DATA 609 HW 3

Daniel Moscoe

9/22/2021

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

Response. Newton's method is an algorithm for determining an extreme value for a function of one variable. The algorithm generates successive approximations of the value of x that optimizes the function. Newton's formula is the expression for the $k+1$th approximation given the kth. The general formula is

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

For the function above, Newton's formula is

$$x_{k+1} = x_k - \frac{x_k^3 - x_k^2}{3x_k^2 - 2x_k}$$

Implementing Newton's formula for this function in R:

```
#Original function:
ex1 <- function(x){
  res <- (3 * x^4 - 4 * x^3) / 12
  return(res)
}

#Newton's formula:
Newt <- function(x){
  res <- x - ((x^3 - x^2)/(3 * x^2 - 2 * x))
  return(res)
}
```

We can build the function `NewtLoop`, which uses `Newt` to perform Newton's algorithm for finding the value of x that minimizes $f(x)$. When successive approximations differ by less than `D`, the loop stops. `NewtLoop` returns a list containing `x`, the near-optimum value of x, `value`, the value of the function defined in exercise 1 at `x`, and `iters`, the number of iterations required to determine `x`.

```
#xk is the initial guess.

#D is the maximum difference between successive guesses permitted
#before reporting results.

NewtLoop <- function(xk, D){
  res <- list()
  D <- D
```

```
  iters <- 0
  cond <- TRUE
  while(cond){
    xkp1 <- Newt(xk)
    d <- abs(xkp1 - xk)
    xk <- xkp1
    iters <- iters + 1
    cond <- d > D
  }
  res$x <- xk
  res$value <- ex1(xk)
  res$iters <- iters
  return(res)
}
```

```
print(NewtLoop(-3, 0.01))
```

```
## $x
## [1] -0.009076013
##
## $value
## [1] 2.509056e-07
##
## $iters
## [1] 10
```

With an initial guess less than 0, Newton's Method returns a value for both x and $f(x)$ near 0. Although $(0, 0)$ is a critical point for the function, it is not either a local or global extremum. This is a common issue when using Newton's Method to locate extrema: other methods must also be used to verify that the critical points located by Newton's Method are in fact optimal.

```
print(NewtLoop(3, 0.01))
```

```
## $x
## [1] 1.000142
##
## $value
## [1] -0.08333332
##
## $iters
## [1] 6
```

With a positive initial guess, we are able to come very close to the true global minimum for the function. The computed values, $(1.000142, -0.083333)$, are very close to the true values, $(1, -1/12)$.

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

Response. We can use `optimize` to locate the global minimum of $f(x)$.

```
ans <- optimize(ex1, lower = -10, upper = 10, maximum = FALSE)
ans
```

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

`optimize()` returns values very close to the true global minimum, $(1, -1/12)$.

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.*

Response. I will use Newton's Method for multivariate functions to find the minimum of $f(x,y)$.

Newton's method for multivariate functions uses the formula below to update an initial guess:

$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{H}^{-1}(\boldsymbol{x}^{(k)})\nabla f(\boldsymbol{x}^{(k)})$.

We can iteratively evaluate the formula above using the function `mvNewtLoop`.

```
#f(x,y)
ex3 <- function(x,y){
  return((x - 1)^2 + 100 * (y - x^2)^2)
}

#Gradient of f(x,y)
grad <- function(x,y){
  return(matrix(c(2 * (x - 1) + 200 * (y - x^2) * (-2 * x), 200 * (y - x^2)), nrow = 2))
}

#Hessian of f(x,y)
hess <- function(x,y){
  xx <- 2 - 400 * y + 1200 * x^2
  yy <- 200
  xy <- -400 * x
  return(matrix(c(xx,xy,xy,yy),nrow = 2))
}

#Calculate x_{k+1}, y_{k+1}
mvNewt <- function(xk, yk){
  Xk <- matrix(c(xk, yk), nrow = 2)
  Hess <- solve(hess(xk, yk))
  grad <- grad(xk, yk)
  return(Xk - Hess %*% grad)
}

#Newton's Method
mvNewtLoop <- function(xk, yk, D){
  res <- list()
  D <- D
  iters <- 0
  cond <- TRUE
  while(cond){
    xkp1 <- mvNewt(xk, yk)[1]
    ykp1 <- mvNewt(xk, yk)[2]
    d <- sqrt((xkp1 - xk)^2 + (ykp1 - yk)^2)
    xk <- xkp1
```

```
    yk <- ykp1
    iters <- iters + 1
    cond <- d > D
  }
  res$X <- c(xk, yk)
  res$value <- ex3(xk, yk)
  res$iters <- iters
  return(res)
}
```

```
print(mvNewtLoop(3,4,0.05))
```

```
## $X
## [1] 1.0000000 0.9999975
##
## $value
## [1] 6.422538e-10
##
## $iters
## [1] 4
```

`mvNewtLoop` returns a value for (x, y) very close to the true optimal value, $(1, 1)$. It returns a minimum value of $f(x, y)$ very close to the true minimum value, 0.

```
print(mvNewtLoop(-1234,-5678,0.05))
```

```
## $X
## [1] 1 1
##
## $value
## [1] 6.727008e-14
##
## $iters
## [1] 4
```

For this function, Newton's Method quickly returns the optimum values even when an initial guess has opposite sign, or is far from the optimum.

Ex. 4. *Explore the* `optimr()` *package for R and try to solve the previous problem.*

Response. `optimr()` is a wrapper that gathers other R tools for optimization under a single interface. It takes as arguments an initial guess `par`, a function to be optimized `fn`, a function to compute the function's gradient `gr`, and optional bounds. The argument `method` can be used to specify a particular optimization method.

Calling `optimr()` with a specified gradient function:

```
library(optimr)
```

```
## Warning: package 'optimr' was built under R version 4.0.5
```

```r
#Initial guess
p = c(2,3)

#Objective function modified for optimr
ex4 <- function(p){
  return((p[1] - 1)^2 + 100 * (p[2] - p[1]^2)^2)
}

#Gradient function modified for optimr
grad4 <- function(p){
  return(matrix(c(
    2 * (p[1] - 1) + 200 * (p[2] - p[1]^2) * (-2 * p[1]),
    200 * (p[2] - p[1]^2)),
    nrow = 2))
}

ans <- optimr(par = p, fn = ex4,
              gr = grad4, lower = -10, upper = 10,
              method = "L-BFGS-B")
ans
```

```
## $par
## [1] 1 1
##
## $value
## [1] 2.938712e-16
##
## $counts
## function gradient
##       31       31
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Calling `optimr()` without a specified gradient function:

```r
#Initial guess
p = c(2,3)

ans <- optimr(par = p, fn = ex4, lower = -10, upper = 10, method = "L-BFGS-B")
ans
```

```
## $par
## [1] 0.9998119 0.9996239
##
## $value
## [1] 3.537743e-08
##
## $counts
## function gradient
```

```
##       56       56
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Without a specified gradient function, the results are still very close to the true values. However, the function made 25 additional calls to `fn` compared to when a gradient function was specified. This suggests that specifying a gradient function may greatly improve the perfomance of `optimr()`.