STAT 374 | HW1

Julian McClellan

April 23, 2016

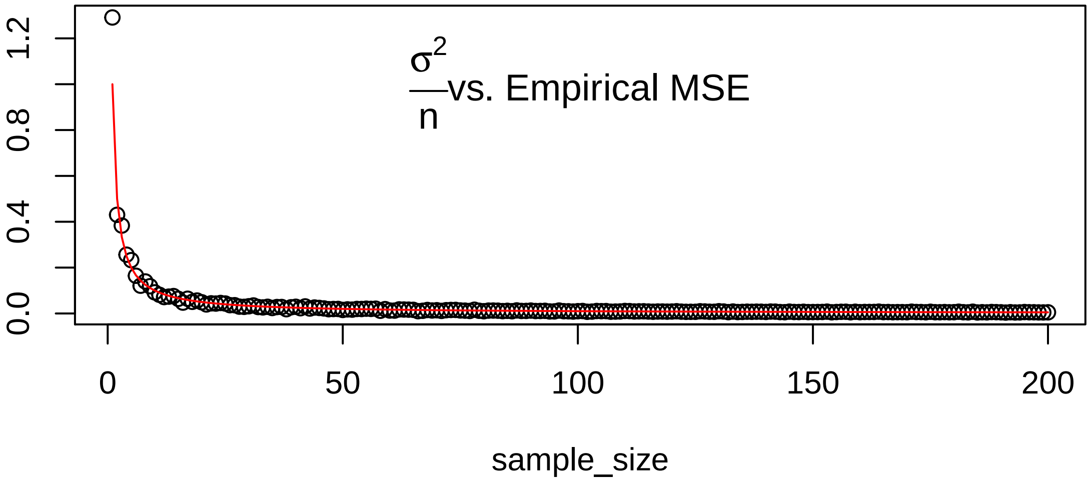
Contents

1. Computing and plotting with R(15 points)

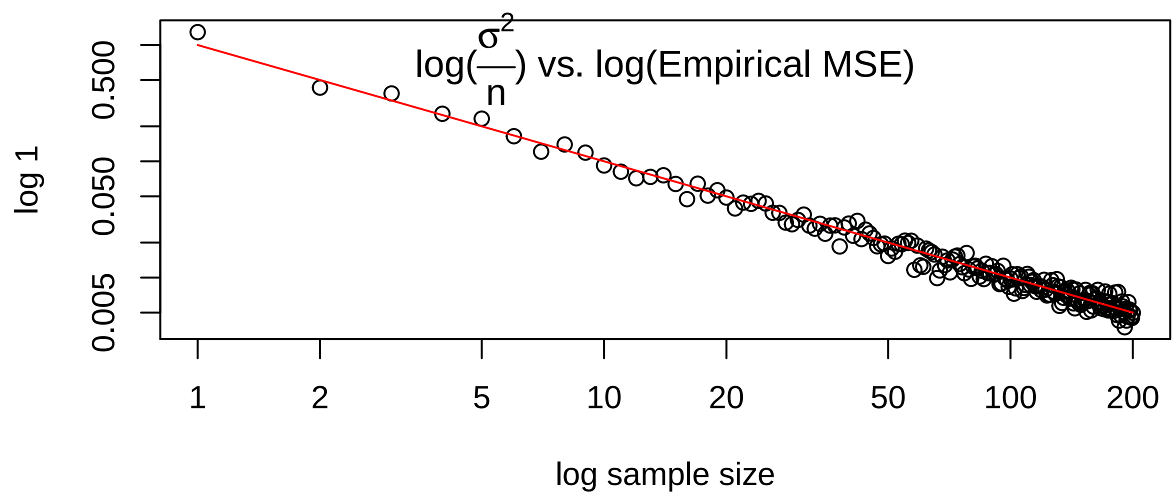
(a)

```
sample_size = c(seq(1, 200))
sim = function(part_b = FALSE, ssize = sample_size, sig = 1,
    b = 100) {
    rv = c()
    for (n in sample_size) {
        stat_n = c()

        for (rep in (1:b)) {
            mew_n = mean(rnorm(n, 1, sig))
            if (part_b) {
                stat_b = sqrt(n) * (mew_n - 1)  # Z
            } else {
                stat_b = (mew_n - 1)^2  # The MSE
            }
            stat_n = append(stat_n, stat_b)
        }

        if (part_b) {
            stat_rv = stat_n
        } else {
            stat_rv = mean(stat_n)
        }
        rv = append(rv, stat_rv)
    }
    if (part_b) {
        x = seq(-4, 4, 0.01)
        f = dnorm(x, sd = sig)
        plot(density(rv), main = "Z")
        lines(x, f, "l", col = "red")
    } else {
        # Normal scale plot
        plot(sample_size, rv, ylab = "")
        lines(sample_size, (sig^2)/sample_size, col = "red")
```

```
        title(expression(paste(frac(sigma^2, n), "vs. Empirical MSE")),
            line = -2)

        print("")

        # Log Log Plot
        plot(sample_size, log = "xy", rv, ylab = "log 1", xlab = "log sample size")
        lines(sample_size, (sig^2)/sample_size, col = "red")
        title(expression(paste("log(", frac(sigma^2, n), ") ",
            "vs. log(Empirical MSE)")), line = -1)
    }
}
sim()
```

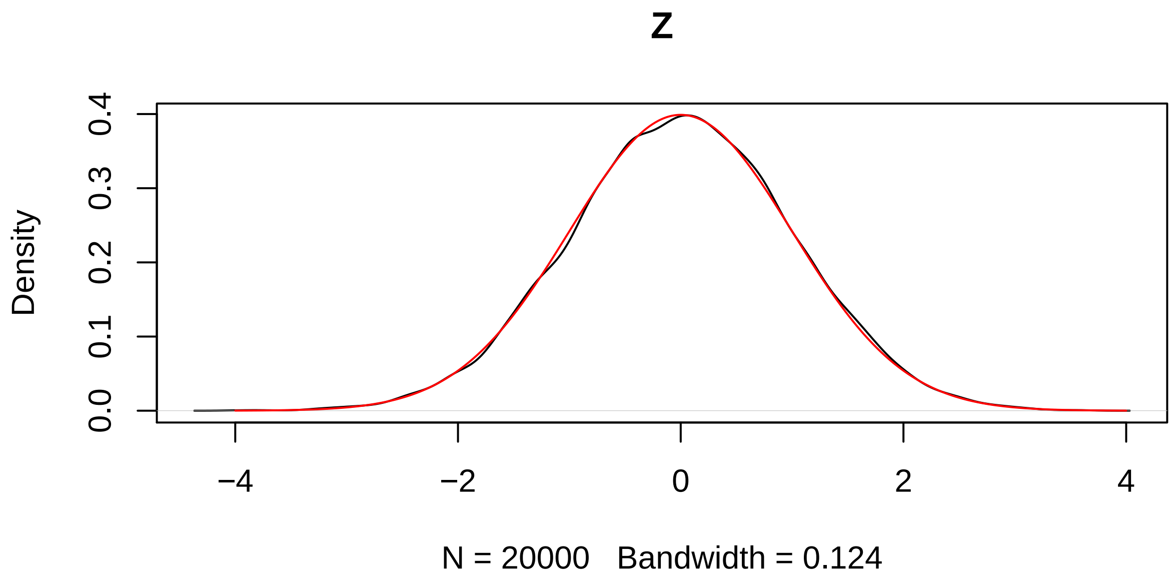



```
## [1] ""
```



(b)

```
zs = sim(TRUE)
```



2. Leave-one-out cross-validation (20 points)

(a)

$\hat{r}_n(x_i) = \sum_{j=1}^{n} L_{ij}Y_j = \sum_{j\neq i}^{n} L_{ij}Y_j + L_{ii}Y_i$ (1)

$\hat{r}_{(-i)}(x_i) = \frac{\sum_{j\neq i}^{n} L_{ij}Y_j}{\sum_{j\neq i}^{n} L_{ij}}$ (2)

where $\sum_{j\neq i}^{n} L_{ij} = 1 - L_{ii}$

$\hat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{r}_{(-i)}(x_i))^2$

from (1) and (2) $\implies \hat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \frac{\hat{r}_n(x_i) - L_{ii}Y_i}{1-L_{ii}})^2$

$\implies \hat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n}(\frac{Y_i(1-L_{ii})}{(1-L_{ii})} - \frac{\hat{r}_n(x_i) - L_{ii}Y_i}{1-L_{ii}})^2$

$\implies \hat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n}(\frac{Y_i - \hat{r}_n(x_i)}{1-L_{ii}})^2$ •

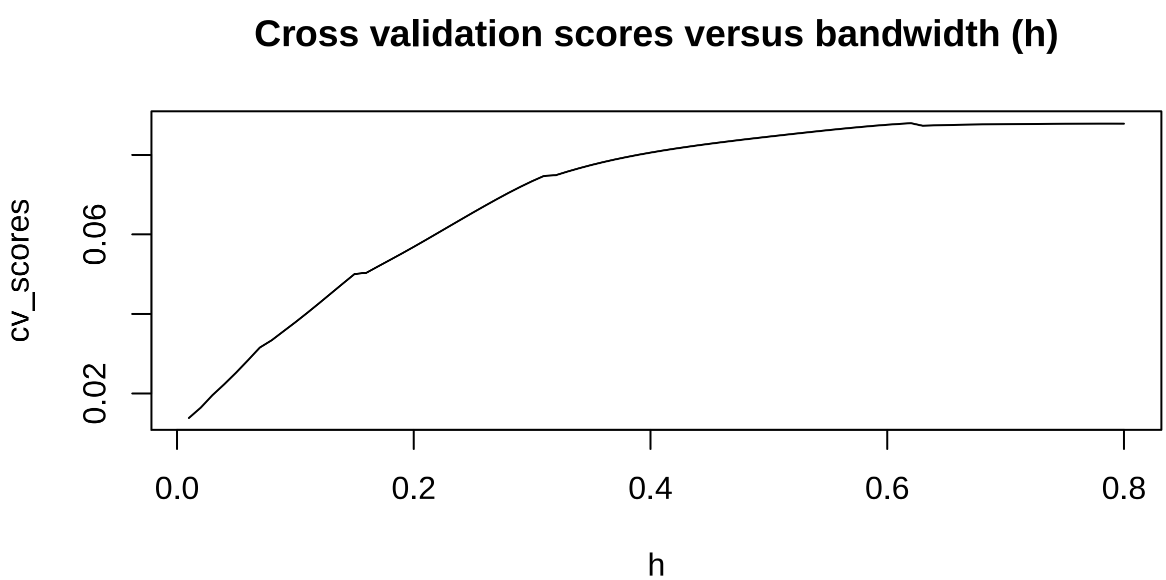
(b)

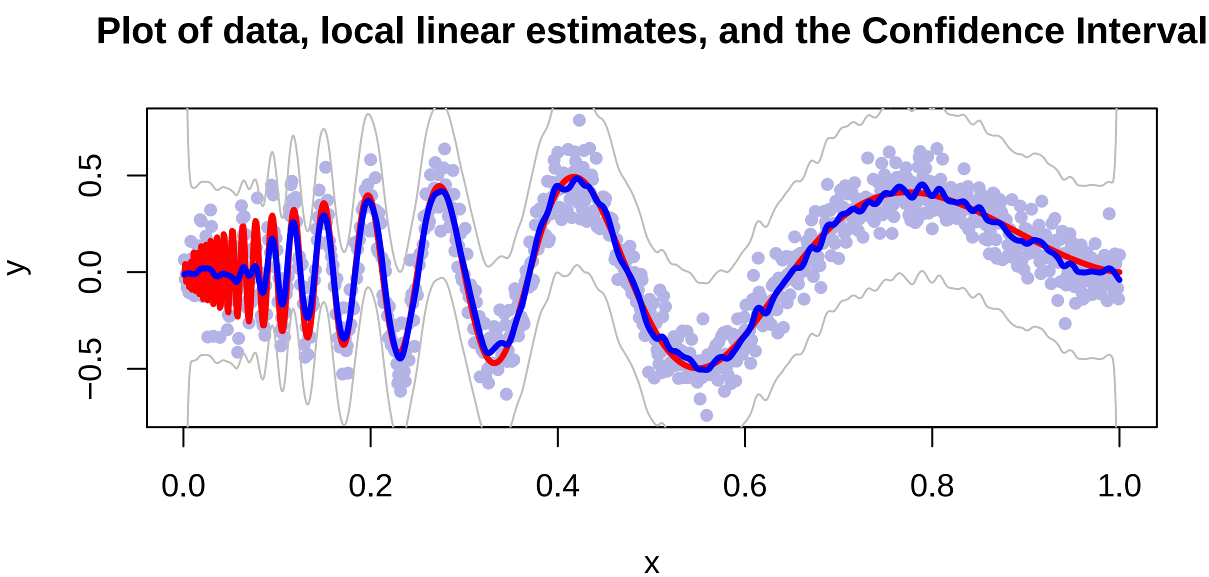
```
doppler = function(x) {
    return(sqrt(x * (1 - x)) * sin(2.1 * pi/(x + 0.05)))
}

cvs = function(h, x, y) {
    llr = locfit(y ~ x, alpha = c(0, h), deg = 1, maxk = 1e+06)

    # From 'smoothdemo.r'
    Lii = predict(llr, where = "data", what = "infl")

    return(mean(((y - fitted(llr))/(1 - Lii))^2))
}
```

```r
model = function(num_obs = 1000, sigma = 0.1) {
    # Generate data
    x = (1:num_obs)/num_obs
    y = doppler(x) + sigma * rnorm(num_obs)

    # Determine optimal h
    h = seq(0.01, 0.8, 0.01)
    alphas = cbind(rep(0, length(h)), h)
    gcvs = gcvplot(y ~ x, alpha = alphas, deg = 1)
    cv_scores = gcvs$values
    plot(h, cv_scores, type = "l", main = "Cross validation scores versus bandwidth (h)")
    hstar = h[cv_scores == min(cv_scores)]

    # Local linear regression fitted using locfit lib
    out = locfit(y ~ x, alpha = c(0, hstar), deg = 1)
    plot(x, y, pch = 16, cex = 0.9, col = rgb(0.7, 0.7, 0.9))
    lines(x, doppler(x), "l", col = "red", lwd = 3)
    lines(x, fitted(out), "l", col = "blue", lwd = 3)

    # From 'smoothdemo.r'
    normell = predict(out, where = "data", what = "vari")
    n = length(x)
    lines(x, fitted(out) + sqrt(n) * 2 * sigma * normell, "l", col = "gray", lwd = 1)
    lines(x, fitted(out) - sqrt(n) * 2 * sigma * normell, "l", col = "gray", lwd = 1)
    title("Plot of data, local linear estimates, and the Confidence Interval")

}
# Run model
library(locfit)
```

```
## locfit 1.5-9.1    2013-03-22
```

```r
model()
```



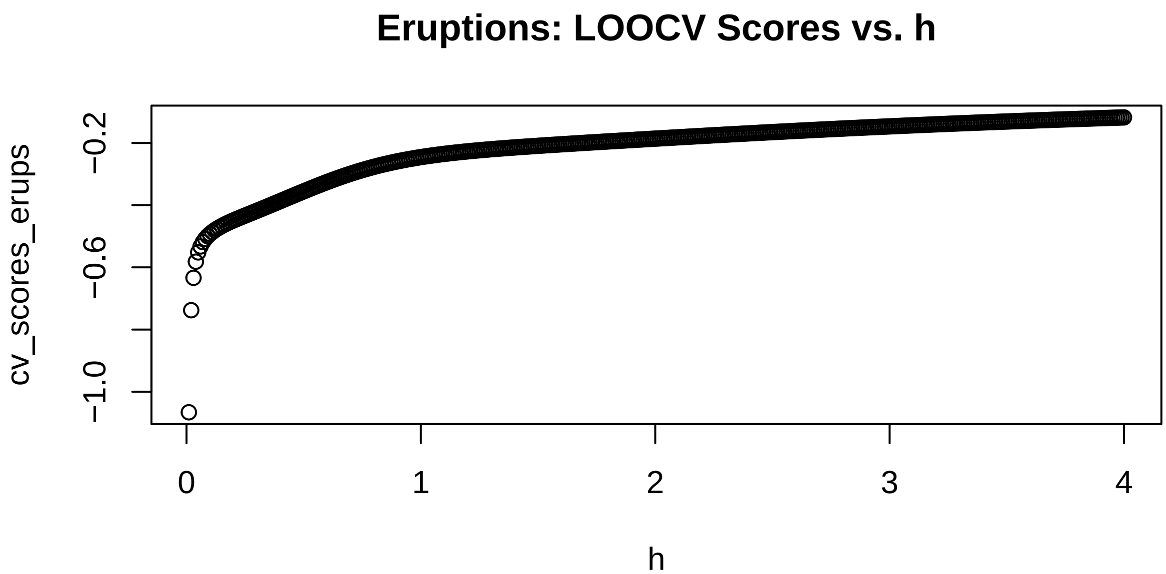


Is $I_n(x)$ the 95 percent pointwise confidence interval for $r(x)$? Sure, why not.

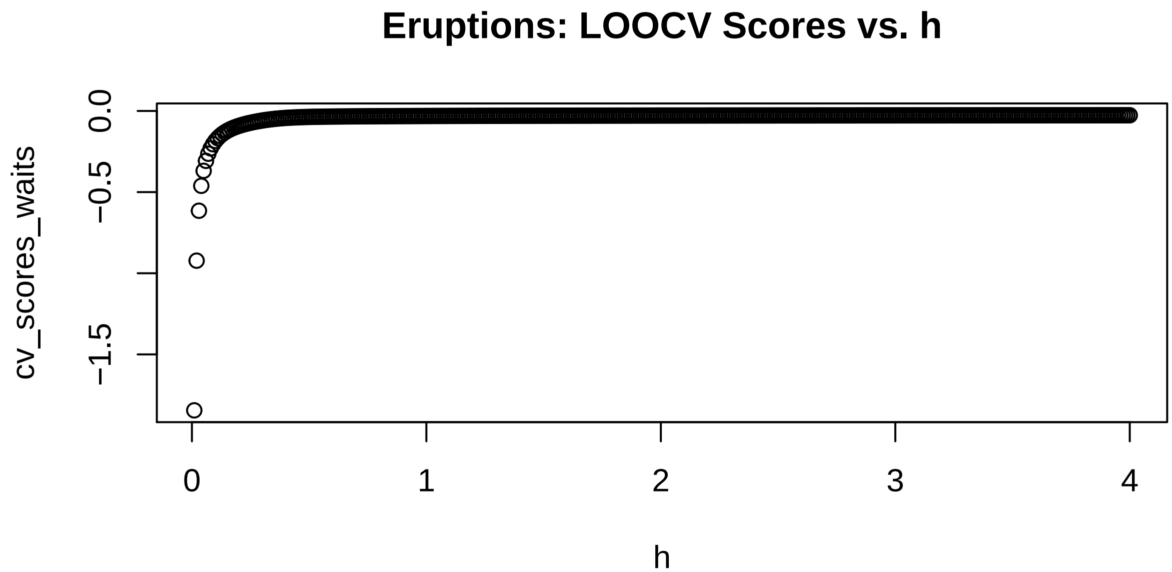
3. Kernel density estimates for Old Faithful (15 points)

```
data(faithful)

# Could have just used dnorm(x). Oh well.
gaus_kern = function(x) {
  rv = (1 / sqrt(2 * pi)) * exp(-(x ^ 2) / 2)
  return(rv)
}

kde = function(data, h, point) {
  n = length(data)
  rv = (1 / (n * h)) * sum(unlist(lapply((point - data) / h, gaus_kern)))
  return(rv)
}

kde_pts = function(data, h) {
  dom_data = seq(from = range(data)[1], to = range(data)[2], length.out = length(data) * 1.5)
  dense_pts = unlist(lapply(dom_data, kde, h = h, data = data))
  return(list(dom_data, dense_pts))
}

# From Theorem 6.35 in AoNPS. (Using a Gaussian Kernel, this is a shortcut formula for
# 6.5
cvs1 = function(data, h) {
  n = length(data)

  sum_ij = c()
  for(x in data) {
    sum_ij = append(sum_ij, sum(kstar((x - data) / h)))
  }
  sum_ij = sum(sum_ij)
```

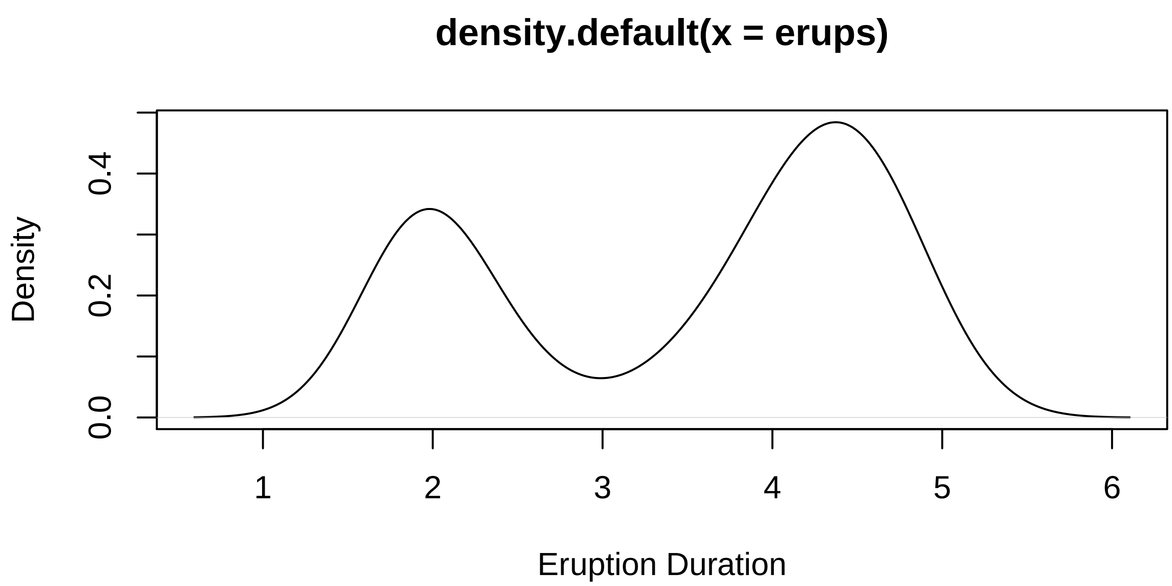
```r
  rv = (1 / (h * n^2)) * sum_ij + (2 / (n * h)) * kstar(0)
  return(rv)
}

# Also From Thoerem 6.35 AoNPS "When K is a N(0,1) Gaussian kernel. . . "
kstar = function(x) {
  return(dnorm(x, sd = sqrt(2)) - 2 * dnorm(x))
}

erups = faithful$eruptions
waits = faithful$waiting
h = seq(.01, 4, .01)

cv_scores_erups = unlist(lapply(h, cvs1, data = erups))
plot(h, cv_scores_erups, main = "Eruptions: LOOCV Scores vs. h")
```



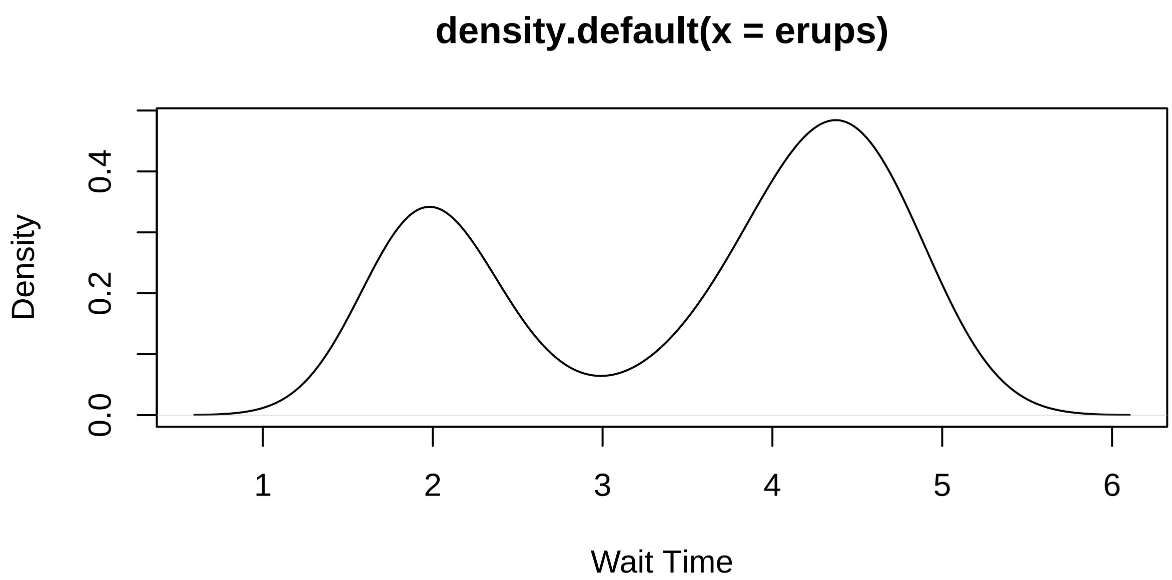

```r
hstar_erups = h[cv_scores_erups == min(cv_scores_erups)]

cv_scores_waits = unlist(lapply(h, cvs1, data = waits))
plot(h, cv_scores_waits, main = "Eruptions: LOOCV Scores vs. h")
```




```
hstar_waits = h[cv_scores_waits == min(cv_scores_waits)]

dense_erups = kde_pts(erups, hstar_erups)
plot(density(erups), type = 'l', ylab = 'Density', xlab = 'Eruption Duration')
```




```
dense_waits = kde_pts(waits, hstar_waits)
plot(density(erups), type = 'l', ylab = 'Density', xlab = 'Wait Time')
```



4. Risk of a two-dimensional Kernel density estimate (20 points)

See attached :(

5. Capital Bikeshare (30 points)

(a)

```
rmsle = function(trues, estimates) {
    diff = trues - estimates
    return(sqrt(mean(diff^2)))
}


train = read.csv("train.csv", colClasses = c(year = "factor",
    month = "factor", hour = "factor", season = "factor", holiday = "factor",
    workingday = "factor", weather = "factor"))
test = read.csv("test.csv", colClasses = c(year = "factor", month = "factor",
    hour = "factor", season = "factor", holiday = "factor", workingday = "factor",
    weather = "factor"))

train$count = log(train$count + 1)

# Split the training data into new training and new test sets
new_train = train[which(train$day <= 15), ]
new_test = train[which(16 <= train$day & train$day <= 19), ]

# Change day to factor
new_train$day = as.factor(new_train$day)
new_test$day = as.factor(new_test$day)
test$day = as.factor(test$day)

lm0 = lm(count ~ workingday + holiday + season + daylabel + hour +
```

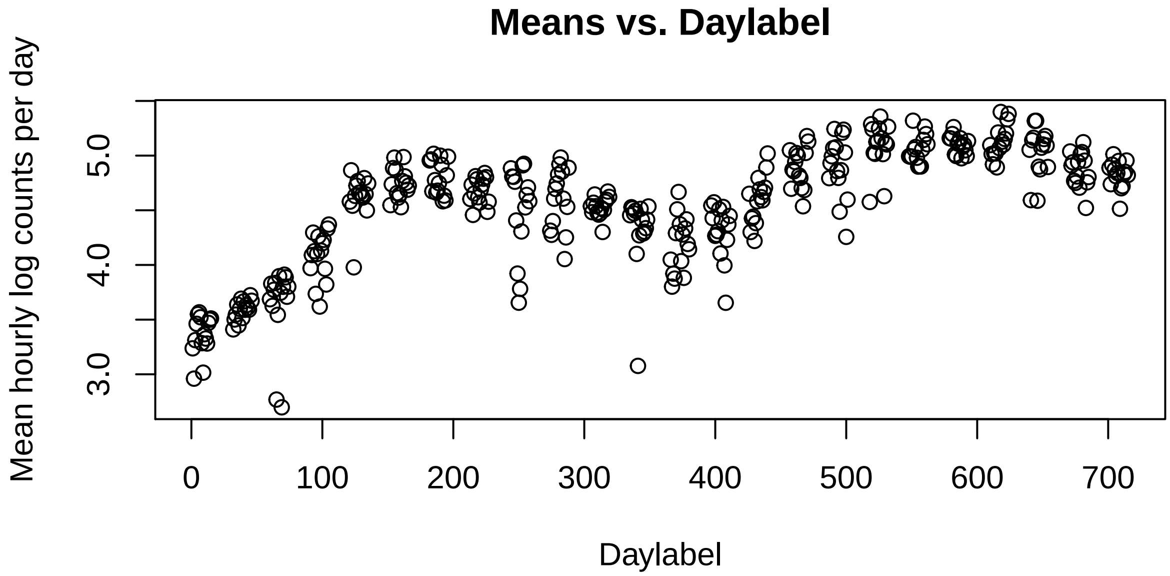
```
    weather + atemp + humidity * windspeed, data = new_train)

rmsle(new_test$count, predict(lm0, new_test))
```

```
## [1] 0.5849585
```

(b)

```
library(MASS)
library(locfit)

for (day_num in unique(new_train$daylabel)) {
    mean_count = mean(new_train$count[which(new_train$daylabel ==
        day_num)])
    new_train$mean_count[which(new_train$daylabel == day_num)] = mean_count
}

plot(unique(new_train$daylabel), unique(new_train$mean_count),
    main = "Means vs. Daylabel", ylab = "Mean hourly log counts per day",
    xlab = "Daylabel")

mcount.model = locfit(mean_count ~ daylabel, data = new_train)
new_train$resids = resid(mcount.model)

resid_model = lm(resids ~ workingday + holiday + season + daylabel +
    hour + weather + atemp + humidity * windspeed, data = new_train)

rmsle(new_test$count, predict(resid_model, new_test) + predict(mcount.model,
    new_test))
```

```
## [1] 1.353147
```



(c)

```r
library(gam)
```

```
## Loading required package: splines
## Loading required package: foreach
## Loaded gam 1.14
##
## Attaching package: 'gam'
## The following object is masked from 'package:locfit':
##
##     gam.slist
```

```r
gam.model = gam(count ~ s(daylabel) + s(atemp) + s(humidity) + s(windspeed) + weather +
    hour + season + workingday + holiday + s(temp), data = new_train)
rmsle(new_test$count, predict(gam.model, new_test))
```

```
## [1] 0.5703581
```

(d)

```r
library(gam)
gam.mcount_model = gam(mean_count ~ s(daylabel), data = new_train)
new_train$resids = resid(gam.mcount_model)
gam.resid_model = gam(resids ~ +s(atemp) + s(humidity) + s(windspeed) + weather +
    hour + season + workingday + holiday, data = new_train)

predictions = predict(gam.model, test)
predictions = exp(predictions) - 1

write.csv(predictions, file = "assn1-jmcclellan.txt", row.names = FALSE)
```