CaseStudy3

Justin Glommen

2/23/2017

Setup

Setting up the dataset and the actual data realization information into accessible variables for analysis.

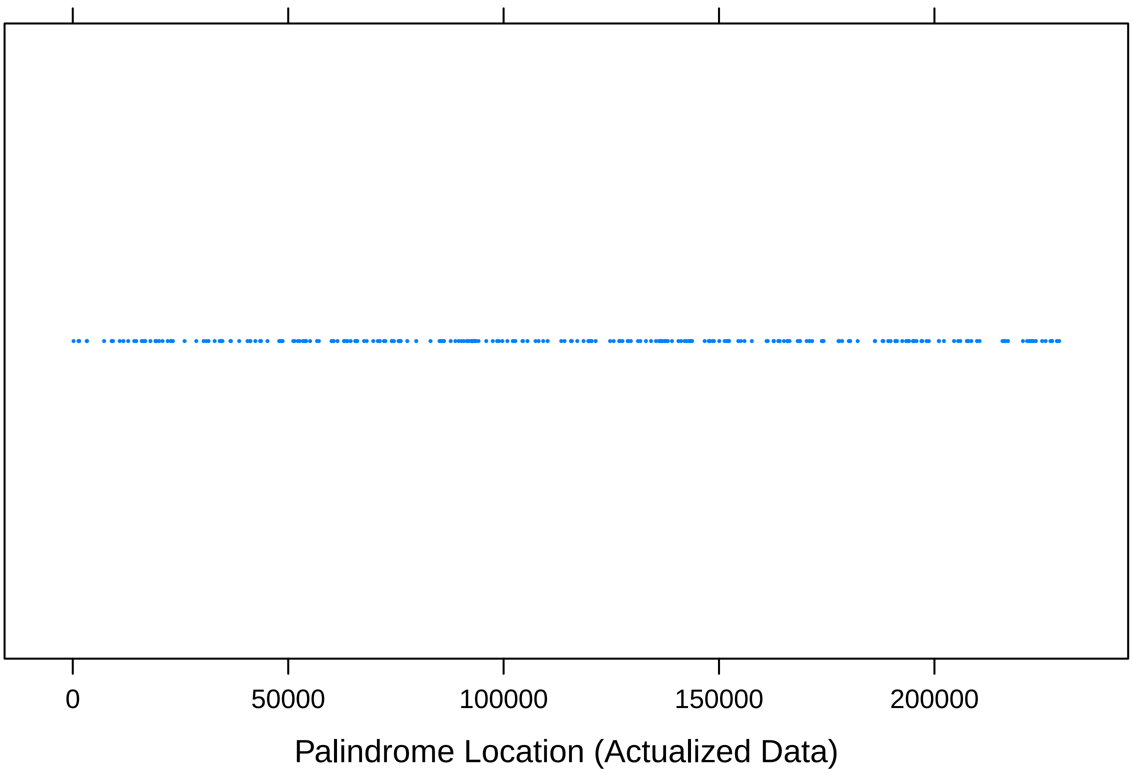
```
# Assuming the data is from the same working directory as this file
gene <- read.table("hcmv.txt", header=TRUE)
data <- gene[,1]
# Actual data
N <- 229354      # Population size
n <- 296         # Sample number of palindromes
editGene <- gene
site.random <- editGene[["location"]]
# Display the data
site.random
```

```
##   [1]    177   1321   1433   1477   3248   3255   3286   7263   9023   9084
##  [11]   9333  10884  11754  12863  14263  14719  16013  16425  16752  16812
##  [21]  18009  19176  19325  19415  20030  20832  22027  22739  22910  23241
##  [31]  25949  28665  30378  30990  31503  32923  34103  34398  34403  34723
##  [41]  36596  36707  38626  40554  41100  41222  42376  43475  43696  45188
##  [51]  47905  48279  48370  48699  51170  51461  52243  52629  53439  53678
##  [61]  54012  54037  54142  55075  56695  57123  60068  60374  60552  61441
##  [71]  62946  63003  63023  63549  63769  64502  65555  65789  65802  66015
##  [81]  67605  68221  69733  70800  71257  72220  72553  74053  74059  74541
##  [91]  75622  75775  75812  75878  76043  76124  77642  79724  83033  85130
## [101]  85513  85529  85640  86131  86137  87717  88803  89586  90251  90763
## [111]  91490  91637  91953  92526  92570  92643  92701  92709  92747  92783
## [121]  92859  93110  93250  93511  93601  94174  95975  97488  98493  98908
## [131]  99709 100864 102139 102268 102711 104363 104502 105534 107414 108123
## [141] 109185 110224 113378 114141 115627 115794 115818 117097 118555 119665
## [151] 119757 119977 120411 120432 121370 124714 125546 126815 127024 127046
## [161] 127587 128801 129057 129537 131200 131734 133040 134221 135361 136051
## [171] 136405 136578 136870 137380 137593 137695 138111 139080 140579 141201
## [181] 141994 142416 142991 143252 143549 143555 143738 146667 147612 147767
## [191] 147878 148533 148821 150056 151314 151806 152045 152222 152331 154471
## [201] 155073 155918 157617 161041 161316 162682 162703 162715 163745 163995
## [211] 164072 165071 165883 165891 165931 166372 168261 168710 168815 170345
## [221] 170988 170989 171607 173863 174049 174132 174185 174260 177727 177956
## [231] 178574 180125 180374 180435 182195 186172 186203 186210 187981 188025
## [241] 188137 189281 189810 190918 190985 190996 191298 192527 193447 193902
## [251] 194111 195032 195112 195117 195151 195221 195262 195835 196992 197022
## [261] 197191 198195 198709 201023 201056 202198 204548 205503 206000 207527
## [271] 207788 207898 208572 209876 210469 215802 216190 216292 216539 217076
## [281] 220549 221527 221949 222159 222573 222819 223001 223544 224994 225812
## [291] 226936 227238 227249 227316 228424 228953
```

The Data

Here a strip plot is shown to visualize where palindromes are distributed, by laying out all possible palindrome location sites and displaying binary values representative of palindrome occurances.

```
library(lattice)
stripplot(site.random, pch=16, cex=0.25, xlab="Palindrome Location (Actualized Data)")
```



Testing

Uniform Random Distribution by Simple Random Sampling

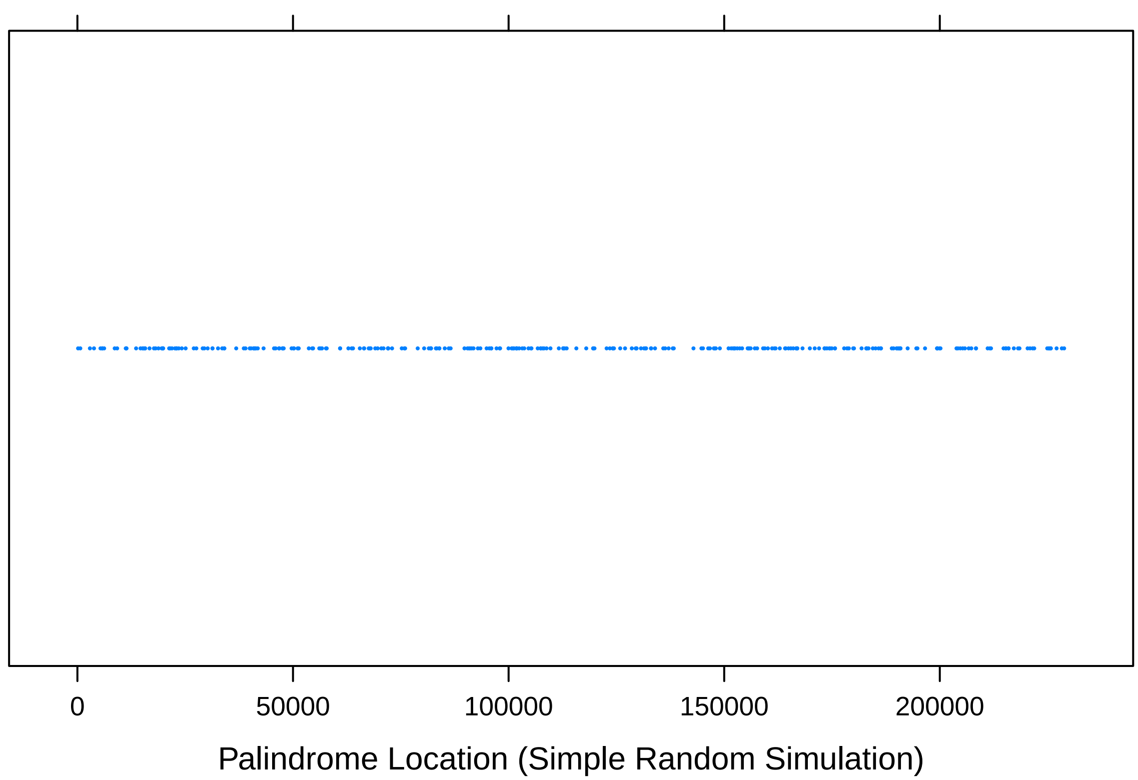
```
# Pseudo data for simulation
set.seed(22217)     # Setting seed for the date at which this analysis was performed
gene <- seq(1, N)
# Produce simple random sample
site.random <- sample.int(N, size=n)
site.random
```

```
##   [1] 174949  85198  54658  83367  72973  91865 106758  15113  19546 119591
##  [11]  96009 164114 157638  70997 220364 144722 131728 181859  53671  90823
##  [21] 130697 124433  72012 190519 183051  95948 123480 224920 204215 225734
##  [31]   6201  26994 215341 172002 131893 142875  15731 186370 206710  15320
##  [41]  89768 133009 196581 215916  97206 135873  57820  16730  21534  72088
##  [51] 175670 100715  91087 148991 129680 189277  47631 157104  70484  63601
##  [61] 105306  41189  63919 124143  67861 225315 204752   2902 208373 225741
##  [71] 208412  82017  18803  90473  86555 166688 194563    677  23050 154151
```

```
##  [81] 199346 151015 112659  67559 221899 188870  91449 136312 118009 112991
##  [91] 177821  22628  40832 174562 133071 174368 190177 107483  98033  25099
## [101] 156135 146687  43177  56412  54455  46738  29438 190864 207306 166013
## [111]  24206  23508  83949 161648 199413  15621  60911 225348 101947 153596
## [121]  38596  40275 165459  80415 133956  78921 190555 211874 108806 183468
## [131] 166963 138078 194812 111652  41423  81951 103166  62846 173352  47548
## [141] 185794 217170  56090  34055  38986  92882 147599  51325  32614  19815
## [151]  76009 107916 146298 101794  19782   5773  93454 119658 170997 179921
## [161] 152061 102443   5355 205314 164279 199427  81979 119920 169866  19893
## [171]  50119 127009  30217 128574 125884 184474  11390 113468 131415 145078
## [181]  57659 129479 152350 104617 138299 178905 166946 105186  29061 227086
## [191]  39966  66443  18149  31320 166789  75231 218477 100994 183115   5768
## [201] 214790 148054  17754  27584 205823  36830 215960 152500 153030 159426
## [211] 173815 112666  46851  49676 161107  69637  11329 221523 228302  86128
## [221] 180094 124225 220926  31336  41832  34085   8653 101249  72057 159021
## [231]  33585 160137 190886 147808  45605 162896  50239  47843 211100  60943
## [241] 192546  83934  13614  81505 151623 182934 215324 211606 228835 168165
## [251] 115714  45974  75841   9223  40997 155648 203873 186298  99977 109712
## [261]  94906  97925 103631 161921  65500 199918 129636   3849 101848  11221
## [271] 175712  22927    163  51070  95523  69072 164911 173253  68029 218164
## [281] 189971 107415  21279 185108 178498 155443 137114  14635  56735 200160
## [291]  21943  66517 122735  83140 155824 108215
```

Plotting the uniform random distribution with a strip plot, we can compare it to the actualized data above.

```r
library(lattice)
stripplot(site.random, pch=16, cex=0.25, xlab="Palindrome Location (Simple Random Simulation)")
```



It's easy to notice that while although similar, the actualized data may appear to have a couple more dense clusters than what was generated via the simple sample. We will need to perform more testing to confirm whether these apparent clusters are statistically significant within the actualized data.

Monte Carlo Uniform Simulation

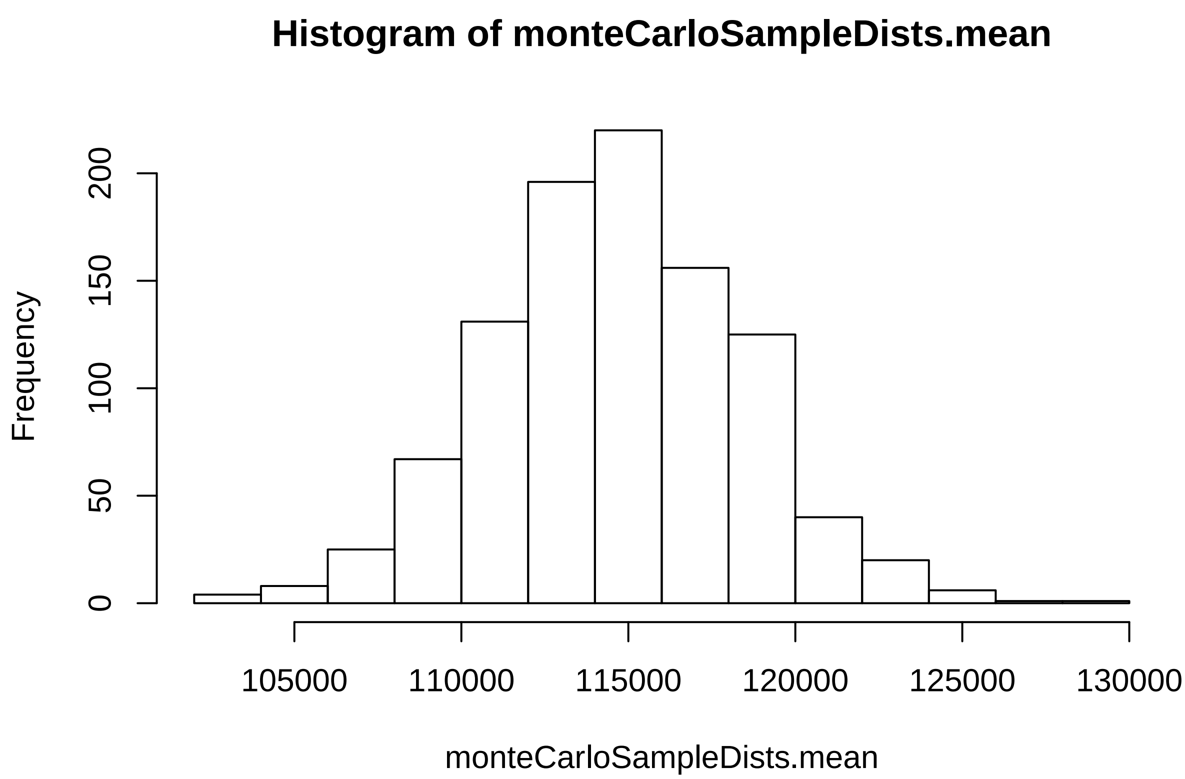
Generating the samples

Here we are generating one thousand instances of 296 simple randomly chosen uniformally distributed vriables, in order to acquire distributions of desired parameters for testing.

```
B <- 1000 # 1000 bootstrap uniform samples
monteCarloSampleDists <- matrix(data = NA, ncol = n, nrow = B)
monteCarloSampleDists.mean <- vector(mode="logical", length = B)
for (i in 1:B) {
  # Row is overall sample, column is data per sample
  monteCarloSampleDists[i,] <- sample.int(N,n)
  # Need to sort them in order to check consecutive palindromes
  monteCarloSampleDists[i,] <- monteCarloSampleDists[i,order(monteCarloSampleDists[i,])]
  monteCarloSampleDists.mean[i] <- mean(monteCarloSampleDists[i,])
}

#print(monteCarloSampleDists[1,])

hist(monteCarloSampleDists.mean)
```



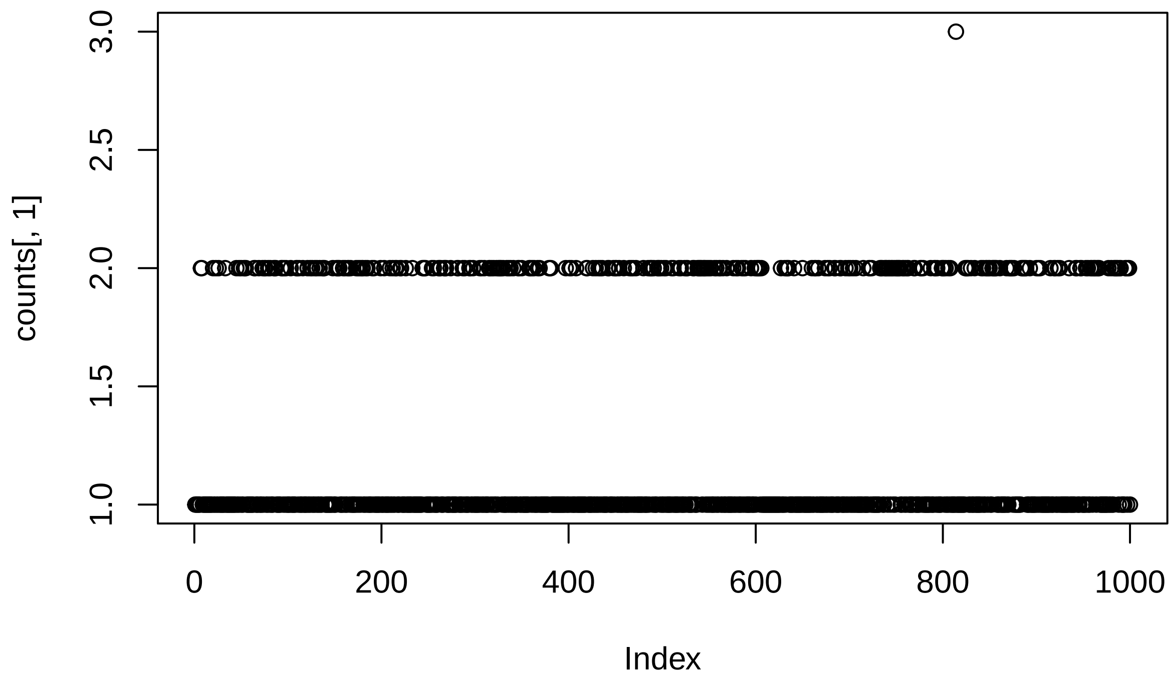
Expected Consecutive Palindrome Occurences

Now, we want to build some statistics surrounding the randomly distributed monte carlo simulation. Here we will count the longest string of consecutive palindromes using the sorted data, and more.

```r
counts <- matrix(data = NA, nrow = B, ncol = n)
for (i in 1:B) {
  indexHighestCount <- 0
  tempFirstIndex <- 0
  # 0 indicates false, 1 indicates true
  booleanIsConsecutiveNow <- 0
  # Counting the amount of consecutive palindromes
  count <- 1           # Initialized to one since we're counting backwards down below in loop
  highestCount <- 1    # Used to track the highest count overall
  # Inefficient, but stable
  # Starts from 2 to compare to last element
  for (j in 2:n) {
    #monteCarloSampleDists[i,j]
    if (monteCarloSampleDists[ i, (j - 1) ] == ((monteCarloSampleDists[i,j]) - 1)) {
      if(booleanIsConsecutiveNow == 0) {
          tempFirstIndex <- (j - 1) # Index at first palindrome in at least 2 consecutive occurances
      }
      count <- (count + 1)
      booleanIsConsecutiveNow <- 1
    }
    else {
      if (count > highestCount) {
        highestCount <- count
        indexHighestCount <- tempFirstIndex
      }
      count <- 1
      tempFirstIndex <- 0
      booleanIsConsecutiveNow <- 0
    }

  }
  # Store highest count into the counts array
  counts[i,1] <- highestCount
  # Store the index at which highestCount occured into the array also.
  counts[i,2] <- indexHighestCount
}

# Clearly, two consecutively is quite normal.
plot(counts[,1])
```




```r
summary(counts[,1])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   1.324   2.000   3.000
```

Now we have to run the same process on the actualized data, to determine its normality.

```r
indexHighestCount <- 0
tempFirstIndex <- 0
# 0 indicates false, 1 indicates true
booleanIsConsecutiveNow <- 0
# Counting the amount of consecutive palindromes
count <- 1           # Initialized to one since we're counting backwards down below in loop
highestCount <- 1    # Used to track the highest count overall
# Inefficient, but stable
# Starts from 2 to compare to last element
for (i in 2:n) {
  if (data[ (i - 1) ] == ((data[i]) - 1)) {
    if(booleanIsConsecutiveNow == 0) {
        tempFirstIndex <- (i - 1) # Index at first palindrome in at least 2 consecutive occurances
    }
    count <- (count + 1)
    booleanIsConsecutiveNow <- 1
  }
  else {
    if (count > highestCount) {
      highestCount <- count
      indexHighestCount <- tempFirstIndex
    }
    count <- 1
    tempFirstIndex <- 0
    booleanIsConsecutiveNow <- 0
  }
```

```
}
# Print out the highest count, and its index.
highestCount
```

```
## [1] 2
```

```
indexHighestCount
```

```
## [1] 221
```

Unfortunately, our realized data matches the above statistics when it comes to consecutive occurances of palindromes, since the max amount of consecutive occurances matched that of the third quartile value above. This means there were no unusually long strings of consecutively occuring palindromes, therefore we'll have to resort to the poisson process in order to help us better determine unusual clusters for the replication site.

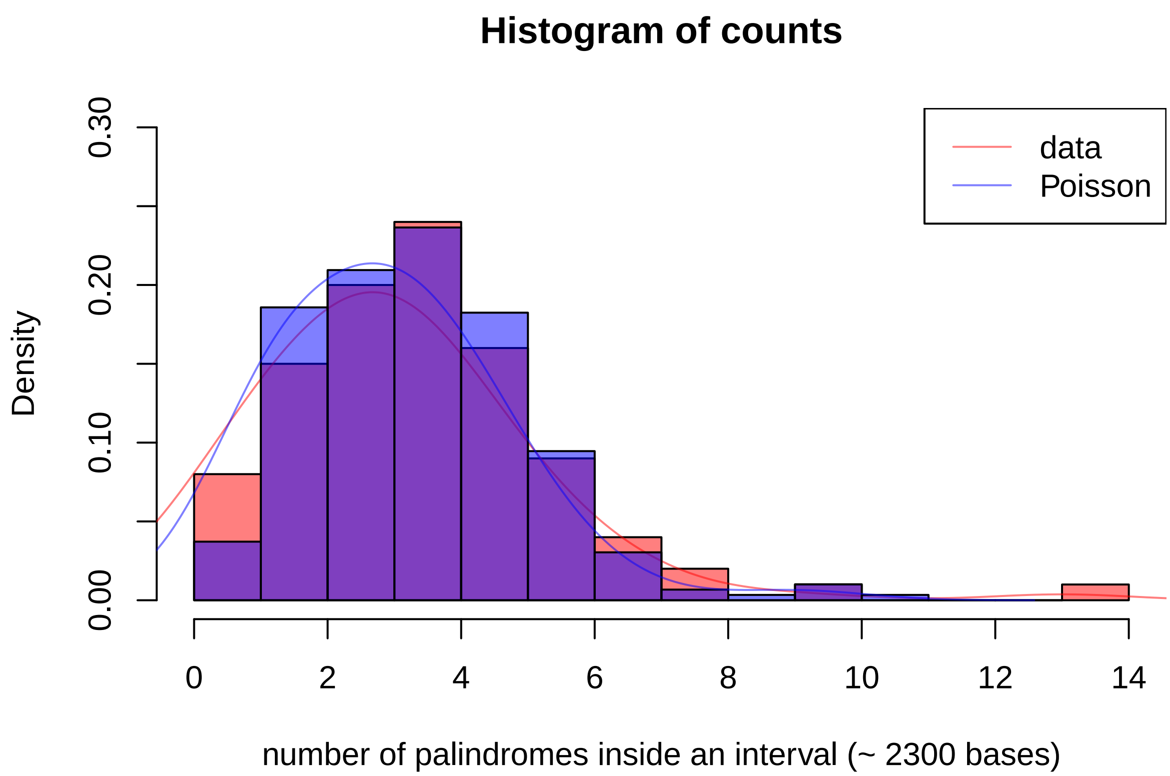
Poisson Process

First, we'll generate a histogram forming the basis of the poisson distribution of clusters.

```
k <- 100;          # The interval length for clusters of palindromes
n <- 296;
# Split the data into the clustered intervals
tab <- table(cut(data, breaks = seq(0, 230000,
    length.out = k+1), include.lowest = TRUE));

counts <- as.vector(tab);

hist(counts, breaks=seq(0,14,by=1), col = rgb(1,0,0,0.5),
     probability = TRUE,
     xlab = "number of palindromes inside an interval (~ 2300 bases)",
     ylim = c(0,0.3), include.lowest = TRUE, right = FALSE);
lines(density(counts, adjust = 2), col = rgb(1,0,0,0.5))
Pois <- rpois(296, lambda = mean(counts))
hist(Pois, breaks=seq(0,13,by=1), col = rgb(0,0,1,0.5), probability = TRUE, add = TRUE,
     include.lowest = TRUE, right = FALSE);
lines(density(Pois, adjust = 2), col = rgb(0,0,1,0.5))
legend("topright", legend = c("data", "Poisson"), lty = c(1,1), col = c(rgb(1,0,0,0.5), rgb(0,0,1,0.5)))
```



Finding the Cluster

Then, we'll calculate the highest likely amount of clusters within any one given interval, and compare it to the highest amount of palindromes we have in any one cluster in the actualized data.

```r
# Here, the 99.95th percentile is calculated to help determine upper outlier.
highestLikelyCluster <- qpois(.9995, mean(tab))   # Using sample statistic of lambda hat = x bar
print('The highest likely amount in a cluster is:')
```

```
## [1] "The highest likely amount in a cluster is:"
```

```r
highestLikelyCluster
```

```
## [1] 10
```

```r
print('and the highest amount of any cluster in our actualized data is:')
```

```
## [1] "and the highest amount of any cluster in our actualized data is:"
```

```r
max(tab)
```

```
## [1] 13
```

It's clear that with 99.95% probability, a maximum of ten palindromes will be in any one cluster. Due to our outlier of 13, and given it's the only above 10, it seems like a likely candidate for our replication site.

Chi-Squared Test

```r
regionsplit <- function(n.region, gene, site){
  count.int <- table(cut(site, breaks = seq(1, length(gene), length.out=n.region+1), include.lowest=TRU
```

```r
  count.vector <- as.vector(count.int)
  count.tab <- table(count.vector)
  return (count.tab)
}

n.region <- 50
regionsplit(n.region, gene, site.random)
```

```
## count.vector
##  2  3  4  5  6  7  8  9 10 11
##  2  2  5 18  6  6  5  3  2  1
```

```r
chisqtable <- function(n.region, site, N){
  n <- length(site)
  # lambda estimate
  lambda.est <- n/n.region
  # cut into n.region number of non-overlapping intervals
  count.int <- table(cut(site, breaks = seq(1, length(gene), length.out=n.region+1), include.lowest=TRU
  # get the count levels range
  count.vector <- as.vector(count.int)
  count.range <- max(count.vector) - min(count.vector) + 1

  # create contingency table
  table <- matrix(rep(NA, count.range*3), count.range, 3)
  for (i in 1:count.range){
    offset <- min(count.vector) - 1
    # first column = count level
    table[i, 1] <- i + offset
    # second column = observed count
    table[i, 2] <- sum(count.vector == i + offset)
    # third column = expected count
    if ((i + offset == min(count.vector)) && (min(count.vector) != 0))
      table[i, 3] <- ppois(i+offset, lambda.est)*n.region
    else if (i + offset == max(count.vector))
      table[i, 3] <- 1 - ppois(i + offset - 1, lambda.est)
    else
      table[i, 3] <- (ppois(i+offset, lambda.est) - ppois(i + offset - 1, lambda.est))*n.region
  }
  return (table)
}
site.random.tabtemp <- chisqtable(n.region, site.random, N)

site.random.tab <- matrix(rep(NA, 7*2), 7, 2)
site.random.tab[1,] <- colSums(site.random.tabtemp[1:2, 2:3])
site.random.tab[2:6,] <- site.random.tabtemp[3:7, 2:3]
site.random.tab[7,] <- colSums(site.random.tabtemp[8:10, 2:3])
site.random.stats <- sum((site.random.tab[,2] - site.random.tab[,1])^2/site.random.tab[,2])
pchisq(site.random.stats, 7 - 2, lower.tail=FALSE) #if lower.tail=TRUE then you're testing something el
```

```
## [1] 0.009900061
```