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(current\_theta + change\_vector) - grad(current\_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 # Set hessian to not supplied  
 }  
   
 obj\_at\_theta <- func(theta) # Evaluate objective function at initial theta  
 grad\_at\_theta <- grad(theta) # Evaluate gradient at initial theta  
   
 if(hess.supplied){ # Checks whether hessian is supplied, if so,  
 hess\_at\_theta <- hess(theta) # evaluate hessian at inital theta  
 } else { #otherwise,  
 # approximate hessian matrix by finite differencing  
 hess\_at\_theta <- hess\_with\_finite\_differencing(theta, grad, eps)  
 }  
   
 # Collate obj, gradient, and hessian values at intial theta into 1 vector,  
 obj\_and\_derivatives <- c(obj\_at\_theta, grad\_at\_theta, hess\_at\_theta)  
   
 # and check that all values are finite.  
 # If not finite,  
   
 if (!all(is.finite(obj\_and\_derivatives)) ){  
 stop("Objective Funciton or Derivatives Not Finite at Initial Theta")  
 }  
   
 i <- 0 # Initialises the number of iterations to be 0.   
 while(i < maxit){ # Tracks the number of iterations up to the maximum allowance.  
 original\_theta <- theta # Creates a new variable passing the value of theta.  
 obj\_at\_theta <- func(theta) # Evaluates the corresponding function value at  
 # theta.  
 grad\_at\_theta <- grad(theta) # Evaluates the corresponding gradient value at  
 # theta.   
   
 # Reckon the Hessian.  
 if(hess.supplied){ # If a user specifies the Hessian matrix.  
 hess\_at\_theta <- hess(theta) # Evaluates the value at theta.  
 } else { # If the user doesn't specify it.  
 hess\_at\_theta <- hess\_with\_finite\_differencing(theta, grad, eps)  
 # Applys the finite differencing function, and evaluates the value at theta.  
 }  
   
 chol\_of\_hess <- try(chol(hess\_at\_theta), silent = TRUE) # Make a trial of   
 # constructing Cholesky decomposition to the Hessian.  
   
 if(all(class(chol\_of\_hess) != 'try-error')){ # If the trial returns a numeric  
 # matrix.  
 hess\_is\_pd <- TRUE # Determines that the Hessian is positive definite.  
 hess\_is\_inv <- TRUE # Determines that the Hessian is invertible.  
 inv\_hess <- chol2inv(chol\_of\_hess) # Reckons the inverse of Hessian from   
 # Cholesky decomposition.  
 } else { # If the trial returns an error.  
 hess\_is\_pd <- FALSE # Determines that the Hessian is not positive definite.   
 eig\_hess <- eigen(hess\_at\_theta) # Computes the Eigen-decomposition on the   
 # Hessian matrix.  
 lambdas <- matrix(eig\_hess$values) # Identifies the eigenvalue matrix of the  
 # Hessian.  
 U <- eig\_hess$vectors # Identifies the matrix of eigenvectors of the Hessian   
 # matrix or the perturbed Hessian matrix.  
 if(0 %in% lambdas){ # If there is an eigenvalue of 0.  
 hess\_is\_inv <- FALSE # Hessian is not invertible.  
 } else { # If all eigenvalues are non-zero.  
 hess\_is\_inv <- TRUE # Hessian is invertible.  
 inv\_hess\_before\_perturb <- U %\*% (diag(1 / lambdas)) %\*% t(U) # Calculates  
 # the inverse of Hessian by Eigen-decomposition.  
 }  
 pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) + 1) \* diag(dim(as.matrix(hess\_at\_theta))[1])   
 # Perturbs the Hessian.  
 inv\_hess <- chol2inv(chol(pert\_hess\_at\_theta))   
 # Calculates the inverse of the perturbed Hessian.  
 }   
   
 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' = original\_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' = original\_theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = inv\_hess\_before\_perturb))  
 }  
 return(list('f' = obj\_at\_theta, 'theta' = original\_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, k = 2)  
print(test)

## $f  
## [1] 4.447203e-29  
##   
## $theta  
## [1] 1 1  
##   
## $iter  
## [1] 17  
##   
## $g  
## [1] 3.819167e-14 -1.509903e-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"

## A function of a cone  
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] 8.889676e-38  
##   
## $theta  
## [1] -5.293956e-23 -2.981556e-19  
##   
## $iter  
## [1] 2  
##   
## $g  
## [1] -1.058791e-22 -5.963112e-19  
##   
## $Hi  
## [,1] [,2]  
## [1,] 0.5 0.0  
## [2,] 0.0 0.5

## Will's function reaching the limit of step halving.

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

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