Computational Statistics (732A90) Lab02

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

2021-11-16

Question 1

Task 1

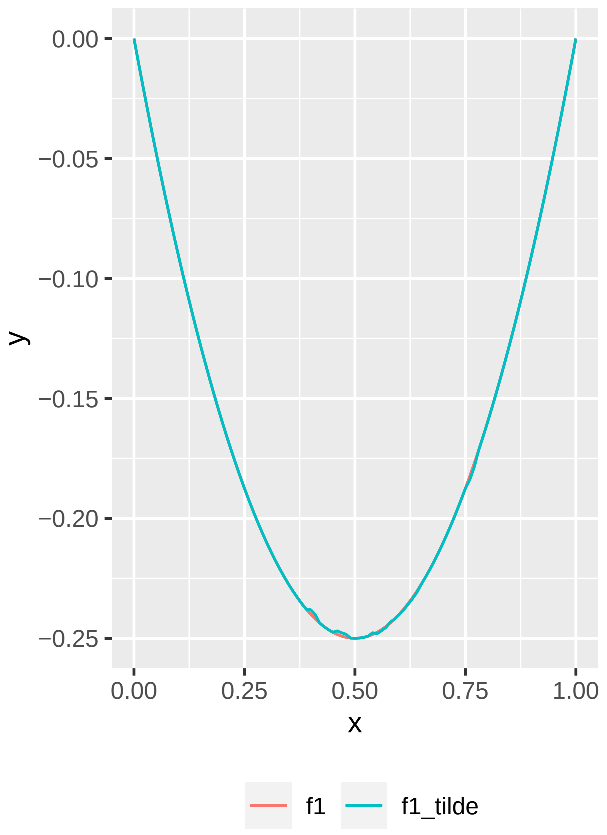
```r
find_a <- function(f, x0, x1, x2){
  stopifnot("x0, x1, x2 should be between [0, 1]" = any(c(x0,x1,x2) <= 1 | c(x0,x1,x2) >= 0),
            "f(x1) < f(x0), f(x2)" = any(c(f(x0),f(x2)) >= f(x0)))

  f_tilde <- function(x, a){
    a0 <- a[1]
    a1 <- a[2]
    a2 <- a[3]
    a0 + a1 * x + a2 * x^2
  }
  sq_error <- function(a){
    a0 <- a[1]
    a1 <- a[2]
    a2 <- a[3]
    return((f(x0) - f_tilde(x0, a))^2 + (f(x1) - f_tilde(x1, a))^2 + (f(x2) - f_tilde(x2, a))^2)
  }
  return(optim(par = c(0,0,0), fn = sq_error))
}
```

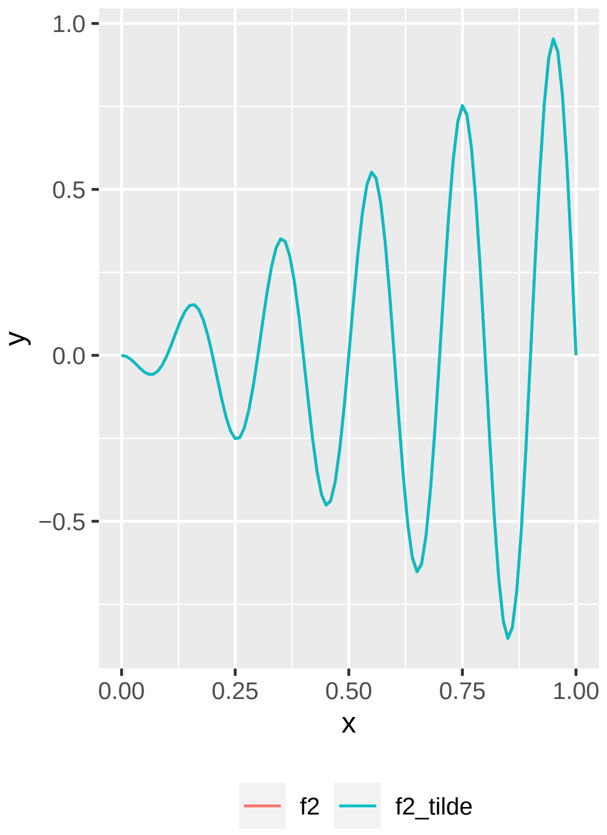
Task 2

```r
appr_fun <- function(f, n){
  interval_length <- 1/n
  a <- matrix(nrow=n, ncol=3)
  for (i in 1:n){
    a[i,] <- find_a(f,
                    (i-1) * interval_length,
                    (i-1) * interval_length + 1 / 2 * interval_length,
                    i* interval_length)$par
  }
  res_fun <- function(x){
    stopifnot(0 <= x & x <= 1)
    if (x == 0) {index <- 1} else {index <- ceiling(x * n)}
    return(a[index, 1] + a[index, 2] * x + a[index, 3] * x^2)
  }
  return(res_fun)
}
```

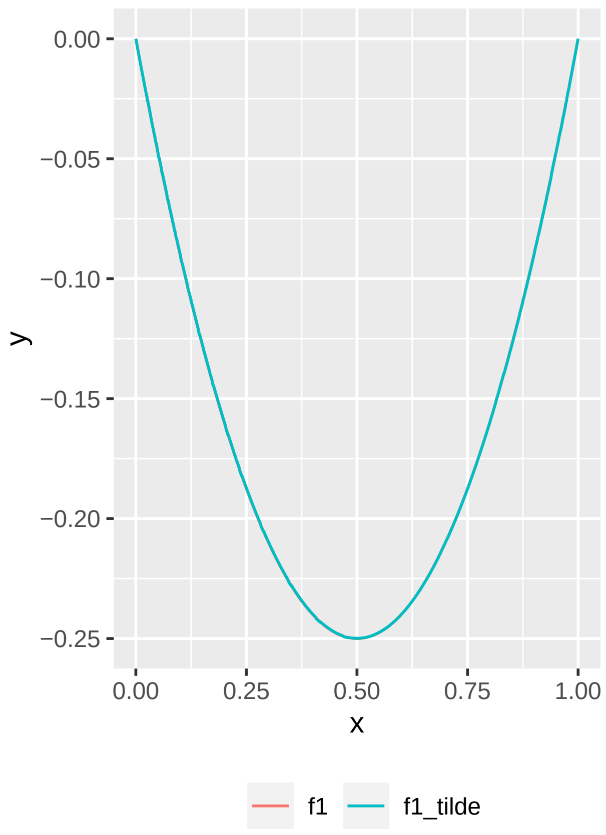
Task 3

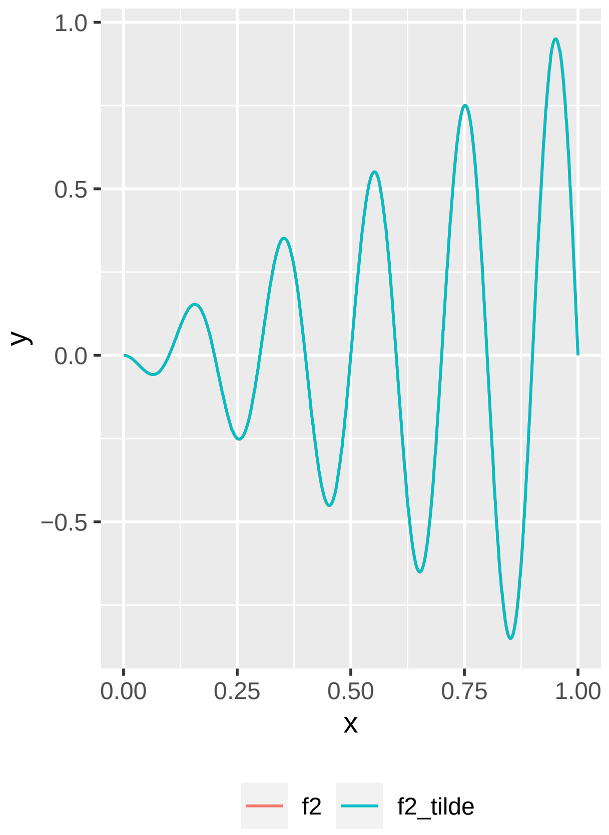
```
library(ggplot2)
library(gridExtra)

f1 <- function(x) {-x * (1 - x)}
f2 <- function(x) {-x * sin(10 * pi * x)}

f1_tilde <- appr_fun(f1, 100)
f2_tilde <- appr_fun(f2, 100)

x_values <- seq(0,1, by=0.01)
data1 <- data.frame(x = c(x_values, x_values),
                    y = c(f1(x_values), sapply(x_values, f1_tilde)),
                    group=c(rep("f1", length(x_values)), rep("f1_tilde", length(x_values))))
data2 <- data.frame(x = c(x_values, x_values),
                    y = c(f2(x_values), sapply(x_values, f2_tilde)),
                    group=c(rep("f2", length(x_values)), rep("f2_tilde", length(x_values))))

plot1 <- ggplot(data1, aes(x, y, color=group)) +
  geom_line() +
  theme(legend.position = 'bottom', legend.title=element_blank())
plot2 <- ggplot(data2, aes(x, y, color=group)) +
  geom_line() +
  theme(legend.position = 'bottom', legend.title=element_blank())
grid.arrange(plot1, plot2, ncol=2)
```



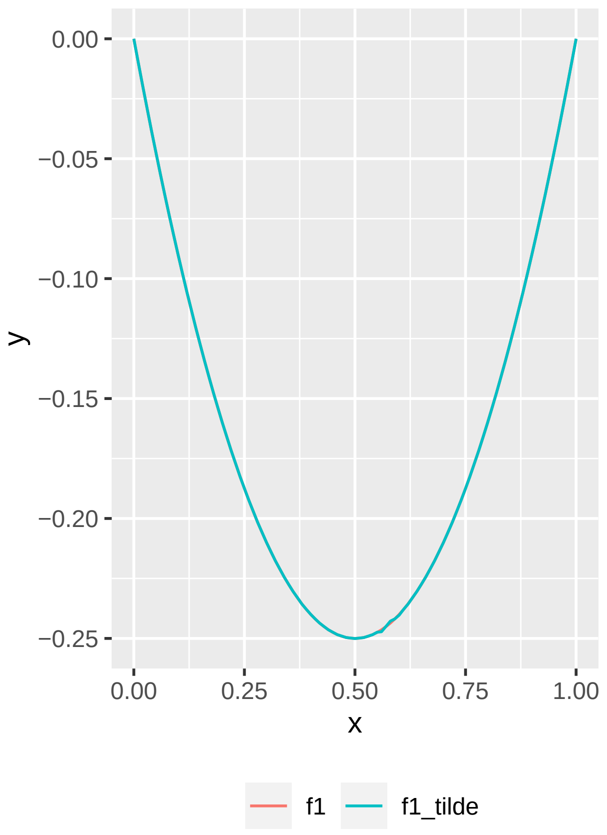


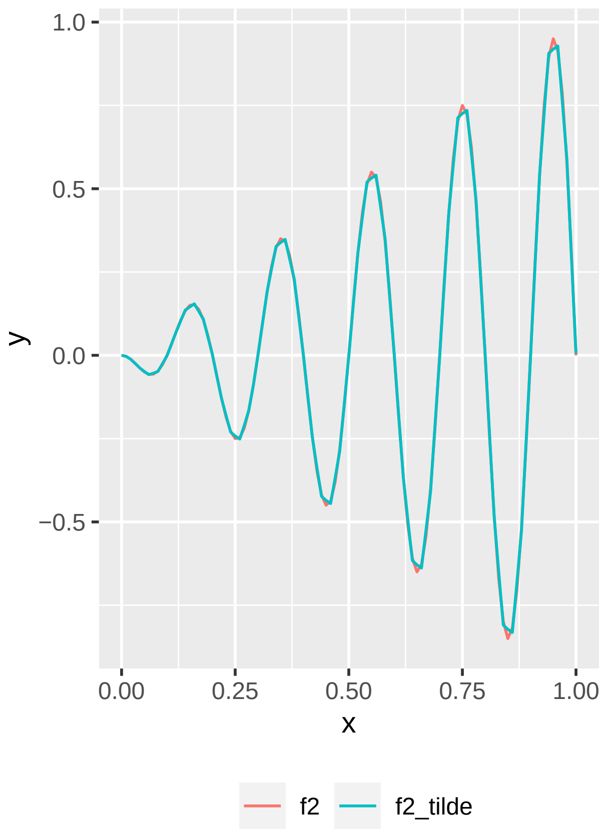
In the plots it can be seen that generally the approximation of the function is good. The approximation can be enhanced by choosing more and therefore smaller intervals.





Running the above code again with n = 1000 and 0.001 steps we can see that the f_tilde is “identical” to the f function.





By lowering the arguments to n = 50 and keeping steps at 0.01 we can see the decrease in the quality of the approximation.

Question 2

Task 1

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

Task 2

The log-likelihood is

$$P(X_i|\sigma,\mu) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x_i-\mu}{\sigma}\right)^2\right)$$

$$P(X_1,\ldots,X_{100}|\sigma,\mu) = \prod_{i=1}^{100} P(X_i|\sigma,\mu)$$

$$P(X_1,\ldots,X_{100}|\sigma,\mu) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^{100} \exp\left(-\frac{1}{2}\sum_{i=1}^{100}\left(\frac{x_i-\mu}{\sigma}\right)^2\right)$$

$$\log(P(X_1,\ldots,X_{100}|\sigma,\mu)) = -100\cdot\log(\sigma\sqrt{2\pi}) - \frac{1}{2}\sum_{i=1}^{100}\left(\frac{x_i-\mu}{\sigma}\right)^2$$

Partial derivative of μ

$$\frac{\partial}{\partial \mu} \log(P(X_1, \dots, X_{100}|\sigma, \mu)) \overset{!}{=} 0$$

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

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

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

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

Partial derivative of σ

$$\frac{\partial}{\partial \sigma} \log(P(X_1, \dots, X_{100}|\sigma, \mu)) \overset{!}{=} 0$$

$$0 = \frac{-100 \cdot \sqrt{2\pi}}{\hat{\sigma} \cdot \sqrt{2\pi}} + \frac{1}{\hat{\sigma}^3} \cdot \sum_{i=1}^{100} (x_i - \mu)^2$$

$$\frac{100}{\hat{\sigma}} = \frac{1}{\hat{\sigma}^3} \cdot \sum_{i=1}^{100} (x_i - \mu)^2$$

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

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

As the previous calculations show, the maximum likelihood estimators of μ and σ are represented by the sample mean $\bar{x}$ and the unadjusted sample variance S_n^2.

So we can calculate the first parameter $\hat{\mu}$ to

```
mean(data)
```

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

and $\hat{\sigma}$ to

```
sqrt(1/(100) * sum((data - mean(data))^2))
```

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

Task 3

```
# abs(sigma) used because it is actually sqrt(sigma^2)
minus_log_likelihood <- function(param, x){
  mu <- param[1]
  sigma <- param[2]
  return(100 * log(abs(sigma) * sqrt(2 * pi)) + 1/2 * sum(((data-mu)/sigma)^2))
}
gradient_fun <- function(param){
  mu <- param[1]
  sigma <- param[2]
  return(c(-sum((data - mu)/abs(sigma)), (100/abs(sigma)) - (1 / abs(sigma)^3) * sum((data - mu)^2)))
}

CG_ngf <- optim(c(0,1), minus_log_likelihood, method = "CG")
CG_ngf
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       87       19
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
CG_wgf <- optim(c(0,1), minus_log_likelihood, method = "CG", gr=gradient_fun)
CG_wgf
```

```
## $par
## [1] 1.275528 2.005976
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       36       11
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
BFGS_ngf <- optim(c(0,1), minus_log_likelihood, method = "BFGS")
BFGS_ngf
```

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

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

```
BFGS_wgf <- optim(c(0,1), minus_log_likelihood, method = "BFGS", gr=gradient_fun)
BFGS_wgf
```

```
## $par
## [1] 1.275521 2.005980
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       36       16
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The likelihoods that are calculated are very close to zero which can lead to underflow problems quickly. Transformation to log-likelihood does not change the relations of order of the likelihood function and can thus be used to calculate the maximum and does not lead to underflow problems as (positive) small numbers are transformed to negative numbers.

Task 4

The algorithms converged in all cases. The optimal values for μ and σ were

```
CG_ngf$par
```

```
## [1] 1.275528 2.005977
```

for the CG method without gradient function.

```
CG_wgf$par
```

```
## [1] 1.275528 2.005976
```

for the CG method with gradient function.

```
BFGS_ngf$par
```

```
## [1] 1.275528 2.005977
```

for the BFGS method without gradient function.

```
BFGS_wgf$par
```

```
## [1] 1.275521 2.005980
```

for the BFGS method without gradient function.

The following amount of evaluations of the function and the gradient function were necessary:

```
CG_ngf$counts
```

```
## function gradient
##       87       19
```

for the CG method without gradient function.

```
CG_wgf$counts
```

```
## function gradient
##       36       11
```

for the CG method with gradient function.

```
BFGS_ngf$counts
```

```
## function gradient
##       37       15
```

for the BFGS method without gradient function.

```
BFGS_wgf$counts
```

```
## function gradient
##       36       16
```

for the BFGS method without gradient function.

We can see that providing the gradient function vastly reduces the function calls needed in the case of the Conjugate Gradient method. It converged to the correct result and had the fewest function and gradient calls. This is why we would recommend the CG method with a provided gradient.

Appendix

```r
knitr::opts_chunk$set(dev = 'pdf')
find_a <- function(f, x0, x1, x2){
  stopifnot("x0, x1, x2 should be between [0, 1]" = any(c(x0,x1,x2) <= 1 | c(x0,x1,x2) >= 0),
            "f(x1) < f(x0), f(x2)" = any(c(f(x0),f(x2)) >= f(x0)))

  f_tilde <- function(x, a){
    a0 <- a[1]
    a1 <- a[2]
    a2 <- a[3]
    a0 + a1 * x + a2 * x^2
  }
  sq_error <- function(a){
    a0 <- a[1]
    a1 <- a[2]
    a2 <- a[3]
    return((f(x0) - f_tilde(x0, a))^2 + (f(x1) - f_tilde(x1, a))^2 + (f(x2) - f_tilde(x2, a))^2)
  }
  return(optim(par = c(0,0,0), fn = sq_error))
}
appr_fun <- function(f, n){
  interval_length <- 1/n
  a <- matrix(nrow=n, ncol=3)
  for (i in 1:n){
    a[i,] <- find_a(f,
                    (i-1) * interval_length,
                    (i-1) * interval_length + 1 / 2 * interval_length,
                    i* interval_length)$par
  }
  res_fun <- function(x){
    stopifnot(0 <= x & x <= 1)
    if (x == 0) {index <- 1} else {index <- ceiling(x * n)}
    return(a[index, 1] + a[index, 2] * x + a[index, 3] * x^2)
  }
  return(res_fun)
}

library(ggplot2)
library(gridExtra)

f1 <- function(x) {-x * (1 - x)}
f2 <- function(x) {-x * sin(10 * pi * x)}

f1_tilde <- appr_fun(f1, 100)
f2_tilde <- appr_fun(f2, 100)

x_values <- seq(0,1, by=0.01)
data1 <- data.frame(x = c(x_values, x_values),
                    y = c(f1(x_values), sapply(x_values, f1_tilde)),
                    group=c(rep("f1", length(x_values)), rep("f1_tilde", length(x_values))))
data2 <- data.frame(x = c(x_values, x_values),
                    y = c(f2(x_values), sapply(x_values, f2_tilde)),
                    group=c(rep("f2", length(x_values)), rep("f2_tilde", length(x_values))))

plot1 <- ggplot(data1, aes(x, y, color=group)) +
  geom_line() +
  theme(legend.position = 'bottom', legend.title=element_blank())
plot2 <- ggplot(data2, aes(x, y, color=group)) +
  geom_line() +
  theme(legend.position = 'bottom', legend.title=element_blank())
grid.arrange(plot1, plot2, ncol=2)
library(ggplot2)
library(gridExtra)

f1 <- function(x) {-x * (1 - x)}
f2 <- function(x) {-x * sin(10 * pi * x)}

f1_tilde <- appr_fun(f1, 1000)
f2_tilde <- appr_fun(f2, 1000)

x_values <- seq(0,1, by=0.001)
data1 <- data.frame(x = c(x_values, x_values),
                    y = c(f1(x_values), sapply(x_values, f1_tilde)),
                    group=c(rep("f1", length(x_values)), rep("f1_tilde", length(x_values))))
data2 <- data.frame(x = c(x_values, x_values),
                    y = c(f2(x_values), sapply(x_values, f2_tilde)),
                    group=c(rep("f2", length(x_values)), rep("f2_tilde", length(x_values))))

plot1 <- ggplot(data1, aes(x, y, color=group)) +
  geom_line() +
  theme(legend.position = 'bottom', legend.title=element_blank())
plot2 <- ggplot(data2, aes(x, y, color=group)) +
  geom_line() +
  theme(legend.position = 'bottom', legend.title=element_blank())
grid.arrange(plot1, plot2, ncol=2)
library(ggplot2)
library(gridExtra)

f1 <- function(x) {-x * (1 - x)}
f2 <- function(x) {-x * sin(10 * pi * x)}

f1_tilde <- appr_fun(f1, 50)
f2_tilde <- appr_fun(f2, 50)

x_values <- seq(0,1, by=0.01)
data1 <- data.frame(x = c(x_values, x_values),
                    y = c(f1(x_values), sapply(x_values, f1_tilde)),
                    group=c(rep("f1", length(x_values)), rep("f1_tilde", length(x_values))))
data2 <- data.frame(x = c(x_values, x_values),
                    y = c(f2(x_values), sapply(x_values, f2_tilde)),
                    group=c(rep("f2", length(x_values)), rep("f2_tilde", length(x_values))))

plot1 <- ggplot(data1, aes(x, y, color=group)) +
  geom_line() +
  theme(legend.position = 'bottom', legend.title=element_blank())
```

```r
plot2 <- ggplot(data2, aes(x, y, color=group)) +
  geom_line() +
  theme(legend.position = 'bottom', legend.title=element_blank())
grid.arrange(plot1, plot2, ncol=2)
load("data.RData")

mean(data)
sqrt(1/(100) * sum((data - mean(data))^2))

# abs(sigma) used because it is actually sqrt(sigma^2)
minus_log_likelihood <- function(param, x){
  mu <- param[1]
  sigma <- param[2]
  return(100 * log(abs(sigma) * sqrt(2 * pi)) + 1/2 * sum(((data-mu)/sigma)^2))
}
gradient_fun <- function(param){
  mu <- param[1]
  sigma <- param[2]
  return(c(-sum((data - mu)/abs(sigma)), (100/abs(sigma)) - (1 / abs(sigma)^3) * sum((data - mu)^2)))
}

CG_ngf <- optim(c(0,1), minus_log_likelihood, method = "CG")
CG_ngf
CG_wgf <- optim(c(0,1), minus_log_likelihood, method = "CG", gr=gradient_fun)
CG_wgf

BFGS_ngf <- optim(c(0,1), minus_log_likelihood, method = "BFGS")
BFGS_ngf
BFGS_wgf <- optim(c(0,1), minus_log_likelihood, method = "BFGS", gr=gradient_fun)
BFGS_wgf
CG_ngf$par
CG_wgf$par
BFGS_ngf$par
BFGS_wgf$par
CG_ngf$counts
CG_wgf$counts
BFGS_ngf$counts
BFGS_wgf$counts
```