Midterm 02 - STAT440

Joseph Sepich (jps6444)

10/23/2020

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

Problem 1 Nelder-Mead

Part a

There are two steps here we need to justify. First there is an equality, then there is an approximation. The equality is simply an expansion of the Kenel Density Estimate function $\hat{f}_\sigma(x)$. Since $\frac{1}{n}$ is a constant and we can sum integrals we pull those two terms out of the integral to get that middle function.

The approximation is done via our Monte Carlo integration. The integral we have in the summation is actually in the form of an expectation $\int g(x)f(x)dx$. In our case the $g(x) = k_\sigma(x, X_i)$ and the f(x) is our true density. Since we have samples $\{X_i\}_{i=1}^n$ from the true density $f(x)$ we can therefore use these samples in our Monte Carlo integration.

In case I did not explain enough the actual integral estimation term is the $\frac{1}{n-1}\Sigma_{j\neq i}k_\sigma(X_j, X_i)$, which is merely the expected value of our KED functoin by summing up the sample points and dividing by the sample size, which is follows the expected value equation. The beginning part of the term $(\frac{1}{n}\Sigma_{i=1}^n)$ comes from the first step mentioned above expanding $\hat{f}_\sigma(x)$.

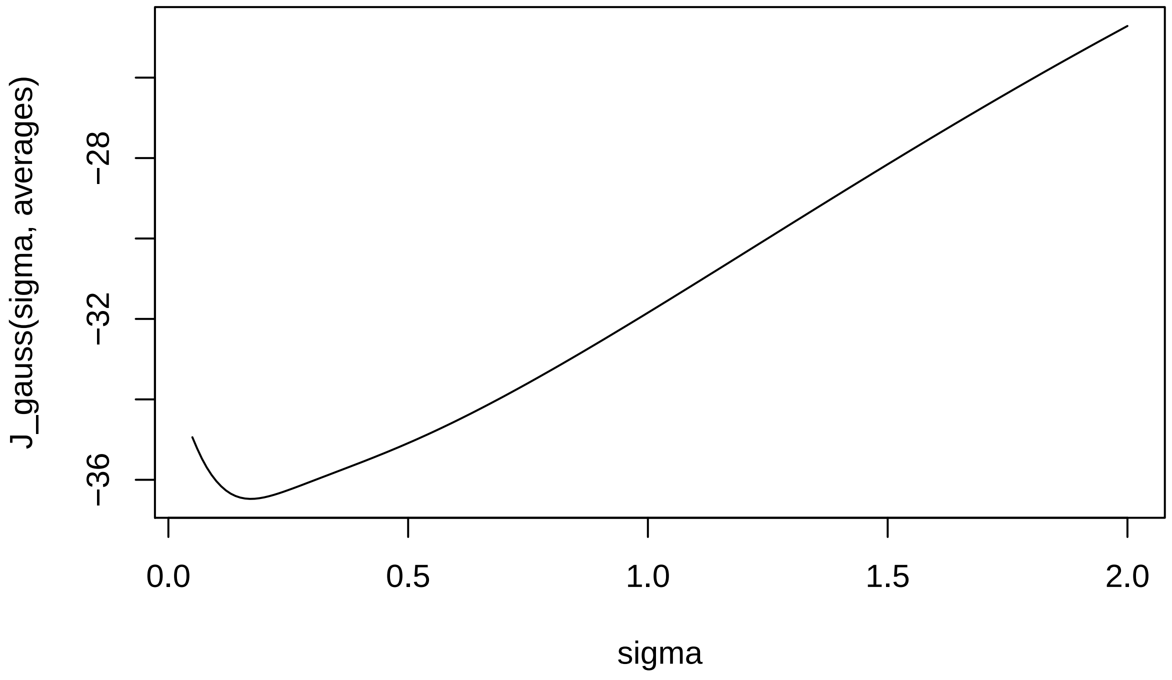
Part b

```r
# gaussian kernel
k_gauss <- function(x, x_prime, sigma=1) {
    dnorm(x, mean=x_prime, sd=sigma)
}

# error function J
J_gauss <- function(sigma, X) {
    n <- length(X)
    first_term <- 0
    for (i in 1:n) {
        for (j in 1:n) {
            first_term <- first_term + k_gauss(X[i], X[j], sqrt(2)*sigma)
        }
    }
    first_term <- first_term * (1 / n^2)
    second_term <- 0
    for (i in 1:n) {
        inner <- 0
        for (j in 1:n) {
            if (j != i) {
                second_term <- second_term + k_gauss(X[j], X[i], sigma)
            }
        }
        second_term <- second_term + (1 / (n - 1)) * inner
    }
    second_term <- second_term * (2 / n)
    return(first_term - second_term)
}
```

```
# load dataset
scores <- data.matrix(read.csv('./data/score.csv'))
averages <- rowMeans(scores) / 10
averages[0:10]
```

```
##  [1] 9.04 9.00 8.86 7.92 7.82 7.80 7.62 7.62 7.54 7.30
```

```
sigma <- seq(0.05, 2, 0.01)
plot(sigma, J_gauss(sigma, averages), type="l")
```



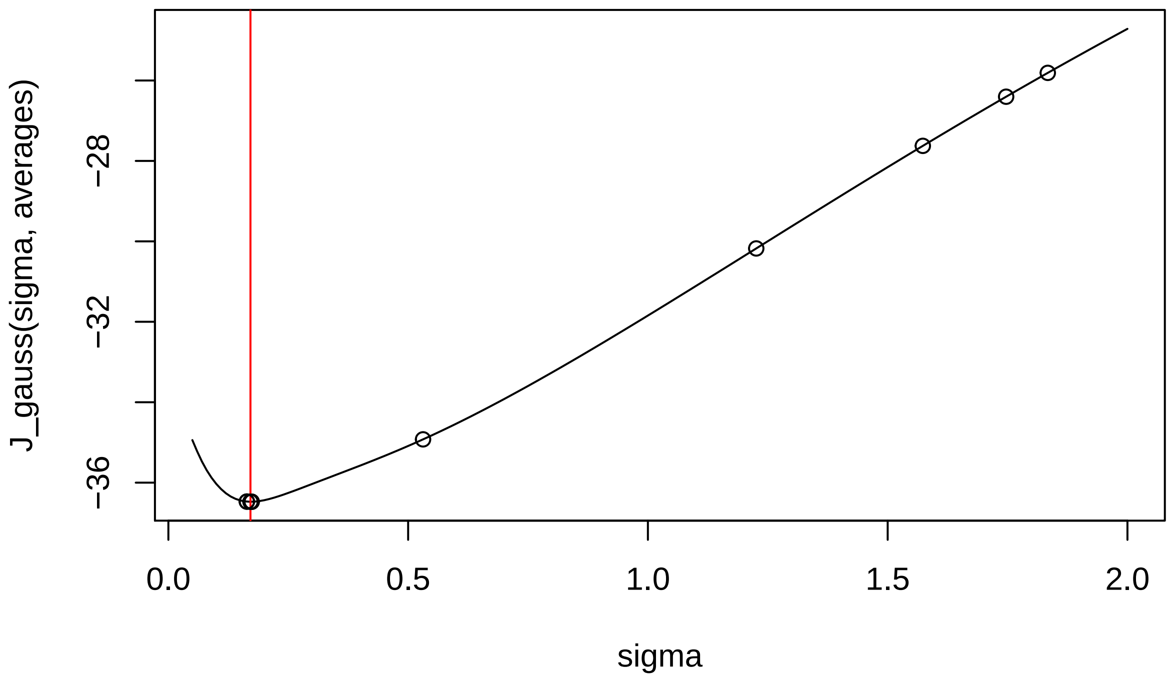
Part c

The error curve above is a convex function along this curve. You can draw a line between any two points on this interval that will always be above the line. A minimiziation algorithm should find the minimum here, since there are no local extrema to draw the algorithm away from the global minimum.

Part d

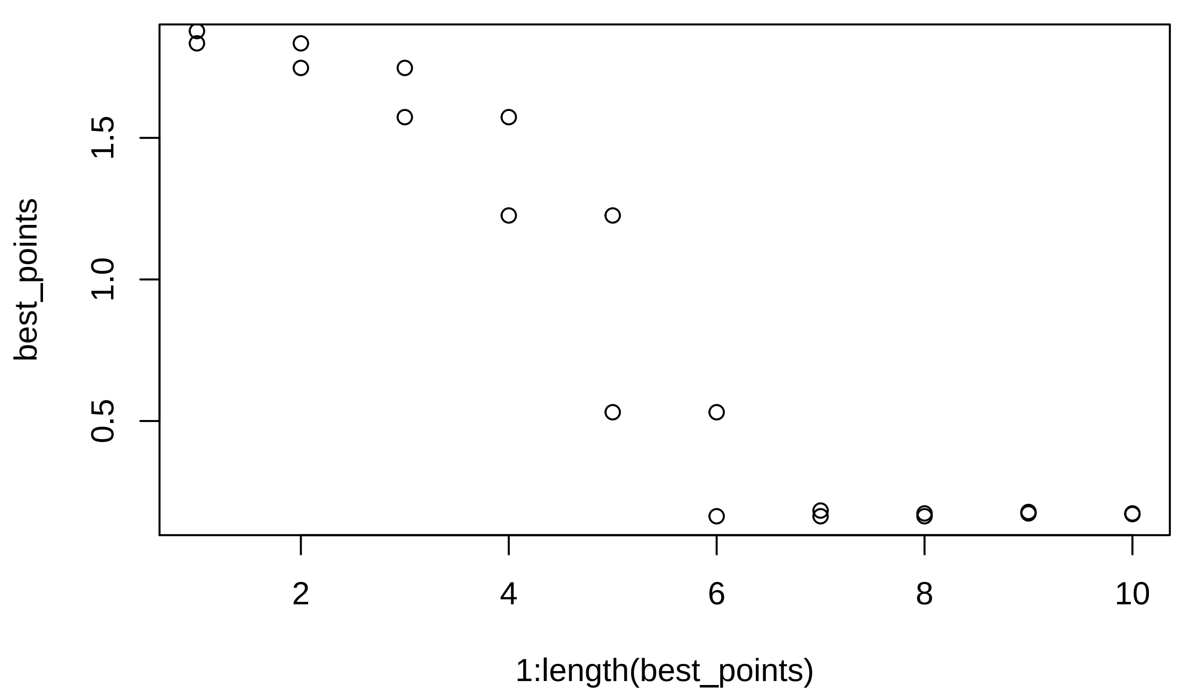
```r
my_loss <- function(sigma) {
    J_gauss(sigma, averages)
}

nm_gauss_kde <- function(par=c(0.05, 2), fn = my_loss, return_points=FALSE) {
    # parameters
    alpha <- 1
    gamma <- 2
    rho <- 0.5
    sigma <- 0.5

    # termination criteria
    max_term <- 100
    sd_threshold <- 0.001

    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(fn(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])
        if (reflected_point < 0) {
            reflected_point <- -1 * reflected_point
        }
        val_r <- fn(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 < fn(points[1])) {
            expanded <- centroid + gamma * (reflected_point - centroid)
            if (expanded < 0) {
                expanded <- -1*expanded
            }
            if (fn(expanded) < val_r) {
                points[dim] <- expanded
            } else {
                points[dim] <- reflected_point
            }
        } else {
            contracted <- centroid + rho * (reflected_point - centroid)
            if (contracted < 0) {
                contracted <- -1*contracted
            }
            if (fn(contracted) < fn(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])
}
```

```r
points <- nm_gauss_kde(return_points = TRUE)
best_points <- points[,1]
plot(sigma, J_gauss(sigma, averages), type="l")
abline(v=points[length(best_points),1], col="red")
points(best_points, J_gauss(best_points, averages))
```



Part e

```r
plot(1:length(best_points), best_points)
points(1:length(best_points), points[,2])
```



In the first portion, from steps 1 through about 5, the step is almost always a reflect and expand. During thi part the algorithm is walking down the large slope to the right of the minimum. After this the reflect and contract happens, so we do not go past 0.05, and then the algorithm pretty much converged through shrinking or some step that is too small to distinguish on this graph.

Problem 2 Permutation Test

Part a

Part b

Problem 3 Cross-Validation

Problem 4 Bootstrap and Regression

Problem 5 Extra Points

Part a

Part b

Part c