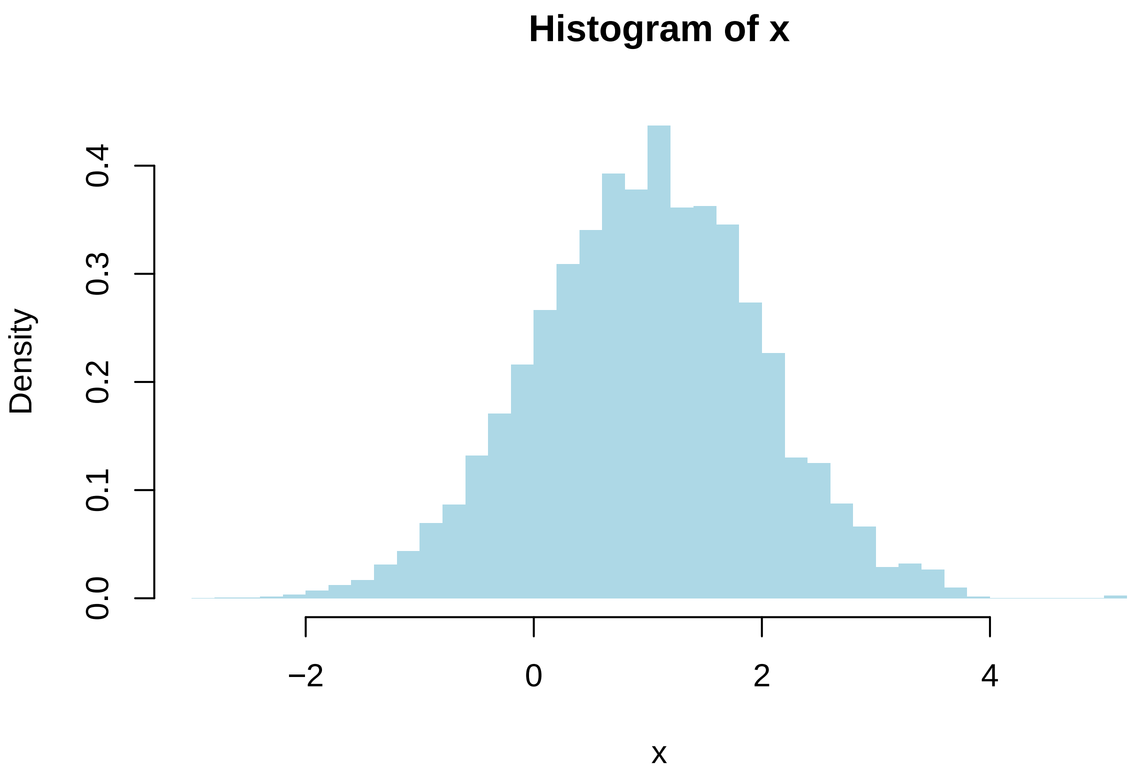
```r
}

x = conditionalAPF(x_ref = simulatedData$x, y = simulatedData$y, N)
p <- plot_ly(x = c(1:T), y = x, name = 'Fitted States', type = 'scatter', mode = 'lines+markers')
add_lines(p, x = c(1:T), y = simulatedData$x, name = 'True States', type = 'scatter', mode = 'lines+markers')
```



IV.3 Conditional importance sampling

(a) With N=2 particles, sample from $\pi(x) = \mathcal{N}\left(x \,\middle|\, 1,\, 1\right)$ by using conditional importance sampling with the proposal $q(x) = \mathcal{N}\left(x \,\middle|\, 0,\, 1\right)$

```r
condImpSamplingKernel <- function(x_ref, N=2)
{
  # Sample from proposal
  particles <- rnorm(N)

  # Set the last sample to the reference
  particles[N] = x_ref

  # Calculate weights
  w = dnorm(particles, mean = 1, sd = 1, log = TRUE) -
      dnorm(particles, mean = 0, sd = 1, log = TRUE)
  w = w - max(w)
  w = exp(w) / sum(exp(w))

  return(particles[sample(N, size = 1, prob = w)])
}

condImpSampling <- function(iter = 100, N = 2)
{
  # Initialize variables
  particles <- rep(0, iter)
  particles[1] <- rnorm(1)
  for (i in 2:iter) {
    particles[i] = condImpSamplingKernel(particles[i-1], N = N)
  }
  return(particles)
}

x <- condImpSampling(50000)
hist(x, breaks = 50, prob = TRUE, col = "lightblue", lty=0)
```



IV.4 An SMC sampler for localization

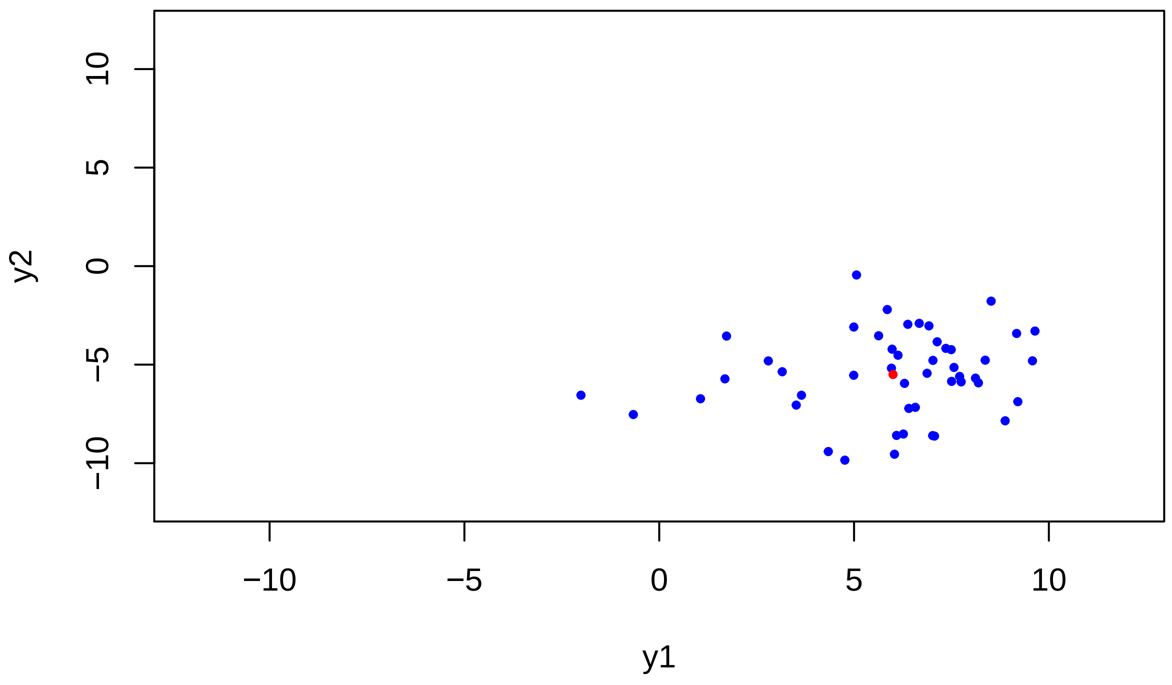
(a) Simulate M independent measurements from the model

$$y_m^1 = x_0^1 + n_m^1 b_m^1$$
$$y_m^2 = x_0^2 + n_m^2 b_m^2$$

where

$$m = 1, 2, \ldots, M$$
$$x_0 = \left(x_0^1, x_0^2\right)$$
$$n_m^1, n_m^2 \sim \mathrm{Exp}\,(2)$$
$$\mathbb{P}\left(b_m^1 = 1\right) = \mathbb{P}\left(b_m^1 = -1\right) = \frac{1}{2}$$
$$\mathbb{P}\left(b_m^2 = 1\right) = \mathbb{P}\left(b_m^2 = -1\right) = \frac{1}{2}$$

```r
M = 50
x0 = c(6, -5.5)
n1 = rexp(M, 1/2)
n2 = rexp(M, 1/2)
n <- cbind(n1, n2)

b1 = sample(c(-1,1), size = M, replace = TRUE, prob = c(0.5, 0.5))
b2 = sample(c(-1,1), size = M, replace = TRUE, prob = c(0.5, 0.5))
b <- cbind(b1, b2)

y1 = x0[1] + n1*b1
y2 = x0[2] + n2*b2

y = cbind(y1, y2)
plot(y, xlim = c(-12,12), ylim = c(-12,12), cex = .5, col = "blue", pch = 19)
lines(x0[1], x0[2], col = "red", type = "p", cex = .5, pch = 19)
```



(b) Likelihood The components of y_m are independent

$$p\left(y_m \mid x_0\right) = p\left(y_m^1 \mid x_0^1\right) \cdot p\left(y_m^2 \mid x_0^2\right)$$

where

$$p\left(y_m^i \mid x_0^i\right) = \frac{1}{4}\exp\left(-\frac{abs(y_m^i - x_0^i)}{2}\right)$$

```r
logLikeliHood <- function(x, y)
{
  sum_ll <- c(0, 0)

  for (m in 1:M) {
    sum_ll <- sum_ll + log(0.25) + 0.5 * c(-1, -1) * abs(y[m, ] - x)
  }
  return(sum(sum_ll))
}
```

(c) Tempered transition $\pi_0, ..., \pi_k$, given

$$\pi_0 \sim N\left(\left(\begin{array}{c} 0 \\ 0 \end{array}\right), 7 * I_2\right)$$

```r
tempLogLikeliHood <- function(x, y, k, K=10)
{
  temp <- (k / K) * logLikeliHood(x, y) + dmvnorm(x, mean=c (0, 0), sigma = 7 * diag(2), log = TRUE)
  return(temp)
}
```

(d) π_n-invariant Metropolis-Hastings (MH) kernel based on a random walk proposal.

```r
MHKernel <- function(x, y, k, K=10, theta = 0.5)
{
  # A new proposal
  xProposed = rmvnorm(1, mean = x, sigma =  theta^2 * diag(2))
  # Acceptance probability
  #print(tempLogLikeliHood(xProposed, y, k, K=K))
  acc_prob <- tempLogLikeliHood(xProposed, y, k, K=K) -
              tempLogLikeliHood(x, y, k, K=K)
  acc_prob <- min(1, exp(acc_prob))
  # Sample from uniform to be compared to the acceptance probability
  u = runif(1)

  # Set next state depending on acceptance probability
  if(u <= acc_prob){
    return(xProposed)
  }
  else{
    return(x)
  }
}

MHKernel(x0, y, 1)
```

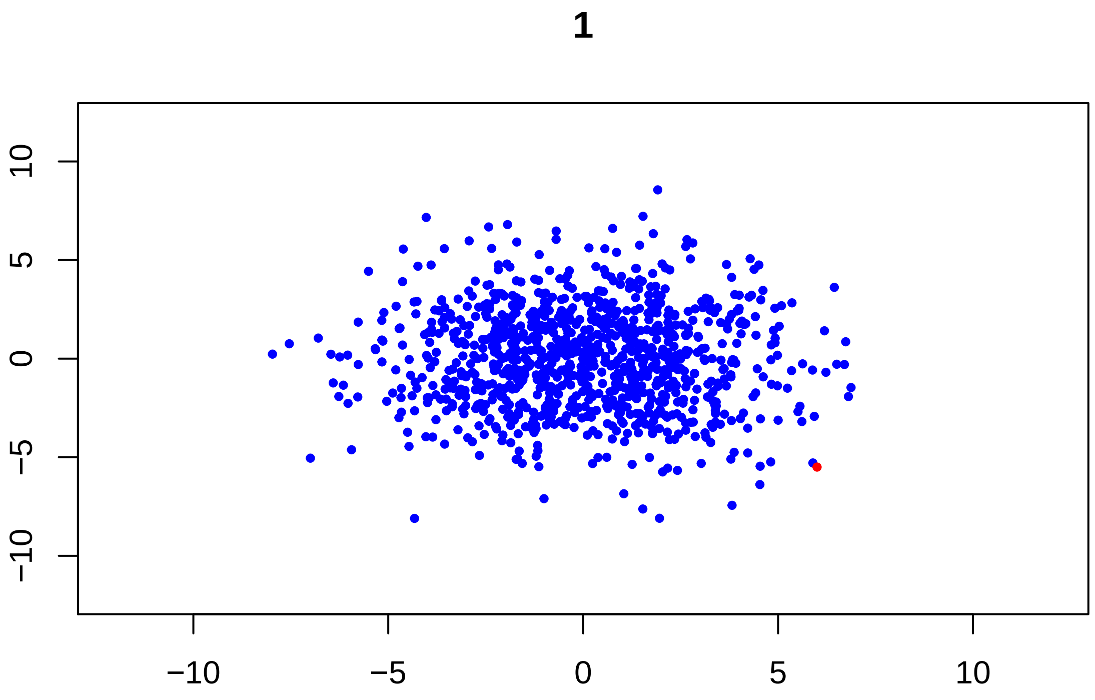
```
##          [,1]      [,2]
## [1,] 6.354334 -5.036059
```

```r
MHKernel(c(0,0), y, 1)
```

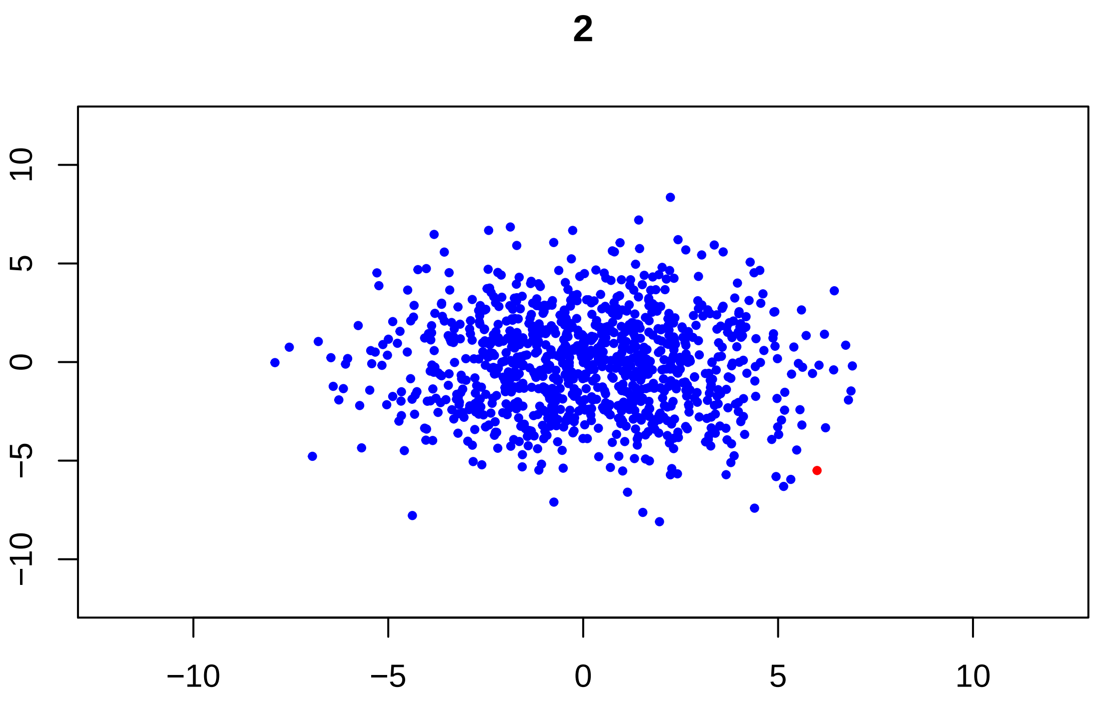
```
##            [,1]       [,2]
## [1,] -0.3577252 -0.2695782
```

(e) Put everything together in an SMC sampler

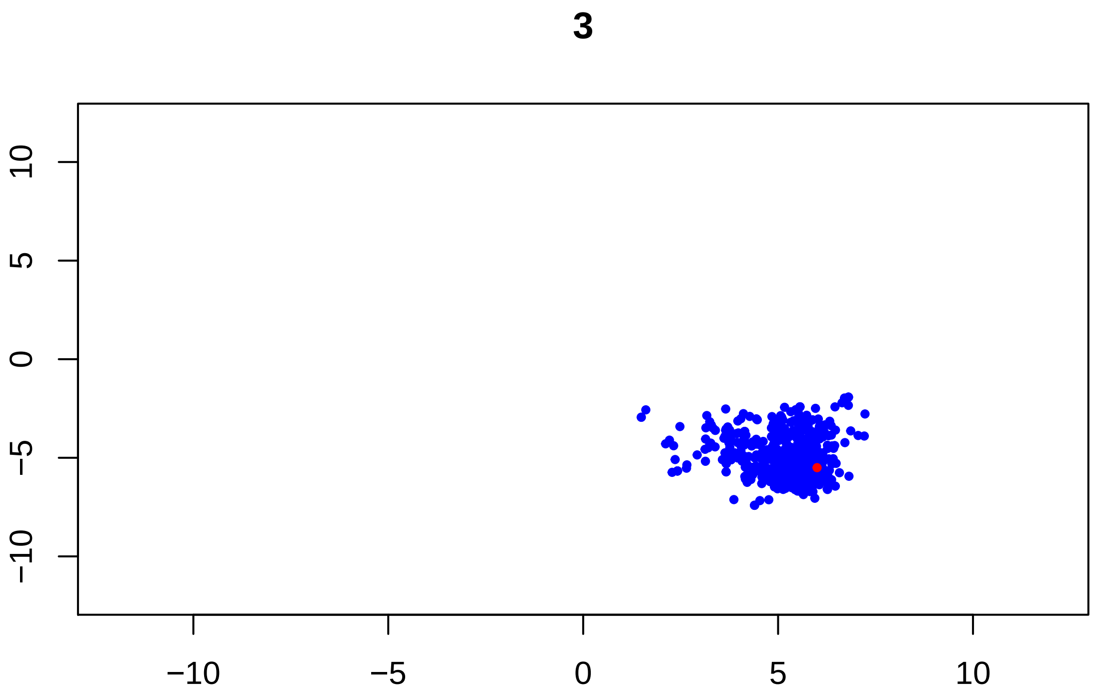
```r
smcSampler <- function(y, K=10, N=100, theta=.5)
{
  # Initialize variables
  particles <- array(0, c(N, K+1, 2))
  ancestorIndices <- matrix(0, nrow = N, ncol = K+1)
  weights <- matrix(1, nrow = N, ncol = K+1)
  normalisedWeights <- matrix(0, nrow = N, ncol = K+1)

  # Initial state
  xInit = rmvnorm(N, mean=c(0,0), 7*diag(2))
  ancestorIndices[, 1] <- 1:N
  particles[ ,1,] <- xInit
  normalisedWeights[, 1] = 1 / N

  for(t in 2:(K+1))
  {
    # Update weights
    for (i in 1:N)
    {
      weights[i, t] = log(normalisedWeights[i, t-1]) +
        tempLogLikeliHood(particles[i, t-1, ], y, k=t, K=K) -
        tempLogLikeliHood(particles[i, t-1, ], y, k=t-1, K=K)
    }
    # normalize weights
    weights[, t] = weights[, t] - max(weights[, t])
    normalisedWeights[, t] = exp(weights[, t]) / sum(exp(weights[, t]))

    newAncestors <- 1:N
    # Resample
    if(1/sum(normalisedWeights[, t-1]^2) < N/2){
      newAncestors <- sample(N, replace = TRUE, prob = normalisedWeights[, t-1])
      normalisedWeights[, t-1] <- 1/N
    }

    ancestorIndices[, t-1] <- newAncestors

    particles[, t, ] <- particles[newAncestors, t-1, ]
    # Propagate  using MHKernel
    for (i in 1:N)
    {
      particles[i, t, ] = MHKernel(particles[i, t, ], y,
                                   k=t, K=K, theta=theta)
    }
  }
  return(list(particles = particles, weights = normalisedWeights))
}

K=10
result = smcSampler(y, N=1000, K = K)

par(mfrow=c(1, 1))
for (i in 1:11) {
  plot(result$particles[, i, 1], result$particles[, i, 2], xlim = c(-12,12), ylim = c(-12,12), cex = .5, col =
  lines(x0[1], x0[2], col = "red", type = "p", cex = .5, pch = 19)
  print(i)
}
```



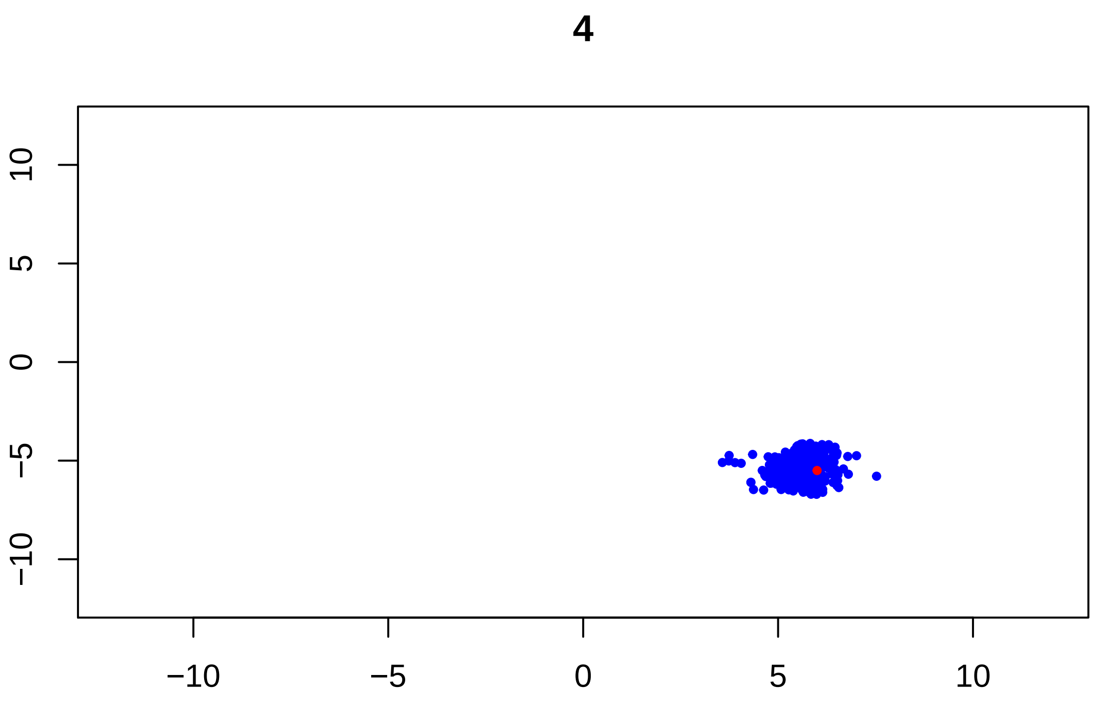
```
## [1] 1
```



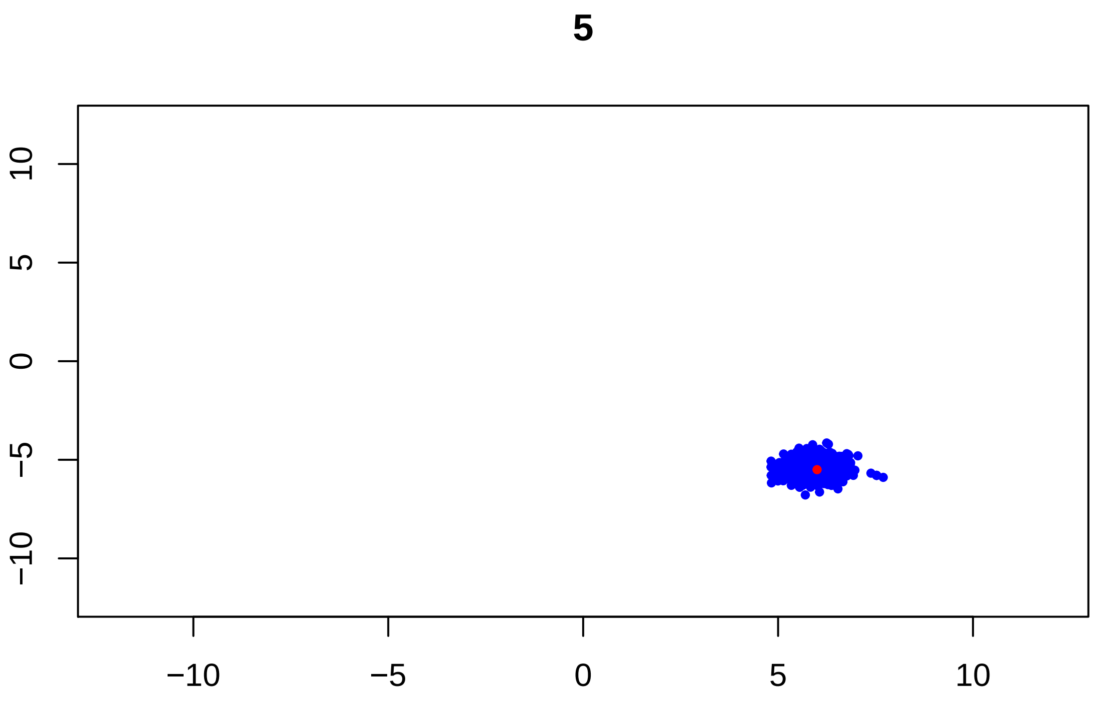
```
## [1] 2
```



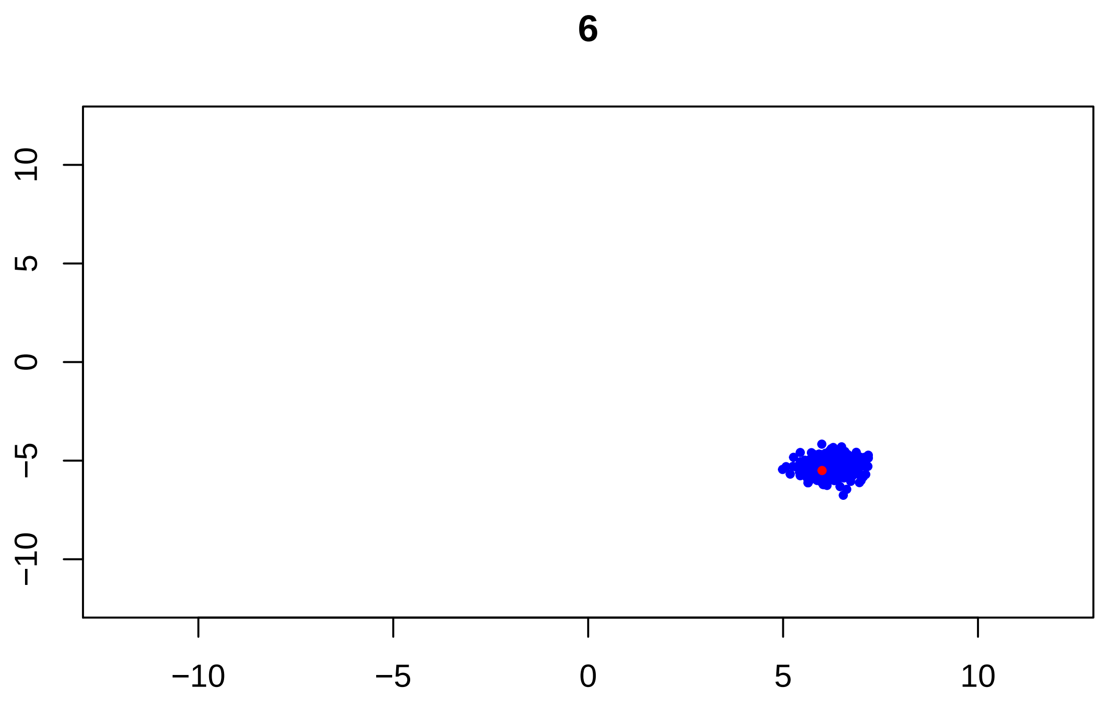
```
## [1] 3
```



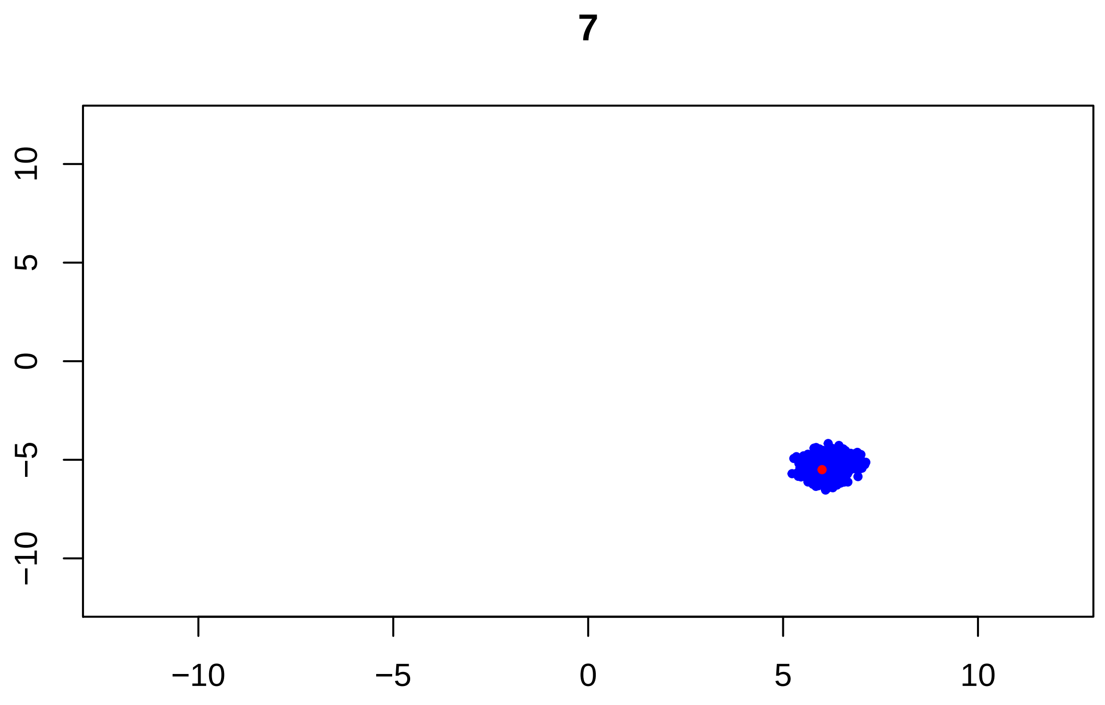
```
## [1] 4
```



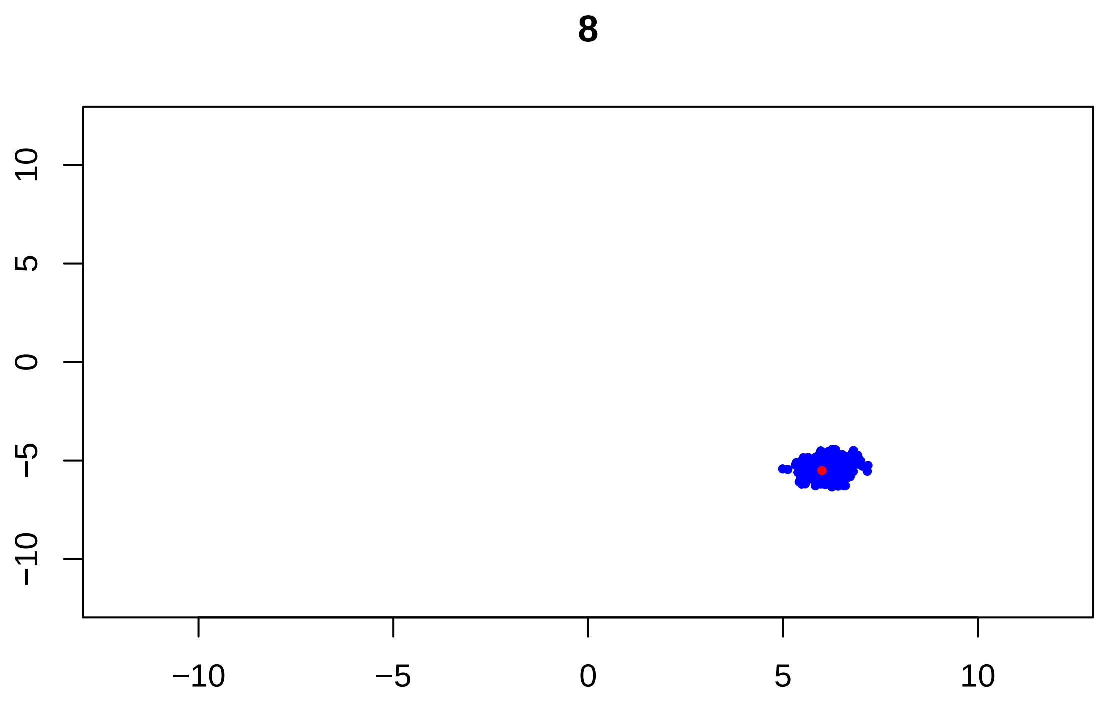
```
## [1] 5
```



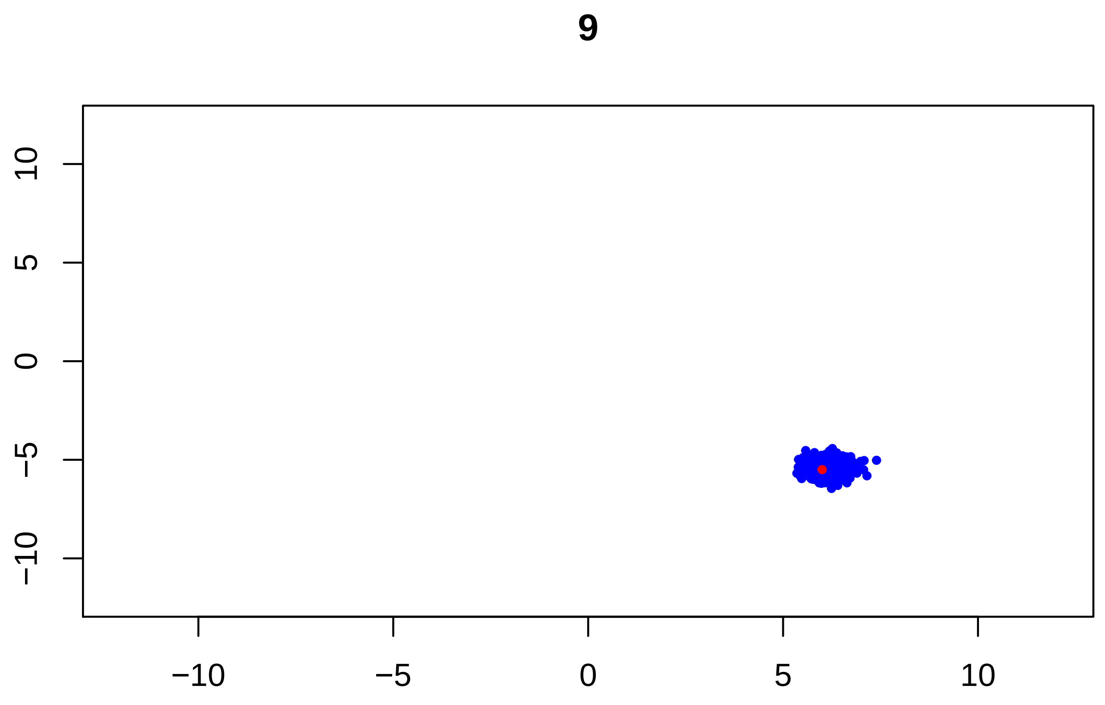
```
## [1] 6
```



```
## [1] 7
```

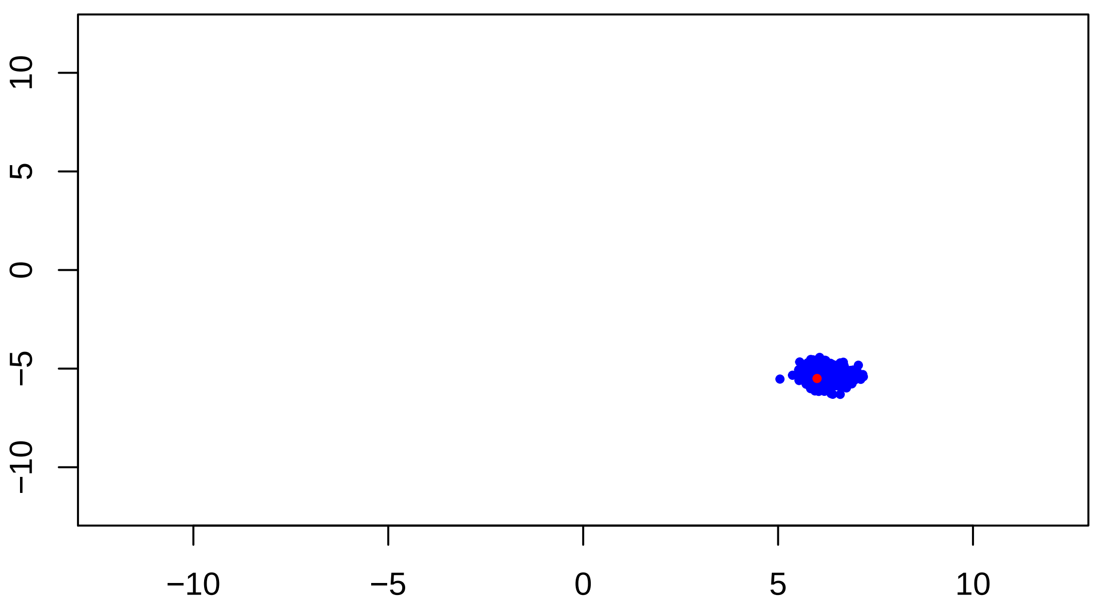


```
## [1] 8
```



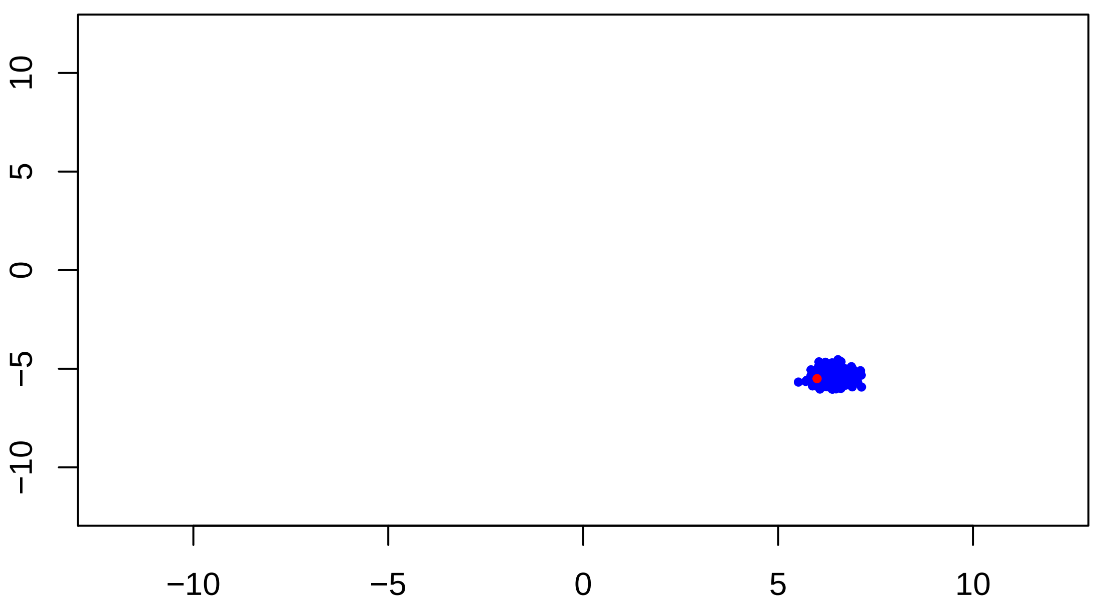
```
## [1] 9
```

10



```
## [1] 10
```

11



```
## [1] 11
```