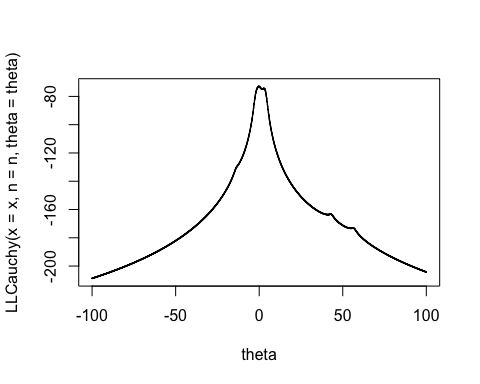
HW 3 - Ian Douglas

### Part A

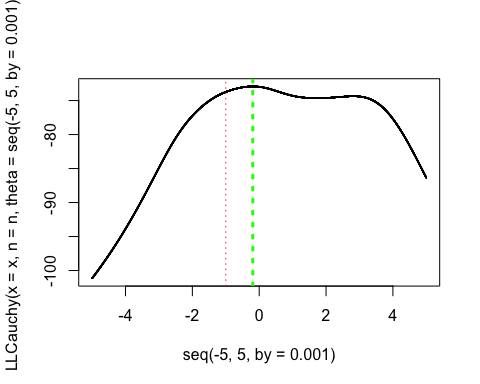
Graph the log-likelihood function

x<-matrix(c(1.77, -0.23, 2.76, 3.80, 3.47, 56.75, -1.34, 4.24, -2.44, 3.29,   
 3.71, -2.40, 4.53, -0.07, -1.05, -13.87, -2.53, -1.75, 0.27,   
 43.21),ncol=1)  
n <- dim(x)[1]  
LLCauchy <- function(theta,x=x, n=n) {  
 y <- c(rep(0,times=length(theta)))  
 for (i in 1:length(theta))  
 y[i] = -n\*log(pi) - sum(log(1 + (x-theta[i])^2))  
 y  
}  
theta <- seq(-100,100, by=.1)  
plot(theta, LLCauchy(x = x, n = n, theta = theta), cex = .05)



### Part B

#redefine function as the negative log likelihood, given x and n as above  
llCauchy = function(theta) {  
 n\*log(pi) + sum(log(1 + (x-theta)^2))  
}  
#Run with 3 sets of lower and upper bounds  
list <- matrix(c(-1,1,-2,-1,-3,3),ncol=2,byrow=TRUE)  
est <- matrix(c(rep(0,times=3)),ncol = 1)  
for (i in 1:3) {  
 est[i,1] <- (optimize(llCauchy, lower = list[i,1], upper = list[i,2]))$minimum  
}  
  
plot(x = seq(-5,5,by=.001),   
 y = LLCauchy(x = x, n = n, theta = seq(-5,5,by=.001)),   
 cex = .05)  
abline(v = est[1,1], col = "green",lty = 3, lwd = 3)  
abline(v = est[2,1],col = "red",lty = 3, lwd = 1)



The method for finding the minimum of the negative log-likelihood function worked well, and even found the maximum over an interval in which two local maxima existed, but where only one of which was the true global maximum.

The optimize function only failed when the search interval did not contain the actual minimum of the negative log-likelihood function. In that case it returned the maximum of the search interval.

### Question 2

Use any multivariate method to maxime the target function with starting points x1 = 0 and x2 = 3

#Target function  
f <- function(x1,x2) {-(x1 - 2)^4 - (x1 - 2\*x2)^2}

#Optim solution  
fbb <- function(x) -(x[1] - 2)^4 - (x[1] - 2\*x[2])^2  
# starting values must be a vector now  
optim(c(0,3), fbb, control=list(fnscale=-1))$par

## [1] 1.9783572 0.9890109

# METHOD: Gradient descent  
#Gradient function from the definition of the derivative  
fx1 <- function(x1,x2,h=0.001) (f(x1+h,x2)-f(x1,x2))/h  
fx2 <- function(x1,x2,h=0.001) (f(x1,x2+h)-f(x1,x2))/h  
f.gradient = function(x1,x2) {c(fx1(x1,x2),fx2(x1,x2))}  
n = 1000000  
alpha = 0.0001  
#Search:  
M = matrix(0, n, 2)  
M[1,1] = 0  
M[1,2] = 3  
  
for (i in 2:n)  
{  
 M[i,] = M[i-1,] + alpha\*f.gradient(M[i-1,1],M[i-1,2])  
}  
  
eval = numeric()  
for (i in 1:n) eval[i] = f(M[i,1], M[i,2])  
tail(cbind(eval,M))

## eval   
## [999995,] -7.070651e-05 1.908627 0.9538134  
## [999996,] -7.070651e-05 1.908627 0.9538134  
## [999997,] -7.070651e-05 1.908627 0.9538134  
## [999998,] -7.070651e-05 1.908627 0.9538134  
## [999999,] -7.070651e-05 1.908627 0.9538134  
## [1000000,] -7.070651e-05 1.908627 0.9538134

#Gradient function derived analytically using the chain rule  
f\_gradient = function(x1, x2) {c((-4\*(x1-2)^3)-2\*(x1-2\*x2), 8\*(x1-2\*x2))}  
n = 1000000  
alpha = 0.0001  
  
#Search:  
m = matrix(0, n, 2)  
# Initial values  
m[1,1] = 0  
m[1,2] = 3  
  
for (i in 2:n)  
{  
 m[i,] = m[i-1,] + alpha\*f\_gradient(m[i-1,1],m[i-1,2])  
}  
  
fval = numeric()  
for (i in 1:n) fval[i] = f(m[i,1], m[i,2])  
tail(cbind(fval,m))

## fval   
## [999995,] -1.970738e-06 1.962533 0.9812606  
## [999996,] -1.970734e-06 1.962533 0.9812606  
## [999997,] -1.970730e-06 1.962533 0.9812606  
## [999998,] -1.970726e-06 1.962533 0.9812606  
## [999999,] -1.970723e-06 1.962533 0.9812607  
## [1000000,] -1.970719e-06 1.962533 0.9812607

All three methods demonstrate that maximum of the function approaches zero, as x approaches 1.97 (or 2) and x2 approaches .98 (or 1).

The two approaches to the gradient method confirm the findings of the optim function, and further demonstrate that the analytical derivative is a more exact approximation than the reiman sum method of interval lengths .001