Computational Statistics: Lab 2

Tim Yuki Washio & Nicolas Taba

17 novembre, 2020

Question 1: Optimizing parameters

1

We are asked to create a function that uses **optim()** to approximate a function using a piecewise quadratic function.

```
interp_fn <- function(boundaries, par){
  result <- c()
  for (i in 1:length(boundaries)){
    result[i] = c(par[1] + par[2] * boundaries[i] + (par[3] * boundaries[i]^2))
  }
  return(result)
}
min_se <- function(boundaries, par, target_fn){
  t_f = target_fn(boundaries)
  i_f = interp_fn(boundaries, par)
  sse = sum((t_f - i_f)^2)
  return(sse)
}
approximate <- function(boundaries, target_fn){
  optimized <- optim(par=c(0,0,0), fn=min_se, target_fn=target_fn, boundaries=boundaries)
  return(optimized$par)
}
```

2

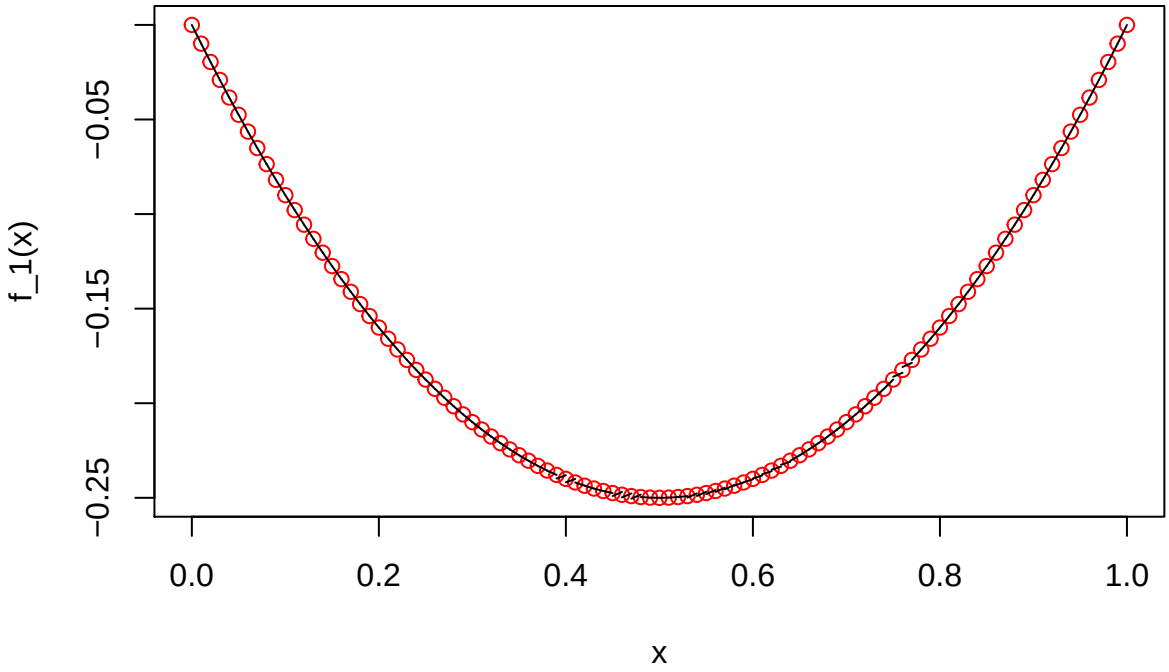
Now we build a function such that it can make this approximation on an interval [0,1] by discretizing it into n intervals.

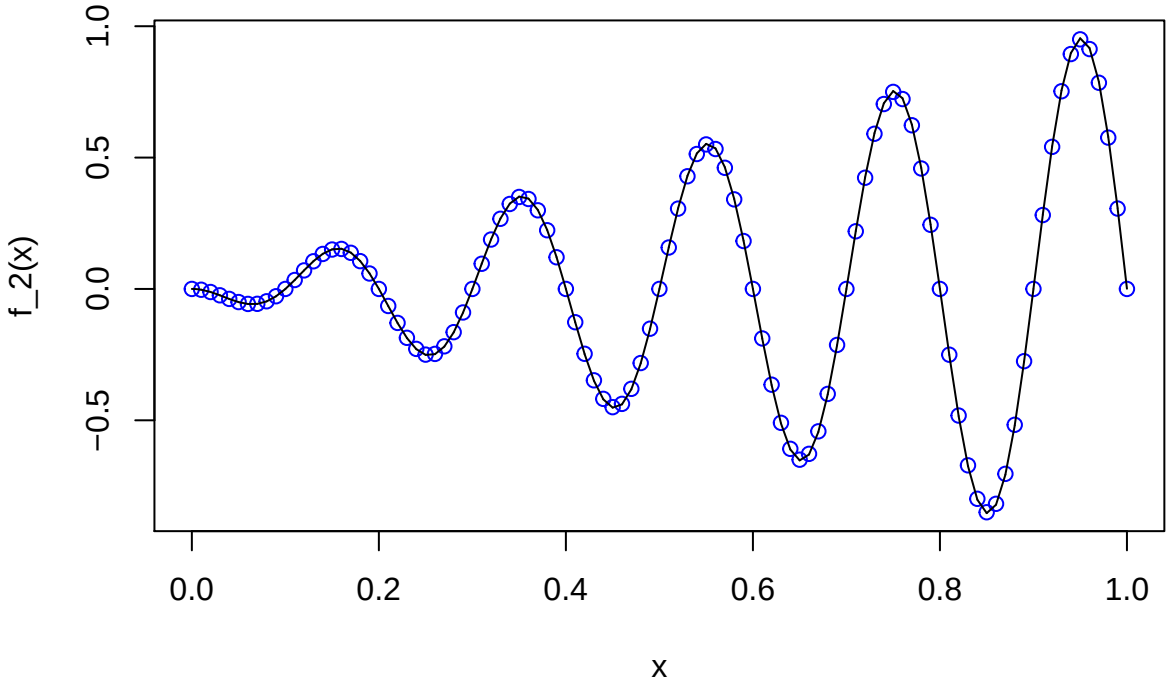
```
approximate_cut <- function(n, fn){
  interval <- seq(0, 1, length.out = n+1)
  interval_targets = list()
  for (i in 1:length(interval)) {
    first_val = interval[i]
    if (first_val == 1){stop}
    else{
      second_val = interval[i+1]
      mid_val = interval[i] + ((second_val - first_val) / 2)
      boundaries = c(first_val, mid_val, second_val)
      interval_targets = append(interval_targets, list(approximate(boundaries, fn)))
    }
  }
  return(interval_targets)
```

```
}
```

3

Finally, we assess our function on $f_1 = -x(1-x)$ and $f_2 = -xsin(10\pi x)$





The quadratic interpolation is good enough when we have intervals that are small enough for the optim function to find suitable parameters. It is difficult to notice a difference between the target function and our interpolated approximation.

Question 2: Maximizing likelihood

1.

Load data into global environment.

```
load("data.RData")
```

2.

We define the log-likelihood function as the logarithm of the likelihood function:

$$\mathcal{L}_x(\vartheta) = ln(L_x(\vartheta))$$

For normal distributions we define:

$$\mathcal{L}_x(\mu, \sigma^2) = -\frac{n}{2} ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2$$

For a given $n = 100$ we get:

$$\mathcal{L}_x(\mu, \sigma^2) = -\frac{100}{2} ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{100} (x_i - \mu)^2$$

We calculate the partial derivative of log-likelihood with respect to μ and set it equal to 0:

$$\frac{\partial \mathcal{L}}{\partial \mu} = \frac{1}{\sigma^2} \sum_{i=1}^{100} (x_i - \mu) = 0$$

We transform our equation and get:

$$\hat{\mu} = \frac{1}{100} \sum_{i=1}^{100} x_i$$

Now we calculate the partial derivative of log-likelihood with respect to σ and set it equal to 0:

$$\frac{\partial \mathcal{L}}{\partial \sigma} = -\frac{100}{2\sigma^2} - \left[\frac{1}{2}\sum_{i=1}^{100}(x_i - \mu)^2\right]\left(-\frac{1}{(\sigma^2)^2}\right) = -\frac{100}{2\sigma^2} + \left[\frac{1}{2}\sum_{i=1}^{100}(x_i - \mu)^2\right]\frac{1}{(\sigma^2)^2} = \frac{1}{2\sigma^2}\left[\frac{1}{\sigma^2}\sum_{i=1}^{100}(x_i - \mu)^2 - 100\right] = 0$$

We transform our equation and get:

$$\hat{\sigma}^2 = \frac{1}{100} \sum_{i=1}^{100} (x_i - \mu)^2$$

```
mu_hat = sum(data)/length(data)
sigma_hat = sqrt(sum((data-mu_hat)^2)/length(data))

cat("mu_hat: ", mu_hat, "\nsigma_hat: ", sigma_hat)
```

```
## mu_hat:  1.275528
## sigma_hat:  2.005976
```

3.

```
negative_log_likelihood <- function(param) {
  mu = param[1]
  sigma = param[2]
  n = length(data)
  log_likelihood = -n*0.5*log(2*pi*sigma^2)-(0.5/sigma^2)*sum((data-mu)^2)
  negative_log_likelihood = -(log_likelihood)
  return (negative_log_likelihood)
}

gradient <- function(params) {
  mu = params[1]
  sigma = params[2]
  n = length(data)
  result = c(-(sum(data)-n*mu)/sigma^2, (n/sigma)-(1/(sigma^3) * sum((data-mu)^2)))
  return(result)
}

cg_1 <- optim(par=c(0, 1), negative_log_likelihood, method="CG")
bfgs_1 <- optim(par=c(0, 1), fn=negative_log_likelihood, method="BFGS")
cg_2 <- optim(par=c(0, 1), fn=negative_log_likelihood, gr=gradient, method="CG")
bfgs_2 <- optim(par=c(0, 1), fn=negative_log_likelihood, gr=gradient, method="BFGS")
```

There could be issues of underflow if the difference in order of magnitude between the elements that are summed is too big. Maximizing the log-likelihood allows to prevent this issue, because very small and very large values can have comparable contributions to the sum.

4.

Algorithm	Mean	Var	Function Counts	Gradient Counts	Gradient Specified	Value
CG	1.27553	2.00598	297.00000	45.00000	NO	211.50695
CG	1.27553	2.00598	53.00000	17.00000	YES	211.50695
BFGS	1.27553	2.00598	37.00000	15.00000	NO	211.50695
BFGS	1.27553	2.00598	39.00000	15.00000	YES	211.50695

All algorithms converged. The optimal values of parameters are pretty similar to the ones we calculated earlier. We would recommend to use the BFGS algorithm since we observed less computations while getting the same result.

Appendix: All code for this report

```
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(huxtable)
library(magrittr)
library(ggplot2)
interp_fn <- function(boundaries, par){
  result <- c()
  for (i in 1:length(boundaries)){
    result[i] = c(par[1] + par[2] * boundaries[i] + (par[3] * boundaries[i]^2))
  }
  return(result)
}
min_se <- function(boundaries, par, target_fn){
  t_f = target_fn(boundaries)
  i_f = interp_fn(boundaries, par)
  sse = sum((t_f - i_f)^2)
  return(sse)
}
approximate <- function(boundaries, target_fn){
  optimized <- optim(par=c(0,0,0), fn=min_se, target_fn=target_fn, boundaries=boundaries)
  return(optimized$par)
}

approximate_cut <- function(n, fn){
  interval <- seq(0, 1, length.out = n+1)
  interval_targets = list()
  for (i in 1:length(interval)) {
    first_val = interval[i]
    if (first_val == 1){stop}
    else{
      second_val = interval[i+1]
      mid_val = interval[i] + ((second_val - first_val) / 2)
      boundaries = c(first_val, mid_val, second_val)
      interval_targets = append(interval_targets, list(approximate(boundaries, fn)))
    }
  }
  return(interval_targets)
}
f_1 <- function(x) {
  y = -x * (1-x)
  return(y)
}
f_2 <- function(x) {
  y = -x * sin(10 * pi * x)
  return(y)
}
approximation1 <- approximate_cut(100, f_1)
x <- seq(0, 1, length.out = 101)

plot1 <- plot(x, f_1(x), col = "red")
for (i in 1: length(approximation1)){
  x_1 <- seq((i*1/100 - 1/100), (1/100*i), by = 0.01/length(approximation1))
```

```
  lines(x_1, sapply(x_1, interp_fn, par = approximation1[[i]]))
}
approximation2 <- approximate_cut(100, f_2)
x <- seq(0, 1, length.out = 101)
plot2 <- plot(x, f_2(x), col = "blue")
for (i in 1: length(approximation1)){
  x_2 <- seq((i*1/100 - 1/100), (1/100*i), by = 0.01/length(approximation1))
  lines(x_2, sapply(x_2, interp_fn, par = approximation2[[i]]))
}
load("data.RData")
mu_hat = sum(data)/length(data)
sigma_hat = sqrt(sum((data-mu_hat)^2)/length(data))

cat("mu_hat: ", mu_hat, "\nsigma_hat: ", sigma_hat)
negative_log_likelihood <- function(param) {
  mu = param[1]
  sigma = param[2]
  n = length(data)
  log_likelihood = -n*0.5*log(2*pi*sigma^2)-(0.5/sigma^2)*sum((data-mu)^2)
  negative_log_likelihood = -(log_likelihood)
  return (negative_log_likelihood)
}

gradient <- function(params) {
  mu = params[1]
  sigma = params[2]
  n = length(data)
  result = c(-(sum(data)-n*mu)/sigma^2, (n/sigma)-(1/(sigma^3) * sum((data-mu)^2)))
  return(result)
}

cg_1 <- optim(par=c(0, 1), negative_log_likelihood, method="CG")
bfgs_1 <- optim(par=c(0, 1), fn=negative_log_likelihood, method="BFGS")
cg_2 <- optim(par=c(0, 1), fn=negative_log_likelihood, gr=gradient, method="CG")
bfgs_2 <- optim(par=c(0, 1), fn=negative_log_likelihood, gr=gradient, method="BFGS")
algorithms = rbind("CG", "CG", "BFGS", "BFGS")
params = rbind(cg_1$par, cg_2$par, bfgs_1$par, bfgs_2$par)
counts = rbind(cg_1$counts, cg_2$counts, bfgs_1$counts, bfgs_2$counts)
gradient = c("NO", "YES", "NO", "YES")
values = rbind(cg_1$value, cg_2$value, bfgs_1$value, bfgs_2$value)

df = data.frame(algorithms, params, counts, gradient, values)
colnames(df) = c("Algorithm", "Mean", "Var", "Function Counts", "Gradient Counts", "Gradient Specified"


pretty_df <-
  hux(df) %>%
  set_bold(row = 1, col = everywhere, value = TRUE) %>%
  set_all_borders(TRUE)

set_number_format(pretty_df, 5)
```