HW3 - DATA 609

Thomas Hill

October 3, 2021

```
library(stats)
library(optimr)
```

Ex. 1 – Write down Newton's formula for finding the minimum of $f(x) = \frac{3x^4 - 4x^3}{12}$ in the range of [-10,10]. Then, implement in R.

Newton's method of finding the minimum or maximum value of a function, f(x), is by solving the formula

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)} = x_k - \frac{g'(x_k)}{g''(x_k)}$$

where $g(x) = f'(x) = 0$. Since we're looking at the range [-10, 10], we can use the start point of -10 and increasing, or 10 and decreasing.

```
function_1 <- expression((3*x^4 - 4*x^3)/12)

g_x <- D(function_1,"x")

g_x
```

```
## (3 * (4 * x^3) - 4 * (3 * x^2))/12
```

```
deriv_g_x <- D(g_x,"x")

deriv_g_x
```

```
## (3 * (4 * (3 * x^2)) - 4 * (3 * (2 * x)))/12
```

It's clear from g(x) that there's at least one zero at x = 0. Factoring also gives the following:

$$x^3 - x^2 = x^2(x - 1)$$

So there's another critical point at x = 1. Substituting into these points shows the true maximum over this range is at x = 1. Let's now use R to find the value using Newton's method.

```
ex_1_newton <- function(x_k) {

  result <- x_k - ((3 * (4 * x_k^3) - 4 * (3 * x_k^2))/12)/((3 * (4 * (3 * x_k^2)) - 4 * (3 * (2
* x_k)))/12)
  return((result))
}





ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(
10))))))))
```

```
## [1] 1.01487
```

```
ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton
(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(-10))))))))))))))
```

```
## [1] -0.007949901
```

Iterating through these several times brings us very close to the two zeroes. However, let's try doing the same thing between [0,1]

```
ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton
(ex_1_newton(0.4)))))))))
```

```
## [1] 0.0003501794
```

```
ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton(ex_1_newton
(ex_1_newton(0.75)))))))))
```

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

Between these two points, we are also given two different answers. So, it appears that Newton's method is unable converge at a single minimum over this range.

Ex. 2 – Explore *optimize()* in R and try to solve the previous problem.

```
ex_2_function <- function(x) {
  (3*x^4 - 4*x^3)/12
}

optimize(ex_2_function, c(-10,10))
```

```
## $minimum
## [1] 0.9999986
##
## $objective
## [1] -0.08333333
```

Using the built-in function R, a single value is returned with no errors raised.

Ex. 3 – Use any optimization algorithm to find the minimum of $f(x, y) = (x - 1)^2 + 100(y - x^2)^2$ in the domain $-10 \leq x, y \leq 10$. Discuss any issues concerning the optimization process.

After inspecting the formula f(x,y), it's clear that f(x,y) is greater than or equal to zero for all real numbers. This is because it is the sum of two squares. It appears it is equal to zero when x = 1, and when $y = x^2$. This gives two global minima at (0,0) and (1,1) Using Newton's method, I need to find the Hessian matrix as well as the gradient:

```
ex_3_F <- expression((x-1)^2 + 100*(y- (x)^2)^2)

g_x3 <- D(ex_3_F,"x")

g_xy3 <- D(g_x3, 'y')
g_xx3 <- D(g_x3, 'x')

g_y3 <-  D(ex_3_F,"y")

g_yx3 <- D(g_y3, 'x')
g_yy3 <- D(g_y3, 'y')

print(g_xx3)
```

```
## 2 - 100 * (2 * (2 * (y - (x)^2) - 2 * (x) * (2 * (x))))
```

```
print(g_xy3)
```

```
## -(100 * (2 * (2 * (x))))
```

```
print(g_yx3)
```

```
## -(100 * (2 * (2 * (x))))
```

```
print(g_yy3)
```

```
## 100 * 2
```

The resulting Hessian is:

$$H = \begin{bmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{bmatrix}$$

Its inverse is:

$$H^{-1} = \frac{1}{200(1200 - x^2 - 400y + 2) - (400x)^2} \begin{bmatrix} 200 & 400x \\ 400x & 1200x^2 - 400y + 2 \end{bmatrix}$$

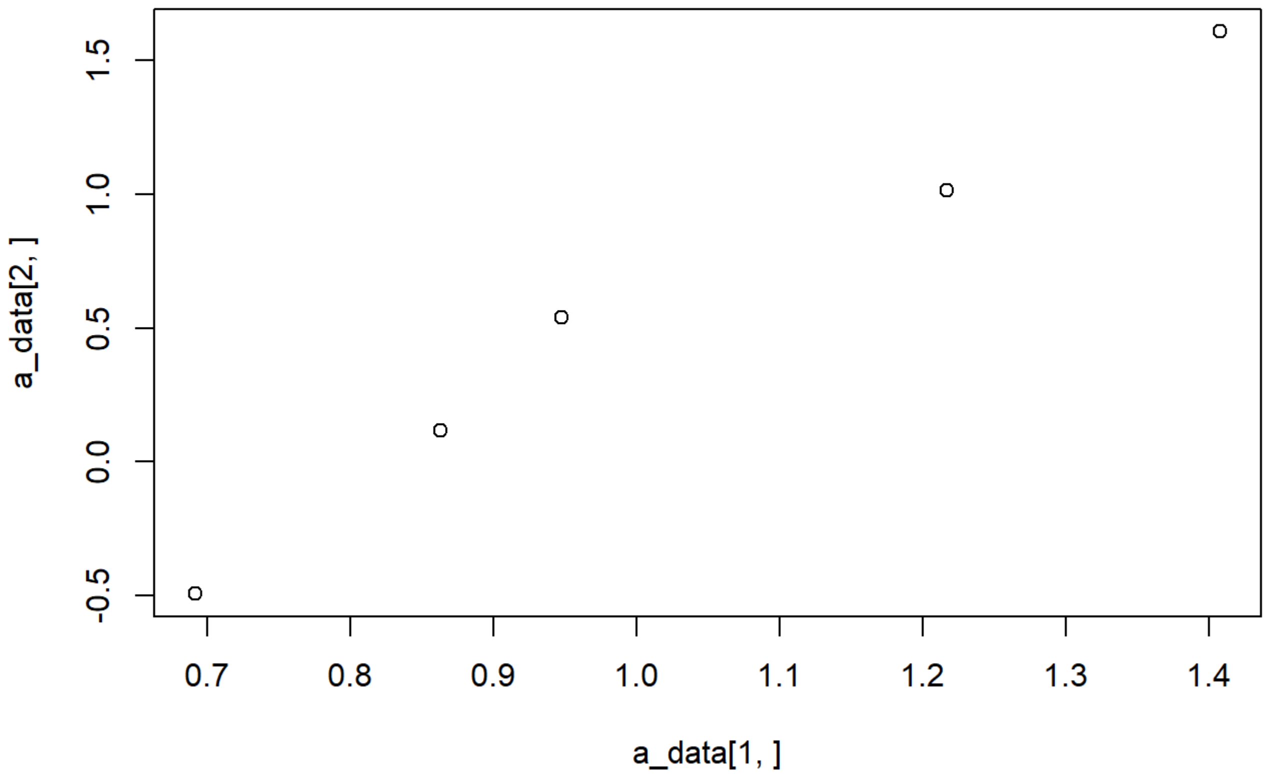
and gradient:

$$\nabla f = \begin{bmatrix} 2x(x-1) + 100(y^2 - 2x^2y + x^4) \\ 200(y - x^2) \end{bmatrix}$$

To avoid too much trial and error, I'll identify the critical points for x and y as well. Setting g(x) and g(y) equal to zero, points of interest include

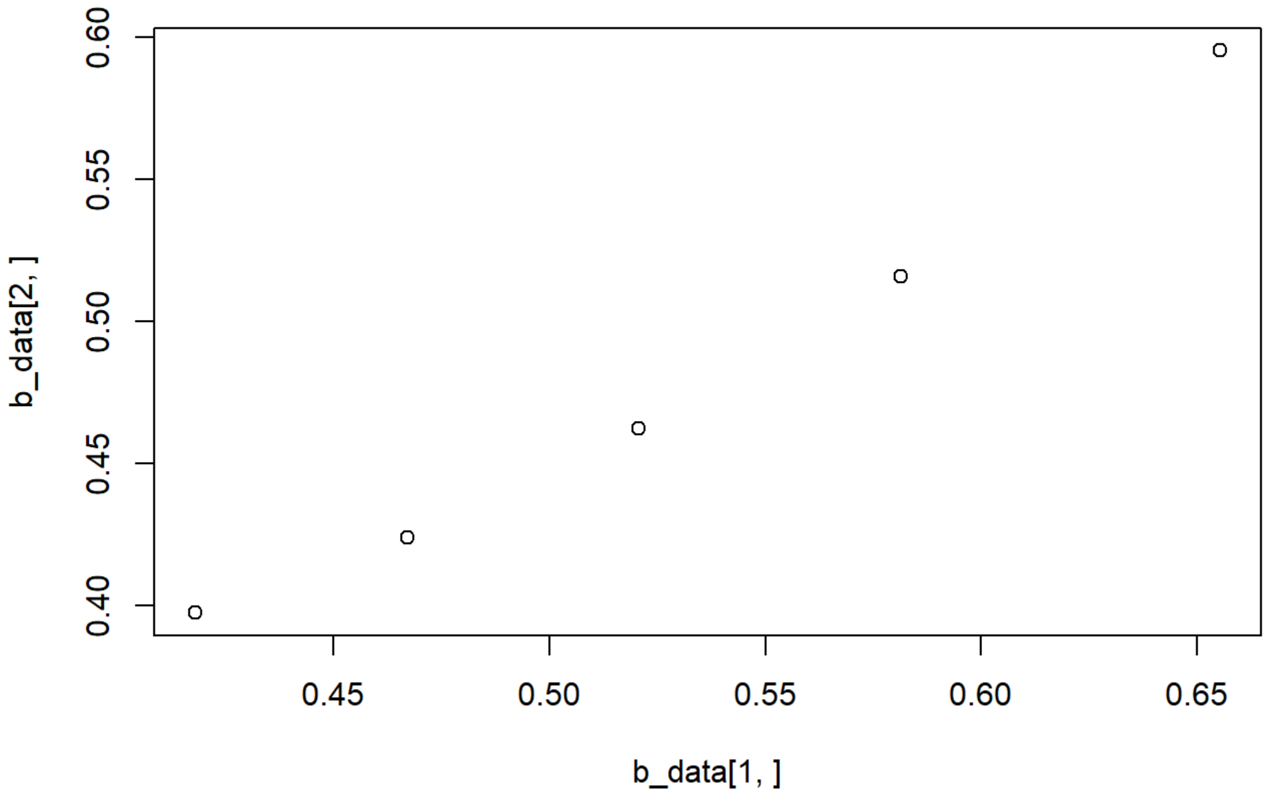
```
ex_3_newton <- function(mat_params) {
  x <- mat_params[1]
  y <- mat_params[2]
  newton <-matrix(c(x,y), nrow = 2) - (1/(200*(1200-x^2-400*y+2)-(400*x)^2)) * matrix(c(200, 400
*x, 400*x, 1200*x^2 -400*y + 2), nrow = 2) %*% matrix(c(2*x*(x-1) + 100*(y^2 - 2*(x^2)*y + x^4),
200*(y-x^2)), nrow = 2)
  return(newton)
  }
```

```
a_1 <- ex_3_newton(c(1.2,1.2))
a_2 <-ex_3_newton(ex_3_newton(c(1.2,1.2)))
a_3 <- ex_3_newton(ex_3_newton(ex_3_newton(c(1.2,1.2))))
a_4 <-  ex_3_newton(ex_3_newton(ex_3_newton(ex_3_newton(c(1.2,1.2)))))
a_5 <- ex_3_newton(ex_3_newton(ex_3_newton(ex_3_newton(ex_3_newton(c(1.2,1.2))))))

a_data <- cbind(a_1,a_2,a_3,a_4,a_5)
plot(x = a_data[1,], y = a_data[2,])
```



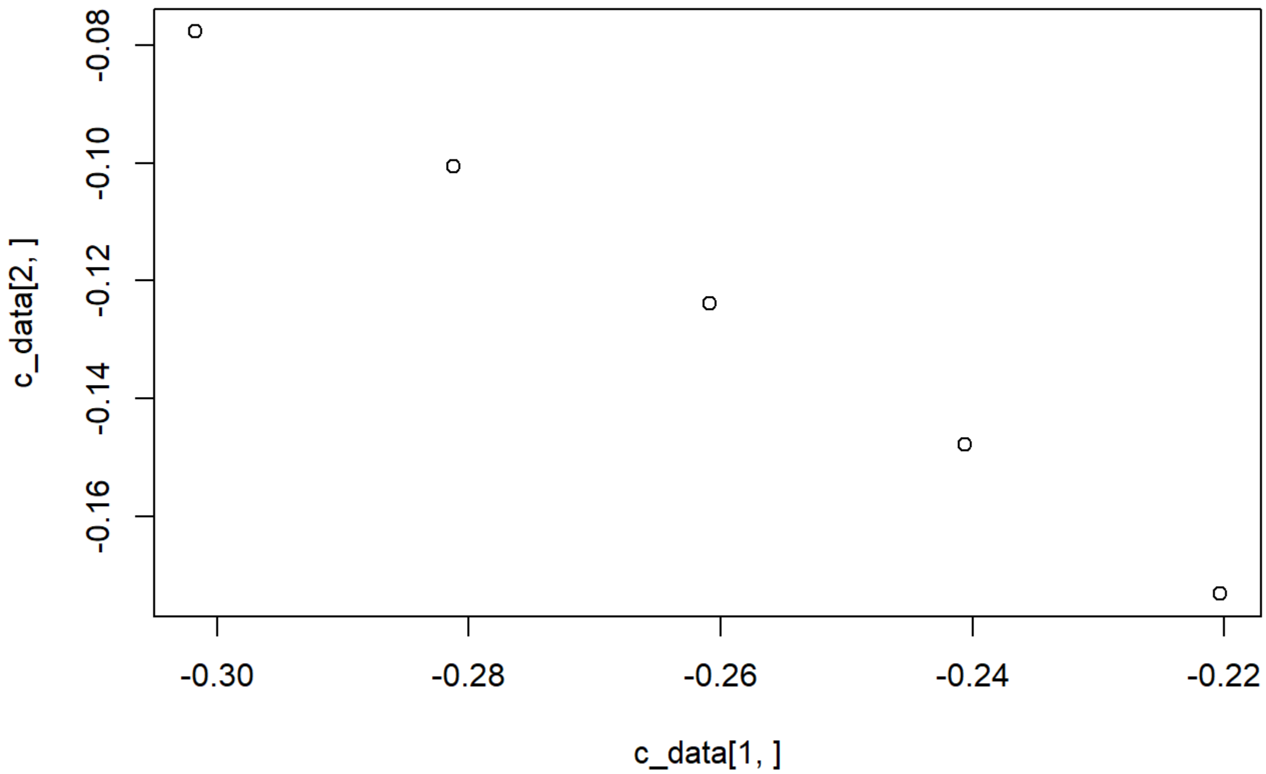
This plot point does not converge to any of the minima.

```
b_1 <-ex_3_newton(c(0.8,0.8))
b_2 <- ex_3_newton(ex_3_newton(c(0.8,0.8)))
b_3 <- ex_3_newton(ex_3_newton(ex_3_newton(c(0.8,0.8))))
b_4 <- ex_3_newton(ex_3_newton(ex_3_newton(ex_3_newton(c(0.8,0.8)))))
b_5 <- ex_3_newton(ex_3_newton(ex_3_newton(ex_3_newton(ex_3_newton(c(0.8,0.8))))))



b_data <- cbind(b_1,b_2,b_3,b_4,b_5)
plot(x = b_data[1,], y = b_data[2,])
```



However, this plot shows convergence to (0,0).

```
c_1 <-ex_3_newton(c(-0.2,-0.2))
c_2 <- ex_3_newton(ex_3_newton(c(-0.2,-0.2)))
c_3 <- ex_3_newton(ex_3_newton(ex_3_newton(c(-0.2,-0.2))))
c_4 <- ex_3_newton(ex_3_newton(ex_3_newton(ex_3_newton(c(-0.2,-0.2)))))
c_5 <- ex_3_newton(ex_3_newton(ex_3_newton(ex_3_newton(ex_3_newton(c(-0.2,-0.2))))))

c_data <- cbind(c_1,c_2,c_3,c_4,c_5)
plot(x = c_data[1,], y = c_data[2,])
```



Looking at negative starting values, these also do not converge. It’s very obvious in the multivariable, unconstrained case that Newton’s method is unreliabile for finding minima.

Ex. 4 – Expore the *optimr* package for R and try to solve the previous problem.

```
ex_4_function <- function(para) {
  matrix.A <- matrix(para, ncol = 2) #matrix with values for x and y
  x <- matrix.A[,1]
  y <- matrix.A[,2]
  f.x <- (x-1)^2 + 100*(y - x^2)^2
  return(f.x)
}
```

```
#par1 <- c(-10, -10)
#par2 <- c(-1, -1)
#par3 <- c(10, 10)
#par4 <- c(2.5,2.5)
par5 <- c(-2.5,-2.5)
par6 <- c(0.1,0.1)

#optimr(par = par1, fn = ex_4_function)
#optimr(par = par2, fn = ex_4_function)
#optimr(par = par3, fn = ex_4_function)
#optimr(par = par4, fn = ex_4_function)
optimr(par = par5, fn = ex_4_function)
```

```
## $par
## [1] 0.9974529 0.9947764
##
## $value
## [1] 8.333996e-06
##
## $counts
## function gradient
##       63       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
optimr(par = par6, fn = ex_4_function)
```

```
## $par
## [1] 0.9999909 0.9999783
##
## $value
## [1] 1.281042e-09
##
## $counts
## function gradient
##      135       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

After some trial and error with starting points, it appears that optimr does find the solution at (1,1). However, some of the moe distant points (e.g, (10,10)) do not have enough iterations to converge on either point.