Lab/Homework 5: Finding Genes

In this lab we will find genes in the genome of *Mycoplasma genitalium*. *M. genitalium* is one of the smallest bacteria, both in actual size and in genome size. It is parasitic, a human pathogen and, as it name suggests, it is sexually transmitted and can cause diseases in both men and women. The bacteria itself is of interest to genome researchers due to its small genome size, and is considered to be a "minimal organism".

1. Why would a parasitic organism require fewer genes than a free living one?

> **Solution:** The parasitic organism may not need to make some molecules necessary for life if the host makes them and they are accessible to the parasite. Viruses are the extreme example of a parasite that can survive with as few as 10 or so genes. Since viruses live inside host cells, they can utilize the host machinery to take care of almost all their needs.

2. Which type of genes do you think are lost?

> **Solution:** The host may not need enzymes that make amino acids or other metabolites available from the host. Bacterial pathogens also don't need the proteins needed for development of a multicellular organism, *i.e.* the different cell types (*e.g.* muscle, bone, and more) and all the regulatory proteins that insure development succeeds. Virus pathogens do not need the polymerases that transcribe their genomes, nor the ribosomal proteins that translate their transcripts, nor the enzymatic complexes that modify their proteins. Membrane-enveloped viruses don't even need to make the lipid membranes that surround them, since they steal bits and pieces the host's membrane.

We will find out the number of genes in *M. genitalium* using the model we learned in class:

1. Estimate the expected number of open reading frames (ORFs) under a simple null model and use the result to infer whether the *M. genetalium* genome is unusually structured with respect to ORFs.
2. Estimate the ORF length distribution under the simple null model.
3. Use that distribution to choose a threshold for gene finding.

Steps:

3. Get the genome file of *M. genitalium* from `/ptmp/bcbio444/genefinding/NC_000908.gb` (use the `cp` command). Use BioPython to convert from Genbank format to FASTA

format. The BioPython library becomes available after you issue the command '`module load python`'. Attach the code you used and record the command you issued.

Solution: I wrote the Python code based on the BioPython manual. The command I used is

```
python gb2fsa.py NC_000908.gb NC_000908.fsa
```

The code is:

```
from Bio import SeqIO
import sys

script_name = sys.argv[0]

if len(sys.argv) != 3:
   print "Incorrect command-line usage.  Correct usage:\n",\
      "python", script_name, "gb_filename fsa_filename\n",\
      "gb_filename\tName of Genbank-formatted input file\n",\
      "fsa_filename\tName of Fasta-formatted file to write\n"
   sys.exit(1)

gb_file = sys.argv[1]
fsa_file = sys.argv[2]

print "Reading", gb_file, "and writing to", fsa_file
SeqIO.convert(gb_file, "genbank", fsa_file, "fasta")
```

4. Use Python to determine the GC content of the *M. genitalium* genome. The human genome GC content is 41%. Is *M. genitalium* likely to have more or less ORFs than human? Longer or shorter ORFs?

Solution: I should reuse previous HW solutions, but I write the following code:

```
from __future__ import print_function

import sys

# handle errors in command-line usage
if len(sys.argv) != 2:
   print("Invalid usage: python " + sys.argv[0] + " fasta_infile",
```

```
      file = sys.stderr)
   sys.exit(1) # convention: exit with non-zero to indicate error

# hard-coded paths and in/out files
fasta_infile = sys.argv[1]

# open in/out files
file_in = open(fasta_infile)

seq_len = 0 # length of current sequence
gc_count = 0   # number of G+C in current sequence

for line in file_in:
   if line.startswith(">"):
      seq_name = line.strip(">")
      seq_name = seq_name.strip()
      if seq_len:
         print(str(float(gc_count) / seq_len) + '\n')
      print(seq_name + '\t')
      seq_len = 0
      gc_count = 0
   else:
      # remove newline and convert to upper case
      seq = line.strip().upper()
      # count all non-gaps
      seq_len += len(seq) - seq.count("-")
      gc_count += seq.count("C") + seq.count("G")

# output last GC content
if seq_len:
   print(str(float(gc_count) / seq_len) + '\n')

print("WARNING: this program does not properly handle IUPAC symbols "
   "other than A, C, G, T, and -!")

file_in.close()
```

I estimate the GC content to be approximately 0.316891. Since stop and start codons are at least 2/3 A/T, one expects more chance occurrences of start and stop codons in AT-rich organisms. As a result, we expect more and shorter ORFs than in human. Of course, natural selection will remove excess start and stop codons, but if humans and this bacterium were subject to the same level of selection, we would overall

expect shorter ORFs in the bacterium.

5. Since we want to perform inference (find the genes) in this genome, the population to which we want to extend our conclusions is the *M. genitalium* genome. We need a model that generates *M. genitalium*-like ORFs. We could use the iid Multinoulli model, but there is another model particularly useful for this case where we have no need to generalize to other genomes. The permutation model can generate whole genomes along with the ORFs in them by randomly shuffling the *M. genitalium* nucleotides. This model generates not quite iid nucleotides, because the probabilities for the next nucleotide depend on which nucleotides have already been placed. Write a function that randomly permutes the *M. genitalium* genome, produce 100 permuted genomes, and output them in FASTA format. Attach the code you used to perform this task, but do not attach the permuted data! Do you expect the ORFs to be longer or shorter in the permuted genomes? Why?

Solution: The code I wrote is show below. It produces 100 separate FASTA files with shuffled genomes.

```
import random, sys
from Bio import SeqIO

if len(sys.argv) < 2:
   print "Usage: python", sys.argv[0], "fasta_file [times] [output_root]"
   print "\tfasta_file\tShuffle sequences in this FASTA-formatted file"
   print "\ttimes\t(optional) Number of shuffled sequences to output"
   print "\toutput_root\t(optional) Sequences will be output one per file"\
      "with filename given by output_root{index}.shuff.fsa"
   sys.exit(1)

# assumes one sequence in input fasta file
seq_obj = SeqIO.parse(sys.argv[1], "fasta").next()
seq_str = str(seq_obj.seq)

times = 100
output_root = sys.argv[1][0:sys.argv[1].index(".")]
if len(sys.argv) >= 3:
   times = int(sys.argv[2])
if len(sys.argv) >= 4:
   output_root = sys.argv[3]

for i in range(times):
   outfilename = output_root + str(i) + ".shuff.fsa"
```

```
    outfile = open(outfilename, "w")
    outfile.write(">Shuffle" + str(i) + "\n")
    outfile.write("".join(random.sample(list(seq_str), k = len(seq_str))))
    outfile.write("\n")
    outfile.close()
```

I expect the ORFs to be shorter in the permuted genomes because there is no selection pressure acting to suppress stop codons in the middle of functional, protein- encoding ORFs.

6. Now we need to find and translate the ORFs. Mycoplasma are weird: they use a slightly different codon table than most other organisms. UGA is commonly used as a stop codon. In mycoplasma, it codes for the amino-acid Tryptophan. This is a serious headache for anyone doing genetics in mycoplasma, but all we have to worry about is using the correct translation table. The translation table you should use is #4 here: `https://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi`.

   Lab: You almost have the code to write this yourself in Python, but others have already done the job. To speed up the lab, you will use the translation program `getorf`, part of the EMBOSS suite of software, which is available on `hpc-class` after you issue the command '`module load emboss`'. The command will be something like:

   ```
   $ getorf -table 4 -find ??
   ```

   What should you put after `-find`? Type '`tfm getorf`' to find out. Be sure to use the mandatory arguments `-sequence` and `-outseq` appropriately.

Solution: The following will output all translated ORFs to a FASTA file.

```
getorf -table 4 -find 1 -sequence NC_000908.fsa -outseq NC_000908.orf.fsa
```

Homework bonus: Write python code to find and output all the ORFs in FASTA format. Please beware that an ORF may be in any of the six frames: three frames on each strand of the genome. Attach your code for credit.

Solution: I write the following code to count the number of ORFs or to extract the length of ORFs (used later) from FASTA files.

```
from __future__ import print_function
import random, sys
```

```
from Bio import SeqIO

# usage statement to help user
if len(sys.argv) < 2:
   print("Usage: python", sys.argv[0], "fasta_files [count|length]",
      file = sys.stderr)
   print("\tfasta_files\tFASTA files to analyze", file = sys.stderr)
   print("\tcount|length\t(optional) Tell program to count ORFs or record"\
      " lengths", file = sys.stderr)
   sys.exit(1)

# warning about which ORFs we are counting
print("NOTE: While other start codons can be used by Mycoplasma species, "\
   "AUG is by far the most common "\
   "[https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4444185/], "\
   "so this program only considers AUG start codons.", file = sys.stderr)

# process command line
stat = "count" # default: count ORFs
fasta_files = sys.argv[1:]
if fasta_files[-1] == "count" or fasta_files[-1] == "length":
   stat = fasta_files[-1]
   del fasta_files[-1]

##
# Function determines whether a codon is a stop codon (in Mycoplasma)
#
# @param codon the codon to test
# @return   True/False
def is_stop(codon):
   if codon == "TAA":
      return True
   elif codon == "TAG":
      return True
   return False

# iterate over a potentially long list of FASTA files
for fasta_file in fasta_files:

   # assumes one sequence in input fasta file
   seq_obj = SeqIO.parse(fasta_file, "fasta").next()

   # get reverse complement
   seq_rc_str = str(seq_obj.reverse_complement().seq)
   seq_str = str(seq_obj.seq)
```

```
cnt = 0
rc_str = "forward"

# for both forward and reverse strand
for seq in [seq_str, seq_rc_str]:

   # consider three frames
   for frame in range(3):
      in_frame = False
      for i in range(frame, len(seq) - frame, 3):
         if in_frame and is_stop(seq[i:i+3]):
            cnt += 1
            in_frame = False

            # output sequence, strand, length
            if stat == "length":
               print(seq_obj.id + ", "\
                  + rc_str + ", "\
                  + str((i - start)/3))
         elif seq[i:i+3] == "ATG":
            in_frame = True
            start = i
   rc_str = "reverse"

# output sequence, ORF count
if stat == "count":
   print(seq_obj.id + ", " + str(cnt))
```

7. The fundamental assumption of the permutation model is that the order of the nucleotides is random. Since ORFs are determined by the order of nucleotides, it seems reasonable to detect biologically functional ORFs as somehow unusual relative to ORFs generated by the random order (permutation) model. How could you critique the assumptions of the permutation model?

Solution: The permutation model very nearly assumes nucleotides are independent. Certainly, it does not model local dependencies common in functional sequences, and not all of these functional dependencies are caused by functional ORFs. A genome without functional ORFs may still have other dependencies that lengthen or shorten the "accidental" ORFs.

The permutation model also assumes the same marginal distribution of nucleotides at every site. In other words, sites are identically distributed. We know, of course, that nucleotide content changes along the genome, partially because of ORFs, but

for other reasons as well. This variation in nucleotide usage may also lead to longer or shorter "accidental" ORFs.

We will use the number of ORFs as a test statistic. Generally speaking, what values of this test statistic reject the null model (hypothesis) of random order?

Solution: Since selection acts to suppress stop codons within ORFs (to avoid breaking functional proteins) and start codons outside of ORFs (to avoid starting translation too early), we expect fewer ORFs in real sequences.

Count the number of ORFs in each of the 100 shuffled genomes.

Solution: After running `python shuffle_fsa.py NC_000908.fsa 100` to create the 100 FASTA files, we run the command

```
python orf_stats.py NC_000908*.shuff.fsa > orf_cnts.csv
```

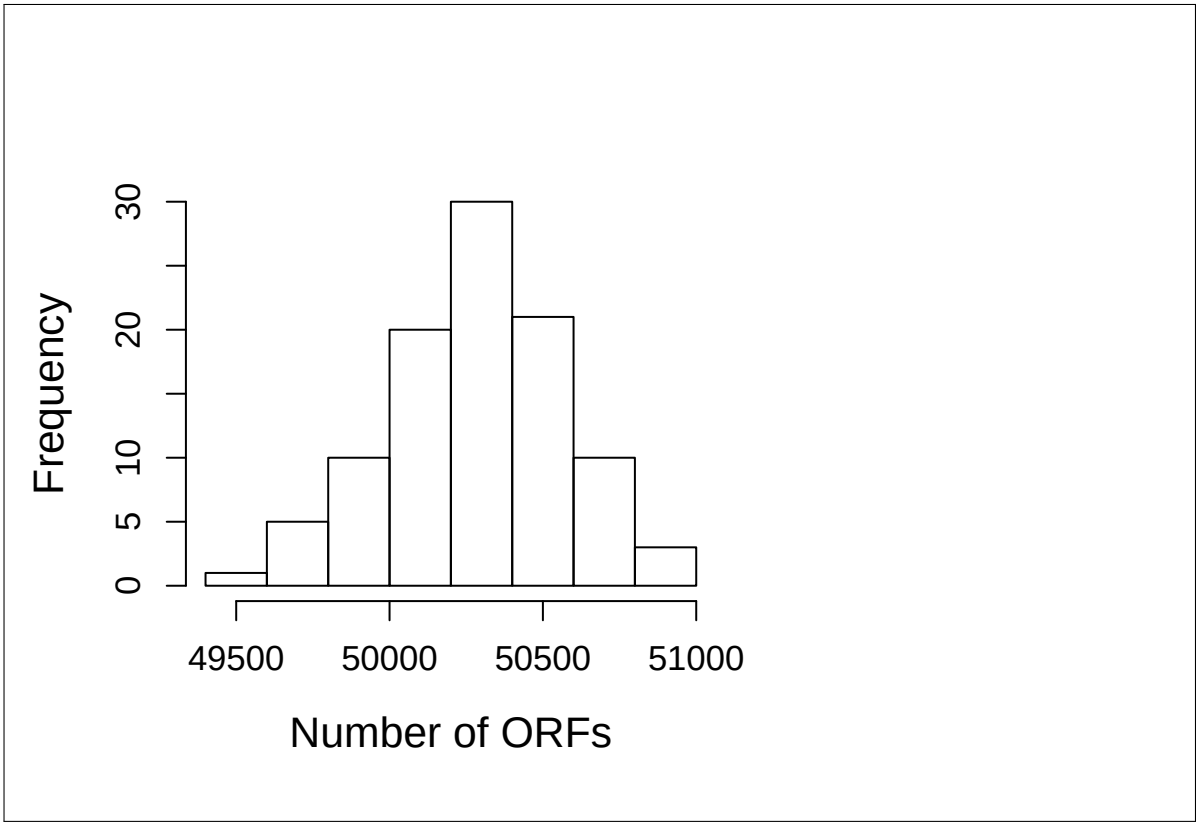
to count the number of ORFs and capture the result in file `orf_cnts.csv`. From here, I will use `R` to do the rest:

```
d.hw5q7 <- read.csv("orf_cnts.csv", header=F)
mean.orf.number <- mean(d.hw5q7$V2)
max.orf.number <- max(d.hw5q7$V2)
```

Report the mean number of ORFs found: 5.027804×10^4. Report the maximum number of ORFs found: **50937**. Plot a histogram of the number of ORFs.

Solution:

```
hist(d.hw5q7$V2, xlab="Number of ORFs", cex.axis=1.1, cex.lab=1.3, main="")
```



8. Count the number of ORFs in the `NC_000908.fsa` file. Can you reject the random order model for the *M. genitalium* genome?

Solution: To count the ORFs in the real genome, we issue command

```
python orf_stats.py NC_000908.fsa
```

The results are shown below.

```
## NC_000908.2, 10716
```

Does the number of ORFs you found in *M. genitalium* genome make sense? Why or why not?

Solution: As we had predicted, the number of ORFs in the real genome is substantially smaller.

Are all of them true ORFs? Why?

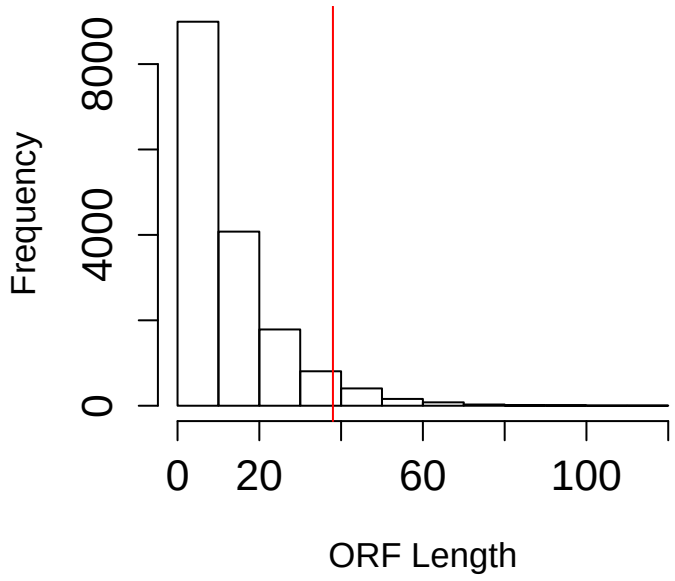
> **Solution:** No. Nature is sloppy. There will be accidential ORFs in the *M. genitalium* genome that are not functional. Some of these may be old ORFs that have lost their previous function. Mutations that create start/stop codons can produce accidental ORFs, and this may be one way that new genes are invented by evolution: see orphan genes.

9. We will now try to find the number of true ORFs in *M. genitalium* using a threshold derived from the null model. Our test statistic is the length of the ORF. Generally speaking, what values of the test statistic would indicate a true ORF?

> **Solution:** By similar arguments we have made above, functional ORFs are likely to be longer because stop codons are explicitly prohibited from occurring in the middle of a functional ORF.

Because our experimentally sampled unit is now the ORF (not the genome) and there are many ORFs in a single genome, we can estimate the ORF length distribution under the null distribution from a single genome permutation. Compute the ORF length null distribution from the first permuted genome. Estimate the 95% percentile (i.e., the ORF length which is longer than 95% of the ORF lengths). Are the ORF lengths computed from the genome independent? Why or why not?

> **Solution:** We use the `orf_stats.py` script to obtain the length of ORFs from the shuffled and real genome with these commands:
>
> ```
> python orf_stats.py NC_000908*shuff.fsa length > orf_lengths.csv
> python orf_stats.py NC_000908.fsa length >> orf_lengths.csv
> ```
>
> Now, we will use `R` to process the results:
>
> ```
> d.hw5q9 <- read.csv("orf_lengths.csv", header=F)
> q95 <- quantile(d.hw5q9$V3[d.hw5q9$V1 == "Shuffle0"], prob=0.95)
> ```
>
> The 0.95 quantile that we estimate is 38.000. A histogram of the values is shown below, with the quantile marked in red.
>
> ```
> hist(d.hw5q9$V3[d.hw5q9$V1 == "Shuffle0"], cex.axis=1.3, cex.lab=1.1,
>         main="", xlab="ORF Length")
> abline(v = q95, col="red")
> ```



The ORF lengths computed from the genome are not independent. To argue this point, consider the forward and reverse strands in the same region. If there is a long ORF on the forward strand, then there can be an effect on the presence or absence of ORFs on the reverse strand, since the reverse strand is made from the reverse complement of the forward strand. Another example is the last ORFs. If all the A, G, and T have been already used in the permutation, then the last C nucleotides will necessarily form an ORF since they cannot make a stop codon.

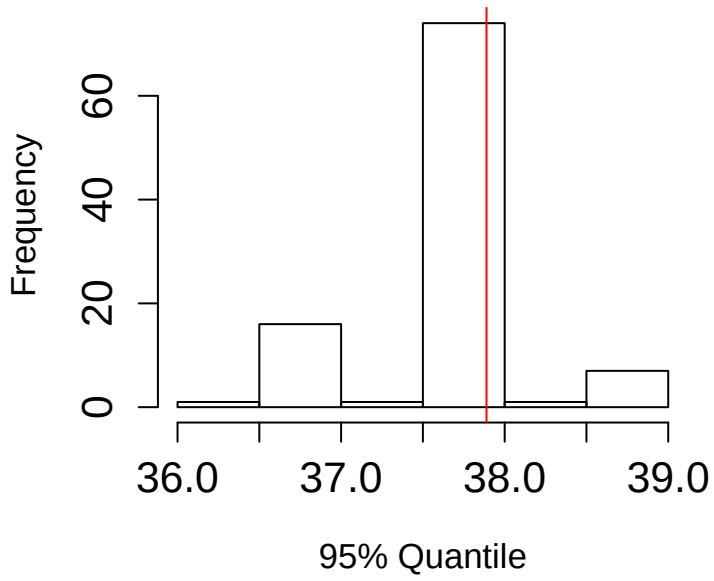
10. Percentiles are notoriously poorly estimated. You can get a better 95% percentile estimate (lower variance) by averaging the estimates from all 100 permuted genomes. What is the new estimate? (Note: Averaging the estimates is mathematically different from estimating the percentile from the combined sample of all ORF lengths from all 100 permutations. The difference would be especially notable when the ORF lengths are highly dependent within a single genome.)

Solution: We will do this in `R`.

```
hw5q10.q95 <- NULL
for (i in 0:99) {
        hw5q10.q95[i+1] <- quantile(d.hw5q9$V3[d.hw5q9$V1
                == paste("Shuffle", i, sep="")], prob=0.95)
}
hw5q10.q95.mean <- mean(hw5q10.q95)
```

The histogram of 100 estimated quantiles is shown below, with the mean 37.889 highlighted in red. In this case, the estimate from the first dataset is the same as the mean combining all 100 datasets, since we reject the null hypothesis of a nonfunctional ORF when the ORF observed length, which is necessarily an integer, is $\geq$ the quantile.

```
hist(hw5q10.q95, cex.axis=1.3, cex.lab=1.1, main="",
        xlab="95% Quantile")
abline(v = hw5q10.q95.mean, col="red")
```



11. Using the 95% percentile from question 8 as a minimum ORF length, how many ORFs are predicted to be in the *M. genitalium* genome? **927**. How did you calculate that (share any code you wrote and command you issued)?

Solution: Again, I use `R` for convenience, but the same logic would be very easy to implement in Python.

```
num.fxnal <- sum(d.hw5q9$V3[d.hw5q9$V1=="NC_000908.2"]
        >= hw5q10.q95.mean)
```

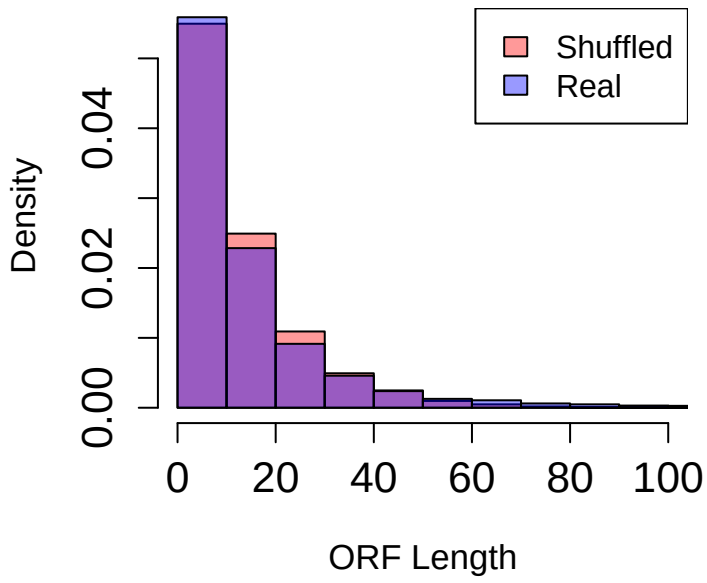
If the random order model were correct, how many of these ORFs would you expect to be false positives (not real ORFs)? How does the proportion change if there are true positives (real ORFs)? Why?

Solution: (This was an unintentional trick question.) If the random order is correct, then we expect all 100% of these ORFs to be false positives. Of all the ORFs in the genome, we expect 5% of them to be labeled (incorrectly) as functional ORFs (false positives).

If there are true ORFs, then we expect the proportion of false positives to drop because some will be true positives. However, there will still be true positives because we selected the threshold such that 5% of the false ORFs exceed it. The exact proportion of true positives will depend on how many true ORFs there are and their length distribution.

We can get a small sense of the two distributions by plotting the histograms of the ORF lengths from the first shuffled genome and the real genome. Since we do not know the true ORFs, the ORF lengths from the real genome include false positives.

```
hist(d.hw5q9$V3[d.hw5q9$V1=="Shuffle0"], xlab="ORF Length", freq=F,
        cex.axis=1.3, cex.lab=1.1, main="", col=rgb(1,0,0,0.4),
        xlim=c(0,100))
hist(d.hw5q9$V3[d.hw5q9$V1=="NC_000908.2"], breaks=200, add=T,
        freq=F, col=rgb(0,0,1,0.4))
legend(x = "topright", legend=c("Shuffled", "Real"),
         fill=c(rgb(1,0,0,0.4), rgb(0,0,1,0.4)))
```



There is very little difference between the distributions. In fact, we see more short ORFs (less than 10) and long ORFs (greater than 60) in the real genome. Meanwhile, shuffled genome ORFs tend more often to be of length 10-30.

12. The annotators of *M. genitalium* have determined that there are 482 protein coding genes. Were you close to the mark? Why do you think that you deviated (if you did)?

Solution: We found almost twice the number of functional ORFs. These extra functional ORFs include false positives, as discussed above. If the random order model were correct for the real genome, then we expect 5% of the 10,716 ORFs to be false positives. Of course there are some true ORFs in the genome, so the expected number of false positives is smaller. From our data, we can conclude there are between $391.2 = 927 - 0.05 \times 10716$ and 927 functional ORFs in the *M. genitalium* genome, which is totally consistent with what the professional annotators have found.