BC205: Algorithms for Bioinformatics.

II. Biological Sequence Analysis

Christoforos Nikolaou

Biological Sequence Analysis

- Biological Sequence Analysis includes all lexicographic, statistical, or modeling types of analyses conducted on biological sequences such as DNA, RNA and proteins. In this class we will start looking into real biological problems, focusing on DNA sequences.
- We will discuss some very basic concepts of computation such as hashing.
- We will then turn to the implementation of the things we learnt last time using Brute Force and Divide and Conquer Approaches.
- We will go back the concept of Binary Search and implement it the context of Genome Analysis.

The biological problems:

- Compare different species on the basis of DNA composition
- Find evidence of horizontal gene transfer in a bacterial genome
- Locate the Origin of Repication of a Bacterial Genome

Bioinformatics Warm-Up

1. You are given a DNA sequence
   - Can you count the number of nucleotides of each of the four bases (A, G, C, T)?
   - How many calculations will you need?
   - How will you implement it?
2. Now consider the same problem only instead of nucleotides we need to count the number of all octanucleotides. What do you need to consider to attack the problem? What is different between the 4 mononucleotides and the 8^4 octanucleotides?

Aspects of DNA Composition

- GC content
- k-mer frequencies
- genomic signatures
- parity distributions

GC content

We call GC content (or GC%) the ratio of (G+C) nucleotides of a given DNA sequence

- Why is it important? G-C pairs are linked with 3 hydrogen bonds, while A-T ones with 2. High GC genomes are more stable in terms of physical chemistry.

GC Content

GC is related to:

- Biochemical level: Thermal stability
- Evolutionary level: Organism Phylogeny, Mutational pressures
- Genomic level: Genome size
- Functional level: Functional role of underlying sequences
- and many more

GC content in Genomic Sequences

- Bacteria: GC% is highly variable **between** species
- Bacteria: GC% is rather homogeneous **within** each genome
- Bacteria: GC% can be used in their classification

GC Content

GC content in Genomic Sequences

- Eukaryotes: Very homogeneous overall GC% (~40-45% in all animals)
- Eukaryotes: Fluctuation of GC contentalong the chromosomes and organization in areas of (rather) stable GC%
- Eukaryotes: Regions of stable high/low GC content that segregate mammalian genomes in isochores

Problem 1: GC content in Bacterial Genomes

- Given the DNA sequence of a Bacterial Genome, calculate its GC content:
  - Read the Sequence
  - Enumerate G
  - Enumerate C
  - Divide (G+C) over length of the sequence

GCContent. Pseudocode

- The idea is to **exhaustively** enumerate all mononucleotides, therefore our approach is a very basic Brute Force approach.
- Given that the content of the sequence is unknown we have no other choice.
- We will proceed by reading each nucleotide in the sequence and check its value. Then increment a variable each time we find a G or a C.

GCContent: Implementation (naive)

In [12]:
```python
# Naive GC content

def naiveGC(genomefile):
    import regex as re
    file = open(genomefile, 'r')

    seq = ""
    window = 1000
    total = 0
    nG = nC = 0
    GCCont = 0
    times = 0
    count = 0
    for line in file:
        count += 1
        if (count > 1): # the first line contains the non-sequence header so we discard
            length=len(line)
            total=total+length
            seq=seq+line[0:length-1]

    for k in range(len(seq)):
        if(seq[k]=="G"):
            nG+=1
        elif(seq[k]=="C"):
            nC+=1
    GCContent=(nG+nC)/len(seq)
    return(GCContent)
```

In [15]:
```
EcoliGC = naiveGC('files/ecoli.fa')
StaurGC = naiveGC('files/Staaur.fa')
print(EcoliGC, StaurGC)
```

```
0.5074167653333127 0.3291873726579277
```

GCContent: Implementation (using Python's count function)

The above approach is extremely naive as it makes use not so much of an exhaustive search (this needs to be done anyway in order to enumerate all instances), but because of the naiva approach to test each nucleotide against the pattern. We ca

In [16]:
```
def fastGC(genomefile):

    import regex as re

    file = open(genomefile, 'r')

    seq = ""
    total = 0
    nG = nC = 0
    GCCont = 0
    times = 0
    count = 0
    for line in file:
        count += 1
        if (count > 1):
            length = len(line)
            total = total+length
            seq = seq+line[0:length-1]
    file.close()

    nC = seq.count("C")
    nG = seq.count("G")
    GCContent = (nG+nC)/len(seq);
    return(GCContent)
```

In [17]:
```
EcoliGC = fastGC('files/ecoli.fa')
StaurGC = fastGC('files/Staaur.fa')
print(EcoliGC, StaurGC)
```

```
0.5074167653333127 0.3291873726579277
```

What we have basically done is use a built-in python function to search and count for all instances of the two nucleotides. We will come back to this later on. We now have a method to calculate the GC% of a given genome and can move on to some practical analyses.

Using sequence composition to infer phylogeny

In bacteria sequence composition is stable throughout the genome but varies widely between species. One can use simple measures of the nucleotide constitution to infer phylogenetic relationships between species.

Below, we will try to do just that, first just by using the GC%.

Hands on #1:

- Download a couple of bacterial genome sequences from ENSEMBL Bacteria (http://bacteria.ensembl.org/index.html)
- Implement GC content
- Report the results

- An example would be

Genome	GC
a-Bac1	0.334
e-Bac2	0.595
e-Bac3	0.668
g-Bac4	0.409
e-Bac5	0.551
a-Bac6	0.352
a-Bac7	0.354
g-Bac8	0.418
g-Bac9	0.434
e-Bac8	0.627

Problem 2: Variability of GC content *between* Bacterial Genomes

- Given a number of bacterial genomes:
  - Get their genome sequences
  - Calculate the GC contents
  - Calculate differences between the GC contents
  - Rank genomes based on their differences
- Pseudocode:
  - Perform GC_content on each of the genomes you downloaded
  - Calculate D_(i,j)=|GC_i-GC_j| over all i,j
  - Sort D_(i,j)

Problem 2: Approach

- Below we will do this on a set of precalculated GC% values from 10 different bacterial genomes.

In [19]:
```
f=open('files/GCContent.tsv', 'r')

i=0
GCC={}
for line in f:
    i=i+1
    if(i>1):
        species=line.split()[0]
        GC=line.split()[1]
        GCC[species]=float(GC)

gcdistances={}
for genome1 in GCC.keys():
    for genome2 in GCC.keys():
        pair=genome1+":"+genome2
        gcdistances[pair]=abs(float(GCC[genome1])-float(GCC[genome2]))
        gcdistances[pair]=round(gcdistances[pair],2)

sorted(gcdistances.items(), key=lambda x: x[1])
```

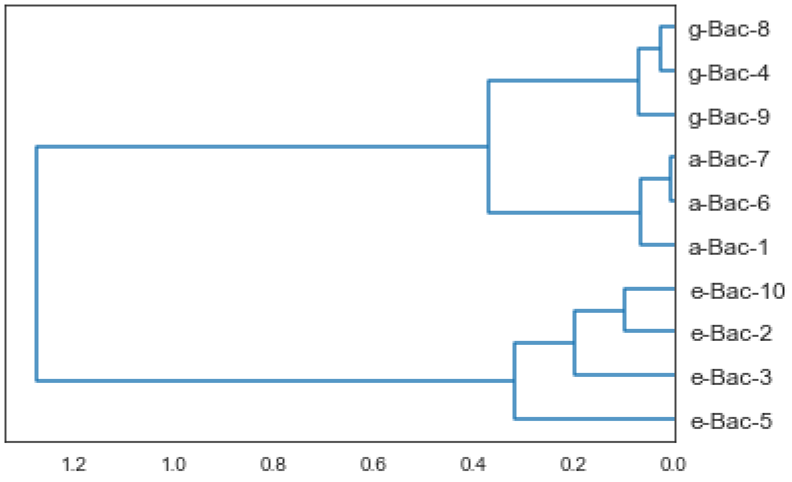
```
Out[19]: [('a-Bac1:a-Bac1', 0.0),
 ('e-Bac2:e-Bac2', 0.0),
 ('e-Bac3:e-Bac3', 0.0),
 ('g-Bac4:g-Bac4', 0.0),
 ('e-Bac5:e-Bac5', 0.0),
 ('a-Bac6:a-Bac6', 0.0),
 ('a-Bac6:a-Bac7', 0.0),
 ('a-Bac7:a-Bac6', 0.0),
 ('a-Bac7:a-Bac7', 0.0),
 ('g-Bac8:g-Bac8', 0.0),
 ('g-Bac9:g-Bac9', 0.0),
 ('e-Bac10:e-Bac10', 0.0),
 ('g-Bac4:g-Bac8', 0.01),
 ('g-Bac8:g-Bac4', 0.01),
 ('a-Bac1:a-Bac6', 0.02),
 ('a-Bac1:a-Bac7', 0.02),
 ('a-Bac6:a-Bac1', 0.02),
 ('a-Bac7:a-Bac1', 0.02),
 ('g-Bac8:g-Bac9', 0.02),
 ('g-Bac9:g-Bac8', 0.02),
 ('e-Bac2:e-Bac10', 0.03),
 ('g-Bac4:g-Bac9', 0.03),
 ('g-Bac9:g-Bac4', 0.03),
 ('e-Bac10:e-Bac2', 0.03),
 ('e-Bac3:e-Bac10', 0.04),
 ('e-Bac10:e-Bac3', 0.04),
 ('e-Bac2:e-Bac5', 0.05),
 ('g-Bac4:a-Bac7', 0.05),
 ('e-Bac5:e-Bac2', 0.05),
 ('a-Bac7:g-Bac4', 0.05),
 ('g-Bac4:a-Bac6', 0.06),
 ('a-Bac6:g-Bac4', 0.06),
 ('a-Bac7:g-Bac8', 0.06),
 ('g-Bac8:a-Bac7', 0.06),
 ('a-Bac1:g-Bac4', 0.07),
 ('e-Bac2:e-Bac3', 0.07),
 ('e-Bac3:e-Bac2', 0.07),
 ('g-Bac4:a-Bac1', 0.07),
 ('a-Bac6:g-Bac8', 0.07),
 ('g-Bac8:a-Bac6', 0.07),
 ('a-Bac1:g-Bac8', 0.08),
 ('a-Bac6:g-Bac9', 0.08),
 ('a-Bac7:g-Bac9', 0.08),
 ('g-Bac8:a-Bac1', 0.08),
 ('g-Bac9:a-Bac6', 0.08),
 ('g-Bac9:a-Bac7', 0.08),
 ('e-Bac5:e-Bac10', 0.09),
 ('e-Bac10:e-Bac5', 0.09),
 ('a-Bac1:g-Bac9', 0.1),
 ('g-Bac9:a-Bac1', 0.1),
 ('e-Bac5:g-Bac9', 0.11),
 ('g-Bac9:e-Bac5', 0.11),
 ('e-Bac5:g-Bac8', 0.12),
 ('g-Bac8:e-Bac5', 0.12),
 ('e-Bac3:e-Bac5', 0.13),
 ('g-Bac4:e-Bac5', 0.13),
 ('e-Bac5:e-Bac3', 0.13),
 ('e-Bac5:g-Bac4', 0.13),
 ('e-Bac2:g-Bac9', 0.16),
 ('g-Bac9:e-Bac2', 0.16),
 ('e-Bac2:g-Bac8', 0.18),
 ('g-Bac8:e-Bac2', 0.18),
 ('e-Bac2:g-Bac4', 0.19),
 ('g-Bac4:e-Bac2', 0.19),
 ('e-Bac5:a-Bac6', 0.19),
 ('e-Bac5:a-Bac7', 0.19),
 ('a-Bac6:e-Bac5', 0.19),
 ('a-Bac7:e-Bac5', 0.19),
 ('g-Bac9:e-Bac10', 0.19),
 ('e-Bac10:g-Bac9', 0.19),
 ('a-Bac1:e-Bac5', 0.21),
 ('e-Bac5:a-Bac1', 0.21),
 ('g-Bac8:e-Bac10', 0.21),
 ('e-Bac10:g-Bac8', 0.21),
 ('g-Bac4:e-Bac10', 0.22),
 ('e-Bac10:g-Bac4', 0.22),
 ('e-Bac3:g-Bac9', 0.23),
 ('g-Bac9:e-Bac3', 0.23),
 ('e-Bac2:a-Bac6', 0.24),
 ('e-Bac2:a-Bac7', 0.24),
 ('a-Bac6:e-Bac2', 0.24),
 ('a-Bac7:e-Bac2', 0.24),
 ('e-Bac3:g-Bac8', 0.25),
 ('g-Bac8:e-Bac3', 0.25),
 ('a-Bac1:e-Bac2', 0.26),
 ('e-Bac2:a-Bac1', 0.26),
 ('e-Bac3:g-Bac4', 0.26),
 ('g-Bac4:e-Bac3', 0.26),
 ('a-Bac7:e-Bac10', 0.27),
 ('e-Bac10:a-Bac7', 0.27),
 ('a-Bac6:e-Bac10', 0.28),
 ('e-Bac10:a-Bac6', 0.28),
 ('a-Bac1:e-Bac10', 0.29),
 ('e-Bac10:a-Bac1', 0.29),
 ('e-Bac3:a-Bac7', 0.31),
 ('a-Bac7:e-Bac3', 0.31),
 ('e-Bac3:a-Bac6', 0.32),
 ('a-Bac6:e-Bac3', 0.32),
 ('a-Bac1:e-Bac3', 0.33),
 ('e-Bac3:a-Bac1', 0.33)]
```

Let's now look how we can use this simple quantity to infer relationships between different genomes. In the following, we make use of some python functions to organize the genomes in a tree structure that resembles the way evolutionary biologists try to infer phylogenetic relationships.

In [20]:
```python
## Clustering of a dataset

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist, squareform
from scipy.cluster.hierarchy import linkage, dendrogram

# Load the dataframe and assign values/labels
df = pd.read_csv('files/GCContent_simple.csv')
dvalues = df['GCContent'].values.reshape(-1,1)
dlabels = list(df['Genome'])

# Calculate the distances
distances = pdist(dvalues)

# Convert the pairwise distances into a square distance matrix
distance_matrix = squareform(distances)
```

```
# Calculate the linkage matrix using Ward's method
linkage_matrix = linkage(distance_matrix, method='ward')

# Plot the dendrogram
sns.set_style('white')
dendrogram(linkage_matrix, labels=dlabels, color_threshold=0, orientation='left')

# Show the plot
plt.show()
```

```
<ipython-input-20-c652b4de09d1>:21: ClusterWarning: scipy.cluster: The symmetric non-neg
ative hollow observation matrix looks suspiciously like an uncondensed distance matrix
  linkage_matrix = linkage(distance_matrix, method='ward')
```



Notice how smaller values are obtained for same bacterial family (a-, g- and e-proteobacteria) which results in them being grouped together in the same clade of the dendrogram.

Problem 3: What about different regions of the genome?

- We just saw how genomic GC% values may be used to draw conclusions for bacterial phylogeny
- But: How representative is the GC% value you calculated above?
- And: How efficiently can it be used to describe a genome?

Problem 3: Why should we care?

- We mentioned that GC% is stable within bacterial genomes
- **But** Some areas of bacterial genomes are special
- Parts of the bacterial genome have been "horizontally" (as opposed to vertically, i.e. from their "mom") transferred from other species.

Horizontal gene transfer (HGT)

Problem 3: Stability of GC content *along* Bacterial Genomes

- Regions of "strange", or "deviating" GC% values in a given genome are red flags of HGT. The problem now is:
  - Given a bacterial genome sequence:
  - Locate regions of the genome where horizontal gene transfer may have occurred.

Problem 3: Approach

- Choose a window to scan your sequence. This will be your resolution.

- Calculate GC per window
- Try to locate GC values that deviate from the genome average

Problem 3: The core

- We basically repeat the approach for GC content but now we calculate one value for each window

In [30]:
```python
def windowGC(genomefile, window):

    import regex as re
    f=open(genomefile, 'r')

    seq = ""
    nG = nC = 0
    total = 0
    window = int(window)

    for line in f:
        x=re.match(">", line)
        if x == None:
            length=len(line)
            total=total+length
            seq=seq+line[0:length-1]
    f.close()

    step=int(window/10) # we use a 10% sliding overlap between windows
    times=int(len(seq)/step);

    GCwin = {}
    for i in range(times):
        DNA=seq[i*step:i*step+window]
        nC=DNA.count("C")
        nG=DNA.count("G")
        GCwin[i*step] = (nG+nC)/window

    return(GCwin)
```

In [38]:
```python
EcoliWGC = windowGC("files/ecoli.fa", 10000)
print(list(EcoliWGC.values())[1:10])
```

```
[0.3277, 0.328, 0.3259, 0.3229, 0.3234, 0.3226, 0.3228, 0.319, 0.318]
```

Hands-on #2:

- Get the genome sequence of St. aureus
- Implement Sliding GC
- Locate positions in the genome with extreme values of GC
- The problem is: *What do we mean by "extreme values"?** How do we define "extreme"?

Problem 3: Statistics Interlude

- Given a set/sample of values, how can we decide on whether a value could be part of that sample or not?
- In our problem: We know that the GC% of bacteria tends to be characteristic of the genome. Can we "spot" regions of the genome that bear GC% values that are *different* from that characteristic value?
- Q1: How will we define that characteristic value?
- Q2: How will we quantify the *difference* as big enough or not?

Problem 3: Theoretical basis (simplified)

- Central Limit Theorem (simplified):
  - Regardless of the underlying distribution, the mean of a large number of samples follow the normal distribution.
  - We can thus model GC values per window based on the normal distribution

Problem 3: The statistics

- We will model the "characteristic value" as the mean of GC values for all windows
- We will also calculate the standard deviation of these values to assess the variance
- We will then apply a technique called Z-transformation

Z-transformation

- Given a value x, we can compare x to a normal distribution with mean=m and standard deviation=std with the z-score: Z(x)= (x - m)/std
  Z(x) is thus the difference of x from m in units of standard deviation.
  Knowing that in a normal distribution ~99.5% of the values fall within +/-3*std a value of Z(x)>3 or Z(x)<-3 makes it highly unlikely that x is part of our distribution.

Normal Distribution

Problem 3. Predicting HGT locations

- We can now combine sliding GC content calculations with a Z-score transformation and a filtering for |Z|>=3

In [44]:
```python
def Z_GC(genomefile, window, threshold):

    import regex as re
    import numpy as np
    f=open(genomefile, 'r')

    seq = ""
    nG = nC = 0
    total = 0
    window = int(window)

    for line in f:
        x=re.match(">", line)
        if x == None:
            length=len(line)
            total=total+length
            seq=seq+line[0:length-1]
    f.close()

    step=int(window/10) # we use a 10% sliding overlap between windows
    times=int(len(seq)/step);

    GCwin = {}
    for i in range(times):
        DNA=seq[i*step:i*step+window]
        nC=DNA.count("C")
        nG=DNA.count("G")
        GCwin[i*step] = (nG+nC)/window

    GCcont =  list(GCwin.values())
    mGC=np.mean(GCcont)
    sdGC=np.std(GCcont)
    zGC=(GCcont-mGC)/sdGC
    for i in range(len(zGC)):
        if abs(zGC[i]) >= threshold:
            print(i*step, zGC[i])
```

In [45]:
```
Z_GC('files/Staaur.fa', 1000, 5)
```

```
610400 5.121776783517211
610500 5.180360319563124
610600 6.000529824205912
610700 6.088405128274782
610800 6.146988664320696
610900 5.883362752114085
611000 5.561153303861561
611100 5.443986231769735
611200 5.736903911999302
611300 5.414694463746778
611400 5.268235623631994
611500 5.180360319563124
613800 5.385402695723821
613900 5.238943855609038
614000 5.268235623631994
614100 5.473277999792691
614200 5.297527391654951
614300 5.268235623631994
615600 5.209652087586081
615700 5.443986231769735
615800 5.912654520137042
615900 6.146988664320696
616000 6.059113360251825
616100 5.795487448045215
616200 5.561153303861561
616300 5.414694463746778
616400 5.590445071884518
616500 5.268235623631994
616600 5.180360319563124
619000 5.356110927700864
619100 5.268235623631994
619200 5.180360319563124
619300 5.502569767815648
619400 5.356110927700864
619500 5.356110927700864
619600 5.092485015494254
659800 5.092485015494254
659900 5.238943855609038
660000 5.473277999792691
660100 5.971238056182955
660200 6.176280432343652
660300 6.059113360251825
660400 5.766195680022258
660500 5.590445071884518
660600 5.473277999792691
660700 5.473277999792691
660800 5.326819159677908
660900 5.121776783517211
663300 5.209652087586081
663400 5.326819159677908
663500 5.092485015494254
663600 5.531861535838605
663700 5.297527391654951
663800 5.297527391654951
663900 5.238943855609038
1687000 5.063193247471297
1687100 5.326819159677908
1687200 5.356110927700864
1687300 5.473277999792691
```

```
1687400 5.209652087586081
1687500 5.238943855609038
1687600 5.326819159677908
1690100 5.151068551540167
1690200 5.326819159677908
1690300 5.473277999792691
1690400 5.619736839907475
1690500 5.473277999792691
1690600 5.707612143976345
1690700 6.029821592228869
1690800 6.205572200366609
1690900 6.146988664320696
1691000 5.678320375953388
1691100 5.180360319563124
1691200 5.063193247471297
1691300 5.121776783517211
1691400 5.033901479448341
1802100 5.092485015494254
1802200 5.297527391654951
1802300 5.209652087586081
1802400 5.531861535838605
1802500 5.121776783517211
1802600 5.297527391654951
1802700 5.297527391654951
1805200 5.121776783517211
1805300 5.326819159677908
1805400 5.473277999792691
1805500 5.590445071884518
1805600 5.502569767815648
1805700 5.736903911999302
1805800 6.088405128274782
1805900 6.264155736412522
1806000 6.146988664320696
1806100 5.649028607930432
1806200 5.209652087586081
1806300 5.092485015494254
1975300 5.297527391654951
1975400 5.326819159677908
1975500 5.502569767815648
1975600 5.238943855609038
1975700 5.033901479448341
1975800 5.268235623631994
1978200 5.033901479448341
1978300 5.121776783517211
1978400 5.326819159677908
1978500 5.443986231769735
1978600 5.707612143976345
1978700 5.385402695723821
1978800 5.561153303861561
1978900 6.000529824205912
1979000 6.176280432343652
1979100 6.176280432343652
1979200 5.912654520137042
1979300 5.180360319563124
1979400 5.121776783517211
1979500 5.004609711425384
1980300 5.473277999792691
1980400 5.238943855609038
1980500 5.443986231769735
1980600 5.356110927700864
1980700 5.502569767815648
1980800 5.268235623631994
1980900 5.121776783517211
1981000 5.151068551540167
1981100 5.209652087586081
1981200 5.004609711425384
```

```
1981300 5.121776783517211
1981400 5.297527391654951
1981500 5.209652087586081
2764200 -5.0424667204487585
2764300 -6.067678601252243
2764400 -6.946431641940943
2764500 -7.913059986698514
```

Bonus Question

In the above output we see a lot of reported values coming from consecutive indices (i.e. coordinates that are adjacent to each other). Complete the function Z_GC() so that it reports ranges, that is genomic intervals that are supported by the **mean Z-score** of the range.

Problem 2: Revisited

- Background DNA composition has some **functional** role besides simply reflecting mutational pressures
- This means that in some cases we need to know why the local composition is guided by *other* aspects of molecular evolution. e.g. why would rRNA genes be G+C-rich even in AT-rich genomes?
- We need to find a way to control for *background nucleotide composition*

Problem 2 Revisited: Distinguishing between genomes through their sequence composition

1. Going beyond the GC content
2. Going beyond simple bases (mononucleotides, k=1)
3. Analyzing all dinucleotide frequencies of k=2

- Pseudocode:
  - For each kmer in *4^k* k-mers
  - Calculate N(kmer)
  - Create a table

Problem: How to count k-mer frequencies

- For mononucleotides we did it with a Brute Force approach. However the mononucleotides are 4. The k-mers are 4^k.
- How can we count the frequencies of k-mers?
  1. Do we need **all** k-mers?
  2. Do we need to check each k-mer at every step?
- How many calculations do we need if we answer "yes" to 1,2 above.

Solution: Hashing Strategy instead of Brute Force pattern comparisons

- Read the sequence in chunks of *k* nucleotides
- For each subsequence increment a dictionary value with the subsequence as key

Problem 2 Revisited: K-mer frequencies

In [76]:
```python
def kmers(genomefile, k):
    import regex as re

    file = open(genomefile, 'r')

    seq = ""
    kmertable = {}
    count = 0
    for line in file:
        count +=1
        if (count > 1) :
            length=len(line)
            seq=seq+line[0:length-1]

    file.close()

    seq = re.sub("[^AGCT]", "", seq)

    for i in range(len(seq)-k):
        DNA=seq[i:i+k]
        if DNA not in kmertable.keys():
            kmertable[DNA]=1
        else:
            kmertable[DNA]+=1

    kmertable = {k: float(v) / len(seq) for k, v in kmertable.items()}
    return(kmertable)
```

In [77]:
```
kmers('files/ecoli.fa', 2)
```

Out[77]:
```
{'TT': 0.07320527322513617,
 'TC': 0.05768315506124623,
 'CG': 0.0727025730253459,
 'GA': 0.057404557901131546,
 'AC': 0.05527541779900505,
 'CC': 0.059307337810664856,
 'CA': 0.07068467831701519,
 'AA': 0.0726425972478212,
 'AG': 0.051321530788210695,
 'GG': 0.05912354107308919,
 'GT': 0.05572233407668903,
 'TA': 0.04519239326568753,
 'AT': 0.06668468089661853,
 'TG': 0.07056117980738102,
 'GC': 0.08145839164311704,
 'CT': 0.05102971316100723}
```

Problem 2 Revisited: A table of *4^k* frequencies of occurrence

Base	A	T	G	C
A	0.090	0.112	0.048	0.053
T	0.095	0.090	0.064	0.053
G	0.052	0.052	0.023	0.034
C	0.066	0.048	0.026	0.023

- Values may be seen as "probabilities" of finding each k-mer in the sequence
- Can we use the notion of the probability to modify the table so that we get rid of the background nucleotide composition?

Problem 2 Revisited: Removing Background Composition

- The problem stated above persists at the level of k-mers: The background DNA composition may affect our results
- At the k-mer level we can remove the background using ratios of observed/expected frequencies
- Which is the expected frequency of a given k-mer?

Problem 2 Revisited: Observed/Expected(o/e) k-mer frequencies

- Mathematics Interlude:
  - Assume two events A, B that are linked with each other
  - We then say that A and B are dependent (or conditioned) and we have a "conditional probability" of A happening given B is also happening
  - We can think of k-mers the same way: a k-mer is more probable to occur if its constituent mono-mers are occurring
  - Bottomline: Any given k-mer's frequency of occurrence is dependent on the frequencies of occurrence of its mononucleotides. Thus:

Given a k-mer of length k the o/e-ratio frequency is defined as:

$$R[N_1N_2..N_k] = F[N_1N_2..N_k]/(F[N_1]F[N_2]..F[N_k])$$

In this way we can define a new table of modified frequencies that is independent of mono-nucleotide composition

Problem 2 Revisited: Observed/Expected K-mer frequencies

In [92]:

```
def rkmers(genomefile, k):
    import regex as re

    file = open(genomefile, 'r')

    seq = ""
    kmertable = {}

    count = 0
    for line in file:
        count +=1
        if (count > 1) :
            length=len(line)
            seq=seq+line[0:length-1]

    file.close()

    seq = re.sub("[^AGCT]", "", seq)

    # calculation of background nucleotide probability
    pnuc = {}
    pnuc['A'] = float(seq.count('A')/len(seq))
    pnuc['C'] = float(seq.count('C')/len(seq))
    pnuc['G'] = float(seq.count('G')/len(seq))
    pnuc['T'] = float(seq.count('T')/len(seq))

    for i in range(len(seq)-k):
        DNA=seq[i:i+k]
        if DNA not in kmertable.keys():
            kmertable[DNA]=1
        else:
            kmertable[DNA]+=1

    kmertable = {k: float(v) / len(seq) for k, v in kmertable.items()}

    rkmertable = kmertable
    for kmer in kmertable.keys():
        for j in range(len(kmer)):
            nuc = list(kmer)[j]
            pkmer = pkmer * pnuc[nuc]
        rkmertable[kmer]=round(kmertable[kmer]/(pkmer),3)
    return(rkmertable)
```

In [98]:
```
rkmers('files/ecoli.fa', 5)
```

Out[98]:
```
{'TTTTT': 2.882,
 'TTTTC': 2.221,
 'TTTCG': 1.345,
 'TTCGA': 0.754,
 'TCGAC': 0.794,
 'CGACC': 0.818,
 'GACCA': 1.058,
 'ACCAA': 0.978,
 'CCAAA': 0.885,
 'CAAAG': 1.15,
 'AAAGG': 1.064,
 'AAGGT': 0.9,
 'AGGTA': 0.652,
 'GGTAA': 1.418,
 'GTAAC': 0.945,
 'TAACG': 1.283,
 'AACGA': 0.959,
 'ACGAG': 0.397,
 'CGAGG': 0.458,
 'GAGGT': 0.474,
 'TAACA': 0.9,
 'AACAA': 1.429,
 'ACAAC': 1.044,
 'CAACC': 1.088,
 'AACCA': 1.408,
 'ACCAT': 1.224,
 'CCATG': 0.845,
 'CATGC': 0.938,
 'ATGCG': 1.279,
 'TGCGA': 0.855,
 'GCGAG': 0.729,
 'CGAGT': 0.411,
 'GAGTG': 0.525,
 'AGTGT': 0.525,
 'GTGTT': 0.941,
 'TGTTG': 1.439,
 'GTTGA': 1.335,
 'TTGAA': 1.18,
 'TGAAG': 1.423,
 'GAAGT': 0.941,
 'AAGTT': 0.855,
 'AGTTC': 0.93,
 'GTTCG': 0.909,
 'TTCGG': 0.909,
 'TCGGC': 1.201,
 'CGGCG': 2.004,
 'GGCGG': 1.843,
 'GCGGT': 1.58,
 'CGGTA': 1.416,
 'GGTAC': 0.707,
 'GTACA': 0.616,
 'TACAT': 0.571,
 'ACATC': 1.123,
 'CATCA': 2.128,
 'ATCAG': 1.796,
 'TCAGT': 1.149,
 'CAGTG': 1.238,
 'AGTGG': 0.812,
 'GTGGC': 1.253,
 'TGGCA': 1.863,
            'GGCAA': 1.874,
            'GCAAA': 1.993,
            'CAAAT': 1.108,
            'AAATG': 1.406,
            'AATGC': 1.469,
            'ATGCA': 1.058,
            'TGCAG': 1.442,
            'GCAGA': 1.524,
            'CAGAA': 1.646,
            'AGAAC': 0.88,
            'GAACG': 1.05,
            'AACGT': 1.164,
            'ACGTT': 1.164,
            'CGTTT': 1.579,
            'GTTTT': 2.037,
            'TTTCT': 1.533,
            'TTCTG': 1.651,
            'TCTGC': 1.494,
            'CTGCG': 1.635,
            'TGCGG': 1.43,
            'GCGGG': 1.141,
            'CGGGT': 0.983,
            'GGGTT': 0.775,
            'GGTTG': 1.105,
            'GTTGC': 1.48,
            'TTGCC': 1.926,
            'TGCCG': 2.167,
            'GCCGA': 1.189,
            'CCGAT': 1.095,
            'CGATA': 1.33,
            'GATAT': 1.356,
            'ATATT': 1.516,
            'TATTC': 1.186,
            'ATTCT': 0.909,
            'TCTGG': 1.641,
            'CTGGA': 1.684,
            'TGGAA': 1.307,
            'GGAAA': 1.504,
            'GAAAG': 1.338,
            'AAAGC': 1.609,
            'AAGCA': 1.05,
            'AGCAA': 1.445,
            'GCAAT': 1.596,
            'CAATG': 1.269,
            'ATGCC': 1.423,
            'TGCCA': 1.86,
            'GCCAG': 2.719,
            'CCAGG': 1.238,
            'CAGGC': 1.814,
            'AGGCA': 0.964,
            'GGCAG': 1.666,
            'GCAGG': 1.69,
            'CAGGG': 1.0,
            'AGGGG': 0.45,
            'GGGGC': 0.627,
            'GGGCA': 0.957,
            'CAGGT': 1.345,
            'AGGTG': 0.873,
            'GGTGG': 1.281,
            'TGGCC': 0.805,
            'GGCCA': 0.836,
            'GCCAC': 1.244,
            'CCACC': 1.264,
            'CACCG': 1.641,
            'ACCGT': 1.23,
            'CCGTC': 1.04,
            'CGTCC': 0.655,
            'GTCCT': 0.368,
            'TCCTC': 0.507,
            'CCTCT': 0.465,
            'CTCTC': 0.439,
            'TCTCT': 0.674,
            'CTCTG': 0.803,
            'CTGCC': 1.623,
            'TGCCC': 1.019,
            'GCCCC': 0.635,
            'CCCCC': 0.365,
            'CCCCG': 0.611,
            'CCCGC': 1.164,
            'CCGCC': 1.789,
            'CGCCA': 2.638,
            'GCCAA': 1.112,
            'CAAAA': 1.843,
            'AAAAT': 2.23,
            'AAATC': 1.683,
            'AATCA': 1.713,
            'ATCAC': 1.407,
            'TCACC': 1.662,
            'CACCA': 1.88,
            'CCAAC': 0.881,
            'CCATC': 1.383,
            'CATCT': 1.009,
            'ATCTG': 1.396,
            'CTGGT': 1.781,
            'TGGTA': 0.944,
            'GGTAG': 0.581,
            'GTAGC': 0.661,
            'TAGCG': 0.814,
            'AGCGA': 1.192,
            'GCGAT': 1.795,
            'CGATG': 1.496,
            'GATGA': 1.76,
            'ATGAT': 1.393,
            'TGATT': 1.682,
            'GATTG': 1.13,
            'ATTGA': 1.299,
            'TGAAA': 1.98,
            'GAAAA': 2.223,
            'AAAAA': 2.981,
            'AAAAC': 2.049,
            'AAACC': 1.515,
            'CCATT': 1.361,
            'CATTA': 1.277,
            'ATTAG': 0.444,
            'TTAGC': 0.725,
            'AGCGG': 1.355,
            'GCGGC': 1.941,
            'CGGCC': 0.662,
            'CAGGA': 1.294,
            'AGGAT': 0.773,
            'GGATG': 1.199,
            'GATGC': 1.572,
            'ATGCT': 1.1,
            'TGCTT': 1.03,
            'GCTTT': 1.656,
            'CTTTA': 1.191,
            'TTTAC': 1.409,
            'TTACC': 1.431,
            'TACCC': 0.728,
            'ACCCA': 0.745,
            'CCCAA': 0.445,
            'CCAAT': 0.805,
            'CAATA': 1.28,
            'AATAT': 1.536,
            'ATATC': 1.371,
            'TATCA': 1.26,
            'TCAGC': 2.006,
            'CAGCG': 2.496,
            'CCGAA': 0.903,
            'CGAAC': 0.889,
            'ACGTA': 0.586,
            'CGTAT': 0.788,
            'GTATT': 1.126,
            'TATTT': 1.468,
            'ATTTT': 2.206,
            'TTTTG': 1.862,
            'TTTGC': 2.012,
            'GAACT': 0.896,
            'AACTT': 0.846,
            'ACTTC': 0.95,
            'CTTCT': 0.929,
            'TCTGA': 1.013,
            'CTGAC': 1.467,
            'TGACG': 1.39,
            'GACGG': 0.999,
            'ACGGG': 0.74,
            'CGGGA': 0.868,
            'GGGAC': 0.313,
            'GGACT': 0.401,
            'GACTC': 0.39,
            'ACTCG': 0.426,
            'CTCGC': 0.748,
            'TCGCC': 1.86,
            'CGCCG': 1.97,
            'GCCGC': 1.897,
            'CGCCC': 1.175,
            'GCCCA': 0.893,
            'CCCAG': 1.007,
            'CCAGC': 2.712,
            'CAGCC': 1.629,
            'AGCCG': 1.038,
            'GCCGG': 1.395,
            'CCGGG': 1.053,
            'GGGAT': 0.895,
            'GGATT': 1.05,
            'GATTT': 1.614,
            'ATTTC': 1.556,
            'TTTCC': 1.539,
            'TTCCG': 1.335,
            'TCCGC': 1.169,
            'CCGCT': 1.382,
            'CGCTG': 2.521,
            'GCTGG': 2.655,
            'CTGGC': 2.709,
            'GGCAC': 0.995,
            'GCACA': 0.909,
            'CACAA': 0.879,
            'ACAAT': 0.983,
            'CAATT': 0.997,
            'AATTG': 0.992,
            'AAACT': 1.186,
            'ACTTT': 1.084,
            'CTTTC': 1.306,
            'TTCGT': 0.909,
            'TCGTC': 0.892,
            'CGTCG': 1.052,
            'GTCGA': 0.775,
            'ACCAG': 1.786,
            'AGGAA': 0.945,
            'GGAAT': 0.944,
            'GAATT': 0.959,
            'AATTT': 1.56,
            'ATTTG': 1.127,
            'AAATA': 1.455,
            'AATAA': 1.644,
            'ATAAA': 1.872,
            'TAAAA': 1.659,
            'AAACA': 1.5,
            'AACAT': 1.168,
            'ACATG': 0.64,
            'CATGT': 0.651,
            'ATGTC': 0.821,
            'TGTCC': 0.572,
            'TCCTG': 1.266,
            'CCTGC': 1.622,
            'CTGCA': 1.414,
            'TGCAT': 1.032,
            'GCATG': 0.919,
            'CATGG': 0.845,
            'ATGGC': 1.594,
            'GGCAT': 1.419,
            'GCATC': 1.54,
            'CAGTT': 1.652,
            'AGTTT': 1.162,
            'GTTTG': 1.278,
            'TTTGT': 1.337,
            'TTGTT': 1.429,
            'GTTGG': 0.899,
            'TTGGG': 0.455,
            'TGGGG': 0.625,
            'GCAGT': 1.213,
            'AGTGC': 0.842,
            'GTGCC': 1.022,
            'GCCCG': 1.17,
            'CCCGG': 1.05,
            'CCGGA': 1.461,
            'CGGAT': 1.24,
            'GGATA': 1.011,
            'GATAG': 0.633,
            'ATAGC': 0.799,
            'TAGCA': 0.463,
            'AGCAT': 1.06,
            'ATCAA': 1.594,
            'TCAAC': 1.305,
            'CAACG': 1.332,
            'AACGC': 1.752,
            'ACGCT': 1.33,
            'GCTGC': 1.845,
            'TGCGC': 1.9,
            'GCGCT': 1.598,
            'GCTGA': 1.973,
            'CTGAT': 1.799,
            'GCCGT': 1.324,
            'CCGTG': 0.896,
            'CGTGG': 0.999,
            'TGGCG': 2.635,
            'GGCGA': 1.87,
            'CGAGA': 0.549,
            'GAGAA': 0.934,
            'AGAAA': 1.583,
            'AATGT': 0.9,
            'TGTCG': 1.013,
            'TCGAT': 1.103,
            'CGATC': 1.115,
            'GATCG': 1.087,
            'ATCGC': 1.783,
            'GCCAT': 1.593,
            'ATTAT': 1.382,
            'TTATG': 1.061,
            'TATGG': 0.834,
            'GGCCG': 0.649,
            'CCGGC': 1.417,
            'GGCGT': 1.615,
            'GCGTG': 1.292,
            'CGTGT': 0.543,
            'TGTTA': 0.897,
            'GTTAG': 0.455,
            'TTAGA': 0.286,
            'TAGAA': 0.386,
            'AGAAG': 0.908,
            'GAAGC': 1.236,
            'AAGCG': 1.364,
            'AGCGC': 1.642,
            'GCGCG': 1.758,
            'CGCGT': 1.191,
            'GTGGT': 1.273,
            'TGGTC': 1.053,
            'GGTCA': 1.198,
            'GTCAC': 0.892,
            'TCACA': 0.668,
            'CGTTA': 1.278,
            'GTTAC': 0.938,
            'TACCG': 1.433,
            'CCGTT': 1.305,
            'GTTAT': 1.257,
            'TTATC': 1.714,
            'TATCG': 1.341,
            'ATCGA': 1.087,
            'GATCC': 0.879,
            'ATCCG': 1.274,
            'TCCGG': 1.491,
            'CCGGT': 1.457,
            'CGGTC': 0.888,
            'GGTCG': 0.82,
            'TCGAA': 0.732,
            'CGAAA': 1.344,
            'AACTG': 1.63,
            'ACTGC': 1.211,
            'CTGCT': 1.507,
            'TGCTG': 2.524,
            'GTGGG': 0.658,
            'TGGGT': 0.734,
            'GGGTC': 0.38,
            'GTCAT': 1.129,
            'TCATT': 1.295,
            'ATTAC': 1.176,
            'TACCT': 0.626,
            'ACCTC': 0.481,
            'CCTCG': 0.462,
            'CTCGA': 0.48,
            'CGAAT': 0.811,
            'GAATC': 0.82,
            'AATCT': 0.891,
            'ATCTA': 0.32,
            'TCTAC': 0.442,
            'CTACC': 0.562,
            'CGTTG': 1.347,
            'TTGAT': 1.67,
            'TGATA': 1.278,
            'TATTG': 1.279,
            'ATTGC': 1.622,
            'TTGCT': 1.438,
            'CTGAA': 1.85,
            'TGAAT': 1.326,
            'AATCC': 1.039,
            'ATCCA': 1.202,
            'TCCAC': 0.879,
            'CACCC': 0.735,
            'ACCCG': 0.959,
            'CCGTA': 0.831,
            'TTGCG': 1.426,
            'CGGCA': 2.154,
            'GCAAG': 0.831,
            'CAAGC': 0.668,
            'AAGCC': 1.123,
            'CCGCA': 1.459,
            'CGCAT': 1.296,
            'GCATT': 1.474,
            'CATTC': 1.041,
            'ATTCC': 0.919,
            'CGGCT': 1.045,
            'GGCTG': 1.613,
            'TGACC': 1.207,
            'ACCAC': 1.275,
            'CCACA': 0.959,
            'CACAT': 0.792,
            'ATGGT': 1.248,
            'TGGTG': 1.885,
            'GGTGC': 1.212,
            'GTGCT': 0.934,
            'TGATG': 2.133,
            'GATGG': 1.387,
            'TGGCT': 1.375,
            'TGGTT': 1.401,
            'GGTTT': 1.483,
            'GTTTC': 1.295,
            'TTTCA': 1.955,
            'TTCAC': 1.408,
            'TCACT': 0.914,
            'CACTG': 1.263,
            'GTAAT': 1.168,
            'TAATG': 1.249,
            'AATGA': 1.264,
            'ATGAA': 1.537,
            'AAAAG': 1.463,
            'AAGGC': 0.997,
            'AGGCG': 1.254,
            'CGAGC': 0.63,
            'GAGCT': 0.531,
            'AGCTG': 1.18,
            'GGTTC': 0.783,
            'GTTCT': 0.884,
            'CTGGG': 0.998,
            'TGGGA': 0.551,
            'GGACG': 0.636,
            'GACGC': 1.205,
            'ACGCA': 1.199,
            'CGCAA': 1.418,
            'GCAAC': 1.492,
            'AACGG': 1.294,
            'ACGGT': 1.198,
            'CGGTT': 1.369,
            'GTTCC': 0.907,
            'TCCGA': 0.424,
            'CCGAC': 0.835,
            'CGACT': 0.624,
            'GACTA': 0.353,
            'ACTAC': 0.473,
            'CTACT': 0.344,
            'TACTC': 0.495,
            'ACTCC': 0.514,
            'CTCCG': 0.603,
            'CGGTG': 1.613,
            'GGCCT': 0.704,
            'GCCTG': 1.76,
            'CCTGT': 0.973,
            'CTGTT': 1.556,
            'TGTTT': 1.482,
            'GTTTA': 1.111,
            'TTACG': 1.058,
            'TACGC': 1.103,
            'ACGCG': 1.186,
            'CGCGC': 1.737,
            'GCGCC': 1.706,
            'CGATT': 1.061,
            'ATTGT': 0.972,
            'GAGAT': 0.835,
            'AGATC': 0.829,
            'GATCT': 0.785,
            'TGGAC': 0.562,
            'ACGGA': 0.745,
            'GATGT': 1.113,
            'ATGTT': 1.15,
            'TTGAC': 0.745,
            'GGTGT': 0.984,
            'TTTAT': 1.837,
            'TTATA': 0.681,
            'TATAC': 0.444,
            'ATACC': 1.03,
            'ACCTG': 1.29,
            'CCGCG': 1.203,
            'GCGTC': 1.189,
            'CGTCA': 1.388,
            'GTCAG': 1.456,
            'TCAGG': 1.502,
            'CCCGA': 0.603,
            'AGGTT': 0.891,
            'GTTGT': 1.093,
            'AAGTC': 0.591,
            'AGTCG': 0.594,
            'TCCTA': 0.137,
            'CCTAT': 0.301,
            'CTATC': 0.605,
            'GGAAG': 1.199,
            'ATGGA': 0.848,
            'TGGAG': 0.646,
            'GGAGC': 0.569,
            'AGCTT': 0.7,
            'TTCTT': 1.284,
            'TCTTA': 0.462,
            'CTTAC': 0.571,
            'TTACT': 0.852,
            'TACTT': 0.566,
            'CTTCG': 0.994,
            'GGCGC': 1.709,
            'CGCTA': 0.767,
            'GCTAA': 0.714,
            'CTAAA': 0.559,
            'TAAAG': 1.198,
            'AAAGT': 1.085,
            'TCTTC': 1.377,
            'CTTCA': 1.437,
            'ACCCC': 0.432,
            'CGCAC': 1.041,
            'GCACC': 1.244,
            'CCCCA': 0.638,
            'CCCAT': 0.731,
            'CATCG': 1.488,
            'CCAGT': 1.659,
            'TTCCA': 1.357,
            'TCCAG': 1.738,
            'CCAGA': 1.635,
            'CAGAT': 1.403,
            'ATCCC': 0.91,
            'TCCCT': 0.528,
            'CCCTT': 0.583,
            'CCTTG': 0.515,
            'CTTGC': 0.834,
            'TGCCT': 0.933,
            'CCTGA': 1.496,
            'GATTA': 1.161,
            'ATTAA': 1.518,
            'TTAAA': 1.381,
            'AATAC': 1.096,
            'ACCGG': 1.453,
            'CGGAA': 1.354,
            'GAAAT': 1.581,
            'TCCCC': 0.629,
            'CCAAG': 0.252,
            'AGCAC': 1.049,
            'GTACG': 0.68,
            'CGCTC': 0.839,
            'GCTCA': 0.964,
            'CTCAT': 0.755,
            'CATTG': 1.29,
            'ATTGG': 0.798,
            'TTGGT': 0.977,
            'CGTGA': 1.052,
            'GTGAT': 1.384,
            'GAAGA': 1.376,
            'AAGAC': 0.629,
            'AGACG': 0.799,
            'GACGA': 0.91,
            'ACGAA': 0.873,
            'AATTA': 1.098,
            'GTCAA': 0.746,
            'TCAAG': 0.531,
            'CAAGG': 0.526,
            'AAGGG': 0.577,
            'AGGGC': 0.636,
            'CATTT': 1.446,
            'TCCAA': 0.275,
            'CAATC': 1.134,
            'GAATA': 1.177,
            'ATAAC': 1.254,
            'TGTTC': 1.251,
            'GTTCA': 1.313,
            'TTCAG': 1.85,
            'AGCGT': 1.337,
            'GCGTT': 1.743,
            'GGCCC': 0.432,
            'CGGGG': 0.597,
            'GGGGA': 0.63,
            'AGGGA': 0.5,
            'TTGGC': 1.106,
            'CGTCT': 0.788,
            'GTCTT': 0.65,
            'TCTTT': 1.383,
            'CTTTG': 1.193,
            'TTGCA': 1.126,
            'GCAGC': 1.904,
            'TGTCA': 0.916,
            'TCACG': 1.044,
            'CACGC': 1.304,
            'CCCGT': 0.77,
            'TCCGT': 0.745,
            'ATCAT': 1.482,
            'TCATC': 1.779,
            'ATCTT': 1.059,
            'CTTCC': 1.165,
            'ATACA': 0.673,
            'TACAG': 0.801,
            'ACAGT': 0.75,
            'CAGTA': 1.082,
            'AGTAT': 0.588,
            'GTATC': 0.892,
            'TGCGT': 1.175,
            'CGTTC': 1.046,
            'GCGAC': 1.121,
            'GACTG': 0.824,
            'ACTGT': 0.774,
            'CTGTG': 0.788,
            'TGTGT': 0.535,
            'GTGTG': 0.627,
            'TGTGC': 0.907,
            'GTGCG': 1.041,
            'TGAAC': 1.269,
            'CGGGC': 1.096,
            'AAGAG': 0.78,
            'AGAGT': 0.528,
            'GAGTT': 0.741,
            'TTCTA': 0.373,
            'CCTGG': 1.223,
            'GGAAC': 0.871,
            'ACTGA': 1.153,
            'AAAGA': 1.384,
            'AAGAA': 1.289,
            'GAAGG': 0.992,
            'AGGCT': 0.717,
            'GGCTT': 1.135,
            'GCTTA': 0.718,
            'TACTG': 1.092,
            'ACTGG': 1.648,
            'GAGCC': 0.459,
            'GGTGA': 1.655,
            'GTGAC': 0.903,
            'ACGGC': 1.282,
            'TATCT': 0.909,
            'ATCTC': 0.807,
            'TCTCG': 0.559,
            'CTCGG': 0.436,
            'TCGGT': 1.034,
            'GTAGG': 0.297,
            'TAGGT': 0.272,
            'GGTAT': 1.069,
            'GTATG': 0.68,
            'TATGC': 0.887,
            'GCGCA': 1.924,
            'CACCT': 0.856,
            'ACCTT': 0.886,
            'CCTTA': 0.493,
            'TACGT': 0.575,
            'ACGTG': 0.607,
            'GGATC': 0.831,
            'GCGAA': 1.534,
            'AAATT': 1.573,
            'AATTC': 0.936,
            'CAACA': 1.44,
            'ACATT': 0.909,
            'TTGTC': 0.943,
            'GTCGC': 1.111,
            'TGCTC': 0.94,
            'CTCAG': 0.626,
            'CTCAA': 0.794,
            'TCAAT': 1.295,
            'TCTGT': 0.822,
            'CTGTC': 0.795,
            'GTCGT': 0.775,
            'TCGTG': 0.689,
            'ACGAT': 1.001,
            'CCACT': 0.816,
            'CGTGC': 0.9,
            'ACTCA': 0.71,
            'TCAGA': 1.027,
            'AGATG': 1.01,
            'GCTGT': 1.358,
            'TTCAA': 1.174,
            'ACCGA': 1.054,
            'GATCA': 1.174,
            'GGTTA': 1.072,
            'CGAAG': 1.015,
            'AAGTG': 0.879,
            'TTGTG': 0.884,
            'TGTGA': 0.666,
            'GTCGG': 0.818,
            'GAGCA': 0.922,
            'CAACT': 0.863,
            'CAGCA': 2.659,
            'AAGCT': 0.733,
            'AGAAT': 0.907,
            'TAAAC': 1.146,
            'ACATA': 0.637,
            'CATAT': 0.885,
            'GACTT': 0.603,
            'ACTTA': 0.487,
            'GTGTC': 0.54,
            'TGTCT': 0.592,
            'GTCTG': 1.154,
            'TGCTA': 0.454,
            'CTAAC': 0.438,
            'TAACT': 0.667,
            'AACTC': 0.734,
            'GCACT': 0.87,
            'CTCAC': 0.616,
            'ATGTA': 0.552,
            'TGTAC': 0.606,
            'GCCTT': 0.97,
            'CTTAA': 0.722,
            'TTAAT': 1.476,
            'TAATC': 1.136,
            'AGCCA': 1.35,
            'AGAGC': 0.655,
            'TTTAA': 1.369,
            'TCGGG': 0.596,
            'GGGCG': 1.154,
            'CGCTT': 1.377,
            'TAATT': 1.058,
            'ATTCG': 0.843,
            'TTCGC': 1.503,
            'CGCCT': 1.224,
            'GCCTC': 0.533,
            'CTCGT': 0.404,
            'GTGAA': 1.414,
            'GAACC': 0.757,
            'AACCC': 0.787,
            'TGACT': 0.745,
            'CTGTA': 0.808,
            'GTACT': 0.605,
            'TGTGG': 0.919,
            'ATATG': 0.881,
            'TTCCT': 0.923,
            'CGCGA': 1.225,
            'CCACG': 0.998,
            'CACGT': 0.591,
            'ACGCC': 1.638,
            'GAACA': 1.204,
            'ACAAA': 1.313,
            'AGGCC': 0.713,
            'AACAC': 0.892,
            'ACACC': 0.936,
            'TGGAT': 1.18,
            'TACTA': 0.249,
            'TACCA': 0.922,
            'AGTTG': 0.912,
            'GCGGA': 1.184,
            'AATCG': 1.123,
            'TCGCG': 1.225,
            'CGCGG': 1.22,
            'GCGTA': 1.103,
            'CGTAA': 1.071,
            'GTAAA': 1.414,
            'TAAAT': 1.212,
            'CTCTA': 0.265,
            'TCTAT': 0.44,
            'CTATG': 0.463,
            'TATGA': 0.753,
            'ATGAC': 1.129,
            'TGACA': 0.896,
            'GACAC': 0.533,
            'GGGCT': 0.697,
            'TTATT': 1.661,
            'TTGAG': 0.78,
            'TGAGA': 0.588,
            'AACCT': 0.87,
            'TGCAA': 1.146,
            'TTCTC': 0.923,
            'TCTCC': 0.681,
            'TCGCT': 1.207,
            'CTTAT': 0.667,
            'TATAT': 0.65,
            'CAAGT': 0.354,
            'AGTTA': 0.687,
            'TAGAC': 0.221,
            'CATGA': 0.993,
            'CCGAG': 0.452,
            'GAGGC': 0.542,
            'CACAC': 0.619,
            'ACACT': 0.542,
            'GGGAA': 0.905,
            'AATGG': 1.308,
            'ATGGG': 0.68,
            'AACCG': 1.374,
            'CGGAC': 0.646,
            'GGACC': 0.332,
            'GACCC': 0.377,
            'TGATC': 1.149,
            'ATGTG': 0.74,
            'GTAAG': 0.59,
            'TAAGC': 0.774,
            'AGCTA': 0.389,
            'GCTAT': 0.807,
            'CTATT': 0.611,
            'GATTC': 0.81,
            'GCTCG': 0.624,
            'GAAAC': 1.278,
            'AAACG': 1.526,
            'GACGT': 0.772,
            'CGCAG': 1.663,
            'CAGAG': 0.822,
            'TTAAC': 1.329,
            'GAGGG': 0.339,
            'AGGGT': 0.514,
            'GGGTG': 0.762,
            'CTTTT': 1.497,
            'TTTTA': 1.639,
            'CACAG': 0.8,
            'ACAGC': 1.338,
            'CAGCT': 1.194,
            'AGCTC': 0.571,
            'CGACG': 1.074,
            'CTCTT': 0.785,
            'AAGGA': 0.631,
            'TATGT': 0.593,
            'AAGAT': 1.07,
            'AGATT': 0.916,
            'GTGGA': 0.837,
            'TCAAA': 1.167,
            'AGTGA': 0.921,
            'GCCCT': 0.64,
            'CCCTG': 0.999,
            'CCTTC': 0.987,
            'CTATA': 0.385,
            'TATAG': 0.37,
            'TAGCC': 0.583,
            'CACTA': 0.46,
            'ACTAT': 0.515,
            'TATTA': 0.959,
            'TTACA': 0.722,
            'GCTAC': 0.642,
            'CTACG': 0.486,
            'CGTAC': 0.704,
            'GTACC': 0.713,
            'ACCCT': 0.483,
            'CCCTC': 0.335,
            'TCTCA': 0.579,
            'TCATG': 1.022,
            'TTAGG': 0.247,
            'TAGGA': 0.126,
            'AGGAG': 0.484,
            'GGAGT': 0.495,
            'GAGTC': 0.382,
            'AGTCT': 0.414,
            'GACAT': 0.785,
            'GTTAA': 1.314,
            'GCTTC': 1.21,
            'ATGAG': 0.737,
            'TGAGC': 0.903,
            'GAGCG': 0.842,
            'TTTGA': 1.188,
            'GTGCA': 0.866,
            'TCGGA': 0.408,
            'CGGAG': 0.6,
            'GGAGA': 0.701,
            'TGTAG': 0.469,
            'GTAGT': 0.511,
            'TAGTC': 0.316,
            'AGTCA': 0.714,
            'CACGG': 0.89,
            'TGAGG': 0.505,
            'AGAGA': 0.706,
            'GAGAC': 0.329,
            'AGACA': 0.587,
            'ATTCA': 1.305,
            'CAGTC': 0.823,
            'GTCTC': 0.338,
            'GATAA': 1.735,
            'ATAAG': 0.73,
            'ATCGT': 1.067,
            'TCGTT': 1.02,
            'GGGAG': 0.459,
            'GCTTG': 0.676,
            'CTTGG': 0.232,
            'ACTCT': 0.511,
            'ATCGG': 1.094,
            'CTTAG': 0.212,
            'TAGGC': 0.283,
            'GGCTC': 0.468,
            'GCTCC': 0.585,
            'CTCCA': 0.658,
            'GAATG': 1.031,
            'CACTC': 0.542,
            'TGGGC': 0.885,
            'TTGGA': 0.287,
            'TTCAT': 1.562,
            'TACGA': 0.507,
            'ACGAC': 0.776,
            'CGACA': 1.012,
            'GACAA': 0.922,
            'TCTTG': 0.542,
            'TGAGT': 0.704,
            'TATCC': 1.046,
            'GGCTA': 0.609,
            'GCACG': 0.902,
            'TTCCC': 0.912,
            'TCCCG': 0.86,
            'AGCCT': 0.697,
            'CTGAG': 0.562,
            'CCCTA': 0.176,
            'CCTAC': 0.28,
            'GCTCT': 0.648,
            'ACAAG': 0.476,
            'GATAC': 0.842,
            'ACCGC': 1.603,
            'GGGTA': 0.736,
            'TAAGA': 0.454,
            'AACTA': 0.382,
            'TCATA': 0.805,
            'ATACG': 0.767,
            'TACGG': 0.786,
            'CACGA': 0.661,
            'GAGTA': 0.49,
            'AGTAC': 0.597,
            'ACTAA': 0.402,
            'CTACA': 0.442,
            'TACAA': 0.578,
            'AGCAG': 1.568,
            'AGGTC': 0.584,
            'TAACC': 1.087,
            'GGGGT': 0.415,
            'GACCT': 0.581,
            'ATCCT': 0.784,
            'GGGCC': 0.441,
            'ATTTA': 1.21,
            'CCATA': 0.844,
            'GACCG': 0.908,
            'CATCC': 1.208,
            'ATACT': 0.619,
            'AGTCC': 0.398,
            'GTCCA': 0.6,
            'CAAGA': 0.539,
            'TCCAT': 0.884,
            'TCGCA': 0.815,
            'GGTCT': 0.553,
            'TGCAC': 0.843,
            'CAAAC': 1.271,
            'AACAG': 1.562,
            'ACAGA': 0.828,
            'CAGAC': 1.179,
            'GTGAG': 0.627,
            'CGTAG': 0.52,
            'CTTGT': 0.498,
            'GAGAG': 0.459,
            'GTAGA': 0.465,
            'GCATA': 0.881,
            'CATAA': 1.093,
            'AGTAA': 0.877,
            'ACGTC': 0.743,
            'GTCTA': 0.199,
            'AGCCC': 0.718,
            'GACAG': 0.795,
            'ACAGG': 1.018,
            'TAGAG': 0.294,
            'TCGTA': 0.541,
            'CACTT': 0.873,
            'CTCCC': 0.443,
            'TCCCA': 0.562,
            'GCTAG': 0.076,
            'CTAGT': 0.06,
            'GTCCC': 0.335,
            'AGATA': 0.919,
            'ATAGG': 0.287,
            'ATAAT': 1.348,
            'GTCCG': 0.659,
            'GGAGG': 0.426,
            'GAGGA': 0.504,
            'AAGTA': 0.548,
            'TTTGG': 0.909,
            'AGACC': 0.57,
            'TCGAG': 0.466,
            'TAAGG': 0.524,
            'CATAC': 0.716,
            'GTGTA': 0.49,
            'CCTTT': 1.058,
            'CTAAT': 0.401,
            'GGGGG': 0.378,
            'TCTAA': 0.239,
            'GTATA': 0.45,
            'TCCTT': 0.613,
            'TTTAG': 0.546,
            'ATAGT': 0.528,
            'TAGTT': 0.427,
            'CTCCT': 0.475,
            'CCCAC': 0.65,
            'TAGAT': 0.354,
            'TAATA': 0.926,
            'AATAG': 0.585,
            'TAGGG': 0.178,
            'GGACA': 0.533,
            'AGGAC': 0.38,
            'CCTCA': 0.531,
            'TTAGT': 0.389,
            'TAGTA': 0.283,
            'TTAAG': 0.736,
            'ATAGA': 0.464,
            'CCCCT': 0.45,
            'CATAG': 0.488,
            'ATATA': 0.656,
            'GGTCC': 0.339,
            'TATAA': 0.708,
            'AGACT': 0.406,
            ...}
```

Problem 2 Revisited: A table of o/e 4^k frequencies of occurrence

Base	A	G	C	T
A	0.800	0.997	0.878	0.949
G	0.848	0.799	1.174	0.957
C	0.946	0.955	0.848	1.252
T	1.183	0.872	0.946	0.841

- Notice how values now go >1. What does this mean?
- How is this table better (or not) than the previous one?

Genomic Signatures: Comparing o/e k-mer composition

- Genomic Signatures are defined as the table of o/e k-mers for a given genome
- We can use these tables to analyze distances between genomes. (Hint: even eukaryote genomes!)

Hands-on #3:

- Get chromosome 1 from (human, mouse, fly, worm, yeast)
- Use a genomic signature approach to cluster genomic signatures from different genomes
- Calculate the distance between rho_xy(p) and rho_xy(s) to create a table of distances.

Problem 4: Finding the DNA Replication in a bacterial genome

- Bacterial Genomes replicated their genome starting at one point and proceeding towards the opposite point in the circular genome from both directions.
- Bacterial genomes also have a particular distribution of nucleotides along their genome
- The difference of A-T (and G-C) complementary nucleotides goes through a sort of "phase transition" that splits the genome approximately in half.
- Do you know what this split is?
- Do you know why it is so?

How is this related to Sequence Analysis?

- Due to the pioneering work of E. Chargaff we know that A~T and G~C in **single-stranded DNA**
- We know that this holds for all complete genomes except very few exceptions
- The exceptions are the few genomes that **do not** replicate symmetrically

- DNA-strand parity:
    - Strand X is replicated in-continuously
    - Accumulates more substitutions
    - If substitutions are biased the strand will guide the change in both strands through base-pairing

Genome of Staphylococcus aureus

Approaching the problem

- We thus expect (and observe) the parity to be violated and that this violation occurs symmetrically on either side of the OriC
- We are looking for a way to locate this *transition* in the parity violation
- We thus need:
    - A measure of the parity
    - A way to monitor this measure along the genome
    - A way to locate abrupt changes in its values

Breaking the problem into pieces

1. Analyze the DNA composition *along* the genome
2. Calculate a quantity that will be informative
3. Create a condition that will test the location of the Ori

- Pseudocode: Given a bacterial genome:
    - Count nucleotides in windows of N base pairs
    - Calculate the scaled AT-skew as (A-T)/(A+T)
    - Create an array of the skew values along the genome
    - Locate the transition point

Problem 4: Parity Measure Implementation

In [7]:
```python
f=open('files/Staaur.fa', 'r')

seq = ""

for line in f:
    x=re.match(">", line)
    if x == None:
        length=len(line)
        seq=seq+line[0:length-1]
f.close()

window=1000
step=100
times=int(len(seq)/step)

ATparity = []
for i in range(times):
    DNA=seq[i*step:i*step+window]
    A=DNA.count("A")
    T=DNA.count("T")
    C=DNA.count("C")
    G=DNA.count("G")
    ATparity.append(float(A-T)/float(A+T))

for k in range(10):
    print(k,":",ATparity[k])
```

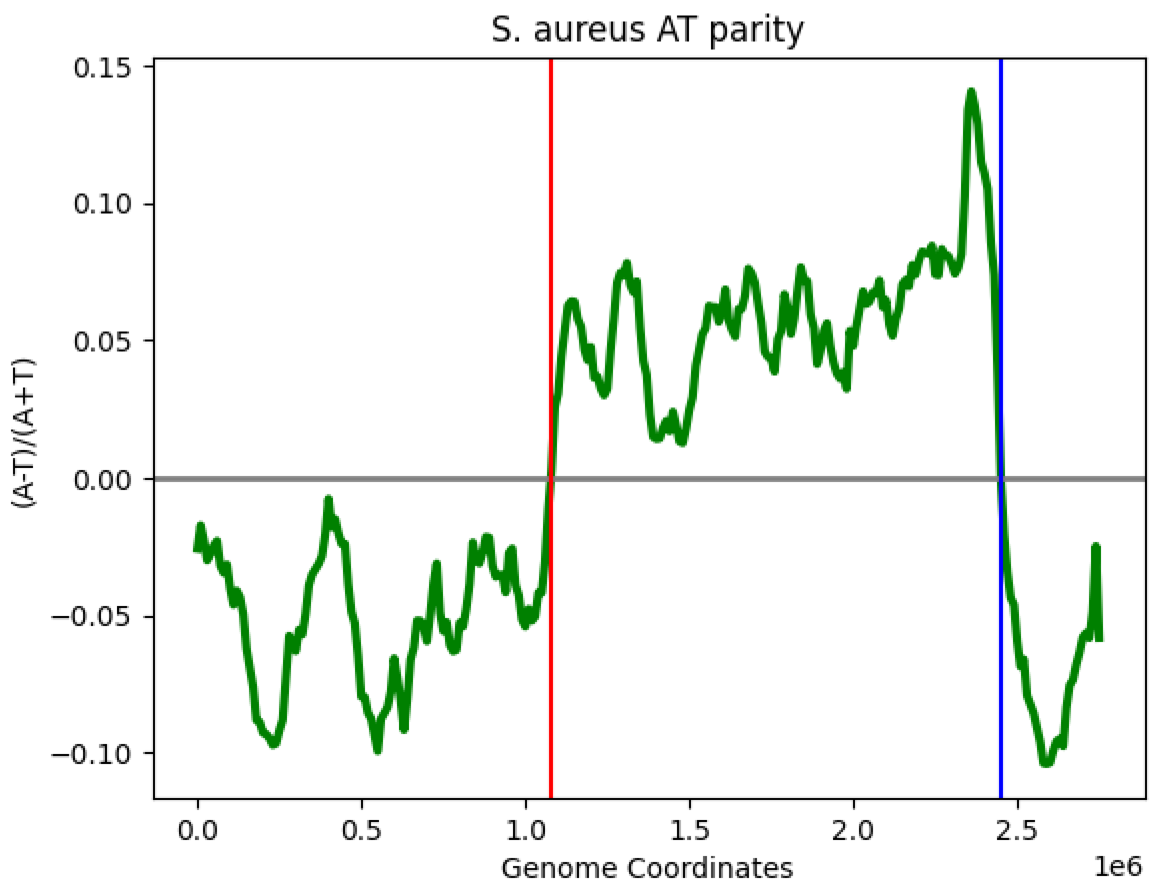
```
0 : -0.06744868035190615
1 : -0.08394698085419734
2 : -0.061946902654867256
3 : -0.061946902654867256
4 : -0.08272859216255443
5 : -0.08613138686131387
6 : -0.07759882869692533
7 : -0.11627906976744186
8 : -0.12954876273653565
9 : -0.1819505094614265
```

Problem 4: Plotting the Values

In [39]:

```
import matplotlib.pyplot as plt

import matplotlib.pyplot as plt
import regex as re

f = open('files/Staaur.fa', 'r')
seq = ""
total = 0
A=T=G=C=[]
times=0;
for line in f:
        x=re.match(">", line)
        if x == None:
                length=len(line)
                total=total+length
                seq=seq+line[0:length-1]
f.close()

x=[]
ATparity=[]
window=100000
step=10000
times=int(len(seq)/step);
for i in range(times):
    x.append(i*step)
    DNA=seq[i*step:i*step+window]
    A=DNA.count("A")
    T=DNA.count("T")
    C=DNA.count("C")
    G=DNA.count("G")
    ATparity.append(float(A-T)/float(A+T))

# plotting points as a scatter plot
plt.plot(x, ATparity, color= "green", linewidth = 3.0)
plt.axhline(y = 0, color = 'grey', linewidth = 2.0)

# plotting vertical lines at Ori and Ter
plt.axvline(x = 1080000, color = 'red')
plt.axvline(x = 2450000, color = 'blue')

#plt.scatter(x, ATparity, color= "green")

# x-axis label
plt.xlabel('Genome Coordinates')
# frequency label
plt.ylabel('(A-T)/(A+T)')
# plot title
plt.title('S. aureus AT parity')
# showing legend
#plt.legend()
```

```python
# function to show the plot
plt.show()
```



Problem 4: Locating the breakpoint(s)

- Not a simple problem. In fact one (breakpoint detection) for which research is ongoing in many fields
- Things you could try:
  - Using derivation (checking the difference between each value and the previous one)
  - Density-based approaches: Trying to locate the region around which changes in the sign occur more robustly (i.e. given many different points around it)

Concept. Binary Searches (revisit this!)

- Let's think of a simpler problem first: Suppose you are given a quadratic equation: f(x)=ax**2+bx+c and you are asked to locate a root of the equation in an interval [k,m].
- How would you proceed?
- A fast and efficient way is to start by checking the values f(k) and f(m). If their product in f(k)f(m)<0 this means that the function "crosses" the x-axis at some point between k and m. (This stems from Lagrange's mean value theorem). The question is how then can we locate that point?
- What we have in our hands is quite similar. We know that AT (or GC) parity values are expected to change their sign at two points in the genome (coinciding with the origin and terminus of replication). We need a way to locate this sign-transitions in a fast and accurate manner.

Take a pause and think.

1. How would you do it with an exhaustive (brute-force) approach?
2. How would you try to do it with a divide-and-conquer approach

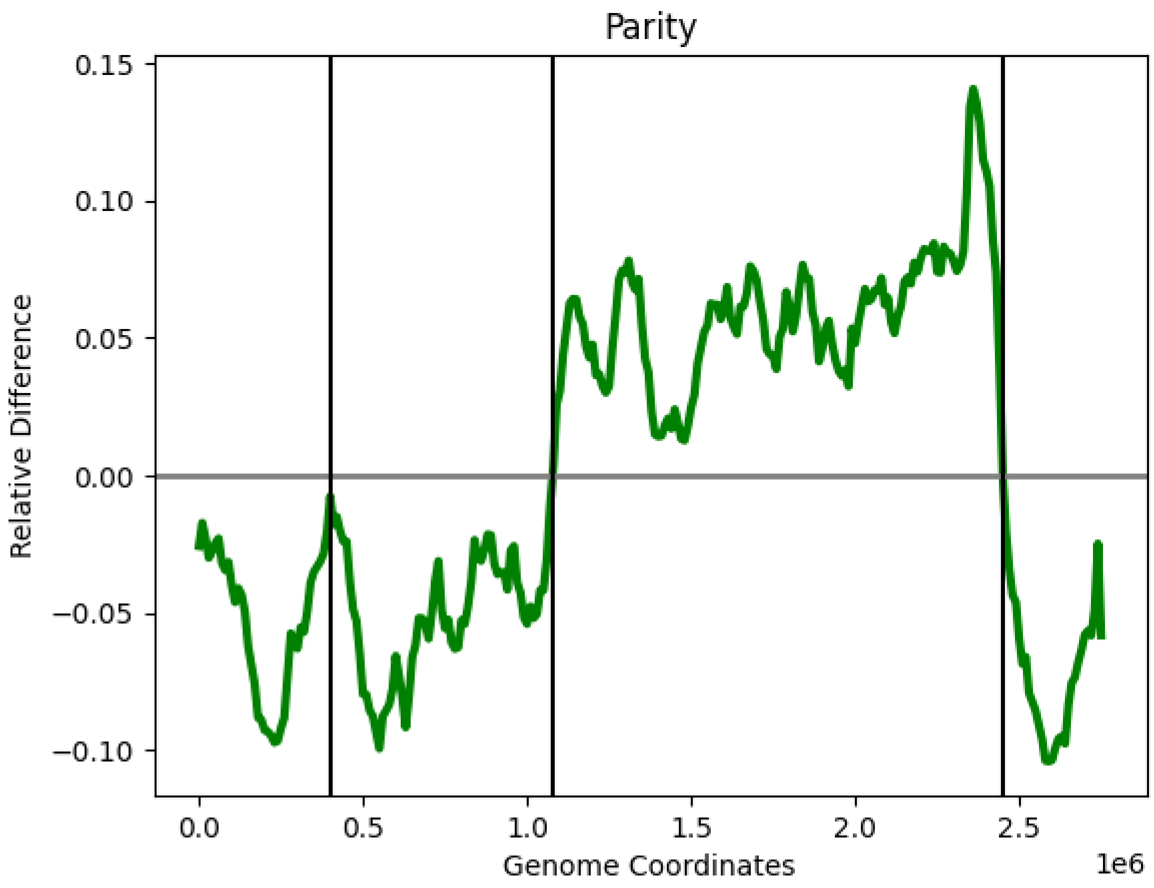
Following, we show a function that takes as input the genomefile, a window of parity calculation, the type of parity (AT or GC) and one additional parameter, which we call "threshold" and which we will use to assess the proximity to a transition point.

In [43]:
```
### A brute-force with a threshold

def bruteParity(genomefile, window, type, threshold):

    import matplotlib.pyplot as plt
    import regex as re

    f = open(genomefile, 'r')
    seq = ""
    total = 0
    A=T=G=C=[]
    times=0;
    for line in f:
        x=re.match(">", line)
        if x == None:
            length=len(line)
            total=total+length
            seq=seq+line[0:length-1]
    f.close()

    x = []
    parity = []
    transitions = []
    step = int(window/10)
    times = int(len(seq)/step);
    for i in range(times):
        x.append(i*step)
        DNA=seq[i*step:i*step+window]
        A=DNA.count("A")
        T=DNA.count("T")
        C=DNA.count("C")
        G=DNA.count("G")
        if type == "AT":
            parity.append(float(A-T)/float(A+T))
            if (abs(float(A-T)/float(A+T)) < threshold):
                transitions.append(i*step)
        if type == "GC":
            parity.append(float(G-C)/float(G+C))
            if (abs(float(G-C)/float(G+C)) < threshold):
                transitions.append(i*step)

    # Plot Parity and Predicted Transitions
    plt.plot(x, parity, color= "green", linewidth = 3.0)
    plt.axhline(y = 0, color = 'grey', linewidth = 2.0)

    # plotting vertical lines at Ori and Ter
    for site in transitions:
        plt.axvline(x = site, color = 'black')

    #plt.scatter(x, parity, color= "green")

    # x-axis label
    plt.xlabel('Genome Coordinates')
    # frequency label
    plt.ylabel('Relative Difference')
    # plot title
    plt.title('Parity')
    # showing legend
    #plt.legend()

    # function to show the plot
    plt.show()

    return(transitions)
```

What we do in this case is very simple:

1. We go on by calculating the parity in a given window
2. We set an absolute value threshold below which we consider the parity to be zero
3. We ask the script to report the positions were this is true.

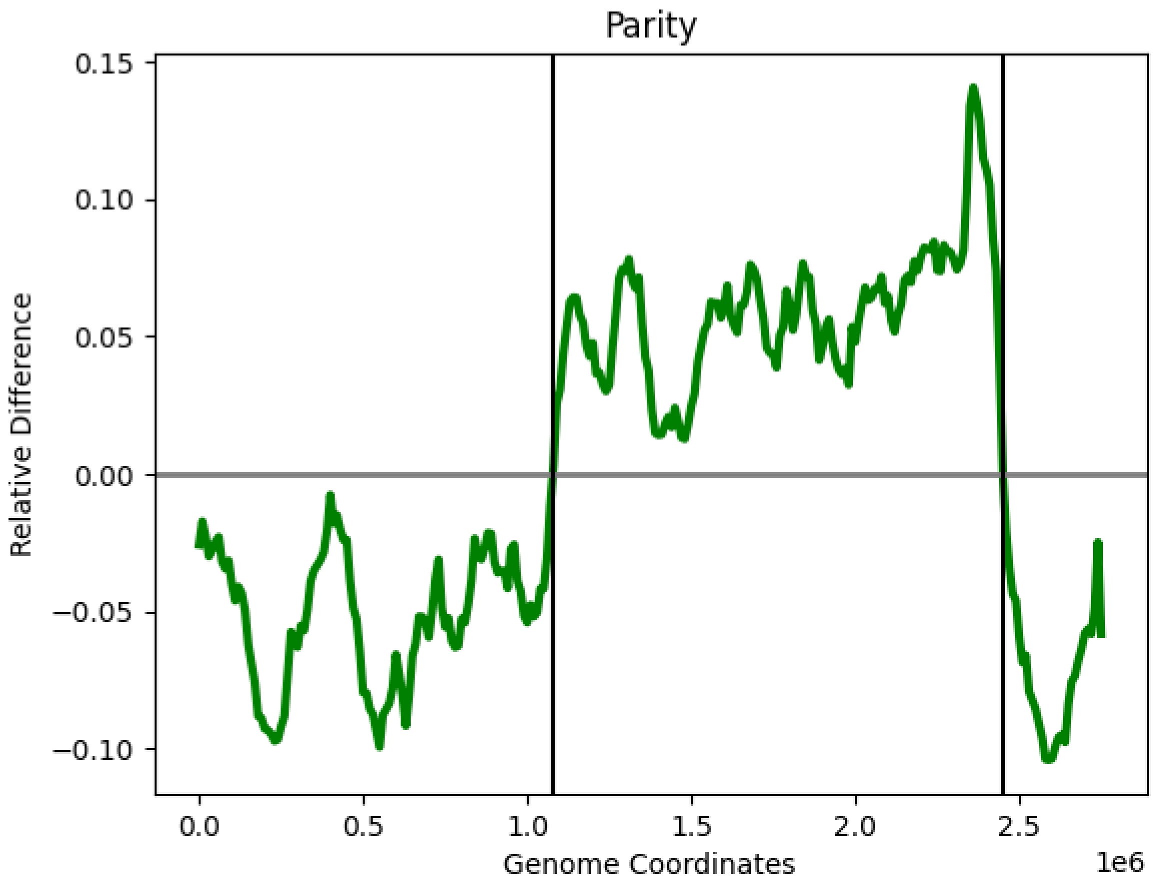
In [47]:
```
bruteParity('files/Staaur.fa', 100000, 'AT', 0.01)
```



Out[47]:
```
[400000, 1080000, 2450000]
```

We see that we locate the two main transition points plus an extra one. We actually have to tune the resolution parameter a bit to get the correct positions.

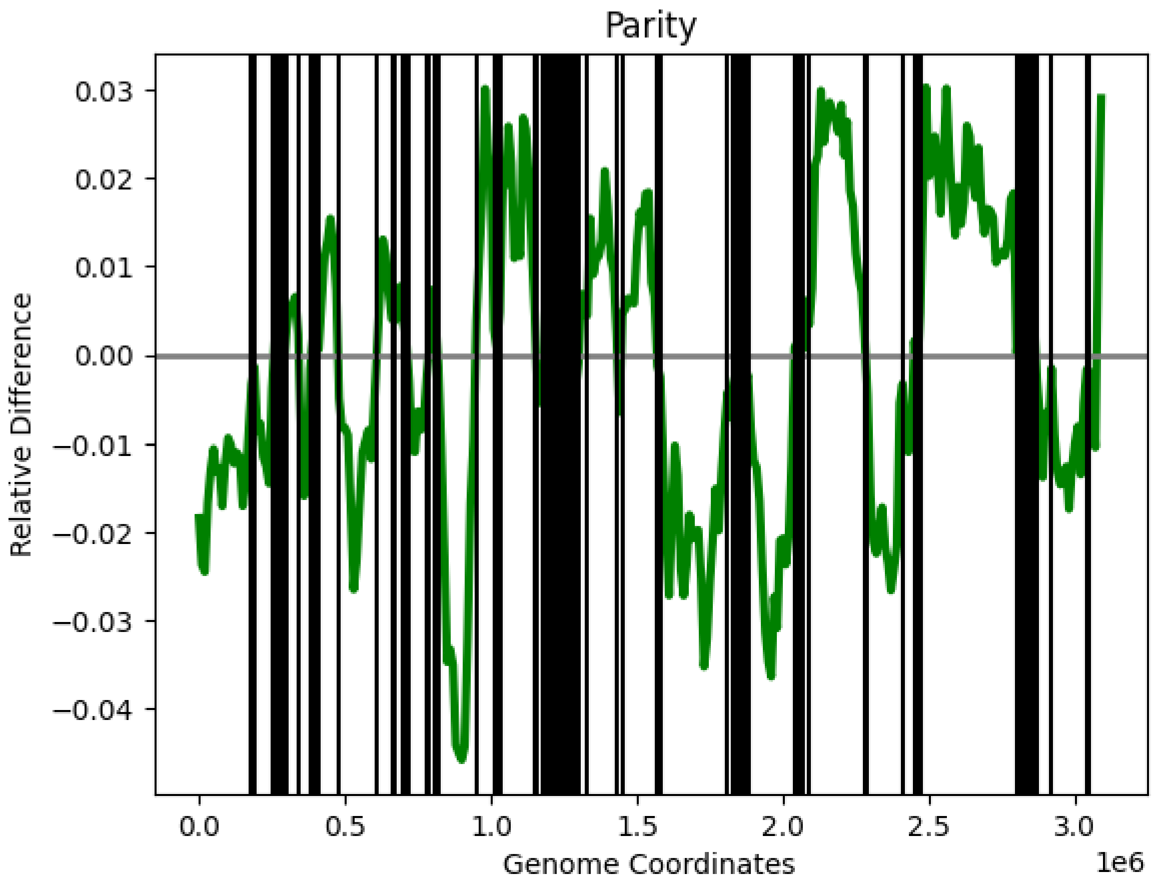
In [49]:
```
bruteParity('files/Staaur.fa', 100000, 'AT', 0.005)
```



Out[49]:
```
[1080000, 2450000]
```

Depending on the genome and its shape of the parity profile we need to tune the parameters accordingly and, obviously, this is not a good strategy for unknown genomes. Let's see for instance how the same parameteres behave in the case of E. coli.

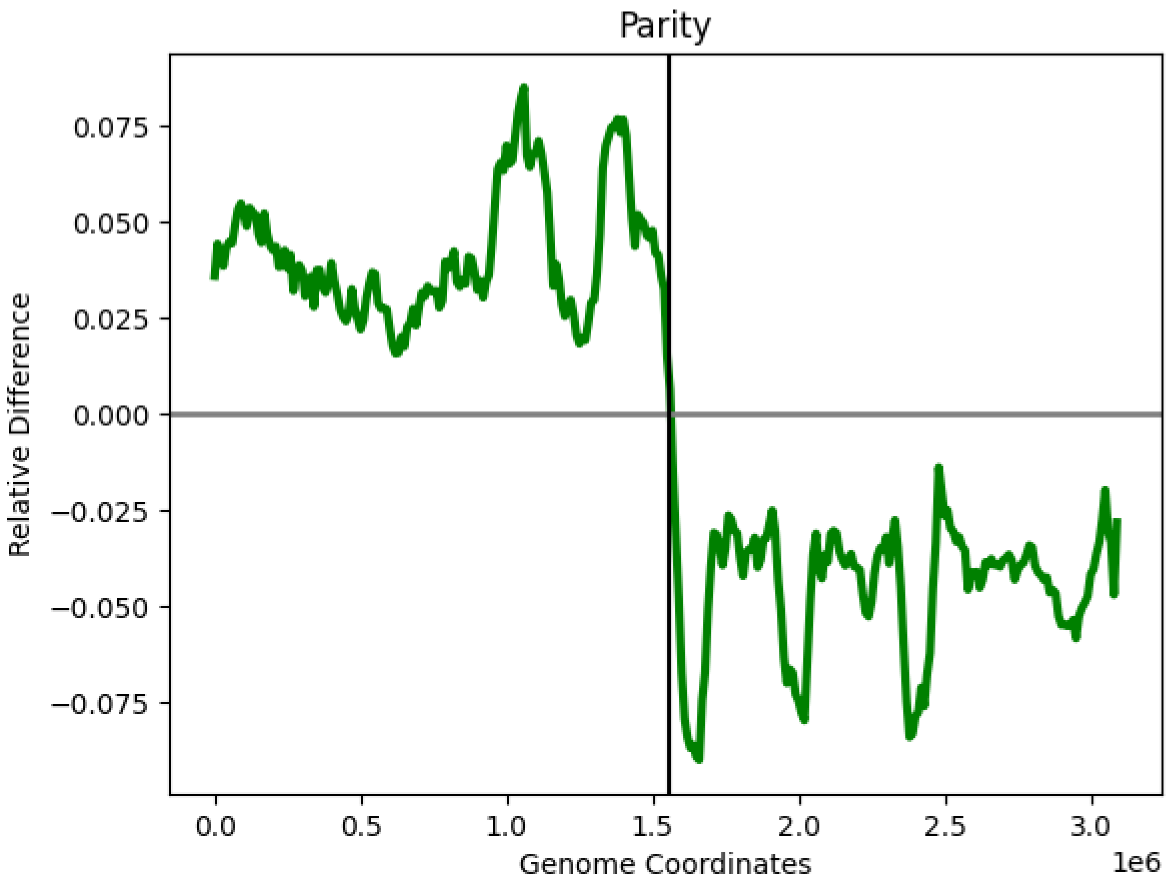
In [50]:
```
bruteParity('files/ecoli.fa', 100000, 'AT', 0.005)
```



```
Out[50]: [180000,
          190000,
          250000,
          260000,
          270000,
          280000,
          290000,
          300000,
          340000,
          380000,
          390000,
          400000,
          410000,
          480000,
          610000,
          660000,
          670000,
          700000,
          710000,
          720000,
          780000,
          790000,
          810000,
          820000,
          950000,
          1010000,
          1020000,
          1030000,
          1150000,
          1160000,
          1180000,
          1190000,
          1200000,
          1210000,
          1220000,
          1230000,
          1240000,
          1250000,
          1260000,
          1270000,
          1280000,
          1290000,
          1300000,
          1330000,
          1430000,
          1450000,
          1570000,
          1580000,
          1810000,
          1830000,
          1840000,
          1850000,
          1860000,
          1870000,
          1880000,
          2040000,
          2050000,
          2060000,
          2070000,
          2090000,
          2280000,
          2290000,
          2410000,
          2450000,
          2460000,
          2470000,
          2800000,
          2810000,
          2820000,
          2830000,
          2840000,
          2850000,
          2860000,
          2870000,
          2920000,
          3040000,
          3050000]
```

AT is not a good choice for E. coli, so we switch to GC parity, for which, relaxing the threshold a bit (at 0.01) allows us to define the major transition point. In this case, the other point coincides with the beginning and end of the sequence.

In [54]:
```
bruteParity('files/ecoli.fa', 100000, 'GC', 0.01)
```



Out[54]:
```
[1560000]
```

The question is, now, if this could be improved in a way that would not require tuning of parameters except from the choice of parity profile (AT or GC).

A Divide and Conquer approach (think about it)

We can think of a way that proceeds iteratively by narrowing down the space where the transition is being searched for. One such approach would be to roughly estimate the location of the transition in a range and then proceed by splitting that range in smaller and smaller pieces. This could work in the case of a single transition as is the one of E. coli.

Exercises

- Use a genomic signature approach to locate possible HGT genes in the genome of *St. aureus*. Do your results of "outliers" differ from those obtained with the GC content approach?
- Employ a similar approach for HGT detection but using Genomic Signatures for k=2 or k=3.
- Write a faster program to locate the origin of replication for a given bacterial genome using the parity rules described in the lecture.