Tristan De Alwis
DSC 465: Intermediate Statistics
Assignment #2
April 16th, 2020

Q1:

$$\pi(\theta \mid x) = \frac{f(x\mid\theta)\,\pi(\theta)}{\int_{\theta=0}^{1} f(x\mid\theta)\,\pi(\theta)\,d\theta} = \frac{f(x\mid\theta)\,\pi(\theta)}{\sum_{\theta} f(x\mid\theta)\,\pi(\theta)}$$

posterior

$$f(x\mid\theta) = \binom{n}{x}\theta^{x}(1-\theta)^{n-x}$$

$$\Rightarrow \pi(\theta\mid x) = \frac{\cancel{\binom{n}{x}}\,\theta^{x}(1-\theta)^{n-x}\,\cancel{\pi(\theta)}}{\sum_{\theta}\cancel{\binom{n}{x}}\,\theta^{x}(1-\theta)^{n-x}\,\cancel{\pi(\theta)}}$$

$$\theta\left(\tfrac14, \tfrac12, \tfrac34\right) = \tfrac13$$

$$\pi(\theta\mid x) = \frac{\theta^{x}(1-\theta)^{n-x}}{\left(\frac14\right)^4\left(\frac34\right)^6 + \left(\frac12\right)^4\left(\frac12\right)^6 + \left(\frac34\right)^4\left(\frac14\right)^6}$$

$$\boxed{\pi(\theta\mid x) = \begin{cases} .397 & \text{for } \theta = \tfrac14 \\ .558 & \text{for } \theta = \tfrac12 \\ .044 & \text{for } \theta = \tfrac34 \end{cases}}$$

Assignment #2: R code

Tristan De Alwis

4/16/2020

Loading Neccessary Libraries

```
suppressPackageStartupMessages(library(MASS))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library("Rlab"))
suppressPackageStartupMessages(library(class))
```

Q2

Let lmabda be the traffic flow rate in vehicles per hour and M be the number of available toll

(c)

```
prior_m <- function(m) {
    if (m == 8) {
        return(4/7)
    } else if (m == 9) {
        return(2/7)
    } else {
        return(1/7)
    }
}

q_m <- function(m, mNew) {
    if (m == 9) {
        return(1)
    } else if (mNew == 9) {
        return(0.5)
    }
}

prior_lam <- function(lambda) {
    return(1/((lambda + 10) - max(c(0, lambda - 10))))
}

get_lam <- function(lambdaOld) {
    return(max(c(0, lambdaOld + runif(1, -10, +10), 1)))
}

q_lam <- function(lambdaOld, lambdaNew) {
    if (lambdaOld == 0) {
        return(abs(lambdaNew - 10)/20)
    } else {
        return(1/20)
    }
```

```r
}

get_m <- function(mOld) {
    if (mOld == 9) {
        return(sample(c(8, 10), 1))
    } else {
        return(9)
    }
}

xDist <- function(x, lambda, m) {
    poisson <- 1 - exp(-1 * lambda * 15 * (1/3600)/m)
    return(max(dbinom(x, m, poisson, 1e-06)))
}

acceptanceProb <- function(lambdaNew, mNew, lambdaOld, mOld, X) {
    pPrime <- xDist(X, lambdaNew, mNew) * prior_m(mNew) * prior_lam(lambdaNew)
    p <- xDist(X, lambdaOld, mOld) * prior_m(mOld)
    rNum <- pPrime * q_lam(lambdaOld, lambdaNew) * q_m(mOld, mNew)
    rDenom <- p * q_lam(lambdaNew, lambdaOld) * q_m(mNew, mOld)
    r <- rNum/rDenom

    return(min(1, r))
}
```

```r
MCMC <- function(M, x, ntrace, interval) {
    mOld <- M
    lambdaOld <- x * 240

    Ms <- c(mOld)
    Lambdas <- c(lambdaOld)

    for (i in 1:ntrace) {
        lambdaNew <- get_lam(lambdaOld)
        mNew <- get_m(mOld)

        if (acceptanceProb(lambdaNew, mNew, lambdaOld, mOld, x) >= runif(1,
            min = 0, max = 1)) {
            mOld <- mNew
            lambdaOld <- lambdaNew
        }
        if (i%%interval == 0) {
            Ms <- c(Ms, mOld)
            Lambdas <- c(Lambdas, lambdaOld)
        }
    }
    return(list(Ms = Ms, Lambdas = Lambdas))
}
```

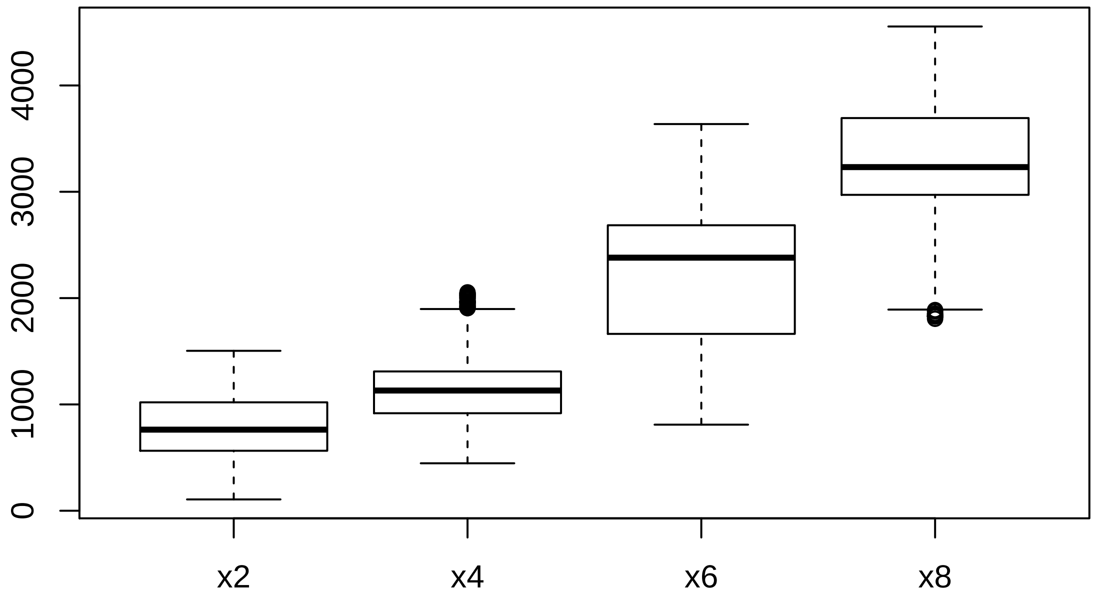
```r
# (iv)
ntrace <- 5e+06
interval = 1000
x2 <- MCMC(8, 2, ntrace, interval)
```

```r
x4 <- MCMC(8, 4, ntrace, interval)
```

```r
x6 <- MCMC(8, 6, ntrace, interval)
```

```r
x8 <- MCMC(8, 8, ntrace, interval)
```

```r
dF = c(x2[2], x4[2], x6[2], x8[2])
boxplot(dF, names = c("x2", "x4", "x6", "x8"))
```



Q3

```r
carsb = Cars93[, c(4, 5, 6, 7, 8, 12, 13, 14, 15, 17, 19:22, 25, 26)]
names(carsb)
```

```
##  [1] "Min.Price"          "Price"              "Max.Price"
##  [4] "MPG.city"           "MPG.highway"        "EngineSize"
##  [7] "Horsepower"         "RPM"                "Rev.per.mile"
## [10] "Fuel.tank.capacity" "Length"             "Wheelbase"
## [13] "Width"              "Turn.circle"        "Weight"
## [16] "Origin"
```

```r
carsb[, -16] = log(carsb[, -16])
```

(a)

```r
lrfp = function(m) {
    (m[1, 1]/(m[1, 1] + m[2, 1]))/(m[1, 2]/(m[2, 2] + m[1, 2]))
}
lrfn = function(m) {
    (m[2, 1]/(m[1, 1] + m[2, 1]))/(m[2, 2]/(m[2, 2] + m[1, 2]))
}
f0 = function(m) {
    c(1 - sum(diag(m))/sum(m), lrfp(m), lrfn(m))
}
```

(b)

```
fit.lda = lda(Origin ~ ., data = carsb)

confusion.matrix.lda = table(predict(fit.lda)$class, carsb$Origin)

m = confusion.matrix.lda

vector = f0(m)
cat("CE:  ", vector[1], "\n")
```

```
## CE:   0.1075269
```

```
cat("LR+: ", vector[2], "\n")
```

```
## LR+:  8.0625
```

```
cat("LR-: ", vector[3], "\n")
```

```
## LR-:  0.1171875
```

(c)

```
fit.qda = qda(Origin ~ ., data = carsb)

confusion.matrix.qda = table(predict(fit.qda)$class, carsb$Origin)

m = confusion.matrix.qda

vector = f0(m)
cat("CE:  ", vector[1], "\n")
```

```
## CE:   0.03225806
```

```
cat("LR+: ", vector[2], "\n")
```

```
## LR+:  43.125
```

```
cat("LR-: ", vector[3], "\n")
```

```
## LR-:  0.04261364
```

Yes, the QDA appears to produce less error

(d)

```
fit.lda = lda(Origin ~ ., data = carsb, CV = TRUE)

confusion.matrix.lda = table(fit.lda$class, carsb$Origin)

m = confusion.matrix.lda

vector = f0(m)
cat("CE:  ", vector[1], "\n")
```

```
## CE:   0.1397849
```

```r
cat("LR+: ", vector[2], "\n")
```

```
## LR+:  6.40625
```

```r
cat("LR-: ", vector[3], "\n\n")
```

```
## LR-:  0.1682692
```

```r
fit.qda = qda(Origin ~ ., data = carsb, CV = TRUE)

confusion.matrix.qda = table(fit.qda$class, carsb$Origin)

m = confusion.matrix.qda

vector = f0(m)
cat("CE:  ", vector[1], "\n")
```

```
## CE:   0.2473118
```

```r
cat("LR+: ", vector[2], "\n")
```

```
## LR+:  3.068182
```

```r
cat("LR-: ", vector[3], "\n")
```

```
## LR-:  0.3308824
```

lda() does better than qda() with CV because it is better at generalizing

Q4

```r
pima1 = rbind(Pima.tr)[, c(2, 3, 4, 5, 6, 8)]
names(pima1)
```

```
## [1] "glu"  "bp"   "skin" "bmi"  "ped"  "type"
```

(a)

```r
yes = length(which(pima1$type == "Yes"))
yes <- yes/200
cat("yes: ", yes, "\n")
```

```
## yes:  0.34
```

```r
no = length(which(pima1$type == "No"))
no <- no/200
cat("no:  ", no, "\n")
```

```
## no:   0.66
```

The classifier would be incorrect 34% of the time.

(b)

```r
knn.function = function(k.list, xtrain, gr) {

    pr.tab = matrix(NA, length(k.list), 3)
```

```r
    for (i in 1:length(k.list)) {

        knn.fit = knn.cv(xtrain, gr, k = k.list[i], use.all = TRUE)
        cm = table(knn.fit, gr)
        # print(cm)
        pr.tab[i, ] = f0(cm)
    }
    cm
    return(pr.tab)
}
```

(c)

```r
k.list = seq(1, 125, 2)
cv.tab = knn.function(k.list, pima1[, 1:5], pima1[, 6])
cv.tab
```

```
##        [,1]     [,2]      [,3]
##  [1,] 0.300 1.639118 0.3974026
##  [2,] 0.295 1.768687 0.3931419
##  [3,] 0.275 1.754735 0.3291246
##  [4,] 0.270 1.663203 0.2966024
##  [5,] 0.260 1.803030 0.2861953
##  [6,] 0.265 1.677922 0.2809917
##  [7,] 0.255 1.742424 0.2575758
##  [8,] 0.265 1.748393 0.2943723
##  [9,] 0.275 1.681818 0.3181818
## [10,] 0.265 1.645623 0.2736742
## [11,] 0.250 1.795225 0.2502165
## [12,] 0.245 1.772727 0.2272727
## [13,] 0.250 1.757576 0.2424242
## [14,] 0.250 1.757576 0.2424242
## [15,] 0.255 1.707359 0.2497704
## [16,] 0.240 1.787879 0.2121212
## [17,] 0.250 1.722078 0.2341598
## [18,] 0.260 1.692641 0.2653811
## [19,] 0.255 1.742424 0.2575758
## [20,] 0.255 1.742424 0.2575758
## [21,] 0.275 1.617003 0.3058712
## [22,] 0.260 1.692641 0.2653811
## [23,] 0.265 1.615070 0.2658847
## [24,] 0.260 1.599681 0.2404040
## [25,] 0.270 1.545455 0.2664577
## [26,] 0.265 1.615070 0.2658847
## [27,] 0.265 1.615070 0.2658847
## [28,] 0.250 1.688552 0.2253788
## [29,] 0.250 1.656839 0.2160313
## [30,] 0.255 1.613238 0.2232323
## [31,] 0.270 1.519697 0.2575758
## [32,] 0.270 1.519697 0.2575758
## [33,] 0.270 1.519697 0.2575758
## [34,] 0.275 1.506818 0.2759740
## [35,] 0.265 1.532576 0.2391775
```

```
## [36,] 0.265 1.532576 0.2391775
## [37,] 0.265 1.532576 0.2391775
## [38,] 0.260 1.571873 0.2309300
## [39,] 0.260 1.571873 0.2309300
## [40,] 0.260 1.571873 0.2309300
## [41,] 0.265 1.532576 0.2391775
## [42,] 0.265 1.532576 0.2391775
## [43,] 0.265 1.532576 0.2391775
## [44,] 0.265 1.532576 0.2391775
## [45,] 0.275 1.459596 0.2575758
## [46,] 0.270 1.495196 0.2480359
## [47,] 0.275 1.459596 0.2575758
## [48,] 0.270 1.471861 0.2377622
## [49,] 0.275 1.416667 0.2361111
## [50,] 0.270 1.449612 0.2266667
## [51,] 0.285 1.355072 0.2575758
## [52,] 0.280 1.366271 0.2341598
## [53,] 0.280 1.348162 0.2207792
## [54,] 0.280 1.348162 0.2207792
## [55,] 0.315 1.198718 0.3541667
## [56,] 0.320 1.119122 0.3090909
## [57,] 0.335 1.055230 0.4292929
## [58,] 0.340 1.007237 0.5151515
## [59,] 0.340 1.000000       NaN
## [60,] 0.340 1.000000       NaN
## [61,] 0.340 1.000000       NaN
## [62,] 0.340 1.000000       NaN
## [63,] 0.340 1.000000       NaN
```

```r
cat("Min C.E.: ", min(cv.tab[, 1]), "\n")
```

```
## Min C.E.:  0.24
```

```r
cat("Min LR+: ", min(cv.tab[, 2]), "\n")
```

```
## Min LR+:  1
```

```r
cat("Max LR+: ", max(cv.tab[, 2]), "\n")
```

```
## Max LR+:  1.80303
```

```r
cat("Min LR-: ", min(cv.tab[, 3], na.rm = TRUE), "\n")
```

```
## Min LR-:  0.2121212
```

```r
cat("Max LR-: ", max(cv.tab[, 3], na.rm = TRUE), "\n")
```

```
## Max LR-:  0.5151515
```

```r
# plot(seq(1, 125, 2), cv.tab[,1], pch=16, ylab='Classification Error',
# xlab='K', main='Normalized Features') lines(seq(1, 125, 2), cv.tab[,1],
# lty=1) abline(h = 0.34, lty = 2, lwd = 3, col = 'dark blue')
```

Using only odd numbers for K avoids the need to break ties within the classification algorithm. The min value of CE is .240 for K = 32. The minumum LR+

(d)

```
norm_feat <- apply(pima1[-6], 2, function(x) {
    return((x - mean(x))/sd(x))
})

k.list = seq(1, 125, 2)
cv.tab = knn.function(k.list, norm_feat, pima1[, 6])
cat("Min C.E.: ", min(cv.tab[, 1]), "\n")
```

```
## Min C.E.:  0.22
```

```
cat("Min LR+: ", min(cv.tab[, 2]), "\n")
```

```
## Min LR+:  1
```

```
cat("Max LR+: ", max(cv.tab[, 2]), "\n")
```

```
## Max LR+:  1.906061
```

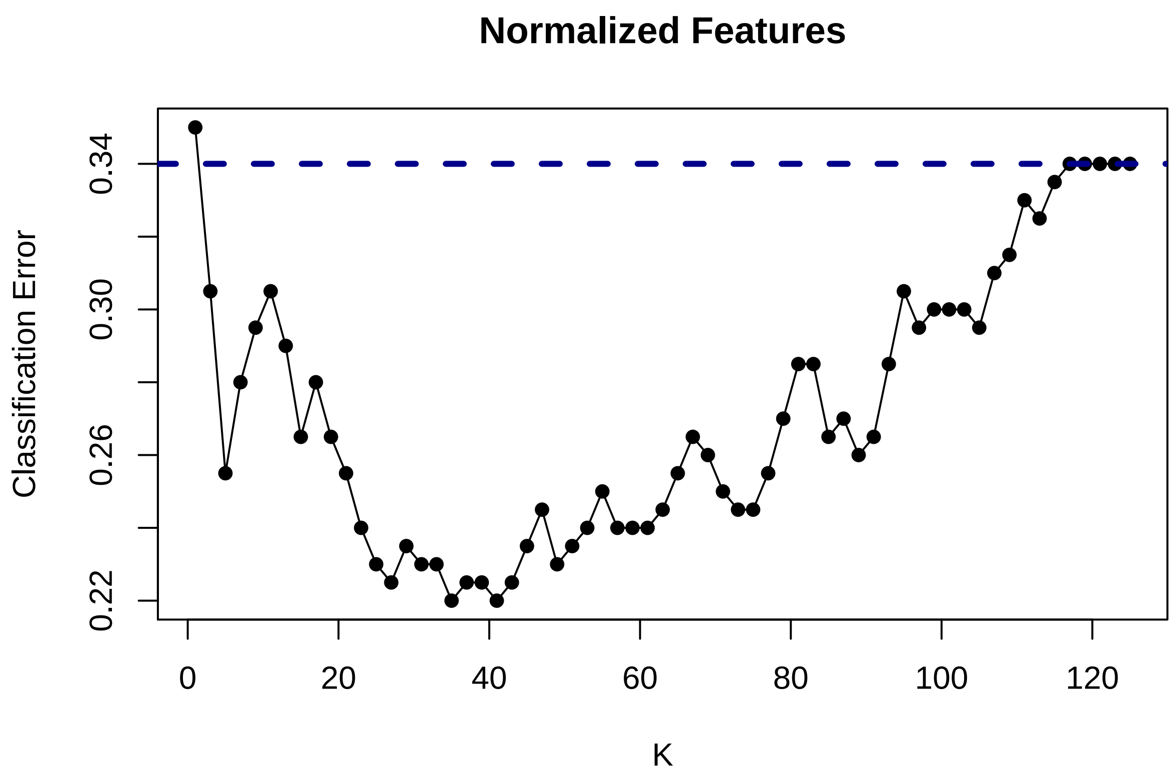
```
cat("Min LR-: ", min(cv.tab[, 3], na.rm = TRUE), "\n")
```

```
## Min LR-:  0.09366391
```

```
cat("Max LR-: ", max(cv.tab[, 3], na.rm = TRUE), "\n")
```

```
## Max LR-:  0.5454545
```

```
knn_mat <- knn.function(k.list, norm_feat, pima1[, 6])
plot(seq(1, 125, 2), knn_mat[, 1], pch = 16, ylab = "Classification Error",
    xlab = "K", main = "Normalized Features")
lines(seq(1, 125, 2), knn_mat[, 1], lty = 1)
abline(h = 0.34, lty = 2, lwd = 3, col = "dark blue")
```



###

Normalize reduces the scale of the variables resulting in reduced error.

Q5

(a)

```
lambda_bim <- function(lam, X, m = 10, t = 15/3600) {
    return(log(choose(m, X) * (1 - exp((-lam * t)/m))^(X) * (exp((-lam * t)/m)^(m -
        X))))
}

probs <- c()
ind <- seq(500, 700, 1)
for (i in ind) {
    probs <- c(probs, lambda_bim(i, X = 6))
}

plot(ind, probs)
```

Q5:

mCx

$\log(\log(\frac{M}{x}) - x)$

$0 = x \cdot \frac{T}{M} \cdot \frac{1}{e^z - 1} + (M-x)\left(\frac{-T}{M}\right)$

$(M-x)\left(\frac{\cancel{T}}{M}\right)\left(\frac{M}{x\cancel{T}}\right) = \frac{1}{e^z-1}$

$\cancel{T}\left(1 - \frac{x}{M}\right)\left(\frac{M}{\cancel{T}x}\right) = \frac{1}{e^z-1}$

$\frac{M-x}{x} \rightarrow \frac{M}{x} - 1$

$\frac{M}{T}\log\left(\frac{M}{M-x}\right)$

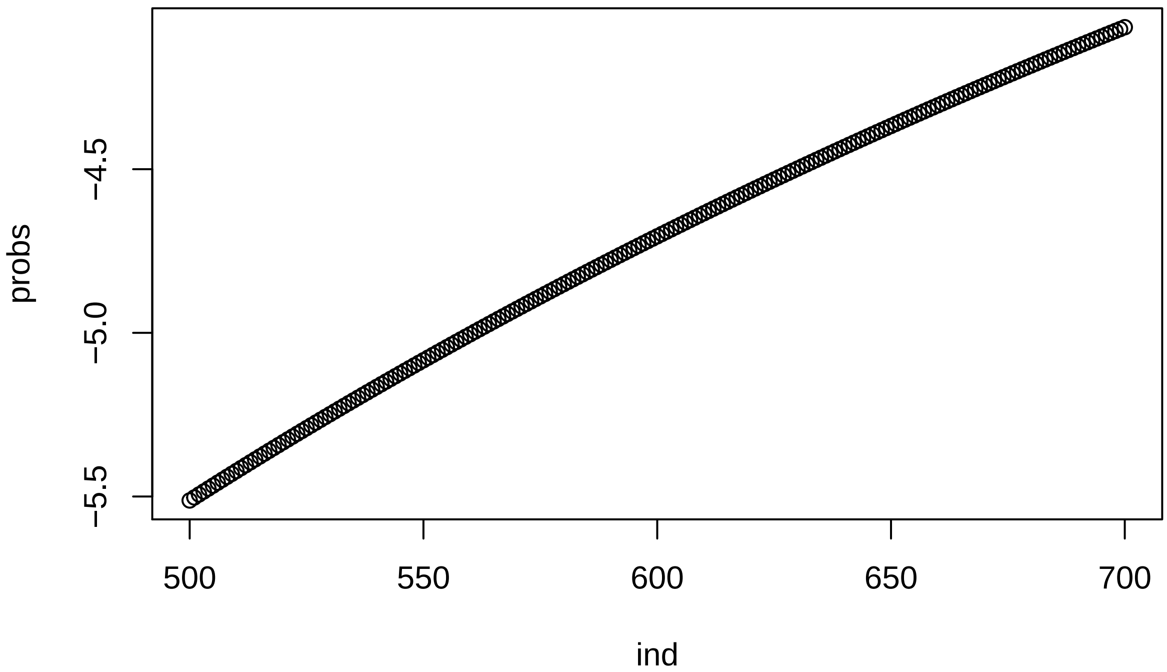
$\lambda = 0 - 5{,}226$

$5{,}526$

$\frac{M}{M-x} = \left(\frac{M}{x} - 1\right)^{-1} + 1$

$= \frac{x}{M-x} + \frac{M-x}{M-x}$

$= \frac{M}{M-x}$



(b)

```
lam_mle <- function(x, M = 10, t = (15/3600)) {
    (M/t) * log(M/(M - x))
}

for (i in 0:9) {
    sprintf("MLE for x=%s: %s", i, round(lam_mle(i), 3)) %>% print(.)
}
```

```
## [1] "MLE for x=0: 0"
## [1] "MLE for x=1: 252.865"
## [1] "MLE for x=2: 535.545"
## [1] "MLE for x=3: 856.02"
## [1] "MLE for x=4: 1225.981"
## [1] "MLE for x=5: 1663.553"
## [1] "MLE for x=6: 2199.098"
## [1] "MLE for x=7: 2889.535"
## [1] "MLE for x=8: 3862.651"
## [1] "MLE for x=9: 5526.204"
```

(c)

```
for (i in 0:10) {
    sprintf("MLE for x=%s: %s", i, round(lam_mle(i), 3)) %>% print(.)
}
```

```
## [1] "MLE for x=0: 0"
## [1] "MLE for x=1: 252.865"
## [1] "MLE for x=2: 535.545"
## [1] "MLE for x=3: 856.02"
## [1] "MLE for x=4: 1225.981"
```

```
## [1] "MLE for x=5: 1663.553"
## [1] "MLE for x=6: 2199.098"
## [1] "MLE for x=7: 2889.535"
## [1] "MLE for x=8: 3862.651"
## [1] "MLE for x=9: 5526.204"
## [1] "MLE for x=10: Inf"
```

Can't divide by 0 so can't give all estimates