Computer Lab 2

Computational Statistics

Group 4

Question 1

1.1

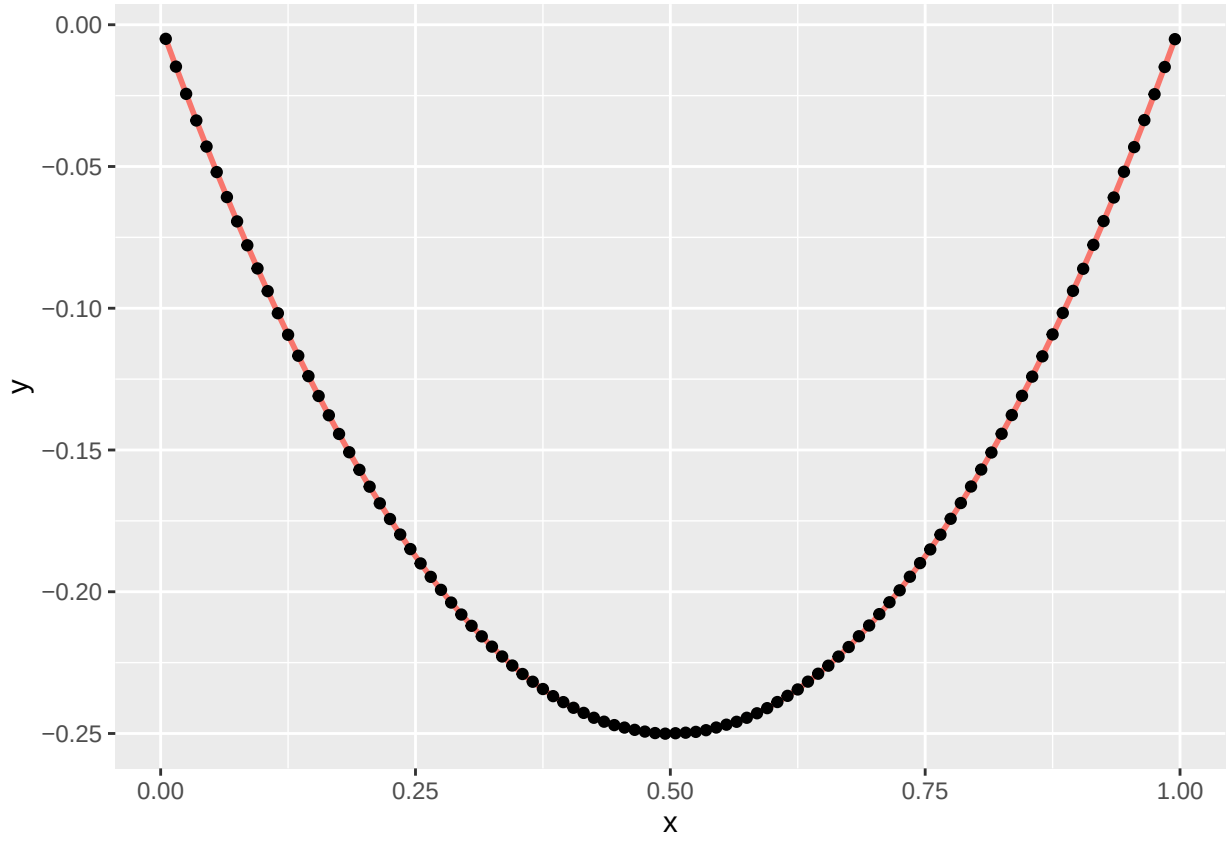
```
squared_error <- function(par, x, y) {
  a0 <- par[1]
  a1 <- par[2]
  a2 <- par[3]

  piecewise_quadratic <- a0 + a1*x + a2*x*x

  return(sum(y - piecewise_quadratic)^2)
}
```

1.2

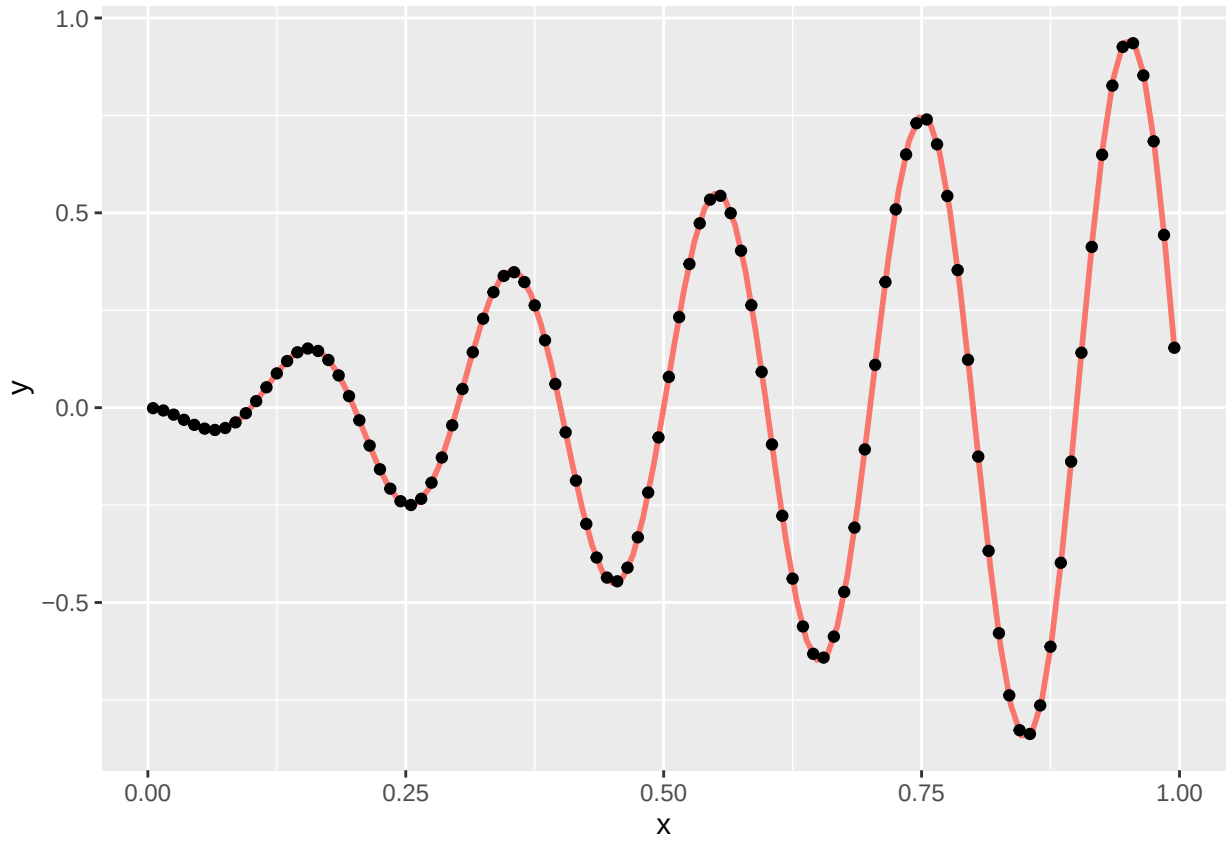
```
func_approx <- function(n_int, func) {
  intervals <- seq(0, 1, length.out = n_int+1)
  res <- as.data.frame(matrix(NA, nrow = n_int, ncol = 6))
  colnames(res) <- c("x0", "x1", "x2", "a0", "a1", "a2")
  for(i in 1:n_int) {
    f_x0 <- func(intervals[i])
    f_x1 <- func(mean(c(intervals[i], intervals[i+1])))
    f_x2 <- func(intervals[i+1])

    opt_par <- optim(par = c(0,1,1),
                     fn = squared_error,
                     x = c(intervals[i],
                           mean(c(intervals[i], intervals[i+1])),
                           intervals[i+1]),
                     y = c(f_x0, f_x1, f_x2))$par

    res[i,1:3] <- c(intervals[i], mean(c(intervals[i], intervals[i+1])), intervals[i+1])
    res[i,4:6] <- opt_par
  }
  return(res)
}
```

1.3

```
func_1 <- function(x) {
  return(-x *(1-x))
}

func_2 <- function(x) {
  return(-x*sin(10*pi*x))
}

res_1 <- func_approx(100, func_1)
res_1$`g(x)` <- res_1[,"a0"] + res_1[,"a1"]*res_1[,"x1"] + res_1[,"a2"]*res_1[,"x1"]^2

res_2 <- func_approx(100, func_2)
res_2$`g(x)` <- res_2[,"a0"] + res_2[,"a1"]*res_2[,"x1"] + res_2[,"a2"]*res_2[,"x1"]^2

library(ggplot2)

ggplot(res_1, aes(x = x1, y = `g(x)`)) +
  geom_function(fun = func_1, aes(color = "red"), size = 1) +
  geom_point() +
  xlab("x") +
  ylab("y") +
  theme(legend.position = "none")
```



```
ggplot(res_2, aes(x = x1, y = `g(x)`)) +
  geom_function(fun = func_2, aes(color = "red"), size = 1) +
  geom_point() +
  xlab("x") +
  ylab("y")+
  theme(legend.position = "none")
```



The piecewise-parabolic interpolate appear to do very well in approximating both $f_1(x) = -x(1-x)$ and $f_2(x) = -x\sin(10\pi x)$. There seem to be very few deviations from the functions.

Question 2

2.1

```
load("C:/Users/Simon/Desktop/Computational Statistics/lab2/data.RData")
```

2.2

The probability density function of the normal distribution is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

The likelihood function is defined as:

$$L(\mu, \sigma|\mathbf{x}) = \prod_{i=1}^{100} f(x_i|\mu, \sigma)$$

$$= \prod_{i=1}^{100} (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

$$= (2\pi\sigma^2)^{-\frac{100}{2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2\right)$$

The log-likelihood function is obtained by taking the natural logarithm of the likelihood function:

$$l(\mu, \sigma^2|\mathbf{x}) = \ln(L(\mu, \sigma^2|\mathbf{x}))$$

$$= \ln\left((2\pi\sigma^2)^{-\frac{100}{2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2\right)\right)$$

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

The maximum likelihood estimator for μ is obtained by taking the partial derivative of $l(\mu, \sigma^2|\mathbf{x})$ with respect to μ, and then setting the obtained partial derivative to 0:

$$\frac{\partial}{\partial\mu} l(\mu, \sigma^2|\mathbf{x}) = \frac{\partial}{\partial\mu}\left(-\frac{100}{2}\ln(2\pi) - \frac{100}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2\right)$$

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

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

$$= \sum_{i=1}^{100} x_i - 100\mu = 0$$

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

The maximum likelihood estimator for σ is obtained by taking the partial derivative of $l(\mu, \sigma^2|\mathbf{x})$ with respect to σ, and then setting the obtained partial derivative to 0:

$$\frac{\partial}{\partial\sigma^2} l(\mu, \sigma^2|\mathbf{x}) = \frac{\partial}{\partial\sigma^2}\left(-\frac{100}{2}\ln(2\pi) - \frac{100}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2\right)$$

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

$$= \frac{1}{2\sigma^2}\left(\frac{1}{\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2 - 100\right) = 0$$

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

The estimated values of μ and σ are:

```
mu <- sum(data)/100
sigma <- sqrt((sum((data-mu)^2))/(100-1))

mu
```

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

```
sigma
```

```
## [1] 2.016082
```

2.3

```
func_neg_loglik <- function(par, x) {
  n <- length(x)
  mu <- par[1]
  sigma <- par[2]

  neg_loglik <- (n/2) * log(2*pi) + (n/2) * log(sigma^2) + (1/(2*sigma^2)) * sum((x-mu)^2)

  return(neg_loglik)
}
```

The gradient can be specified by taking the second derivative of the negative log-likelihood function with respect to μ and σ:

```
neg_loglik_grad <- function(par, x) {
  n <- length(x)
  mu <- par[1]
  sigma <- par[2]

  mu_grad <- -(1/sigma^2) * sum(x-mu)
  sigma_grad <- (n/sigma) - (1/sigma^3) * sum((x-mu)^2)

  return(c(mu_grad, sigma_grad))
}
```

With the *Conjugate Gradient method* without gradient specified:

```r
optim(par = c(0,1), fn = func_neg_loglik, method = "CG", x = data)
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      180       33
##
## $convergence
## [1] 0
##
## $message
## NULL
```

With the *Conjugate Gradient method* and gradient specified:

```r
optim(par = c(0,1), fn = func_neg_loglik, method = "CG", gr = neg_loglik_grad, x = data)
```

```
## $par
## [1] 1.275528 2.005976
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       53       17
##
## $convergence
## [1] 0
##
## $message
## NULL
```

With the *BFGS method* without gradient specified:

```r
optim(par = c(0,1), fn = func_neg_loglik, method = "BFGS", x = data)
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       37       15
```

```
##
## $convergence
## [1] 0
##
## $message
## NULL
```

With the *BFGS method* and gradient specified:

```
optim(par = c(0,1), fn = func_neg_loglik, method = "BFGS", gr = neg_loglik_grad, x = data)
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       38       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The reason why it can be a bad idea to maximize the likelihood rather than the log-likelihood is that parts of the expressions evaluated can become very computationally expensive and potentially cause over- or underflow. In this case $\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i-\mu)^2\right)$ can easily cause underflow which in turn might harm the numerical precision.

2.4

The algorithms did converge in all cases. The optimal value in all four cases were 211.5069.

For the *Conjugate Gradient method* the number of function and gradient evaluations required for the algorithm to converge were 180 and 33 wihtout gradient specified, and 53 and 17 with gradient specified.

For the *BFGS method* the number of function and gradient evaluations required for the algorithm to converge were 37 and 15 without gradient specified, and 38 and 15 with gradient specified.

The algorithms did converge in all four cases, but the *BGGS method* without gradient specified required the least function and gradient evaluations and is therefore the recommended settings in this particular case.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
squared_error <- function(par, x, y) {
  a0 <- par[1]
```

```
  a1 <- par[2]
  a2 <- par[3]

  piecewise_quadratic <- a0 + a1*x + a2*x*x

  return(sum(y - piecewise_quadratic)^2)
}
func_approx <- function(n_int, func) {
  intervals <- seq(0, 1, length.out = n_int+1)
  res <- as.data.frame(matrix(NA, nrow = n_int, ncol = 6))
  colnames(res) <- c("x0", "x1", "x2", "a0", "a1", "a2")
  for(i in 1:n_int) {
    f_x0 <- func(intervals[i])
    f_x1 <- func(mean(c(intervals[i], intervals[i+1])))
    f_x2 <- func(intervals[i+1])

    opt_par <- optim(par = c(0,1,1),
                     fn = squared_error,
                     x = c(intervals[i],
                           mean(c(intervals[i], intervals[i+1])),
                           intervals[i+1]),
                     y = c(f_x0, f_x1, f_x2))$par

    res[i,1:3] <- c(intervals[i], mean(c(intervals[i], intervals[i+1])), intervals[i+1])
    res[i,4:6] <- opt_par
  }
  return(res)
}
func_1 <- function(x) {
  return(-x *(1-x))
}

func_2 <- function(x) {
  return(-x*sin(10*pi*x))
}

res_1 <- func_approx(100, func_1)
res_1$`g(x)` <- res_1[,"a0"] + res_1[,"a1"]*res_1[,"x1"] + res_1[,"a2"]*res_1[,"x1"]^2

res_2 <- func_approx(100, func_2)
res_2$`g(x)` <- res_2[,"a0"] + res_2[,"a1"]*res_2[,"x1"] + res_2[,"a2"]*res_2[,"x1"]^2

library(ggplot2)

ggplot(res_1, aes(x = x1, y = `g(x)`)) +
  geom_function(fun = func_1, aes(color = "red"), size = 1) +
  geom_point() +
  xlab("x") +
  ylab("y") +
  theme(legend.position = "none")

ggplot(res_2, aes(x = x1, y = `g(x)`)) +
  geom_function(fun = func_2, aes(color = "red"), size = 1) +
```

```
  geom_point() +
  xlab("x") +
  ylab("y")+
  theme(legend.position = "none")
load("C:/Users/Simon/Desktop/Computational Statistics/lab2/data.RData")
mu <- sum(data)/100
sigma <- sqrt((sum((data-mu)^2))/(100-1))

mu
sigma
func_neg_loglik <- function(par, x) {
  n <- length(x)
  mu <- par[1]
  sigma <- par[2]

  neg_loglik <- (n/2) * log(2*pi) + (n/2) * log(sigma^2) + (1/(2*sigma^2)) * sum((x-mu)^2)

  return(neg_loglik)
}
neg_loglik_grad <- function(par, x) {
  n <- length(x)
  mu <- par[1]
  sigma <- par[2]

  mu_grad <- -(1/sigma^2) * sum(x-mu)
  sigma_grad <- (n/sigma) - (1/sigma^3) * sum((x-mu)^2)

  return(c(mu_grad, sigma_grad))
}
optim(par = c(0,1), fn = func_neg_loglik, method = "CG", x = data)
optim(par = c(0,1), fn = func_neg_loglik, method = "CG", gr = neg_loglik_grad, x = data)
optim(par = c(0,1), fn = func_neg_loglik, method = "BFGS", x = data)
optim(par = c(0,1), fn = func_neg_loglik, method = "BFGS", gr = neg_loglik_grad, x = data)
```