Group 12 Lab Report - Lab 2

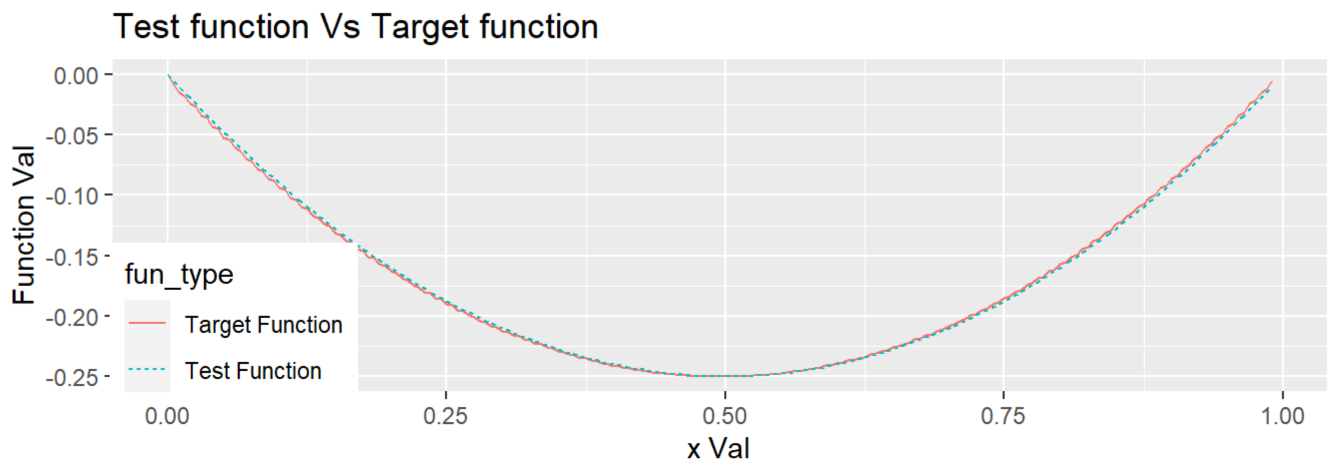
Aswath Mandakath Gopinath, Varun Singapura Ravi

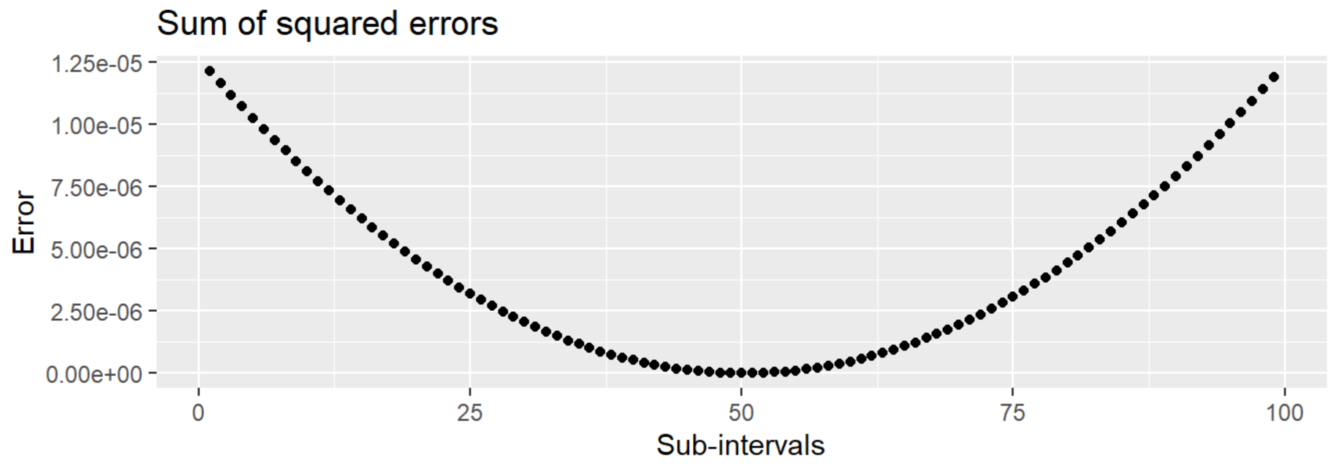
2023-01-03

Question 1

Plot Test vs Target function

Case 1: $f_1(x) = -x(1-x)$

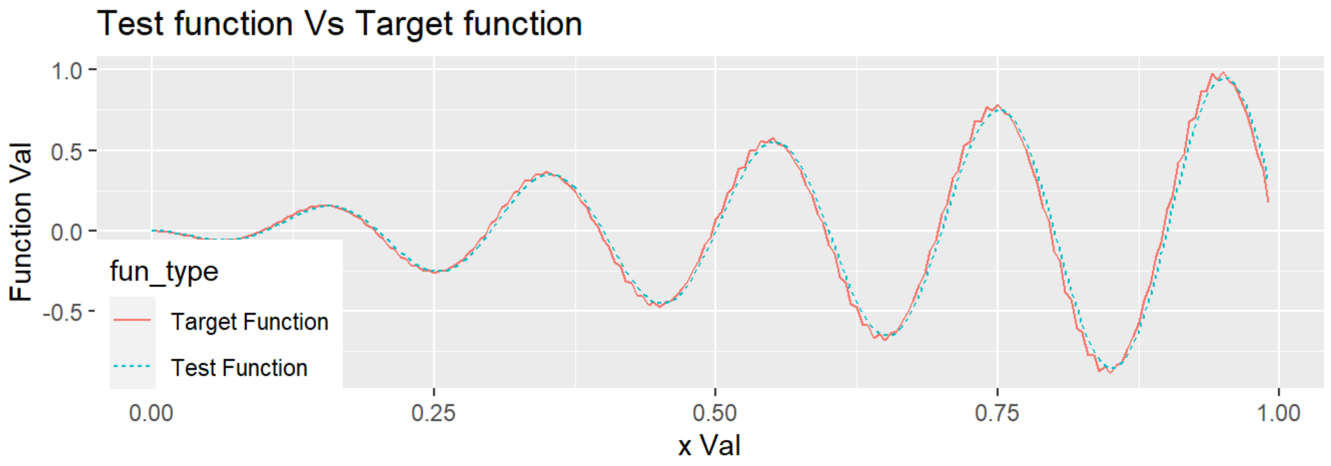


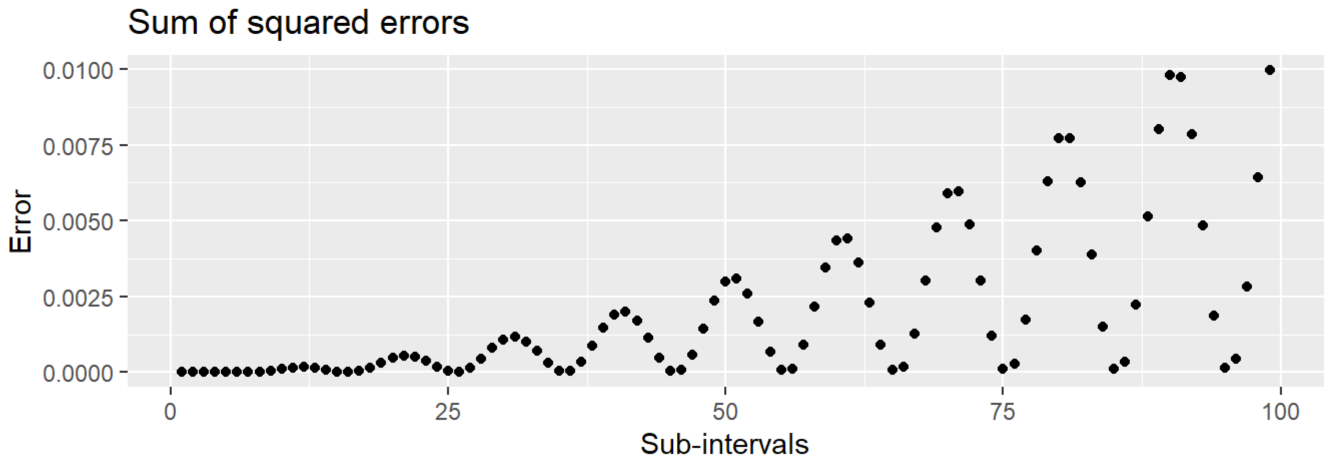


In this graph, the piecewise-parabolic interpolarter is able to follow the same pattern as the test function. At the midpoint, at x = 0.50, there is very little(almost 0) difference between the interpolarter and the test function. This is because the optimization results in very less squared error at mid-point,as shown in the error graph. This is because at the mid-point is where the curve very accurately resembles the approximator.

From the mid-point, as we move towards the ends, either 0 or 1, the piecewise-interpolater doesn't fit the test function that well. This is because, the test function acquires a shape different from the approximator and this keeps increasing as we go from the center and the error graph supports this behaviour.

Case 2: $f_1(x) = -xsin(10\pi x)$





In this case, at each peak and trough the target function closely follows the test function. This is because at these points the squared error is the minimum(as seen in the error graph) because of its close resemblance to the approximator. But as we move away from these points, the error increases and collapses at the peak/trough.

Question 2

Sub Question 1.

See appendix for code.

Sub Question 2

The joint density of the given sample is written as:

$$L(\mu,\sigma) = f(y_1, y_2, \dots, y_n|\mu,\sigma) = f(y_1|\mu,\sigma) \times f(y_2|\mu,\sigma) \times \dots \cdot \times f(y_n|\mu,\sigma)$$

The above equation can be written as:

$$L(\mu,\sigma) = \prod_{i=1}^{n} f(y_i|\mu,\sigma) \tag{}$$

Therefore likelihood function for 100 observations is as follows:

$$L(\mu,\sigma) = \prod_{i=1}^{n} f(y_i|\mu,\sigma) = \frac{e^{-\frac{\sum_{i=1}^{n}(y_i-\mu)^2}{2\sigma^2}}}{(\sigma\sqrt{2\pi})^n}$$

The log-likelihood function for 100 observations is as follows:

$$

$$ln[L(\mu,\sigma)] = -\frac{\sum_{i=1}^{n}(y_i-\mu)^2}{2\sigma^2} - nln\sigma - nln\sqrt{2\pi} \tag{1}$$

$$

$$where, n = 100$$

The maximum likelihood estimators are obtained by partially differentiating with respect to μ and σ, they are as follows:

Partial Derivative of μ: $$

$$\frac{\partial ln[L(\mu,\sigma)]}{\partial \mu} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(y_i-\mu) \tag{2}$$

$$

$$where, n = 100$$

Partial Derivative of σ:

\$\$

$$\frac{\partial ln[L(\mu,\sigma)]}{\partial\sigma} = \frac{\sum_{i=1}^{n}(y_i-\mu)^2}{\sigma^3} - \frac{n}{\sigma} \tag{3}$$

\$\$

$$where, n = 100$$

By setting equation 2 and 3 equal to zero, we obtain the Maximum Likelihood estimators for μ and σ as the following:

For μ, the MLE is:

\$\$

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n}(y_i) = \bar{y} \tag{4}$$

\$\$

$$where, n = 100$$

Obtaining parameter estimate, μ, for loaded data, from above derived formula:

```
mu_mle <- sum(y)/length(y)
mu_mle
```

```
## [1] 1.275528
```

For σ, the MLE is:

\$\$

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(y_i-\bar{y})^2}{n}} \tag{5}$$

\$\$

$$where, n = 100$$

Obtaining parameter estimate, σ, for loaded data, from above derived formula and implementing a square root to obtain standard deviation:

```
std_dev_mle <- sqrt(sum((y - mean(y))^2)/(length(y)))
std_dev_mle
```

```
## [1] 2.005976
```

Sub Question 3:

The sub question 4 shows the results of optimizing minus log-likelihood function with initial parameters $\mu = 0, \sigma = 1$. It also shows the methods BFGS and Conjugate Gradient with and without gradient specified.

It is a bad idea to maximize likelihood function rather than the maximizing log-likelihood because, the likelihood function contains the product of the likelihoods for each point in a sample and computing this becomes computationally expensive. On the other hand, when we maximize the log-likelihood function, we end up with the sum of likelihoods for each point which is much easier to compute and less expensive. Since, $log(L(\mu, \sigma^2))$ i.e, log-likelihood is a monotonically increasing function and hence has the same objective as maximizing the likelihood function $L(\mu, \sigma^2)$.

Sub Question 4:

As it is visible in the data frame from the previous sub question, the algorithms **do converge** in all cases as shown in the data frame below.

```
##   Method Gradient Convergence     Mean  Std_Dev Function.Call.Count
## 1   BFGS       NO         Yes 1.275528 2.005977                  37
## 2     CG       NO         Yes 1.275528 2.005976                 199
## 3   BFGS      YES         Yes 1.275528 2.005977                  40
## 4     CG      YES         Yes 1.275528 2.005976                  53
##   Gradient.Call.Count
## 1                  15
## 2                  35
## 3                  15
## 4                  17
```

The recommended settings would be to use the **BFGS method without gradient specified** as it leads to convergence in least number of iterations as shown in the row below and the optimal values of the Mean(μ) and Standard Deviation(σ):

```
##   Method Gradient Convergence     Mean  Std_Dev Function.Call.Count
## 1   BFGS       NO         Yes 1.275528 2.005977                  37
##   Gradient.Call.Count
## 1                  15
```

Appendix : All code for this report.

```r
library(ggplot2)
library(dplyr)
library(gridExtra)
knitr::opts_chunk$set(echo = TRUE, warning = FALSE)

#Function which is fed to the optim function to get the optimal a0, a1, a2
optimize_fn <- function(a, x, test_fn_val, i){
  #Get the a0, a1, a2 from a. These value are minimized
  a0 <- a[1]
  a1 <- a[2]
  a2 <- a[3]

  x0 <- x[1]
  x1 <- x[2]
  x2 <- x[3]
  fx0 <- test_fn_val[1]
  fx1 <- test_fn_val[2]
  fx2 <- test_fn_val[3]

  error <- (fx0-(a0+a1*x0+a2*x0^2))^2 +
    (fx1-(a0+a1*x1+a2*x1^2))^2 +
    (fx2-(a0+a1*x1+a2*x1^2))^2
  .GlobalEnv$k = .GlobalEnv$k + 1
  .GlobalEnv$error_df = rbind(.GlobalEnv$error_df,
                              data.frame(i_val = i, k = .GlobalEnv$k, x0 = x0, error = error))

  return(error)
}
#Function to get the values of the test function given x
test_fun <- function(type, x){
  if(type == 1){
    return(-x + x^2)
  }else if(type == 2){
    return(-x * sin(10 * pi * x))
  }else if(type == 3){
    return(x*(x+1))
  }
}
#Function to evaluate the parabolic estimator using the optimal a0, a1 & a2 at x
```

```r
approx_fun <- function(a0, a1, a2, x){
  return(a0 + a1*x + a2*x^2)
}
#Function which will return the plot of the test vs target function as well as the underlying data
appoximate_fn <- function(fn_type, sub_intervals){
  x0 <- 0.0#Starting point
  final_df <- as.data.frame(matrix(nrow = 0, ncol = 5))
  colnames(final_df) <- c('i_val', 'x', 'fun_type', 'fun_val', 'a0')
  for (i in 1:sub_intervals) {
    x0 <- x0
    x1 <- x0 + 0.005
    x2 <- x1 + 0.005
    if(x0 > 1 || x1 > 1 || x2 > 1) break
    res <- optim(par = c(0,0,0),#Starting point
                 fn = optimize_fn,
                 method = ("BFGS"),
                 x = c(x0, x1, x2), #x0, x1 & x2 values for approximator function
                 test_fn_val = c( #test function values for each x0, x1, x2
                   test_fun(fn_type, x0),
                   test_fun(fn_type, x1),
                   test_fun(fn_type, x2)
                 ), i = i
    )
    if(i == 1){
      final_df <- rbind(final_df,
                        data.frame(i_val = i, x = x0, fun_type = 'Test Function', #Plot for x0 for Test function
                                   fun_val = test_fun(fn_type, x0), a0 = res$par[1]),
                        data.frame(i_val = i, x = x1, fun_type = 'Test Function', #Plot for x1 for Test function
                                   fun_val = test_fun(fn_type, x1), a0 = res$par[1]),
                        data.frame(i_val = i, x = x2, fun_type = 'Test Function', #Plot for x2 for Test function
                                   fun_val = test_fun(fn_type, x2), a0 = res$par[1])
      )
      final_df <- rbind(final_df,
                        data.frame(i_val = i, x = x0, fun_type = 'Target Function', #Plot for x0 for Target funct
ion
                                   fun_val = approx_fun(a0 = res$par[1], a1 = res$par[2], a2 = res$par[3], x = x
0), a0 = res$par[1]),
                        data.frame(i_val = i, x = x1, fun_type = 'Target Function', #Plot for x1 for Target funct
ion
                                   fun_val = approx_fun(a0 = res$par[1], a1 = res$par[2], a2 = res$par[3], x = x
1), a0 = res$par[1]),
                        data.frame(i_val = i, x = x2, fun_type = 'Target Function', #Plot for x2 for Target funct
ion
                                   fun_val = approx_fun(a0 = res$par[1], a1 = res$par[2], a2 = res$par[3], x = x
2), a0 = res$par[1])
      )
    }else{
      final_df <- rbind(final_df,
                        data.frame(i_val = i, x = x1, fun_type = 'Test Function', #Plot for x1 for Test function
                                   fun_val = test_fun(fn_type, x1), a0 = res$par[1]),
                        data.frame(i_val = i, x = x2, fun_type = 'Test Function', #Plot for x2 for Test function
                                   fun_val = test_fun(fn_type, x2), a0 = res$par[1])
      )
      final_df <- rbind(final_df,
                        data.frame(i_val = i, x = x1, fun_type = 'Target Function', ##Plot for x1 for Target func
tion
                                   fun_val = approx_fun(a0 = res$par[1], a1 = res$par[2], a2 = res$par[3], x = x
1), a0 = res$par[1]),
                        data.frame(i_val = i, x = x2, fun_type = 'Target Function', #Plot for x2 for Target funct
ion
                                   fun_val = approx_fun(a0 = res$par[1], a1 = res$par[2], a2 = res$par[3], x = x
2), a0 = res$par[1])
      )
    }
    x0 <- x2
  }
  final_df <- final_df %>% arrange(fun_type, i_val)

  plot <- ggplot(final_df, aes(x = x, y = fun_val)) +
    geom_line(aes(color = fun_type, linetype = fun_type)) +
    xlab('x Val') + ylab('Function Val') +
    theme(legend.position = c(0.1,0.20)) +
    ggtitle('Test function Vs Target function')
  return(list(plot = plot, data = final_df))
}
plot_error <- function(){
  error_final <- data.frame()
  for (i in 1:100) {
    error_temp <- .GlobalEnv$error_df %>%
      filter(i_val == i)
    error_temp <- error_temp %>%
      filter(k == max(error_temp$k))
    error_final <- rbind(error_final, error_temp)
  }
```

```r
  return(ggplot(error_final, aes(x = i_val, y = error)) +
    geom_point() +
    xlab('Sub-intervals') + ylab('Error') +
    ggtitle('Sum of squared errors')
  )
}
#f1(x) = -x(1-x)
error_df <- as.data.frame(matrix(nrow=0, ncol=4))
colnames(error_df) <- c('i_val', 'k', 'x0', 'error')
k=0

ret_obj <- appoximate_fn(1, 100)
error_plot <- plot_error()
grid.arrange(ret_obj$plot, error_plot, nrow=2)
error_df <- as.data.frame(matrix(nrow=0, ncol=4))
colnames(error_df) <- c('i_val', 'k', 'x0', 'error')
k=0

ret_obj <- appoximate_fn(2, 100)
error_plot <- plot_error()
grid.arrange(ret_obj$plot, error_plot, nrow=2)
load('C:/Users/Varun/Desktop/LiU - STIMA/Computational Statistics/Lab2/data.RData')
y <- data
mu_mle <- sum(y)/length(y)
mu_mle
std_dev_mle <- sqrt(sum((y - mean(y))^2)/(length(y)))
std_dev_mle

neg_likelihood <- function(x){
  mu <- x[1]
  std_dev <- x[2]
  n <- length(y)
  # Change made below
  return((sum((y - mu) ^ 2)/(2 * std_dev^2)) + (n * log(std_dev)) + (n * log(sqrt(2 * pi))))
}


grad_fn <- function(x){
  mu <- x[1]
  std_dev <- x[2]
  n <- length(y)
  mu_grad <- sum(y - mu)/(std_dev^2)
  std_dv_grad <- (((sum((y - mu)^2))/(std_dev^3)) - (n/std_dev))
  return(c(-mu_grad, -std_dv_grad))
}
optimize_df <- data.frame(matrix(ncol = 7, nrow = 0))
colnames(optimize_df) <- c('Method',
                           'Gradient',
                           'Convergence',
                           'Mean',
                           'Std_Dev',
                           'Function Call Count',
                           'Gradient Call Count')
optimizer <- function(t_method, fun_opt, gr_fn = NULL){

  optim_vals <- optim(par = c(0,1), fn = fun_opt, gr = gr_fn, method = t_method)
  optimize_df <<- rbind(optimize_df, data.frame('Method' = t_method,
                                                'Gradient' = ifelse(is.null(gr_fn), 'NO', 'YES'),
                                                'Convergence' = ifelse(optim_vals$convergence == 0, 'Yes', 'No'),
                                                'Mean' = optim_vals$par[1],
                                                'Std_Dev' = optim_vals$par[2],
                                                'Function Call Count' = optim_vals$counts[['function']],
                                                'Gradient Call Count' = optim_vals$counts[['gradient']]))
  return(optimize_df)
}
optim(par = c(0,1), fn = neg_likelihood, method = 'BFGS')
optimizer('BFGS', fun_opt = neg_likelihood)
optimizer('CG', fun_opt = neg_likelihood)
optimizer('BFGS', fun_opt = neg_likelihood, gr_fn = grad_fn)
optimizer('CG', fun_opt = neg_likelihood, gr_fn = grad_fn)

optimize_df
optimize_df[1,]
```