Math189 HW3 Code

May 9, 2021

1 Setup

[4]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

#improve resolution
%config InlineBackend.figure_format ='retina'

# set plotting size parameter
plt.rcParams['figure.figsize'] = (17, 7)
plt.rcParams.update({'font.size': 16})

import warnings
warnings.filterwarnings('ignore')
```

[54]:
```
df = pd.read_csv("hcmv.txt", delim_whitespace = True)
```

[55]:
```
N = 229_354
n_pal = 296
```

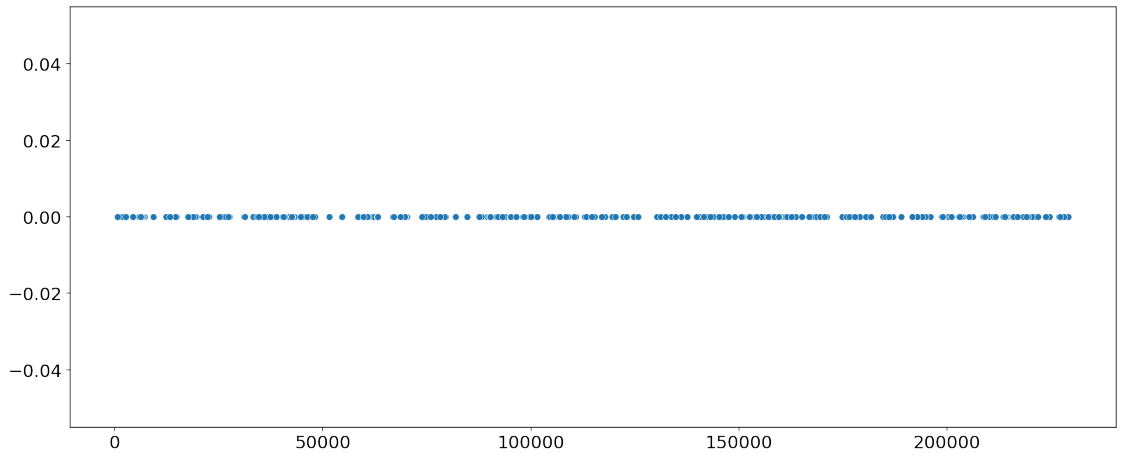
2 1

[Random scatter] Use a computer simulation to see what random scattering looks like. Simulate 296 palindrome sites chosen at random along a DNA sequence of 229,354 bases using a pseudo random number generator. When this is done several times, by making sets of simulated palindrome locations, then the real data can be compared to the simulated data.

[56]:
```
runif = np.random.random_integers(1, high = N, size = n_pal)
```

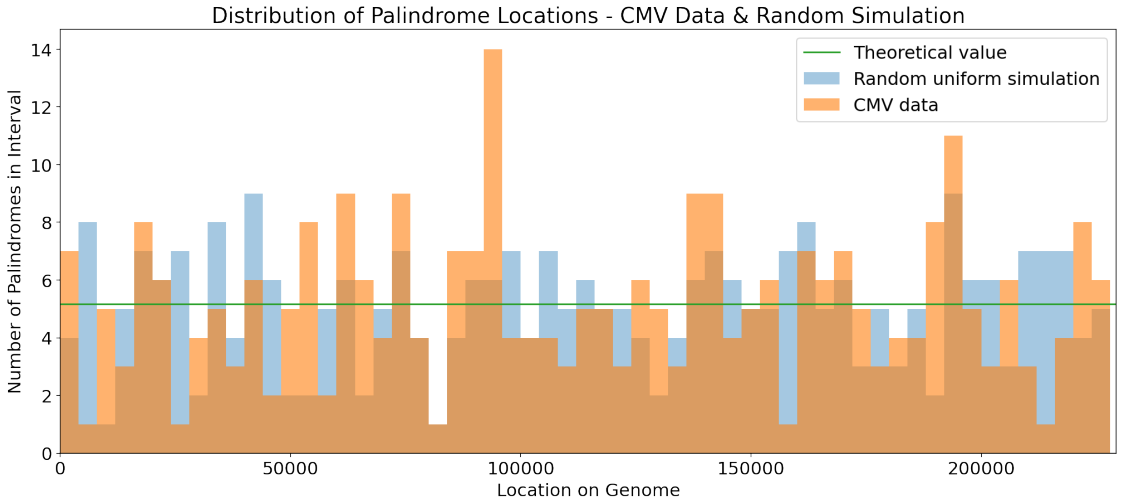
[136]:
```
sns.scatterplot(runif, np.zeros(296))
```

[136]:
```
<AxesSubplot:>
```



[439]:
```
interval_len = 4000

f, ax = plt.subplots(1, 1)
sns.distplot(runif, kde = False, bins = range(0, N, interval_len),
             label = "Random uniform simulation", ax = ax, hist_kws={"alpha": 0.
 ↪4})
sns.distplot(df['location'], kde = False, bins = range(0, N, interval_len),
             label = "CMV data", ax = ax, hist_kws={"alpha": 0.6})
ax.plot([0, N], [n_pal/(N/interval_len), n_pal/(N/interval_len)], label =␣
 ↪"Theoretical value")
ax.legend()
ax.set_title(label = "Distribution of Palindrome Locations - CMV Data & Random␣
 ↪Simulation")
ax.set(xlabel = "Location on Genome", ylabel = "Number of Palindromes in␣
 ↪Interval", xlim = [0, N])
```

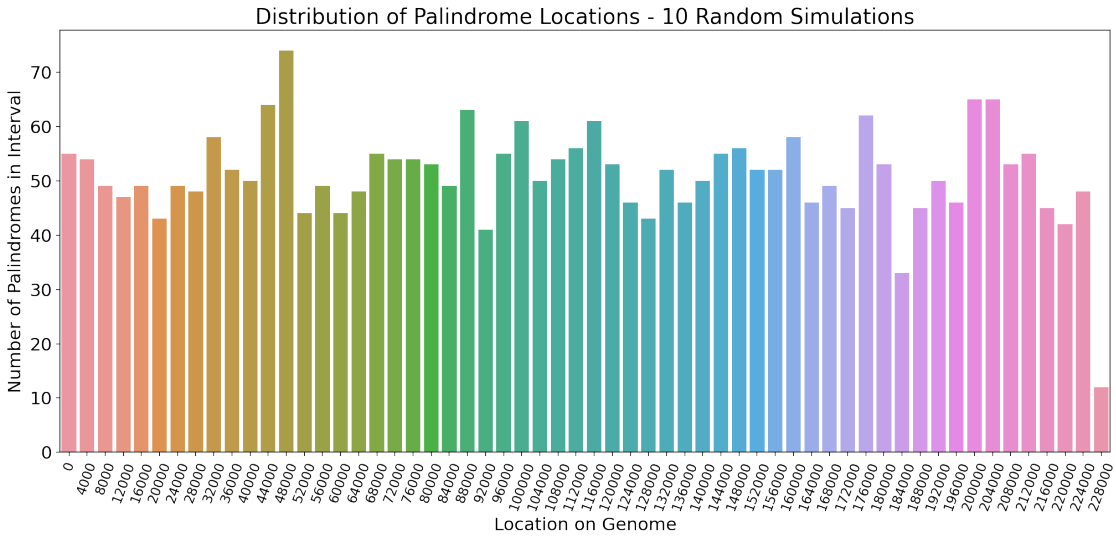
[439]:
```
[Text(0.5, 0, 'Location on Genome'),
 Text(0, 0.5, 'Number of Palindromes in Interval'),
 (0.0, 229354.0)]
```



[442]:
```
#Simulating 10 random samples and storing frequency in each bin

#Slow

bins = range(0, N, interval_len)
freq_dict = {b : 0 for b in bins}
points_dict = {b : [0 for i in range(10)] for b in bins}

for i in range(10):
    rsample = np.random.random_integers(1, high = N, size = n_pal)
    for k, v in freq_dict.items():
        for loc in rsample:
            if loc in range(k, k + interval_len):
                freq_dict[k] += 1
                points_dict[k][i] += 1
```

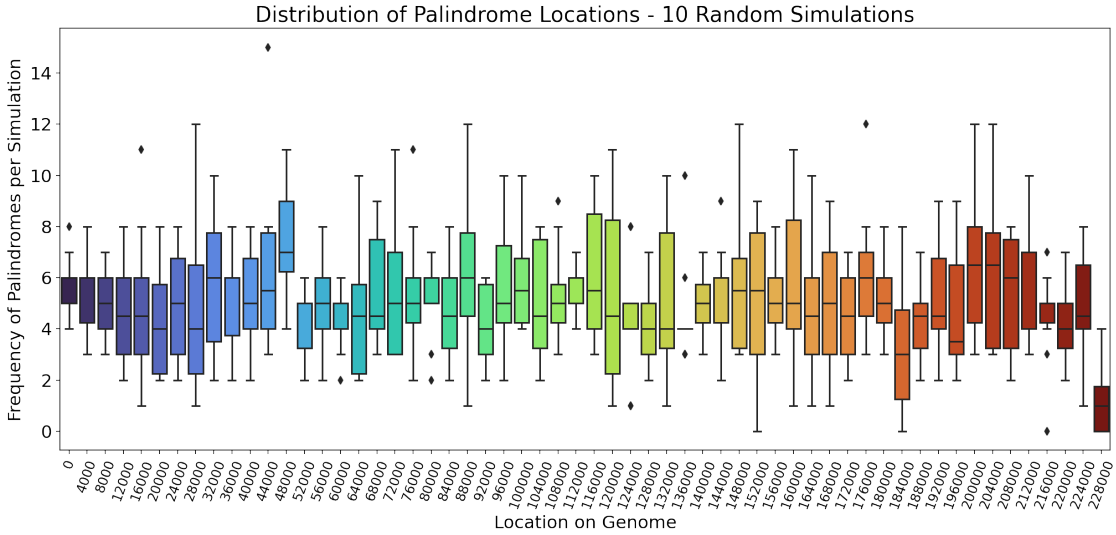
[443]:
```
df_freqs = pd.DataFrame({"Bin" : list(freq_dict.keys()), "Frequency" :␣
 ↪list(freq_dict.values())})
```

[444]:
```
f, ax = plt.subplots(1, 1)
ax = sns.barplot(df_freqs["Bin"], df_freqs["Frequency"])
ax.set_title(label = "Distribution of Palindrome Locations - 10 Random␣
 ↪Simulations")
ax.set(xlabel = "Location on Genome", ylabel = "Number of Palindromes in␣
 ↪Interval")
ax.set_xticklabels(labels = bins, rotation=70, size = 12)
plt.show()
```

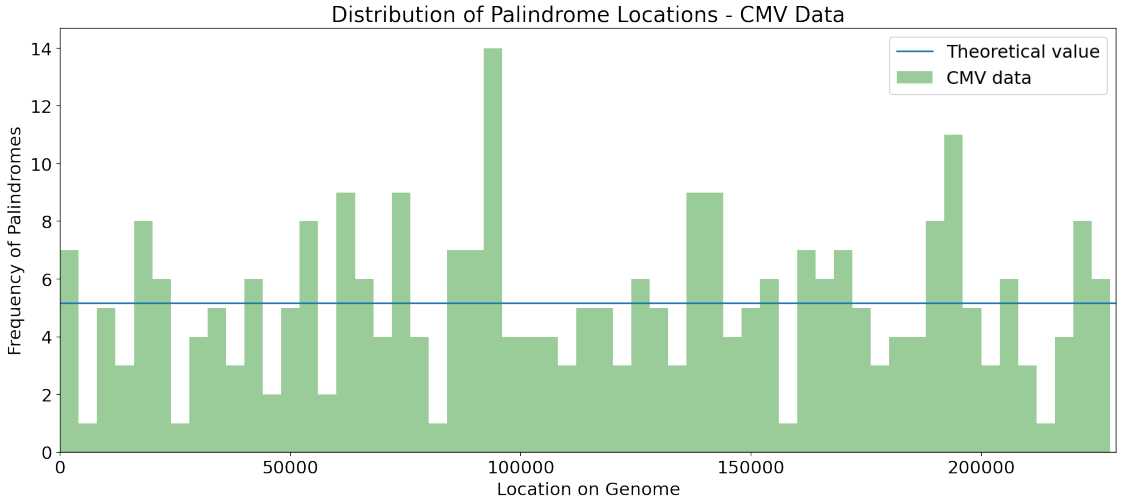


[445]:
```
simulations_df = pd.DataFrame.from_dict(points_dict, orient="columns")
```

[446]:
```
f, ax = plt.subplots(1, 1)
ax = sns.boxplot(x="variable", y="value", data=pd.melt(simulations_df), palette␣
↪= "turbo")
ax.set_title(label = "Distribution of Palindrome Locations - 10 Random␣
↪Simulations")
ax.set(xlabel = "Location on Genome", ylabel = "Frequency of Palindromes per␣
↪Simulation")
ax.set_xticklabels(labels = bins, rotation=70, size = 12)
plt.show()
```



[447]:
```
f, ax = plt.subplots(1, 1)
sns.distplot(df['location'], kde = False, bins = range(0, N, interval_len),␣
↪label = "CMV data", color = "green",
             ax = ax)
ax.plot([0, N], [n_pal/(N/interval_len), n_pal/(N/interval_len)], label =␣
↪"Theoretical value")
ax.set_title(label = "Distribution of Palindrome Locations - CMV Data")
ax.legend()
ax.set(xlabel = "Location on Genome", ylabel = "Frequency of Palindromes", xlim␣
↪= [0, N])
plt.show()
```



We can see that in our data a couple of bins have high frequencies of palindromes. Comparing to the graph generated by the random model, we can see that bins with such high frequencies do occur by chance but are not really that frequent. Eg. The boxplot at position 116000 had a simulation with a pretty high frequency (11), but this was exceptional, as the 75% percentile is at 8 palindromes.

Furthermore, we can see that the 75% percentiles never exceed 8, indicating that it is unlikely for frequencies of palindromes within a bin to be as high as was observed in our dataset. This suggests that these clusters should be investigated.

3 2

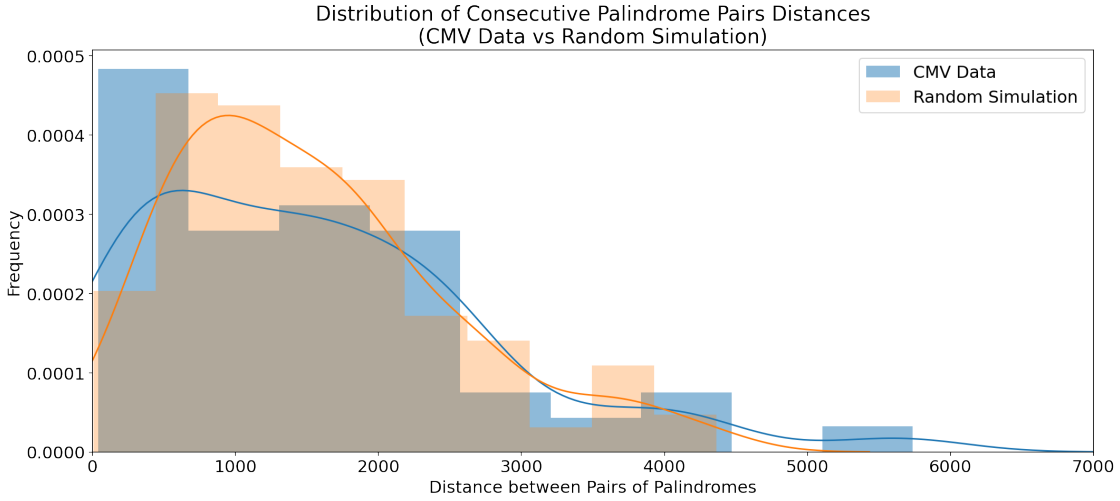
[502]:
```
runif = np.random.random_integers(1, high = N, size = n_pal)
```

[503]:
```
# Distances b/w pairs
spacings = np.array([])
for i in range(0, 294, 2):
    space = df['location'].iloc[i+2] - df['location'].iloc[i]
```

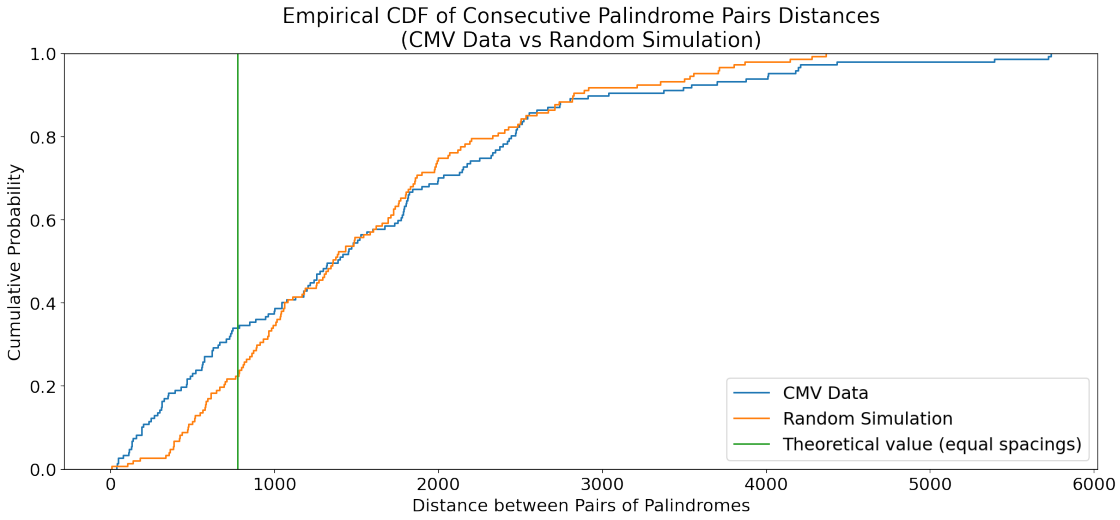
```
    spacings = np.append(space, spacings)
```

[504]:
```
# Distances b/w consecutive palindromes (Simulated Data)
rspacings = np.array([])
for i in range(0, 294, 2):
    space = sorted(runif)[i+2] - sorted(runif)[i]
    rspacings = np.append(space, rspacings)
```

[505]:
```
f, ax = plt.subplots(1, 1)
ax = sns.distplot(spacings, hist_kws={"alpha": 0.5}, label = "CMV Data")
ax = sns.distplot(rspacings, hist_kws={"alpha": 0.3}, label = "Random␣
 ↪Simulation")
ax.set_title(label = "Distribution of Consecutive Palindrome Pairs␣
 ↪Distances\n(CMV Data vs Random Simulation)")
ax.set(xlabel = "Distance between Pairs of Palindromes", ylabel = "Frequency",␣
 ↪xlim = [0, 7000])
ax.legend()
plt.show()
```



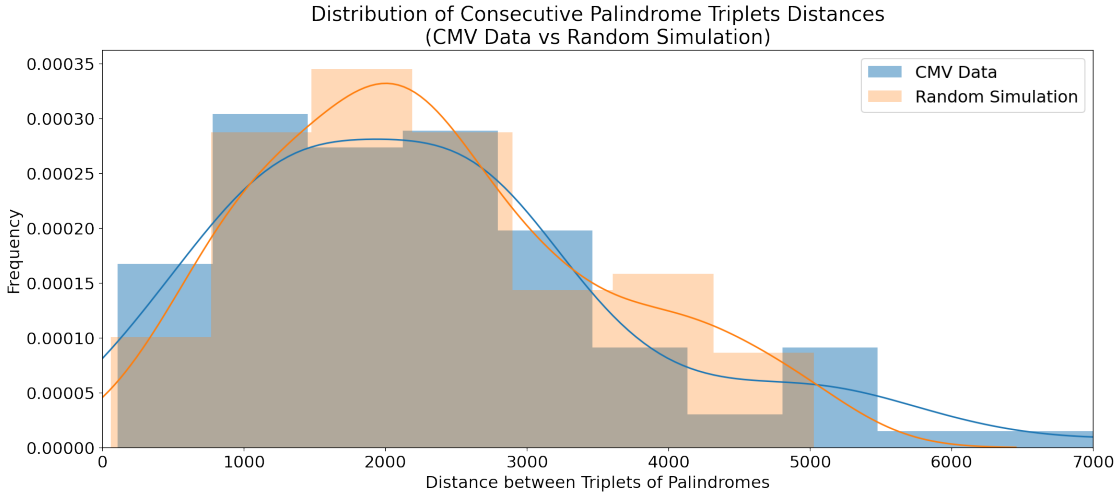
[506]:
```
f, ax = plt.subplots(1, 1)
ax = sns.ecdfplot(spacings, label = "CMV Data")
ax = sns.ecdfplot(rspacings, label = "Random Simulation")
ax = sns.ecdfplot([N/n_pal], label = "Theoretical value (equal spacings)")
ax.set_title(label = "Empirical CDF of Consecutive Palindrome Pairs␣
 ↪Distances\n(CMV Data vs Random Simulation)")
ax.set(xlabel = "Distance between Pairs of Palindromes", ylabel = "Cumulative␣
 ↪Probability")
ax.legend()
plt.show()
```



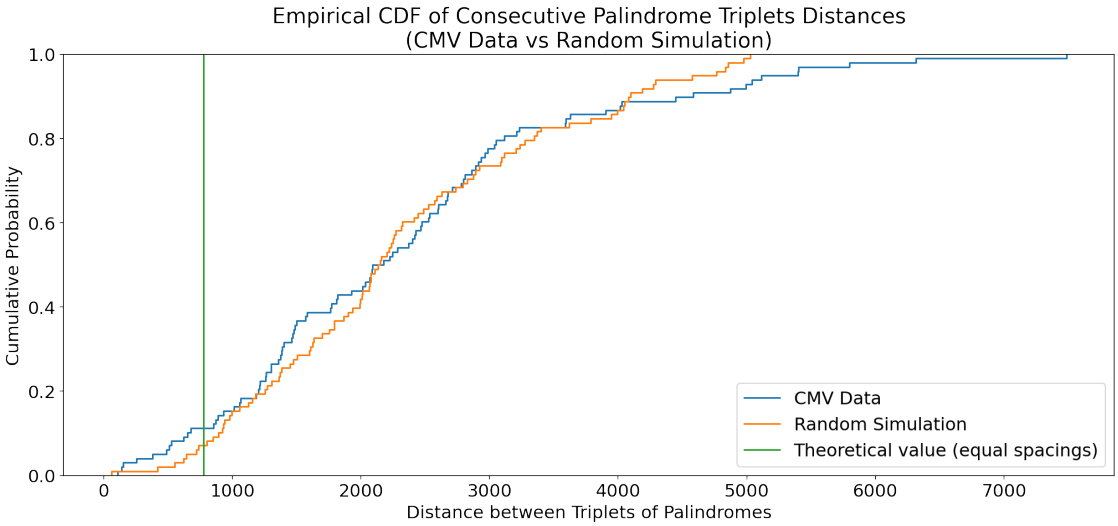
[507]:
```
# Distances b/w pairs
spacings = np.array([])
for i in range(0, 293, 3):
    space = df['location'].iloc[i+3] - df['location'].iloc[i]
    spacings = np.append(space, spacings)
```

[508]:
```
# Distances b/w consecutive palindromes (Simulated Data)
rspacings = np.array([])
for i in range(0, 293, 3):
    space = sorted(runif)[i+3] - sorted(runif)[i]
    rspacings = np.append(space, rspacings)
```

[509]:
```
f, ax = plt.subplots(1, 1)
ax = sns.distplot(spacings, hist_kws={"alpha": 0.5}, label = "CMV Data")
ax = sns.distplot(rspacings, hist_kws={"alpha": 0.3}, label = "Random␣
 ↪Simulation")
ax.set_title(label = "Distribution of Consecutive Palindrome Triplets␣
 ↪Distances\n(CMV Data vs Random Simulation)")
ax.set(xlabel = "Distance between Triplets of Palindromes", ylabel =␣
 ↪"Frequency", xlim = [0, 7000])
ax.legend()
plt.show()
```



[510]:
```
f, ax = plt.subplots(1, 1)
ax = sns.ecdfplot(spacings, label = "CMV Data")
ax = sns.ecdfplot(rspacings, label = "Random Simulation")
ax = sns.ecdfplot([N/n_pal], label = "Theoretical value (equal spacings)")
ax.set_title(label = "Empirical CDF of Consecutive Palindrome Triplets␣
↪Distances\n(CMV Data vs Random Simulation)")
ax.set(xlabel = "Distance between Triplets of Palindromes", ylabel =␣
↪"Cumulative Probability")
ax.legend()
plt.show()
```



4 Advanced Analysis

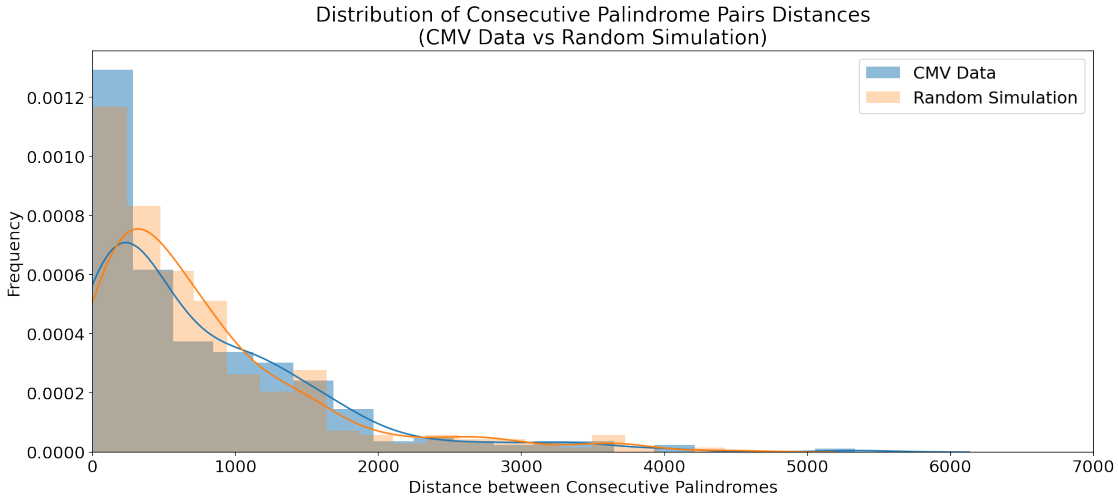
[Locations and spacings] Use graphical methods to examine the spacings between consecutive palindromes and sum of consecutive pairs, triplets, etc, spacings. Compare what you find to what would you expect to find in a random scatter. Also, use graphical methods to compare locations of the palindromes.

4.0.1 Pairs:

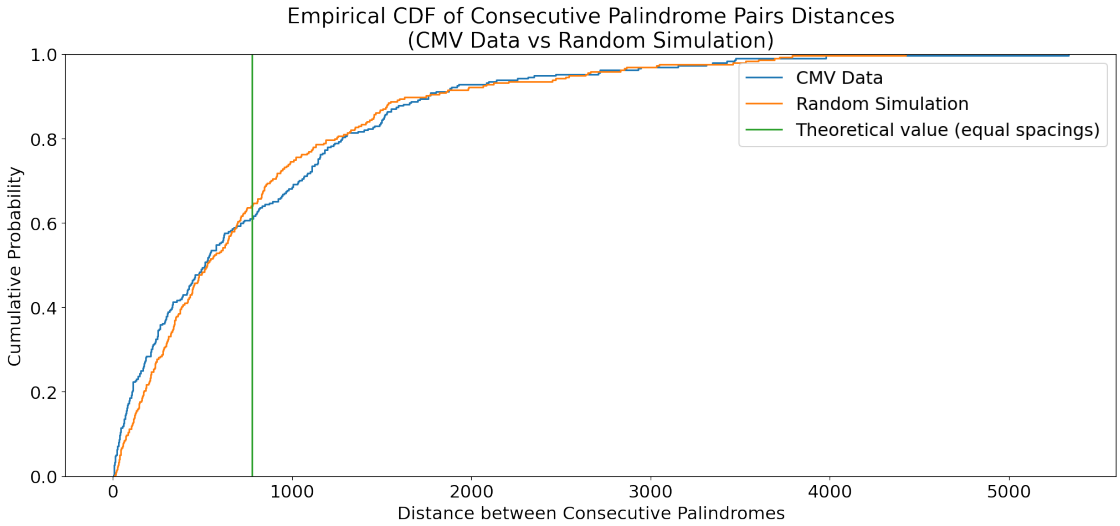
[285]:
```
# Distances b/w consecutive palindromes (CMV Data)
spacings = np.array([])
for i in range(295):
    space = df['location'].iloc[i+1] - df['location'].iloc[i]
    spacings = np.append(space, spacings)
```

[286]:
```
# Distances b/w consecutive palindromes (Simulated Data)
rspacings = np.array([])
for i in range(295):
    space = sorted(runif)[i+1] - sorted(runif)[i]
    rspacings = np.append(space, rspacings)
```

[287]:
```
f, ax = plt.subplots(1, 1)
ax = sns.distplot(spacings, hist_kws={"alpha": 0.5}, label = "CMV Data")
ax = sns.distplot(rspacings, hist_kws={"alpha": 0.3}, label = "Random␣
 ↪Simulation")
ax.set_title(label = "Distribution of Consecutive Palindrome Pairs␣
 ↪Distances\n(CMV Data vs Random Simulation)")
ax.set(xlabel = "Distance between Consecutive Palindromes", ylabel =␣
 ↪"Frequency", xlim = [0, 7000])
ax.legend()
plt.show()
```



[288]:
```
f, ax = plt.subplots(1, 1)
ax = sns.ecdfplot(spacings, label = "CMV Data")
ax = sns.ecdfplot(rspacings, label = "Random Simulation")
ax = sns.ecdfplot([N/n_pal], label = "Theoretical value (equal spacings)")
ax.set_title(label = "Empirical CDF of Consecutive Palindrome Pairs␣
↪Distances\n(CMV Data vs Random Simulation)")
ax.set(xlabel = "Distance between Consecutive Palindromes", ylabel =␣
↪"Cumulative Probability")
ax.legend()
plt.show()
```



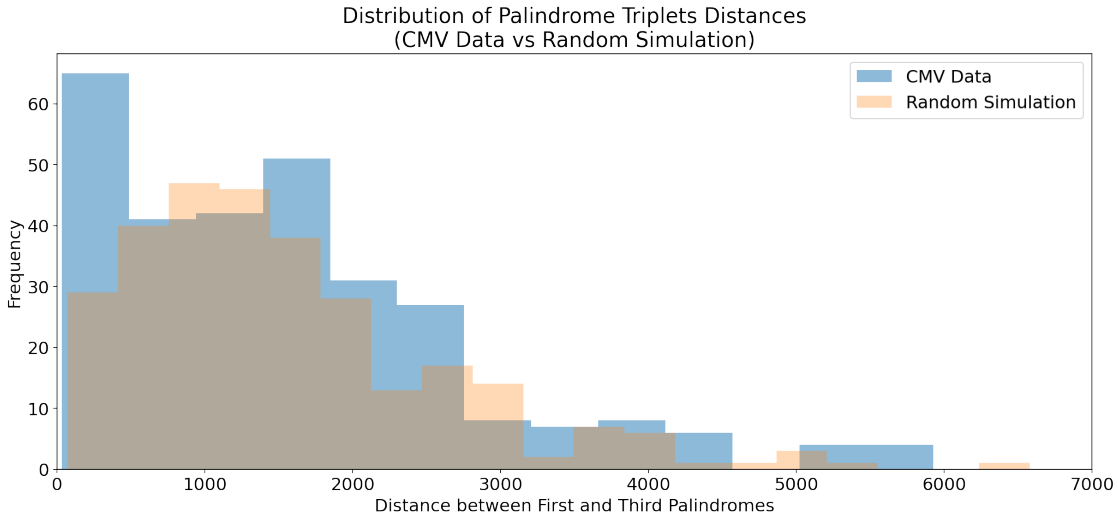
We can see that our dataset has a higher probability of having consecutive palindromes closer together (0 - 500 bases apart) and a lower probability of having consecutive palindromes further apart ($>$ 800 bases apart) than the random simulation.

4.0.2 Triplets:

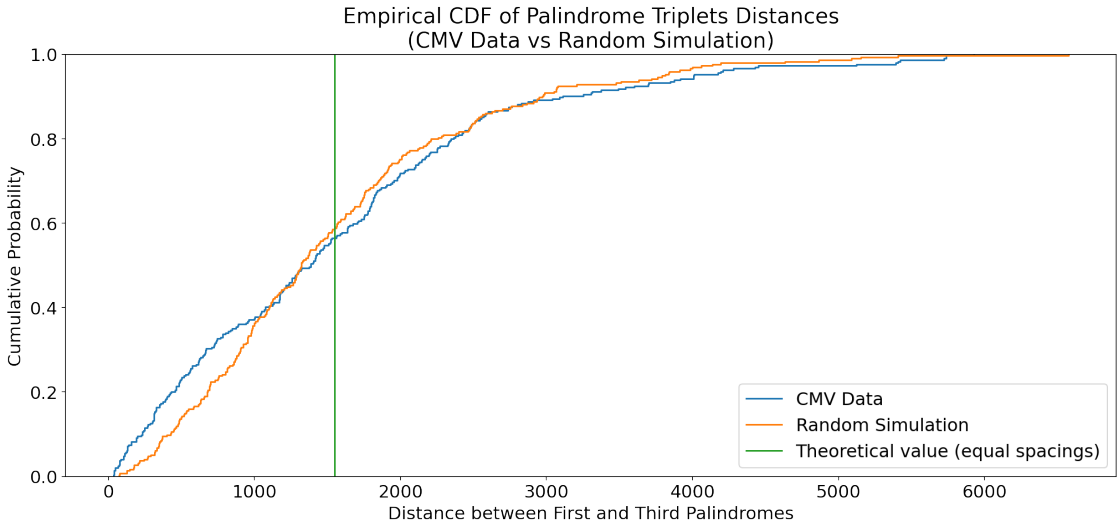
[289]:
```
# Distances b/w consecutive palindromes (CMV Data)
spacings = np.array([])
for i in range(294):
    space = df['location'].iloc[i+2] - df['location'].iloc[i]
    spacings = np.append(space, spacings)
```

[290]:
```
# Distances b/w consecutive palindromes (Simulated Data)
rspacings = np.array([])
for i in range(294):
    space = sorted(runif)[i+2] - sorted(runif)[i]
    rspacings = np.append(space, rspacings)
```

[293]:
```
f, ax = plt.subplots(1, 1)
ax = sns.distplot(spacings, hist_kws={"alpha": 0.5}, label = "CMV Data", kde =␣
↪False)
ax = sns.distplot(rspacings, hist_kws={"alpha": 0.3}, label = "Random␣
↪Simulation", kde = False)
ax.set_title(label = "Distribution of Palindrome Triplets Distances\n(CMV Data␣
↪vs Random Simulation)")
ax.set(xlabel = "Distance between First and Third Palindromes", ylabel =␣
↪"Frequency", xlim = [0, 7000])
ax.legend()
plt.show()
```



[294]:
```
f, ax = plt.subplots(1, 1)
ax = sns.ecdfplot(spacings, label = "CMV Data")
ax = sns.ecdfplot(rspacings, label = "Random Simulation")
ax = sns.ecdfplot([2*N/n_pal], label = "Theoretical value (equal spacings)")
ax.set_title(label = "Empirical CDF of Palindrome Triplets Distances\n(CMV Data␣
↪vs Random Simulation)")
ax.set(xlabel = "Distance between First and Third Palindromes", ylabel =␣
↪"Cumulative Probability")
ax.legend()
plt.show()
```



Here we can see that it is much more likely to see palindrome triplets close to each other in our dataset than it is in a random simulation.

5 3

[Counts] Use graphical methods and more formal statistical tests to examine the counts of palindromes in various regions of the DNA. Split the DNA into nonoverlaping regions of equal length to compare the number of palindromes in an interval to the number of that would you expect from uniform random scatter. The counts for shorter regions will be more variable than those for longer regions. Also consider classifying the regions according to their number of counts.

[311]:
```
# Finding counts for 10 simulations
counts_d = {}

for i in range(10):
    if i == 9:
        d = simulations_df.T.groupby(i).count()[0].to_dict()
    else:
        d = simulations_df.T.groupby(i).count()[i+1].to_dict()
    for k, v in d.items():
        if k in counts_d.keys():
            counts_d[k][i] = v
        else:
            counts_d[k] = np.zeros(10)
            counts_d[k][i] = v
```

[333]:
```
#Convert array from numpy to list and values from float to int
for k, v in counts_d.items():
```

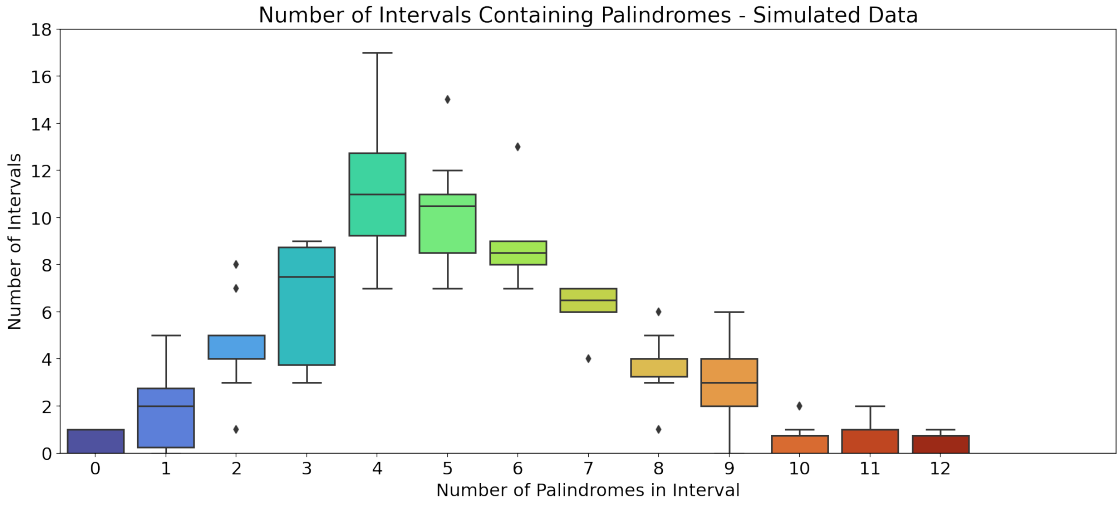
```
    counts_d[k] = list(map(lambda x: int(x), list(counts_d[k])))
```

[335]:
```
counts_df = pd.DataFrame.from_dict(counts_d, orient = "columns")
```

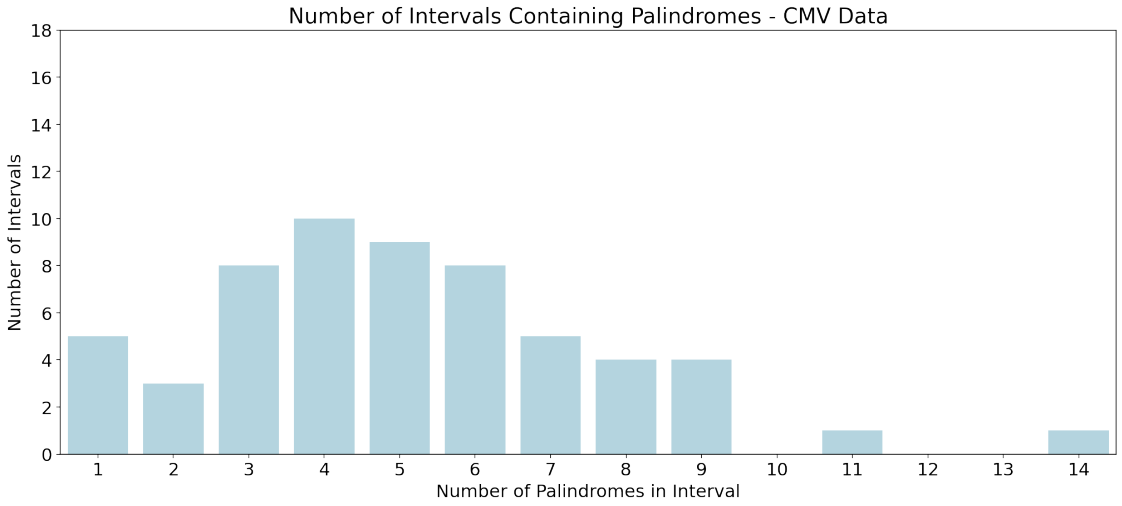
[339]:
```
#Finding counts for CMV dataset
bins = range(0, N, interval_len)
freq_dict_cmv = {b : 0 for b in bins}

for k, v in freq_dict_cmv.items():
    for loc in df['location']:
        if loc in range(k, k + interval_len):
            freq_dict_cmv[k] += 1
```

[377]:
```
counts_df_cmv = pd.DataFrame(freq_dict_cmv.values(),
                             freq_dict_cmv.keys()).reset_index().groupby(0).
↪count().reset_index()
counts_df_cmv = counts_df_cmv.append({0:10,'index':0}, ignore_index = True)
counts_df_cmv = counts_df_cmv.append({0:12,'index':0}, ignore_index = True)
counts_df_cmv = counts_df_cmv.append({0:13,'index':0}, ignore_index = True)
```

[395]:
```
f, ax = plt.subplots(1, 1)
ax = sns.boxplot(x="variable", y="value", data=pd.melt(counts_df), palette =␣
↪"turbo")
ax.set_title(label = "Number of Intervals Containing Palindromes - Simulated␣
↪Data")
ax.set(xlabel = "Number of Palindromes in Interval", ylabel = "Number of␣
↪Intervals", ylim = [0, 18], xlim = [-0.5, 14.5])
plt.show()
```



[380]:
```
f, ax = plt.subplots(1, 1)
ax = sns.barplot(counts_df_cmv[0], counts_df_cmv['index'], color = 'lightblue')
ax.set_title(label = "Number of Intervals Containing Palindromes - CMV Data")
ax.set(xlabel = "Number of Palindromes in Interval", ylabel = "Number of␣
↪Intervals", ylim = [0, 18])
plt.show()
```



6 4

[The biggest cluster] Does the interval with the greatest number of palindromes indicate a potential origin of replication? Be careful in making your intervals, for any small, but significant, deviations from random scatter, such as a tight cluster of a few palindromes, could easily go undetected if the regions examined are too large. Also, if the regions are too small, a cluster of palindromes may be split between adjacent intervals and not appear as a high-count interval.

[418]:
```
#Simulations
dfs = []

for interval_len in [500, 1000, 2000, 3000, 4000, 5000, 6000, 8000]:

    bins = range(0, N, interval_len)
    freq_dict = {b : 0 for b in bins}
    points_dict = {b : [0 for i in range(10)] for b in bins}

    for i in range(10):
        rsample = np.random.random_integers(1, high = N, size = n_pal)
        for k, v in freq_dict.items():
            for loc in rsample:
```

```
                if loc in range(k, k + interval_len):
                    freq_dict[k] += 1
                    points_dict[k][i] += 1

    dfs += [pd.DataFrame.from_dict(points_dict, orient="index").max()]
```

[420]:
```
df_maxes = pd.concat(dfs, axis = 1)
df_maxes.columns = [500, 1000, 2000, 3000, 4000, 5000, 6000, 8000]
df_maxes
```

[420]:

	500	1000	2000	3000	4000	5000	6000	8000
0	4	6	6	9	11	14	13	16
1	5	5	8	11	15	12	16	20
2	4	5	9	10	10	12	15	16
3	4	5	8	10	11	14	15	17
4	5	5	7	9	11	12	13	16
5	4	5	7	9	13	11	15	15
6	3	7	8	9	12	13	16	16
7	4	5	8	9	11	10	17	17
8	4	5	7	10	11	15	14	19
9	3	5	8	11	10	11	15	18

[473]:
```
#Real Data
dfs = []
df = pd.read_csv("hcmv.txt", delim_whitespace = True)

for interval_len in [500, 1000, 2000, 3000, 4000, 5000, 6000, 8000]:

    bins = range(0, N, interval_len)
    freq_dict = {b : 0 for b in bins}
    points_dict = {b : [0 for i in range(10)] for b in bins}

    rsample = df['location']
    for k, v in freq_dict.items():
        for loc in rsample:
            if loc in range(k, k + interval_len):
                freq_dict[k] += 1
                points_dict[k][i] += 1


    dfs += [pd.DataFrame.from_dict(points_dict, orient="index").max()]
```

[476]:
```
df_maxes = pd.concat(dfs, axis = 1)
df_maxes.columns = [500, 1000, 2000, 3000, 4000, 5000, 6000, 8000]
df_maxes
```

[476]:

	500	1000	2000	3000	4000	5000	6000	8000
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	8	8	12	13	14	18	19	21