Computational Statistics (732A90) Lab03

Christophoros Spyretos, Marc Braun, Marketos Damigos, Patrick Siegfried Hiemsch & Prakhar

2021-11-26

Question 1

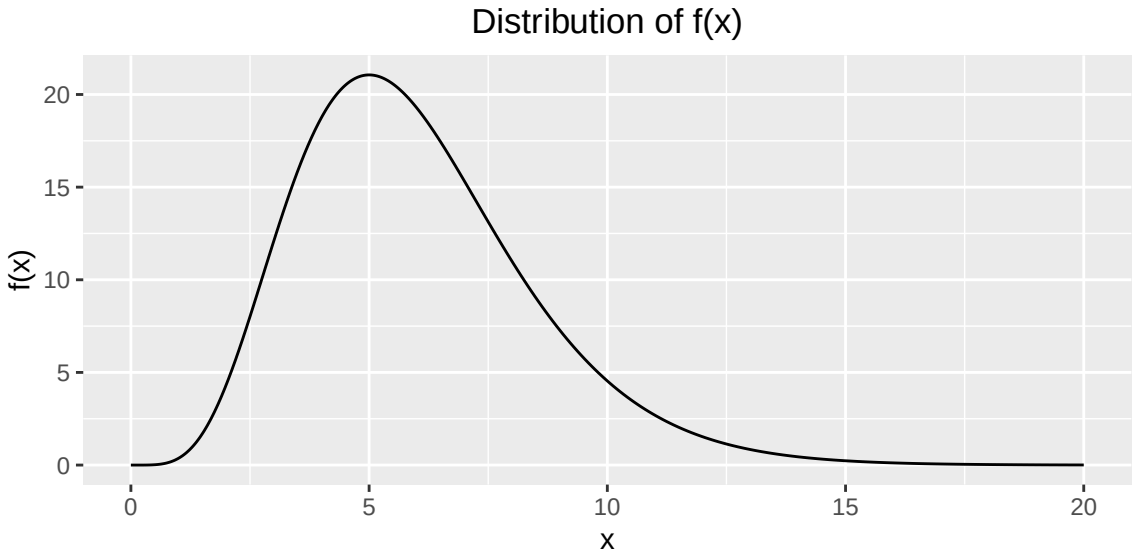
In this exercise, we will be sampling from a distribution f using the Metropolis-Hastings-method with two different proposal distributions, the Lognormal and the Chi-squared distribution. The function f we want to sample from is defined by the following density:

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

Task 1

We first implement function f and plot it, to get a first overview:

```
library(ggplot2)

f <- function(x) {
  if (x > 0) {result <- x^5 * exp(-x)}
  else {result <- 0}
  return(result)
}

x <- seq(0, 20, 0.01)
to_plot <- data.frame(x = x,
                      f = sapply(x, f),
                      lognorm = sapply(x, dlnorm))


f_plot <- ggplot(to_plot) +
  geom_line(mapping = aes(x, f)) +
  labs(x = "x", y = "f(x)", title = "Distribution of f(x)") +
  theme(plot.title = element_text(hjust = 0.5))

f_plot
```



Similar to the lecture, we estimate the 2.5% and 97.5% percentile of the function f in order to use it during the Metropolis-Hastings method to plot two horizontal reference lines between which approximately 95% of the produced sample data should lie. The following code shows, that $P_{2.5\%} \approx 2.2$ and $P_{97.5\%} \approx 11.7$:

```
# compute 2.5% percentile
integrate(f, lower = 0, upper = 2.2)$value/integrate(f, lower = 0, upper = Inf)$value
```

```
## [1] 0.02490975
```

```
# compute 97.5% percentile
integrate(f, lower = 0, upper = 11.7)$value/integrate(f, lower = 0, upper = Inf)$value
```

```
## [1] 0.9754842
```

Now we implement the Metropolis-Hastings algorithm with $LN(X_t, 1)$ as proposal distribution.

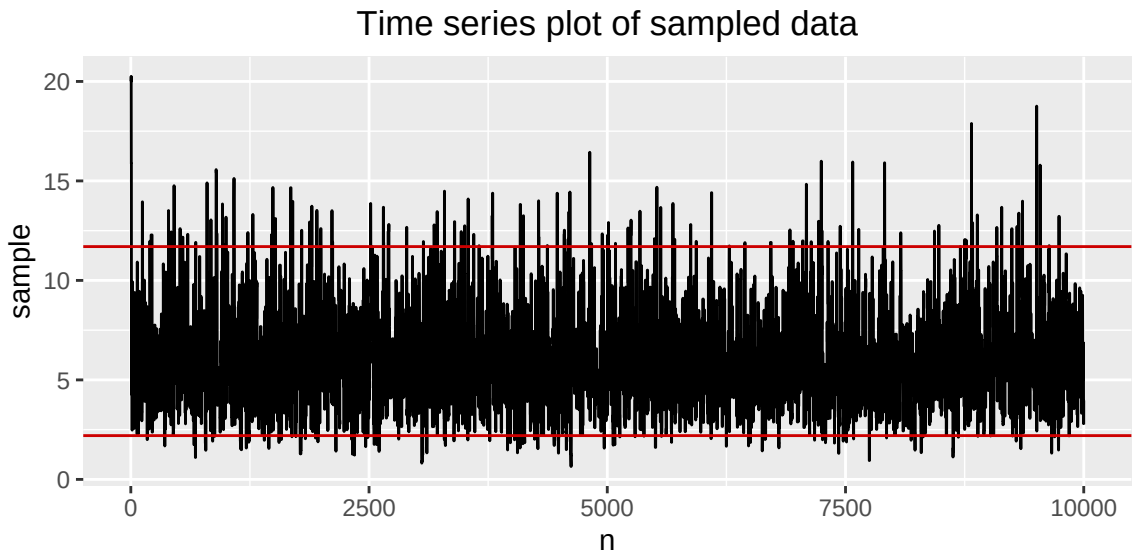
```
f_log_norm_MH <- function(n, x_0, sd, print_plot = FALSE) {
  trials <- 1:n
  samples <- c(x_0)
  for (i in 2:n) {
    x_t <- samples[i-1]
    y <- rlnorm(1, meanlog = log(x_t), sdlog = sd)
    u <- runif(1)

    # avoid denominator = 0
    if ((f(x_t) == 0) | dlnorm(x = y, meanlog = log(x_t), sdlog = sd) == 0) {
      acc_value <- 1
    }
    else {
      acc_value <- (f(y) * dlnorm(x = x_t, meanlog = log(y), sdlog = sd))/
        ((f(x_t) * dlnorm(x = y, meanlog = log(x_t), sdlog = sd)))
    }

    acc <- min(c(1, acc_value))

    if (u <= acc) {samples <- append(samples, y)}
    else {samples <- append(samples, x_t)}
  }
```

```
  data <- data.frame(n = trials,
                     sample = samples)

  plot <- ggplot(data) +
    geom_line(mapping = aes(n, sample)) +
    geom_hline(yintercept = 11.7,  color = "red3") +  # reference lines
    geom_hline(yintercept = 2.2,  color = "red3") +
    ggtitle("Time series plot of sampled data") +
    theme(plot.title = element_text(hjust = 0.5))

  if (print_plot == T) {print(plot)}

  return(data)
}
```

During the implementation of the algorithm, we first used the normal value of X_t for the "meanlog"-parameter in the lnorm-function. But this method produced strange output and lead to a range of sample values starting from 1 until approximately 5, which is very different from the real range that sampled values from function f should fall into. That is why we use $\log(X_t)$ as "meanlog", which solves this issue leading to approximately the same spread of values as observed in the function f.

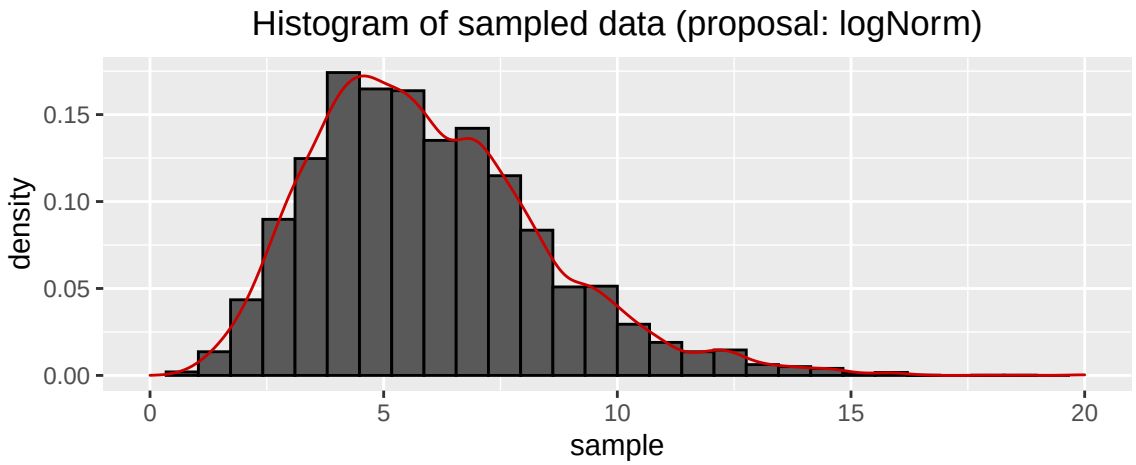
Using this defined algorithm, we now sample 1000 data points and plot function f and the histogram of the obtained samples. We use the value 20 as a starting point.

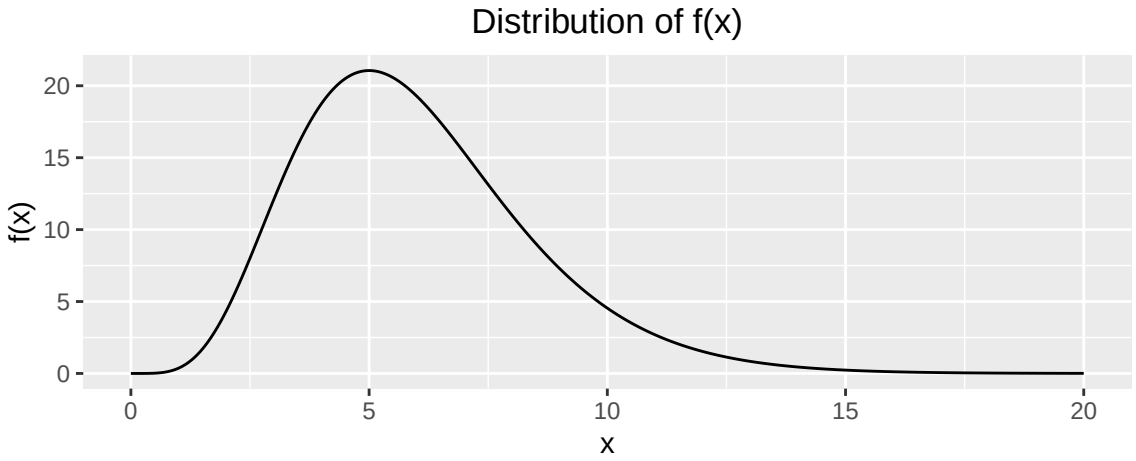
```
set.seed(123123)
data_log_norm <- f_log_norm_MH(10000, 20, sd=1, print_plot = TRUE)
```



From the above plot we can see that the big majority of sampled values fall in the interval between the two red reference lines, which represent the area where 95% of the data should lie, when sampled from function f. This is a first indicator that the Metropolis-Hastings method worked and we actually sampled data from f. Also it can be observed, that there is no real burn-in period.

To get a more detailed view on the results, we plot the histogram of the sampled data including a kernel density estimation alongside the plot of f to compare them:





We can see that both distributions look pretty similar, which is a further indicator that the sampling process worked.

Task 2

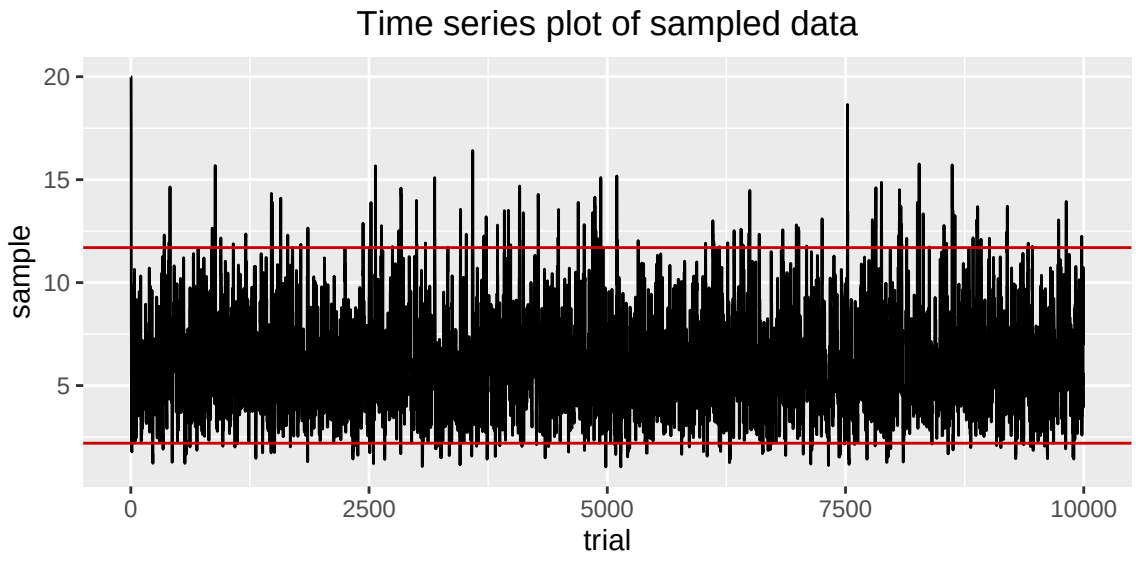
Now we repeat the same process but use $\chi^2(\lfloor X_t + 1 \rfloor)$ as a proposal distribution.

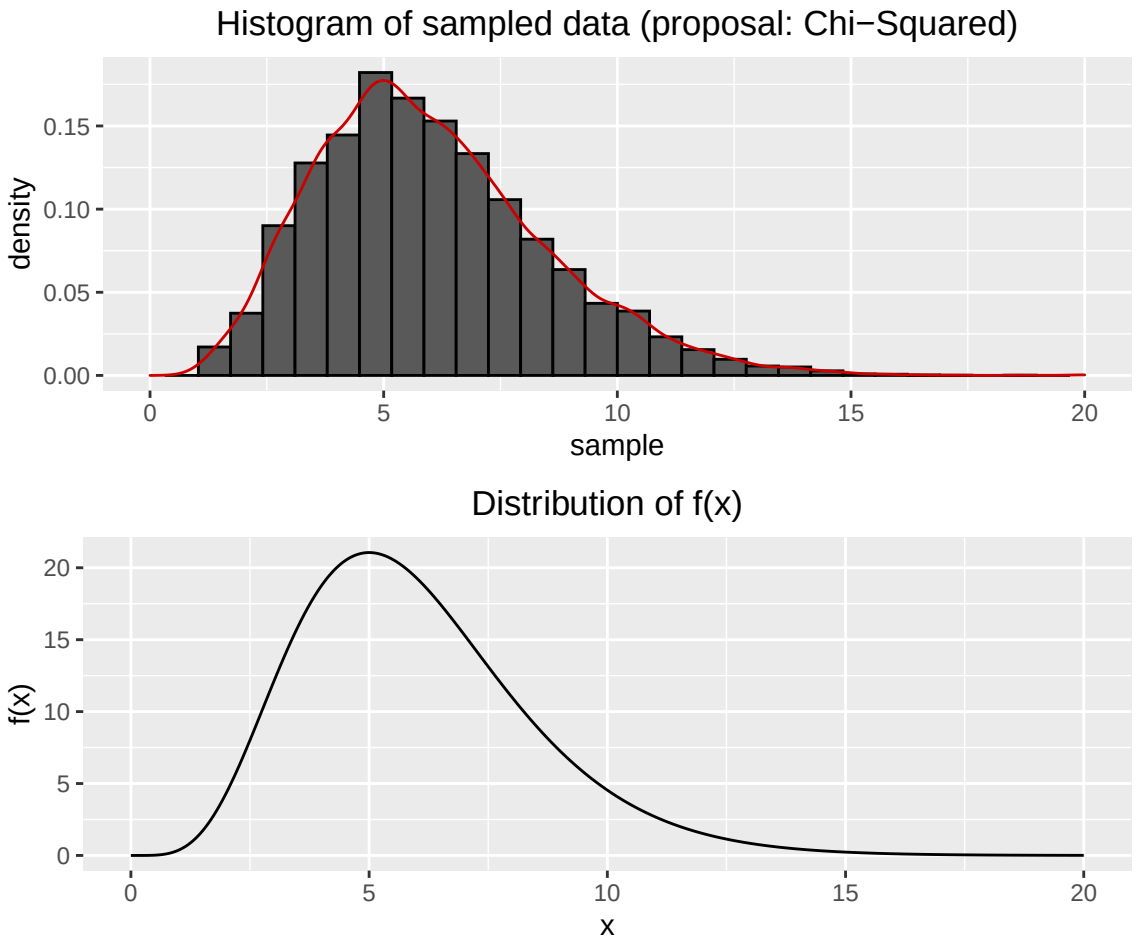
```
f_chi_sq_MH <- function(n, x_0, print_plot = FALSE) {
  trials <- 1:n
  samples <- c(x_0)
  for (i in 2:n) {
    x_t <- samples[i-1]
    y <- rchisq(1, df = floor(x_t + 1))
    u <- runif(1)

    # avoid denominator = 0
    if ((f(x_t) == 0)) {
      acc_value <- 1
    }

    else {
      acc_value <- (f(y) * dchisq(x_t, df = floor(y + 1)))/
        ((f(x_t) * dchisq(y, df = floor(x_t + 1))))
    }
```

```
    acc <- min(c(1, acc_value))

    if (u <= acc) {samples <- append(samples, y)}
    else {samples <- append(samples, x_t)}
  }

  data <- data.frame(trial = trials,
                     sample = samples)

  plot <- ggplot(data) +
    geom_line(mapping = aes(trial, sample)) +
    geom_hline(yintercept = 11.7, color = "red3") +
    geom_hline(yintercept = 2.2,  color = "red3") +
    ggtitle("Time series plot of sampled data") +
    theme(plot.title = element_text(hjust = 0.5))


  if (print_plot == T) {print(plot)}

  return(data)
}
```

Similar as before, we sample 10000 data points and use 20 as the starting value.

```
set.seed(12345)
data_chi <- f_chi_sq_MH(10000, 20, print_plot = TRUE)
```





Similar to the first approach there is no burn in period and the sampled values and their distribution indicate, that the algorithm using the new proposal function worked as well and we sampled values from f.

Task 3

As a next step, we use the Metropolis-Hastings method with $\chi^2(\lfloor X_t + 1 \rfloor)$ as a proposal distribution, to generate 10 MCMC sequences and apply the Gelman-Rubin method to analyze convergence of these sequences. We use values from 1 to 10 as starting points for the different sequences. The following implementation is based on the code in the lecture:

```
library(coda)

n <- 100
k <- 10
f1 <- mcmc.list()

X1 <- matrix(NA, ncol=k, nrow=n)

for (i in 1:10) {
  X1[,i] <- f_chi_sq_MH(n, x_0 = i)$sample
}

for (i in 1:k) {
  f1[[i]]<-as.mcmc(X1[,i])
```

```
}

print(gelman.diag(f1))
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       1.04       1.09
```

As we can see from the output, the value of the Gelman–Rubin factor is smaller than 1.2, which indicates convergence.

Task 4

Using the samples from Task 1 and Task 2, we estimate the value of the integral

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

This integral is just the formula for the expected value of the random variable X under the assummption that it is distributed with density function $f(x)$:

$$\begin{aligned} E[X] &= \int_{-\infty}^{\infty} x f(x) \; dx \\ &= \int_{-\infty}^{0} x f(x) \; dx + \int_0^\infty x f(x) \; dx \\ &= 0 + \int_0^\infty x f(x) \; dx \end{aligned}$$

So we can approximate this integral by computing the mean value of the sample data:

```
# mean value for first sample
integral_1 <- mean(data_log_norm$sample)

# mean value for second sample
integral_2 <- mean(data_chi$sample)
```

```
## [1] "Approximation of integral using first sample: 5.97"
```

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

```
## [1] "Approximation of integral using second sample: 6.01"
```

Task 5

Now we want to compute the real value if the above mentioned integral in order to compare it with the approximations. Since we know, that the generated distribution is actually a gamma distribution $\Gamma(\alpha, \beta)$ with $\alpha = 6$ and $\beta = 1$, we can simply compute it in the following way:

$$\begin{aligned} E[X \sim \Gamma(\alpha, \beta)] &= \alpha \cdot \beta \\ &= 6 \cdot 1 \\ &= 6 \end{aligned}$$

As we can see, the real value of the integral is very close to the two approximations, which again verifies that the sampling process using Metropolis-Hastings method was successfull.