Project 3

Will Graham; Richelle Lee; Robin Lin

2022-11-14

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  
   
 delta <- eps # use a static delta for finite differencing  
   
 # Calculate differences in gradient function  
 grad\_diff <- grad(current\_theta + delta/2) - grad(current\_theta - delta/2)  
   
 # Repeat the gradient differences column m times, where m is the number of  
 # thetas. The resulting matrix is now of size mxm  
 grad\_diff <- matrix(grad\_diff, length(grad\_diff), length(grad\_diff))  
   
 hess\_approx <- grad\_diff / delta # Approximate the Hessian matrix  
   
 # Ensure Hessian approximation is symmetric  
 hess\_approx <- (t(hess\_approx) + hess\_approx) / 2   
   
 return(hess\_approx)  
}

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  
 if(is.null(hess)){   
 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){  
 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)  
 }  
   
   
 if(sum(abs(grad\_at\_theta) < tol \* abs(obj\_at\_theta) + fscale) == length(theta)){  
 chol\_of\_hess <- try(chol(hess\_at\_theta), silent = TRUE)  
 if(all(class(chol\_of\_hess) == 'try-error')){  
 warning('Hessian is not positive definite at convergence.')  
 return(list('f' = obj\_at\_theta, 'theta' = theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = "Hessian is not invertible"))  
   
 }  
 return(list('f' = obj\_at\_theta, 'theta' = theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = inv\_hess))  
 }  
   
 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.  
   
 if(min(lambdas) <= 0){ # If the smallest value of the lambdas is smaller # than zero,  
 pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) + 1)\* diag(length(hess\_at\_theta)) # 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  
   
 inv\_hess <- U %\*% (diag(1/lambdas)) %\*% t(U) # Calculates the inverse of # the hessian matrix.  
   
 descent\_direction <- -inv\_hess %\*% grad\_at\_theta  
 stepsize <- 1  
 halves <- 0  
 theta\_hat <- theta + stepsize \* descent\_direction  
   
   
 while(obj\_at\_theta < func(theta\_hat) && halves <= max.half){  
 stepsize <- stepsize / 2  
 theta\_hat <- theta + stepsize \* descent\_direction  
 halves <- halves + 1  
   
 }  
 if(halves == (max.half + 1)){  
 stop("Maximum number of step halvings reached")  
 }  
 theta <- theta\_hat  
 i <- i + 1  
   
 }  
 warning("Maximum iterations reached without convergence")  
 ## CHECK FOR INVERSE HESSIAN  
 return(list('f' = obj\_at\_theta, 'theta' = theta, 'iter' = i, 'g' = grad\_at\_theta, 'Hi' = inv\_hess))  
}

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, 2)

rb <- function(th) {  
 th  
  
}  
gb <- function(th) {  
 1  
}  
  
hb <- function(th) {  
0  
}  
  
theta = c(10)  
  
test <- newt(theta, rb, gb, hb, fscale = 0.5, maxit = 4)

## Warning in newt(theta, rb, gb, hb, fscale = 0.5, maxit = 4): Maximum iterations  
## reached without convergence

debug(newt)  
newt(theta, rb, gb, hb, fscale = 0.5, maxit = 4)

## debugging in: newt(theta, rb, gb, hb, fscale = 0.5, maxit = 4)  
## debug at <text>#2: {  
## hess.supplied <- TRUE  
## if (is.null(hess)) {  
## hess.supplied <- FALSE  
## }  
## obj\_at\_theta <- func(theta)  
## grad\_at\_theta <- grad(theta)  
## if (hess.supplied) {  
## hess\_at\_theta <- hess(theta)  
## }  
## else {  
## hess\_at\_theta <- hess\_with\_finite\_differencing(theta,   
## grad, eps)  
## }  
## 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  
## while (i < maxit) {  
## obj\_at\_theta <- func(theta)  
## grad\_at\_theta <- grad(theta)  
## if (hess.supplied) {  
## hess\_at\_theta <- hess(theta)  
## }  
## else {  
## hess\_at\_theta <- hess\_with\_finite\_differencing(theta,   
## grad, eps)  
## }  
## if (sum(abs(grad\_at\_theta) < tol \* abs(obj\_at\_theta) +   
## fscale) == length(theta)) {  
## chol\_of\_hess <- try(chol(hess\_at\_theta), silent = TRUE)  
## if (all(class(chol\_of\_hess) == "try-error")) {  
## warning("Hessian is not positive definite at convergence.")  
## return(list(f = obj\_at\_theta, theta = theta,   
## iter = i, g = grad\_at\_theta, Hi = "Hessian is not invertible"))  
## }  
## return(list(f = obj\_at\_theta, theta = theta, iter = i,   
## g = grad\_at\_theta, Hi = inv\_hess))  
## }  
## eig\_hess <- eigen(hess\_at\_theta)  
## lambdas <- eig\_hess$values  
## if (min(lambdas) <= 0) {  
## pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) +   
## 1) \* diag(length(hess\_at\_theta))  
## eig\_hess <- eigen(pert\_hess\_at\_theta)  
## lambdas <- eig\_hess$values  
## }  
## U <- eig\_hess$vectors  
## inv\_hess <- U %\*% (diag(1/lambdas)) %\*% t(U)  
## descent\_direction <- -inv\_hess %\*% grad\_at\_theta  
## stepsize <- 1  
## halves <- 0  
## theta\_hat <- theta + stepsize \* descent\_direction  
## while (obj\_at\_theta < func(theta\_hat) && halves <= max.half) {  
## stepsize <- stepsize/2  
## theta\_hat <- theta + stepsize \* descent\_direction  
## halves <- halves + 1  
## }  
## if (halves == (max.half + 1)) {  
## stop("Maximum number of step halvings reached")  
## }  
## theta <- theta\_hat  
## i <- i + 1  
## }  
## warning("Maximum iterations reached without convergence")  
## return(list(f = obj\_at\_theta, theta = theta, iter = i, g = grad\_at\_theta,   
## Hi = inv\_hess))  
## }  
## debug at <text>#5: hess.supplied <- TRUE  
## debug at <text>#6: if (is.null(hess)) {  
## hess.supplied <- FALSE  
## }  
## debug at <text>#10: obj\_at\_theta <- func(theta)  
## debug at <text>#11: grad\_at\_theta <- grad(theta)  
## debug at <text>#13: if (hess.supplied) {  
## hess\_at\_theta <- hess(theta)  
## } else {  
## hess\_at\_theta <- hess\_with\_finite\_differencing(theta, grad,   
## eps)  
## }  
## debug at <text>#14: hess\_at\_theta <- hess(theta)  
## debug at <text>#21: obj\_and\_derivatives <- c(obj\_at\_theta, grad\_at\_theta, hess\_at\_theta)  
## debug at <text>#23: if (Inf %in% obj\_and\_derivatives | -Inf %in% obj\_and\_derivatives) {  
## stop("Objective Funciton or Derivatives Not Finite at Initial Theta")  
## }  
## debug at <text>#27: i <- 0  
## debug at <text>#28: while (i < maxit) {  
## obj\_at\_theta <- func(theta)  
## grad\_at\_theta <- grad(theta)  
## if (hess.supplied) {  
## hess\_at\_theta <- hess(theta)  
## }  
## else {  
## hess\_at\_theta <- hess\_with\_finite\_differencing(theta,   
## grad, eps)  
## }  
## if (sum(abs(grad\_at\_theta) < tol \* abs(obj\_at\_theta) + fscale) ==   
## length(theta)) {  
## chol\_of\_hess <- try(chol(hess\_at\_theta), silent = TRUE)  
## if (all(class(chol\_of\_hess) == "try-error")) {  
## warning("Hessian is not positive definite at convergence.")  
## return(list(f = obj\_at\_theta, theta = theta, iter = i,   
## g = grad\_at\_theta, Hi = "Hessian is not invertible"))  
## }  
## return(list(f = obj\_at\_theta, theta = theta, iter = i,   
## g = grad\_at\_theta, Hi = inv\_hess))  
## }  
## eig\_hess <- eigen(hess\_at\_theta)  
## lambdas <- eig\_hess$values  
## if (min(lambdas) <= 0) {  
## pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) +   
## 1) \* diag(length(hess\_at\_theta))  
## eig\_hess <- eigen(pert\_hess\_at\_theta)  
## lambdas <- eig\_hess$values  
## }  
## U <- eig\_hess$vectors  
## inv\_hess <- U %\*% (diag(1/lambdas)) %\*% t(U)  
## descent\_direction <- -inv\_hess %\*% grad\_at\_theta  
## stepsize <- 1  
## halves <- 0  
## theta\_hat <- theta + stepsize \* descent\_direction  
## while (obj\_at\_theta < func(theta\_hat) && halves <= max.half) {  
## stepsize <- stepsize/2  
## theta\_hat <- theta + stepsize \* descent\_direction  
## halves <- halves + 1  
## }  
## if (halves == (max.half + 1)) {  
## stop("Maximum number of step halvings reached")  
## }  
## theta <- theta\_hat  
## i <- i + 1  
## }  
## debug at <text>#29: obj\_at\_theta <- func(theta)  
## debug at <text>#30: grad\_at\_theta <- grad(theta)  
## debug at <text>#32: if (hess.supplied) {  
## hess\_at\_theta <- hess(theta)  
## } else {  
## hess\_at\_theta <- hess\_with\_finite\_differencing(theta, grad,   
## eps)  
## }  
## debug at <text>#33: hess\_at\_theta <- hess(theta)  
## debug at <text>#39: if (sum(abs(grad\_at\_theta) < tol \* abs(obj\_at\_theta) + fscale) ==   
## length(theta)) {  
## chol\_of\_hess <- try(chol(hess\_at\_theta), silent = TRUE)  
## if (all(class(chol\_of\_hess) == "try-error")) {  
## warning("Hessian is not positive definite at convergence.")  
## return(list(f = obj\_at\_theta, theta = theta, iter = i,   
## g = grad\_at\_theta, Hi = "Hessian is not invertible"))  
## }  
## return(list(f = obj\_at\_theta, theta = theta, iter = i, g = grad\_at\_theta,   
## Hi = inv\_hess))  
## }  
## debug at <text>#49: eig\_hess <- eigen(hess\_at\_theta)  
## debug at <text>#50: lambdas <- eig\_hess$values  
## debug at <text>#52: if (min(lambdas) <= 0) {  
## pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) +   
## 1) \* diag(length(hess\_at\_theta))  
## eig\_hess <- eigen(pert\_hess\_at\_theta)  
## lambdas <- eig\_hess$values  
## }  
## debug at <text>#53: pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) + 1) \*   
## diag(length(hess\_at\_theta))  
## debug at <text>#55: eig\_hess <- eigen(pert\_hess\_at\_theta)  
## debug at <text>#57: lambdas <- eig\_hess$values  
## debug at <text>#59: U <- eig\_hess$vectors  
## debug at <text>#61: inv\_hess <- U %\*% (diag(1/lambdas)) %\*% t(U)  
## debug at <text>#63: descent\_direction <- -inv\_hess %\*% grad\_at\_theta  
## debug at <text>#64: stepsize <- 1  
## debug at <text>#65: halves <- 0  
## debug at <text>#66: theta\_hat <- theta + stepsize \* descent\_direction  
## debug at <text>#69: while (obj\_at\_theta < func(theta\_hat) && halves <= max.half) {  
## stepsize <- stepsize/2  
## theta\_hat <- theta + stepsize \* descent\_direction  
## halves <- halves + 1  
## }  
## debug at <text>#75: if (halves == (max.half + 1)) {  
## stop("Maximum number of step halvings reached")  
## }  
## debug at <text>#78: theta <- theta\_hat  
## debug at <text>#79: i <- i + 1  
## debug at <text>#28: (while) i < maxit  
## debug at <text>#29: obj\_at\_theta <- func(theta)  
## debug at <text>#30: grad\_at\_theta <- grad(theta)  
## debug at <text>#32: if (hess.supplied) {  
## hess\_at\_theta <- hess(theta)  
## } else {  
## hess\_at\_theta <- hess\_with\_finite\_differencing(theta, grad,   
## eps)  
## }  
## debug at <text>#33: hess\_at\_theta <- hess(theta)  
## debug at <text>#39: if (sum(abs(grad\_at\_theta) < tol \* abs(obj\_at\_theta) + fscale) ==   
## length(theta)) {  
## chol\_of\_hess <- try(chol(hess\_at\_theta), silent = TRUE)  
## if (all(class(chol\_of\_hess) == "try-error")) {  
## warning("Hessian is not positive definite at convergence.")  
## return(list(f = obj\_at\_theta, theta = theta, iter = i,   
## g = grad\_at\_theta, Hi = "Hessian is not invertible"))  
## }  
## return(list(f = obj\_at\_theta, theta = theta, iter = i, g = grad\_at\_theta,   
## Hi = inv\_hess))  
## }  
## debug at <text>#49: eig\_hess <- eigen(hess\_at\_theta)  
## debug at <text>#50: lambdas <- eig\_hess$values  
## debug at <text>#52: if (min(lambdas) <= 0) {  
## pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) +   
## 1) \* diag(length(hess\_at\_theta))  
## eig\_hess <- eigen(pert\_hess\_at\_theta)  
## lambdas <- eig\_hess$values  
## }  
## debug at <text>#53: pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) + 1) \*   
## diag(length(hess\_at\_theta))  
## debug at <text>#55: eig\_hess <- eigen(pert\_hess\_at\_theta)  
## debug at <text>#57: lambdas <- eig\_hess$values  
## debug at <text>#59: U <- eig\_hess$vectors  
## debug at <text>#61: inv\_hess <- U %\*% (diag(1/lambdas)) %\*% t(U)  
## debug at <text>#63: descent\_direction <- -inv\_hess %\*% grad\_at\_theta  
## debug at <text>#64: stepsize <- 1  
## debug at <text>#65: halves <- 0  
## debug at <text>#66: theta\_hat <- theta + stepsize \* descent\_direction  
## debug at <text>#69: while (obj\_at\_theta < func(theta\_hat) && halves <= max.half) {  
## stepsize <- stepsize/2  
## theta\_hat <- theta + stepsize \* descent\_direction  
## halves <- halves + 1  
## }  
## debug at <text>#75: if (halves == (max.half + 1)) {  
## stop("Maximum number of step halvings reached")  
## }  
## debug at <text>#78: theta <- theta\_hat  
## debug at <text>#79: i <- i + 1  
## debug at <text>#28: (while) i < maxit  
## debug at <text>#29: obj\_at\_theta <- func(theta)  
## debug at <text>#30: grad\_at\_theta <- grad(theta)  
## debug at <text>#32: if (hess.supplied) {  
## hess\_at\_theta <- hess(theta)  
## } else {  
## hess\_at\_theta <- hess\_with\_finite\_differencing(theta, grad,   
## eps)  
## }  
## debug at <text>#33: hess\_at\_theta <- hess(theta)  
## debug at <text>#39: if (sum(abs(grad\_at\_theta) < tol \* abs(obj\_at\_theta) + fscale) ==   
## length(theta)) {  
## chol\_of\_hess <- try(chol(hess\_at\_theta), silent = TRUE)  
## if (all(class(chol\_of\_hess) == "try-error")) {  
## warning("Hessian is not positive definite at convergence.")  
## return(list(f = obj\_at\_theta, theta = theta, iter = i,   
## g = grad\_at\_theta, Hi = "Hessian is not invertible"))  
## }  
## return(list(f = obj\_at\_theta, theta = theta, iter = i, g = grad\_at\_theta,   
## Hi = inv\_hess))  
## }  
## debug at <text>#49: eig\_hess <- eigen(hess\_at\_theta)  
## debug at <text>#50: lambdas <- eig\_hess$values  
## debug at <text>#52: if (min(lambdas) <= 0) {  
## pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) +   
## 1) \* diag(length(hess\_at\_theta))  
## eig\_hess <- eigen(pert\_hess\_at\_theta)  
## lambdas <- eig\_hess$values  
## }  
## debug at <text>#53: pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) + 1) \*   
## diag(length(hess\_at\_theta))  
## debug at <text>#55: eig\_hess <- eigen(pert\_hess\_at\_theta)  
## debug at <text>#57: lambdas <- eig\_hess$values  
## debug at <text>#59: U <- eig\_hess$vectors  
## debug at <text>#61: inv\_hess <- U %\*% (diag(1/lambdas)) %\*% t(U)  
## debug at <text>#63: descent\_direction <- -inv\_hess %\*% grad\_at\_theta  
## debug at <text>#64: stepsize <- 1  
## debug at <text>#65: halves <- 0  
## debug at <text>#66: theta\_hat <- theta + stepsize \* descent\_direction  
## debug at <text>#69: while (obj\_at\_theta < func(theta\_hat) && halves <= max.half) {  
## stepsize <- stepsize/2  
## theta\_hat <- theta + stepsize \* descent\_direction  
## halves <- halves + 1  
## }  
## debug at <text>#75: if (halves == (max.half + 1)) {  
## stop("Maximum number of step halvings reached")  
## }  
## debug at <text>#78: theta <- theta\_hat  
## debug at <text>#79: i <- i + 1  
## debug at <text>#28: (while) i < maxit  
## debug at <text>#29: obj\_at\_theta <- func(theta)  
## debug at <text>#30: grad\_at\_theta <- grad(theta)  
## debug at <text>#32: if (hess.supplied) {  
## hess\_at\_theta <- hess(theta)  
## } else {  
## hess\_at\_theta <- hess\_with\_finite\_differencing(theta, grad,   
## eps)  
## }  
## debug at <text>#33: hess\_at\_theta <- hess(theta)  
## debug at <text>#39: if (sum(abs(grad\_at\_theta) < tol \* abs(obj\_at\_theta) + fscale) ==   
## length(theta)) {  
## chol\_of\_hess <- try(chol(hess\_at\_theta), silent = TRUE)  
## if (all(class(chol\_of\_hess) == "try-error")) {  
## warning("Hessian is not positive definite at convergence.")  
## return(list(f = obj\_at\_theta, theta = theta, iter = i,   
## g = grad\_at\_theta, Hi = "Hessian is not invertible"))  
## }  
## return(list(f = obj\_at\_theta, theta = theta, iter = i, g = grad\_at\_theta,   
## Hi = inv\_hess))  
## }  
## debug at <text>#49: eig\_hess <- eigen(hess\_at\_theta)  
## debug at <text>#50: lambdas <- eig\_hess$values  
## debug at <text>#52: if (min(lambdas) <= 0) {  
## pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) +   
## 1) \* diag(length(hess\_at\_theta))  
## eig\_hess <- eigen(pert\_hess\_at\_theta)  
## lambdas <- eig\_hess$values  
## }  
## debug at <text>#53: pert\_hess\_at\_theta <- hess\_at\_theta + (-(min(lambdas)) + 1) \*   
## diag(length(hess\_at\_theta))  
## debug at <text>#55: eig\_hess <- eigen(pert\_hess\_at\_theta)  
## debug at <text>#57: lambdas <- eig\_hess$values  
## debug at <text>#59: U <- eig\_hess$vectors  
## debug at <text>#61: inv\_hess <- U %\*% (diag(1/lambdas)) %\*% t(U)  
## debug at <text>#63: descent\_direction <- -inv\_hess %\*% grad\_at\_theta  
## debug at <text>#64: stepsize <- 1  
## debug at <text>#65: halves <- 0  
## debug at <text>#66: theta\_hat <- theta + stepsize \* descent\_direction  
## debug at <text>#69: while (obj\_at\_theta < func(theta\_hat) && halves <= max.half) {  
## stepsize <- stepsize/2  
## theta\_hat <- theta + stepsize \* descent\_direction  
## halves <- halves + 1  
## }  
## debug at <text>#75: if (halves == (max.half + 1)) {  
## stop("Maximum number of step halvings reached")  
## }  
## debug at <text>#78: theta <- theta\_hat  
## debug at <text>#79: i <- i + 1  
## debug at <text>#28: (while) i < maxit  
## debug at <text>#82: warning("Maximum iterations reached without convergence")

## Warning in newt(theta, rb, gb, hb, fscale = 0.5, maxit = 4): Maximum iterations  
## reached without convergence

## debug at <text>#84: return(list(f = obj\_at\_theta, theta = theta, iter = i, g = grad\_at\_theta,   
## Hi = inv\_hess))  
## exiting from: newt(theta, rb, gb, hb, fscale = 0.5, maxit = 4)

## $f  
## [,1]  
## [1,] 7  
##   
## $theta  
## [,1]  
## [1,] 6  
##   
## $iter  
## [1] 4  
##   
## $g  
## [1] 1  
##   
## $Hi  
## [,1]  
## [1,] 1

undebug(newt)