Computational Statistics - Lab 04

Annalena Erhard (anner218) and Maximilian Pfundstein (maxpf364)

2019-02-04

Contents

1 Question 1: Computations with Metropolis-Hastings

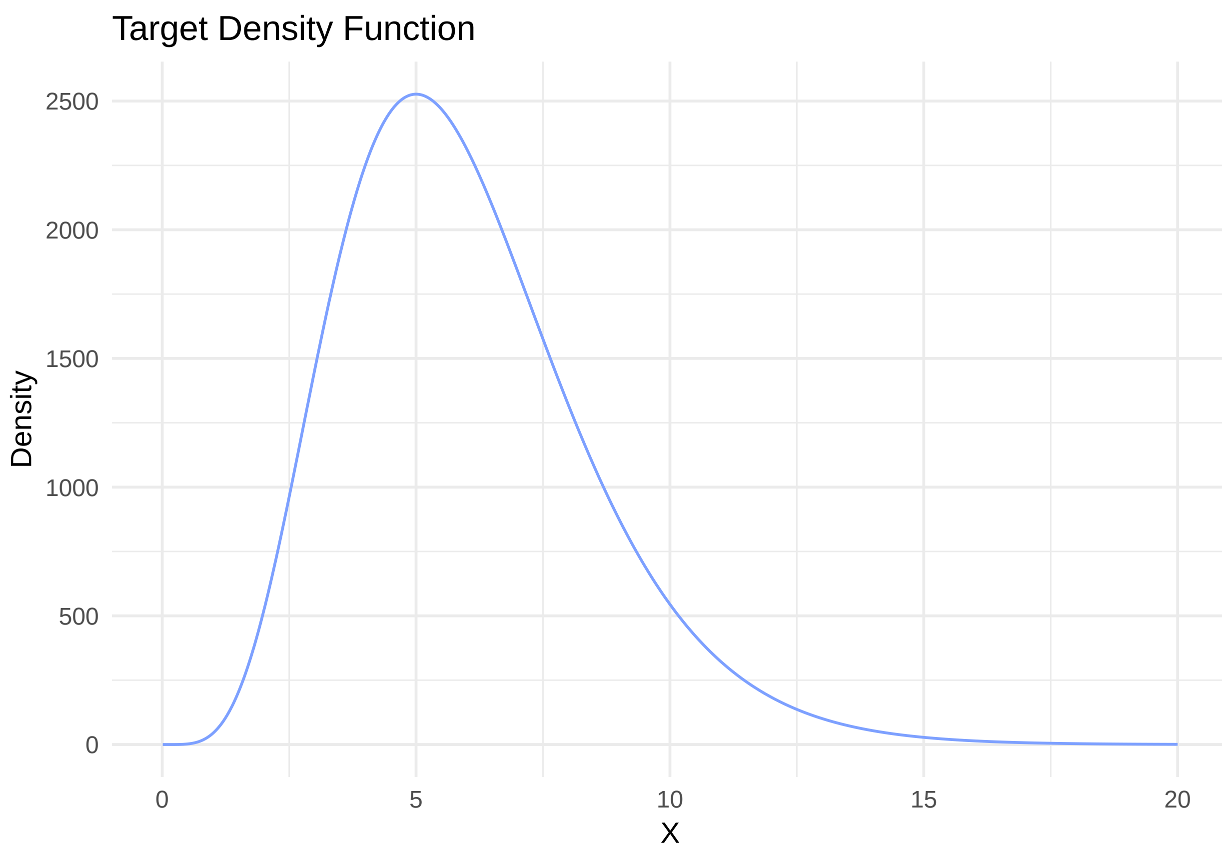
Consider the following probability density function:

$$f(x) \propto x^5 e^{-x}, \quad x > 0.$$

You can see that the distribution is known up to some constant of proportionality. If you are interested (**NOT** part of the Lab) this constant can be found by applying integration by parts multiple times and equals 120.

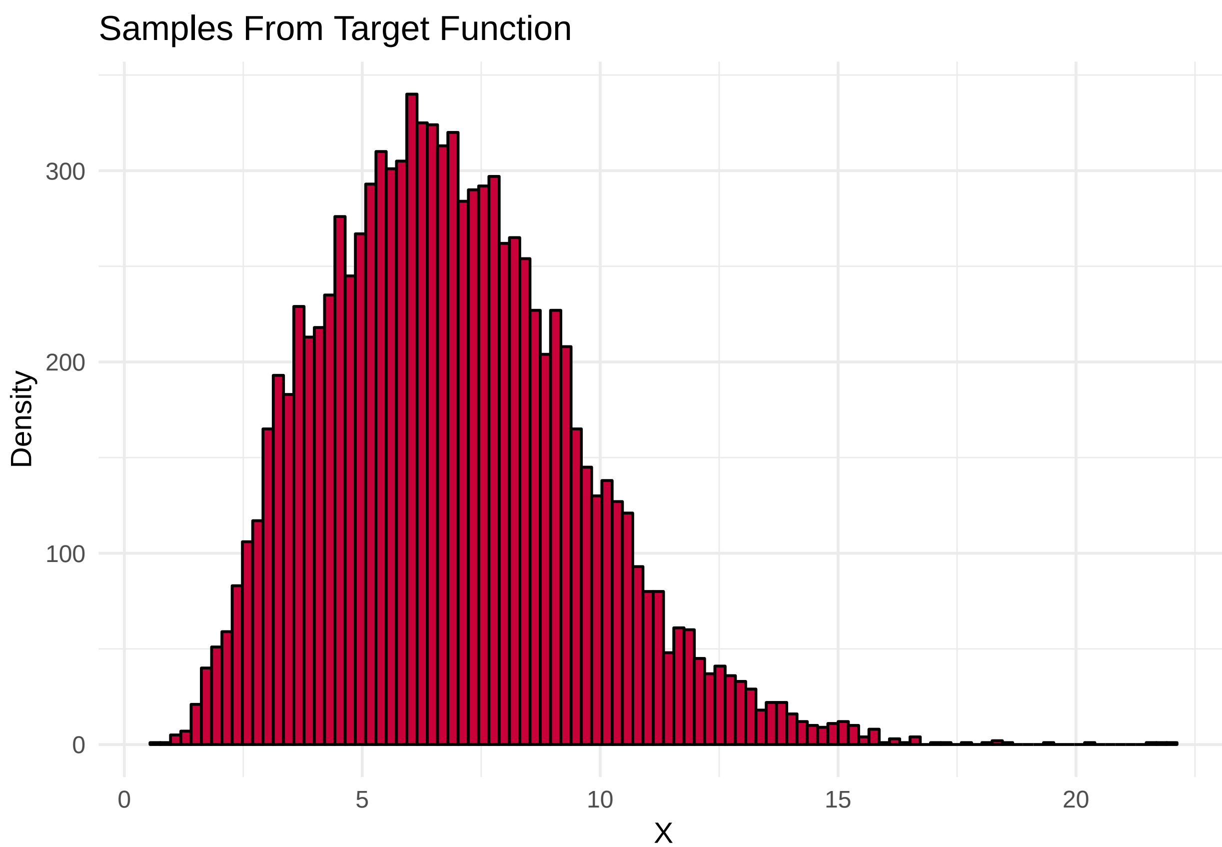
1. Use Metropolis–Hastings algorithm to generate samples from this distribution by using proposal distribution as log–normal $LN(X_t, 1)$, take some starting point. Plot the chain you obtained as a time series plot. What can you guess about the convergence of the chain? If there is a burn–in period, what can be the size of this period?

```
# Target function with original scaling
f = function(x) {
  return(120 * x^5 * exp(-x))
}

# Target function
df = function(x) {
  if (sum(c(x) <= 0) > 0) stop("x must be larger than 0.")
  return(x^5 * exp(-x))
}
```




```r
#' Metropolis Hasting Algorithm
#'
#' @param n Number of samples from the target distribution.
#' @param x_0 Initial state from Pi.
#' @param b Burn-in steps before taking samples.
#'
#' @return Returns a list containing the burned and normal samples.
#' @export
#'
#' @examples
metropolis_hasting = function(n = 1, x_0 = 1, b = 50) {

  # Vectors to store the samples in.
  burn_in_samples = c()
  samples = c()

  # Samples from proposel from the random walk (MC)
  xt = x_0
  xt_1 = x_0

  walk = function(burn) {
    # Generate proposal state
    x_star = rlnorm(n = 1, meanlog = xt, sdlog = 1)

    # Calculate correction factor C
    c = dlnorm(xt_1, meanlog = xt, sdlog = 1) /
```

```r
      dlnorm(x_star, meanlog = xt, sdlog = 1)

    # Calculate acceptance probability alpha
    alpha = min(1, df(x_star)/df(xt_1))

    # Generate u from uniform
    u = runif(n = 1, min = 0, max = 1)

    # Decide if to accept or reject the proposal
    if (u <= alpha) {
      # Accept
      xt_1 <<- xt
      xt <<- x_star

      # Depending on if we're still buring save in different vector
      if (burn) {
        burn_in_samples <<- c(burn_in_samples, x_star)
      }
      else {
        samples <<- c(samples, x_star)
      }
    }
    else {
      # Reject
      xt <<- xt_1
    }
  }

  # Burn-in period
  while (length(burn_in_samples) < b) {
    walk(TRUE)
  }

  # Draw real samples
  while (length(samples) < n) {
    walk(FALSE)
  }

  # Return the burned and normal samples
  return(list(samples = samples, burn_in_samples = burn_in_samples))
}

results = metropolis_hasting(10000, x_0 = 5, b = 100)
```



2. Perform Step 1 by using the chi–square distribution $\chi^2(\lfloor X_t + 1 \rfloor)$ as a proposal distribution, where $\lfloor x \rfloor$ is the floor function, meaning the integer part of x for positive x, i.e. $\lfloor 2.95 \rfloor = 2$.
3. Compare the results of Steps 1 and 2 and make conclusions.
4. Generate 10 MCMC sequences using the generator from Step 2 and starting points 1, 2, . . . , or 10. Use the Gelman–Rubin method to analyze convergence of these sequences.
5. Estimate

$$\int_0^\infty x f(x) dx$$

using the samples from Steps 1 and 2.

6. The distribution generated is in fact a gamma distribution. Look in the literature and define the actual value of the integral. Compare it with the one you obtained.

2 Question 2: Gibbs Sampling

concentration of a certain chemical was measured in a water sample, and the result was stored in the data `chemical.RData` having the following variables:

- X : day of the measurement
- Y : measured concentration of the chemical.

The instrument used to measure the concentration had certain accuracy; this is why the measurements can be treated as noisy. Your purpose is to restore the expected concentration values.

1. Import the data to R and plot the dependence of Y on X. What kind of model is reasonable to use here?
2. A researcher has decided to use the following (random–walk) Bayesian model (n=numberof observations, $\vec{\mu} = (\mu_1, ..., \mu_n)$ are unknown parameters):

$$Y_i = \mathcal{N}(\mu_i, \text{variance} = 0.2), \quad i = 1, ..., n$$

where the prior is

$$p(\mu_1) = 1$$

$$p(\mu_{i+1}|\mu_i) = \mathcal{N}(\mu_i, 0.2), i = 1, ..., n1.$$

Present the formulae showing the likelihood $p(\vec{Y}|\vec{\mu})$ and the prior $p(\vec{\mu})$. **Hint:** a chain rule can be used here $p(\vec{\mu}) = p(\mu_1)p(\mu_2|\mu_1)p(\mu_3|\mu_2)...p(\mu_n|\mu_n 1)$

3. Use Bayes' Theorem to get the posterior up to a constant proportionality, and then find out the distributions of $(\mu_i|\vec{\mu}_{-i}, \vec{Y})$, where μ_{-i} is a vector containing all μ values except of μ_i.

Hint A: consider for separate formulae for $(\mu_1|\vec{\mu}_{-1}, \vec{Y}), (\mu_n|\vec{\mu}_{-n}, \vec{Y})$ and then a formula for all remaining $(\mu_i|\vec{\mu}_{-i}, \vec{Y})$.

Hint B:

$$e^{-\frac{1}{d}\left((x-a)^2+(x-b)^2\right)} \propto e^{-\frac{(x-(a+b)/2)^2}{d/2}}$$

Hint C:

$$e^{-\frac{1}{d}\left((x-a)^2+(x-b)^2+(x-c)^2\right)} \propto e^{-\frac{(x-(a+b+c)/3)^2}{d/3}}$$

4. Use the distributions derived in Step 3 to implement a Gibbs sampler that uses $\vec{\mu}^0 = 0, ..., 0$ as a starting point. Run the Gibbs sampler to obtain 1000 values of $\vec{\mu}$ and then compute the expected value of $\vec{\mu}$ versus X and Y versus X in the same graph. Does it seem that you have managed to remove the noise? Does it seem that the expected value of $\vec{\mu}$ can catch the true underlying dependence between Y and X?
5. Make a trace plot for μ_n and comment on the burn–in period and convergence.

```
# Loading RData
#data2 = get(load("data.RData"))
#head(data2)
```

3 Source Code

```
knitr::opts_chunk$set(echo = TRUE, cache = FALSE, include = TRUE, eval = TRUE)
library(knitr)
library(readxl)
library(ggplot2)
library(gridExtra)

# Target function with original scaling
f = function(x) {
```

```r
  return(120 * x^5 * exp(-x))
}

# Target function
df = function(x) {
  if (sum(c(x) <= 0) > 0) stop("x must be larger than 0.")
  return(x^5 * exp(-x))
}


sequence = seq(from = 0.01, to = 20, by = 0.01)

real_f = f(sequence)
plotdf = data.frame(sequence, real_f)

ggplot(plotdf) +
  geom_line(aes(x = sequence, y = real_f), color = "#7da0ff") +
  labs(title = "Target Density Function", y = "Density",
  x = "X", color = "Legend") +
  theme_minimal()


#' Metropolis Hasting Algorithm
#'
#' @param n Number of samples from the target distribution.
#' @param x_0 Initial state from Pi.
#' @param b Burn-in steps before taking samples.
#'
#' @return Returns a list containing the burned and normal samples.
#' @export
#'
#' @examples
metropolis_hasting = function(n = 1, x_0 = 1, b = 50) {

  # Vectors to store the samples in.
  burn_in_samples = c()
  samples = c()

  # Samples from proposel from the random walk (MC)
  xt = x_0
  xt_1 = x_0

  walk = function(burn) {
    # Generate proposal state
    x_star = rlnorm(n = 1, meanlog = xt, sdlog = 1)

    # Calculate correction factor C
    c = dlnorm(xt_1, meanlog = xt, sdlog = 1) /
      dlnorm(x_star, meanlog = xt, sdlog = 1)

    # Calculate acceptance probability alpha
    alpha = min(1, df(x_star)/df(xt_1))
```

```r
    # Generate u from uniform
    u = runif(n = 1, min = 0, max = 1)

    # Decide if to accept or reject the proposal
    if (u <= alpha) {
      # Accept
      xt_1 <<- xt
      xt <<- x_star

      # Depending on if we're still buring save in different vector
      if (burn) {
        burn_in_samples <<- c(burn_in_samples, x_star)
      }
      else {
        samples <<- c(samples, x_star)
      }
    }
    else {
      # Reject
      xt <<- xt_1
    }
  }

  # Burn-in period
  while (length(burn_in_samples) < b) {
    walk(TRUE)
  }

  # Draw real samples
  while (length(samples) < n) {
    walk(FALSE)
  }

  # Return the burned and normal samples
  return(list(samples = samples, burn_in_samples = burn_in_samples))
}

results = metropolis_hasting(10000, x_0 = 5, b = 100)


plotdf = data.frame(results$samples)

ggplot(plotdf) +
  geom_histogram(aes(x = results.samples),
                 color = "#000000", fill = "#C70039", bins = length(results$samples)/100) +
  labs(title = "Samples From Target Function", y = "Density",
  x = "X", color = "Legend") +
  theme_minimal()


# Loading RData
#data2 = get(load("data.RData"))
#head(data2)
```