Project 4

Will Graham; Richelle Lee; Robin Lin

2022-11-17

hess\_with\_finite\_differencing <- function(current\_theta, grad, eps){  
 # This function uses finite differencing to approximate the hessian matrix  
 # of an objective function with respect to a supplied parameter vector of   
 # thetas.   
 #  
 # Inputs:  
 # current\_theta - a parameter vector that the hessian will be evaluated at  
 # grad - the gradient function. Returns the gradient vector of the objective  
 # w.r.t. the elements of parameter vector  
 #   
 # Returns:  
 # a Hessian matrix approximated with finite differencing  
   
 p = length(current\_theta)  
   
 hess <- matrix(0,p,p)  
   
   
 for (i in 1:p){  
 for (j in 1:i){  
 change\_vector <- matrix(0,p)  
 change\_vector[j] <- eps / 2  
 hess[i,j] <- (grad(theta + change\_vector) - grad(theta - change\_vector))[i]  
 }  
 }  
   
 hess <- hess / eps  
   
 hess[upper.tri(hess)] <- hess[lower.tri(hess)]  
   
 return(hess)  
}

newt <- function(theta, func, grad, hess = NULL, ..., tol = 1e-8, fscale = 1, maxit = 100, max.half = 20, eps = 1e-6){  
   
 # if no hessian matrix function is supplied, default to finite differencing  
 hess.supplied <- TRUE # Default to assume hessian is supplied  
 if(is.null(hess)){ # If hessian is not supplied  
 hess.supplied <- FALSE #   
 }  
   
 obj\_at\_theta <- func(theta) # evaluate objective function at initial theta  
 grad\_at\_theta <- grad(theta) # evaluate gradient at initial theta  
   
 if(hess.supplied){  
 hess\_at\_theta <- hess(theta) # evaluate hessian at inital theta  
 } else {  
 hess\_at\_theta <- hess\_with\_finite\_differencing(theta, grad, eps)  
 }  
   
   
 # combine obj, gradient, and hessian values at theta into 1 vector  
 obj\_and\_derivatives <- c(obj\_at\_theta, grad\_at\_theta, hess\_at\_theta)  
   
 if (Inf %in% obj\_and\_derivatives | -Inf %in% obj\_and\_derivatives){  
 stop("Objective Funciton or Derivatives Not Finite at Initial Theta")  
 }  
   
 i <- 0 # Tracks the iterations.   
 while(i < maxit){  
 original\_theta <- theta  
 obj\_at\_theta <- func(theta)  
 grad\_at\_theta <- grad(theta)  
   
 if(hess.supplied){  
 hess\_at\_theta <- hess(theta) # evaluate hessian at inital theta  
 } else {  
 hess\_at\_theta <- hess\_with\_finite\_differencing(theta, grad, eps)  
 }  
   
 chol\_of\_hess <- try(chol(hess\_at\_theta), silent = TRUE)  
 eig\_hess <- eigen(hess\_at\_theta) # Computes the eigen-decomposition on the # hessian matrix.  
 lambdas <- eig\_hess$values # Identifies the lambdas from the # eigen-decomposition.  
 U <- eig\_hess$vectors # Identifies the U from the eigen-decomposition of # the hessian matrix or the perturbed hessian matrix  
   
 if(all(class(chol\_of\_hess) == 'try-error')){  
 hess\_is\_pd <- FALSE  
 if(0 %in% lambdas){  
 hess\_is\_inv <- FALSE  
 } else {  
 hess\_is\_inv <- TRUE  
 inv\_hess\_before\_perturb <- U %\*% (diag(1 / lambdas)) %\*% t(U) # Calculates the inverse of # the hessian matrix.  
 }   
 pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) + 1) \* diag(dim(as.matrix(hess\_at\_theta))[1]) # perturb the hessian matrix.  
   
 eig\_hess <- eigen(pert\_hess\_at\_theta) # Computes the   
 # eigen-decomposition on the perturbed hessian matrix.  
 lambdas <- eig\_hess$values # Identifies the lambdas from the # eigen-decomposition of the perturbed hessian matrix.  
 U <- eig\_hess$vectors # Identifies the U from the eigen-decomposition of # the hessian matrix or the perturbed hessian matrix  
   
 } else {  
 hess\_is\_pd <- TRUE  
 }  
   
 inv\_hess <- U %\*% (diag(1 / lambdas)) %\*% t(U) # Calculates the inverse of # the hessian matrix.  
   
 if (all(abs(grad\_at\_theta) < tol\*(abs(obj\_at\_theta) + fscale))){  
 if(hess\_is\_pd){  
 return(list('f' = obj\_at\_theta, 'theta' = theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = inv\_hess))  
 }   
 warning('Hessian is not positive definite at convergence.\n')  
 if(hess\_is\_inv){  
 return(list('f' = obj\_at\_theta, 'theta' = theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = inv\_hess\_before\_perturb))  
 }  
 return(list('f' = obj\_at\_theta, 'theta' = theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = "Hessian is not invertible"))  
 }  
   
   
 descent\_direction <- as.vector(-inv\_hess %\*% grad\_at\_theta)  
 stepsize <- 1  
 halves <- 0  
 theta\_hat <- theta + stepsize \* descent\_direction  
   
 suppressWarnings(  
 while((obj\_at\_theta < func(theta\_hat) && halves <= max.half) | (!(is.finite(func(theta\_hat))) && halves <= max.half)){  
 stepsize <- stepsize / 2  
 theta\_hat <- theta + stepsize \* descent\_direction  
 halves <- halves + 1  
 }  
 )   
   
 if(halves == (max.half + 1)){  
 warning("Maximum number of step halvings reached \n")  
 if(hess\_is\_pd){  
 return(list('f' = obj\_at\_theta, 'theta' = theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = inv\_hess))  
   
 }   
 warning('Hessian is not positive definite at convergence.\n')  
 if(hess\_is\_inv){  
 return(list('f' = obj\_at\_theta, 'theta' = theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = inv\_hess\_before\_perturb))  
 }  
 return(list('f' = obj\_at\_theta, 'theta' = theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = "Hessian is not invertible"))  
   
 }  
 theta <- theta\_hat  
 i <- i + 1  
   
   
 }  
 warning("Maximum iterations reached without convergence \n")  
   
 if(hess\_is\_pd){  
 return(list('f' = obj\_at\_theta, 'theta' = original\_theta, 'iter' = maxit, 'g' = grad\_at\_theta, 'Hi' = inv\_hess))  
   
 }   
 warning('Hessian is not positive definite at max iterations.\n')  
 if(hess\_is\_inv){  
 return(list('f' = obj\_at\_theta, 'theta' = original\_theta, 'iter' = maxit, 'g' = grad\_at\_theta, 'Hi' = inv\_hess\_before\_perturb))  
 }  
 return(list('f' = obj\_at\_theta, 'theta' = original\_theta, 'iter' = maxit, 'g' = grad\_at\_theta, 'Hi' = "Hessian is not invertible"))  
   
}

## Typical function with a minimum with no special attributes  
rb <- function(th,k=2) {  
 k\*(th[2]-th[1]^2)^2 + (1-th[1])^2  
}  
gb <- function(th,k=2) {  
 c(-2\*(1-th[1])-k\*4\*th[1]\*(th[2]-th[1]^2),k\*2\*(th[2]-th[1]^2))  
}  
  
hb <- function(th,k=2) {  
 h <- matrix(0,2,2)  
 h[1,1] <- 2-k\*2\*(2\*(th[2]-th[1]^2) - 4\*th[1]^2)  
 h[2,2] <- 2\*k  
 h[1,2] <- h[2,1] <- -4\*k\*th[1]  
 h  
}  
  
  
theta = c(10,10)  
test <- newt(theta, rb, gb, hb, k=2)  
print(test)

## $f  
## [1] 3.643551e-29  
##   
## $theta  
## [1] 1 1  
##   
## $iter  
## [1] 17  
##   
## $g  
## [1] 3.419487e-14 -1.332268e-14  
##   
## $Hi  
## [,1] [,2]  
## [1,] 0.5 1.00  
## [2,] 1.0 2.25

## Linear function y = x, using finite differencing  
rb <- function(th) {  
 th  
   
}  
gb <- function(th) {  
 1  
}  
  
hb <- function(th) {  
 0  
}  
  
theta = c(10)  
  
test <- newt(theta, rb, gb, fscale = 0.5)

## Warning in newt(theta, rb, gb, fscale = 0.5): Maximum iterations reached without convergence

## Warning in newt(theta, rb, gb, fscale = 0.5): Hessian is not positive definite at max iterations.

print(test)

## $f  
## [1] -89  
##   
## $theta  
## [1] -89  
##   
## $iter  
## [1] 100  
##   
## $g  
## [1] 1  
##   
## $Hi  
## [1] "Hessian is not invertible"

rb <- function(th) {  
 th[1]^2 + th[2]^2  
   
}  
gb <- function(th) {  
 c(th[1]\*2, th[2]\*2)  
}  
  
hb <- function(th) {  
 matrix(c(2,0,0,2),2,2)  
}  
  
theta = c(100, 1000)  
  
test <- newt(theta, rb, gb, fscale = 0.5)  
print(test)

## $f  
## [1] 1.526956e-22  
##   
## $theta  
## [1] 6.374399e-16 1.235701e-11  
##   
## $iter  
## [1] 2  
##   
## $g  
## [1] 1.274880e-15 2.471401e-11  
##   
## $Hi  
## [,1] [,2]  
## [1,] 0.5 0.0000000  
## [2,] 0.0 0.4999999

# debug(newt)  
# newt(theta, rb, gb)  
# undebug(newt)

# Rprof()  
# newt(theta, rb, gb, hb, fscale = 0.5, maxit = 4)  
# Rprof(NULL)  
# summaryRprof()