Homework 07 - STAT440

Joseph Sepich (jps6444)

10/18/2020

```r
set.seed(42)
```

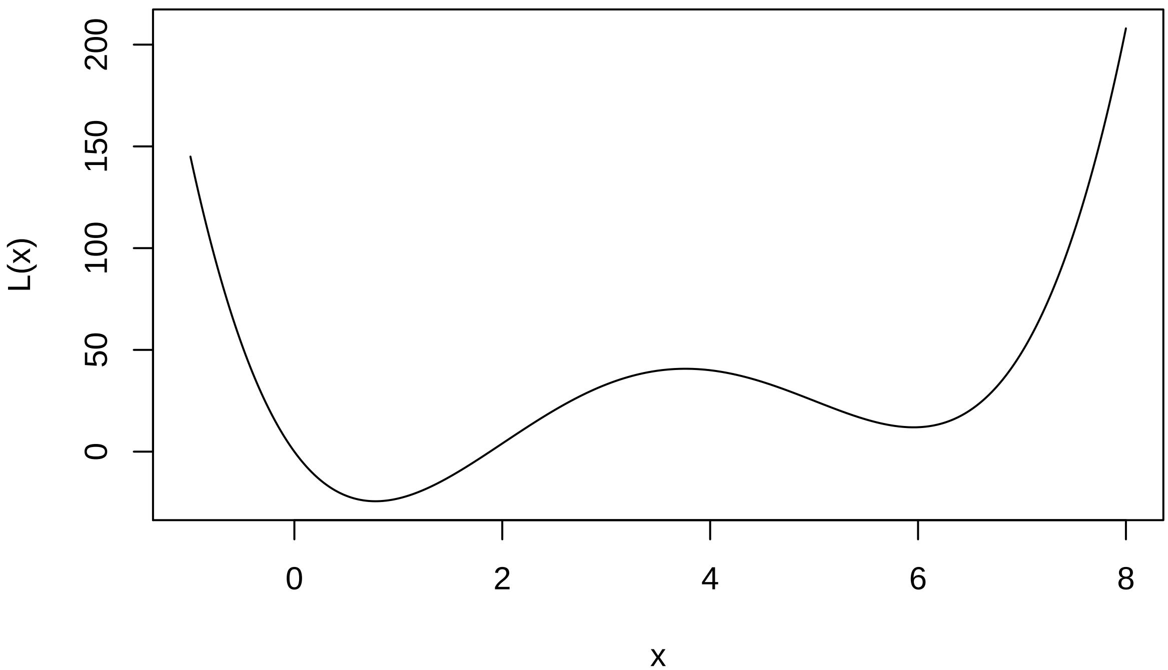
Problem 1

Define loss function in R:

```r
L <- function(x) {
    x^4 - 14*x^3 + 60*x^2 - 70*x
}
```

Part a

```r
# plot function
x <- seq(-1, 8, 0.001)
plot(x, L(x), type="l")
```



The function is not convex. You cannot draw a line between any two points on the line that remains above the line. The bump in the middle would prevent the point at x = 0 to a point at x = 6 from being above the function. (The second derivative is not always positive.) There are regions where it is locally convex. You could split the function approximately near where x take the value just below four and each side of the function (left and right) would be convex.

Part b

I would expect both Nelder-Mead and Newton's method to find local extrema. Since we are going to be looking for minimums I would expect both to either find the actual global minimum on the left or the local minimum on the right. The found minimum would depend on the initialization of the method. For example, if we started Newton's Method at x = 7 we would expect to find the suboptimal solution as opposed to started at x = 0, where would find the optimal solution.

Part c

```
example_nm <- function(par=c(0, 2), fn = L, return_points=FALSE) {
    # parameters
    alpha <- 1
    gamma <- 2
    rho <- 0.5
    sigma <- 0.5
```

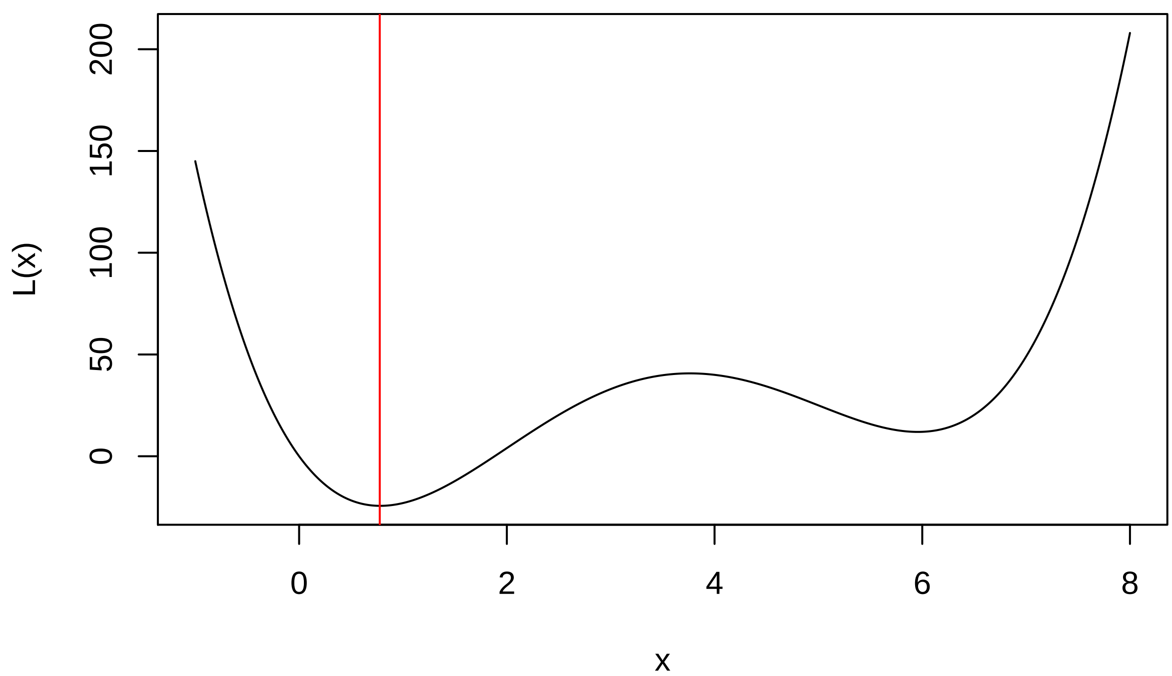
```r
    # termination criteria
    max_term <- 100
    sd_threshold <- 0.01

    dim <- 2
    points <- runif(dim, min=par[1], max=par[2])

    point_mat <- NA

    iter <- 1
    while(sd(points) >= sd_threshold & iter <= max_term) {
        # Order points
        points <- points[order(L(points))]
        if (return_points) {
            if (iter == 1) {
                point_mat <- matrix(points, nrow=1, ncol=dim)
            } else {
                point_mat <- rbind(point_mat, matrix(points, nrow=1, ncol=dim))
            }
        }

        # Compute centroid - this is merely the first point (2 points only)
        centroid <- points[1:dim-1] / (dim-1)
        # Reflect about centroid
        reflected_point <- centroid +  alpha * (centroid - points[dim])
        val_r <- L(reflected_point)
        # don't need to check reflected
        # criteria; if better than x_n but not better than x_1; isn't possible
        # since x_1 = x_n here

        # Expand step
        if (val_r < L(points[1])) {
            expanded <- centroid + gamma * (reflected_point - centroid)
            if (L(expanded) < val_r) {
                points[dim] <- expanded
            } else {
                points[dim] <- reflected_point
            }
        } else {
            contracted <- centroid + rho * (reflected_point - centroid)
            if (L(contracted) < L(points[dim])) {
                # contract
                points[dim] <- contracted
            } else {
                # shrink
                points <- points[1] + sigma * (points - points[1])
            }
        }
        # next iteration
        iter <- iter + 1
    }
    if (return_points) {
        return(point_mat)
```

```
    }
    return(points[1])
}
```

```
print(example_nm())
```

```
## [1] 0.7829517
```

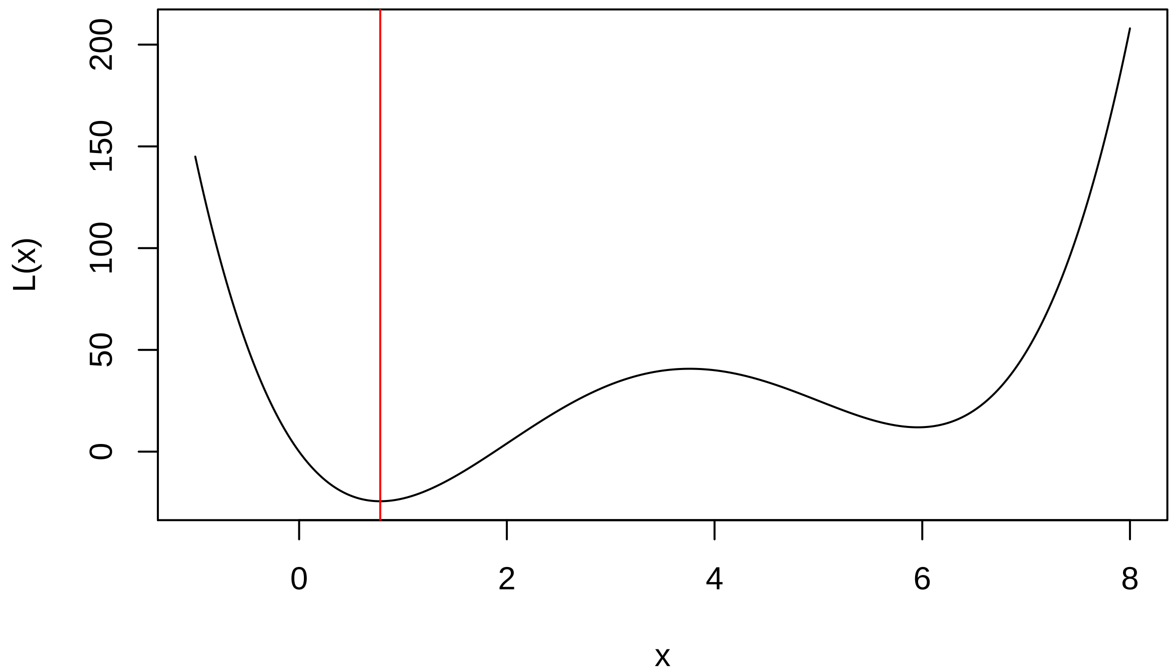
```
plot(x, L(x), type="l")
abline(v=example_nm(), col="red")
```




```
my_loss <- function(x) {
    L(x)[1]
}
val <- optim(par = c(0, 2), fn = my_loss, method="Nelder-Mead")$par[1]
print(val)
```

```
## [1] 0.7808841
```

```
plot(x, L(x), type="l")
abline(v=val, col="red")
```

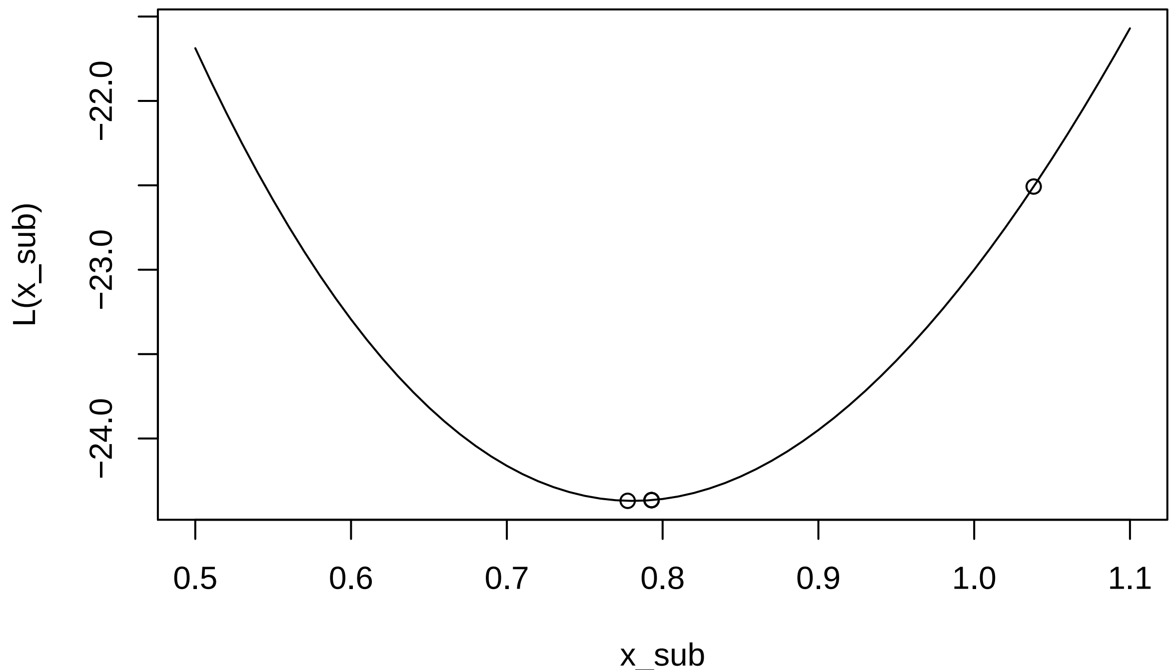


Part d

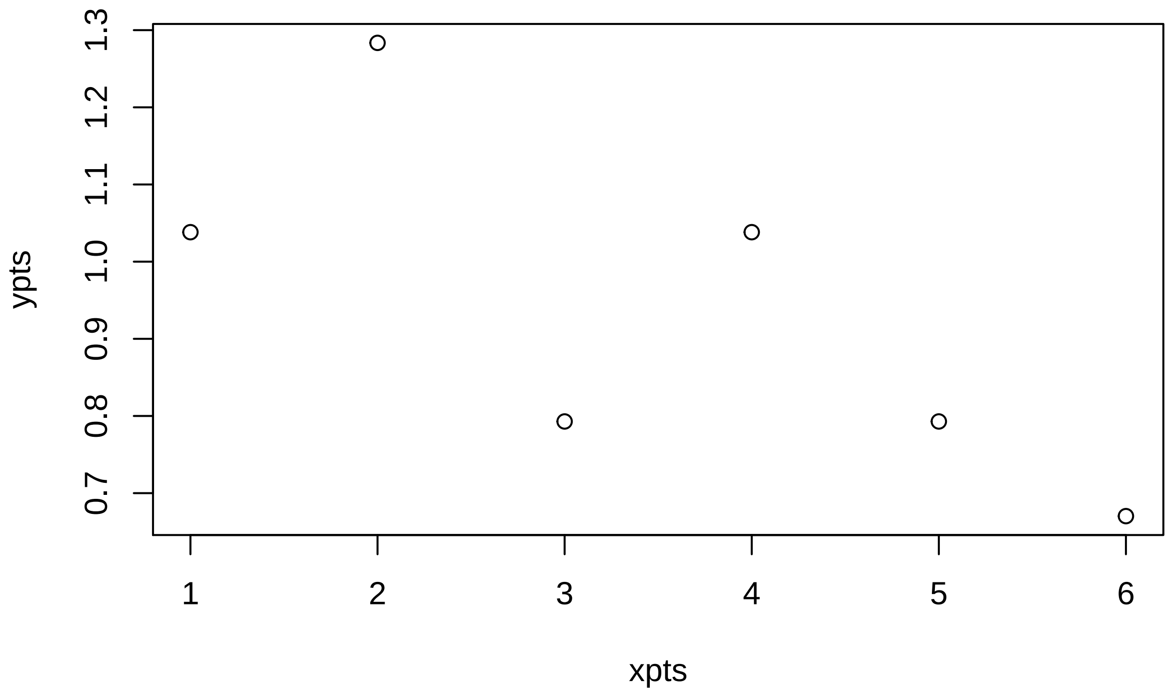
```
all_points <- example_nm(return_points = TRUE)
best_points <- all_points[,1]
print(best_points)
```

```
## [1] 1.0381919 0.7928928 0.7928928 0.7928928 0.7928928 0.7775616
```

```
x_sub <- seq(0.5, 1.1, 0.01)
plot(x_sub, L(x_sub), type="l")
points(best_points, L(best_points))
```




```r
xpts <- rep(1:length(best_points),each=dim)
ypts <- vector(mode="numeric",length=length(xpts))
for (i in 1:length(xpts)) {
    if (i %% 2 == 0) {
        ypts[i] <- all_points[i/2,2]
    } else {
        ypts[i] <- all_points[floor(i/2)+1,1]
    }
}
plot(xpts, ypts)
```

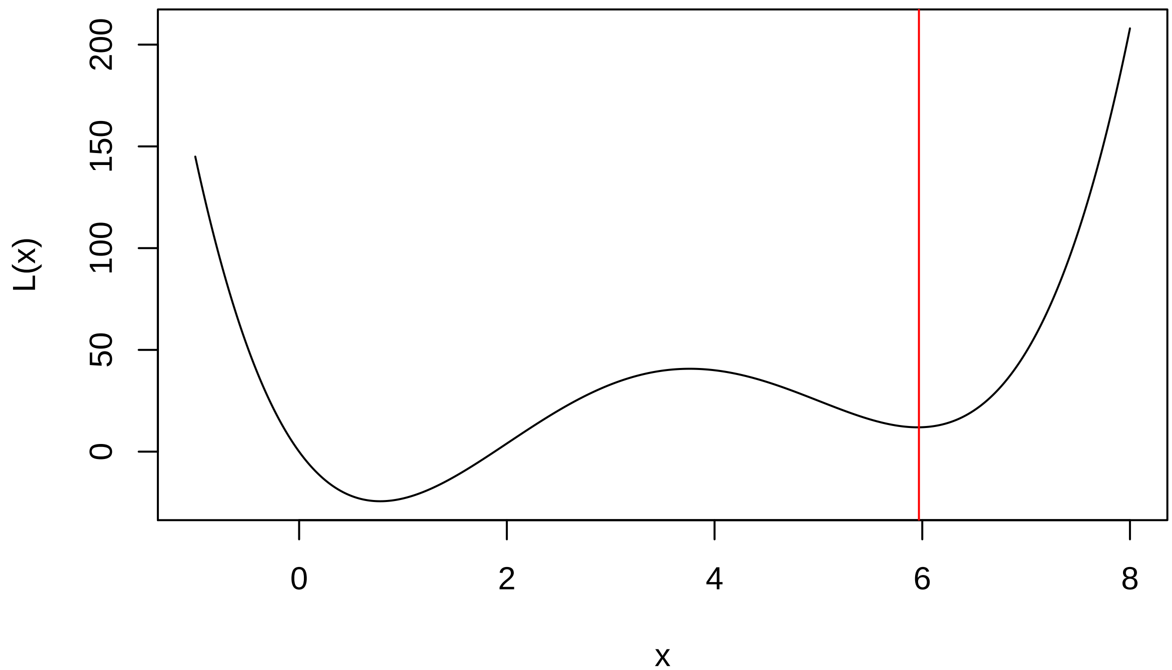


Part e

```
print(example_nm(par=c(6,8)))
```

```
## [1] 5.949562
```

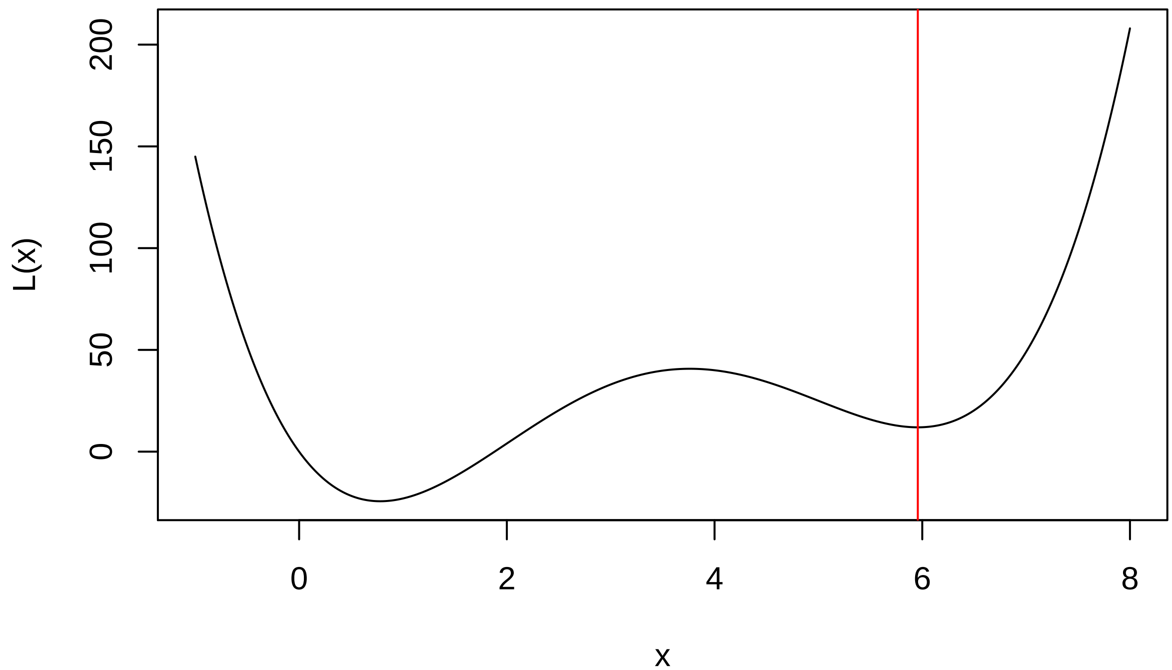
```
plot(x, L(x), type="l")
abline(v=example_nm(par=c(6,8)), col="red")
```




```r
my_loss <- function(x) {
    L(x)[1]
}
val <- optim(par = c(6, 8), fn = my_loss, method="Nelder-Mead")$par[1]
print(val)
```

```
## [1] 5.957156
```

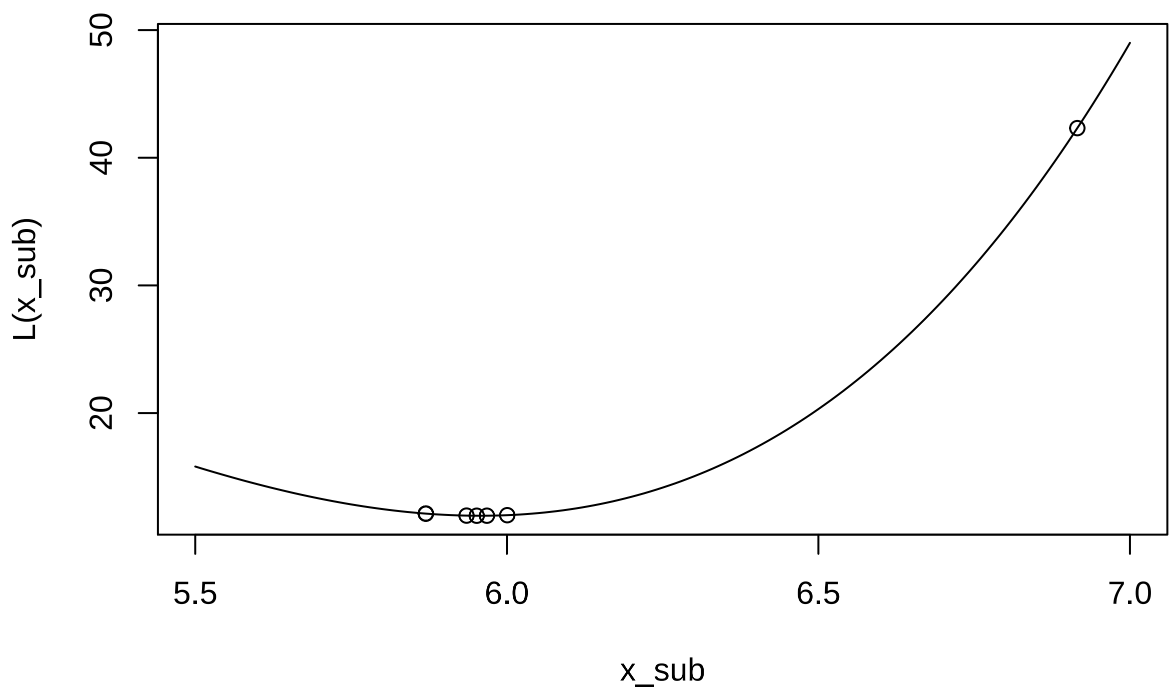
```r
plot(x, L(x), type="l")
abline(v=val, col="red")
```



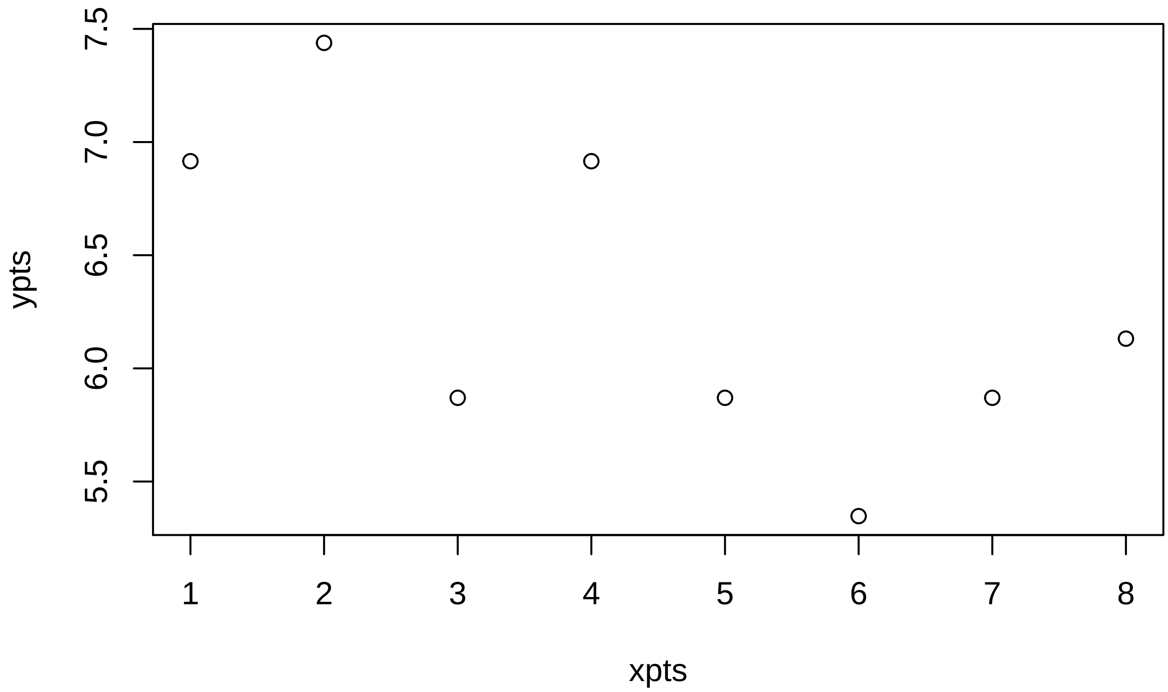

```
all_points <- example_nm(par = c(6,8), return_points = TRUE)
best_points <- all_points[,1]
print(best_points)
```

```
## [1] 6.915484 5.870002 5.870002 5.870002 6.000687 5.935344 5.968016 5.951680
```

```
x_sub <- seq(5.5, 7, 0.01)
plot(x_sub, L(x_sub), type="l")
points(best_points, L(best_points))
```




```
xpts <- rep(1:length(best_points),each=dim)
ypts <- vector(mode="numeric",length=length(xpts))
for (i in 1:length(xpts)) {
    if (i %% 2 == 0) {
        ypts[i] <- all_points[i/2,2]
    } else {
        ypts[i] <- all_points[floor(i/2)+1,1]
    }
}
plot(xpts, ypts)
```



The results are what you would expect. The algorithm got closer and closer the the minimum it was near. In this part where we started on the far right of the function we go the suboptimal minimum, because the algorithm doesn't reflect/expand far enough to overcome this locality. We can see it starts to go to the other side of the local minimum here on step 3, but the loss function increases so the points come back toward the actual minimum.

Problem 2

Problem 3