BC205: Algorithms for Bioinformatics.

II. Sequence Analysis

Christoforos Nikolaou

Sequence Analysis

- In this class we will start looking into real biological problems, focusing on sequence analysis
- 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 discuss a new approach (Binary Search) and implement it the context of a 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 8-nucleotides. What do you need to consider to attack the problem?

Aspects of DNA Composition

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

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 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 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 [4]:
```
# Naive GC content

import regex as re
f=open('files/ecoli.fa', 'r')

seq = ""
window=1000
total = 0
nG=nC=0
GCCont=0
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]

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

```
0.5074167653333127
```

GCContent: Implementation (using Python's count function)

In [5]:
```
import regex as re

f=open('files/staaur.fa', 'r')

seq = ""
window=1000
total = 0
nG=nC=0
GCCont=0
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()
nC=seq.count("C")
nG=seq.count("G")
GCCont=(nG+nC)/len(seq);
print(GCCont)
```

```
0.3291873726579277
```

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

- We could do this very easily with R but it can also be done otherwise

For example

In [1]:

```
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)
        #print(pair, round(gcdistances[pair],3))

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

Out[1]:
```
[('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)]
```

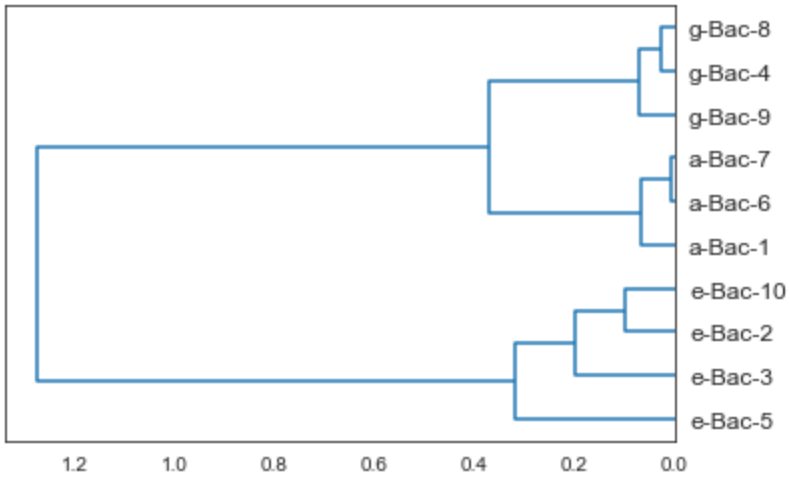
- Notice how smaller values are obtained for same bacterial family (a-, g- and e-proteobacteria)

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 [4]:
```
## 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='l

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

```
<ipython-input-4-c652b4de09d1>:21: ClusterWarning: scipy.cluster: The symme
tric non-negative hollow observation matrix looks suspiciously like an unco
ndensed distance matrix
  linkage_matrix = linkage(distance_matrix, method='ward')
```



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.

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 [9]:
```
import regex as re
f=open('files/Staaur.fa', 'r')

seq = ""
window=1000
nG=nC=0
GCCont=[]
total=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()

step=100
```

```
times=int(len(seq)/step);

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

for k, value in enumerate(GCCont[1:10]):
    print(k*step, "\t", value)
```

```
0        0.321
100      0.322
200      0.322
300      0.311
400      0.315
500      0.317
600      0.312
700      0.313
800      0.313
```

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...

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.

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 [16]:
```
import regex as re
import numpy as np

f=open('files/Staaur.fa', 'r')

seq = ""
window=1000
nG=nC=0
GCCont=[]

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

step=100
times=int(len(seq)/step);

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

# Calculate Z-scores
mGC=np.mean(GCCont)
sdGC=np.std(GCCont)
zGC=(GCCont-mGC)/sdGC
for i in range(len(zGC)):
    if abs(zGC[i])>=3:
        print(i*step, zGC[i])
```

2764699
311700 3.3056871660938962
311800 3.510729542254593
311900 3.7450636864382467
312000 3.80364722248416
312100 3.6571883823693767
312200 3.422854238185723
446200 -3.138501798956574
446300 -3.109210030933617
610000 3.0713530219102427
610100 4.125856670736683
610200 4.272315510851466
610300 4.77027556724173
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
611600 4.975317943402427
611700 4.887442639333557
611800 4.740983799218774
611900 4.77027556724173
612000 4.77027556724173
612100 4.740983799218774
612200 4.213731974805553
612300 4.184440206782596
612400 4.096564902713726
612500 4.037981366667814
612600 3.62789661434642
612700 3.1006447899331993
613200 3.0127694858643292
613300 3.129936557956156
613400 3.2763953980709397
613500 3.5693130783005067
613600 4.096564902713726
613700 4.653108495149903
613800 5.385402695723821
613900 5.238943855609038
614000 5.268235623631994
614100 5.473277999792691
614200 5.297527391654951
614300 5.268235623631994
614400 4.94602617537947
614500 3.9793978306219007
614600 3.422854238185723
614700 3.2471036300479827
615200 3.1006447899331993
615300 3.2763953980709397
615400 3.6571883823693767

615500 4.623816727126947
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
616700 4.975317943402427
616800 4.975317943402427
616900 4.887442639333557
617000 4.77027556724173
617100 4.77027556724173
617200 4.711692031195817
617300 4.59452495910399
617400 4.184440206782596
617500 4.272315510851466
617600 4.037981366667814
617700 4.125856670736683
617800 3.5400213102775497
617900 3.3935624701627662
618100 3.0713530219102427
618200 3.2471036300479827
618300 3.2471036300479827
618400 3.4521460062086797
618500 3.510729542254593
618600 3.6864801503923337
618700 4.008689598644858
618800 4.68240026317286
618900 4.828859103287644
619000 5.356110927700864
619100 5.268235623631994
619200 5.180360319563124
619300 5.502569767815648
619400 5.356110927700864
619500 5.356110927700864
619600 5.092485015494254
619700 4.33089904689738
619800 4.272315510851466
619900 4.301607278874423
620000 4.360190814920336
620100 4.59452495910399
620200 4.067273134690771
620300 3.832938990507117
620400 4.067273134690771
620500 3.9793978306219007
620600 3.832938990507117
620700 3.7743554544612037
620800 3.1006447899331993
659300 3.129936557956156
659400 3.422854238185723
659500 3.9793978306219007

659600 4.360190814920336
659700 4.535941423058077
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
661000 4.975317943402427
661100 4.8581508713106
661200 4.8581508713106
661300 4.8581508713106
661400 4.77027556724173
661500 4.623816727126947
661600 4.50664965503512
661700 4.301607278874423
661800 4.184440206782596
661900 4.125856670736683
662000 4.037981366667814
662100 3.3349789341168528
663000 3.129936557956156
663100 3.8622307585300737
663200 4.448066118989206
663300 5.209652087586081
663400 5.326819159677908
663500 5.092485015494254
663600 5.531861535838605
663700 5.297527391654951
663800 5.297527391654951
663900 5.238943855609038
664000 4.389482582943293
664100 3.3935624701627662
901300 3.1885200940020693
1211300 3.0420612538872858
1211400 3.62789661434642
1211500 3.9501060625989437
1211600 4.067273134690771
1211700 3.3056871660938962
1686800 3.1592283259791127
1686900 4.125856670736683
1687000 5.063193247471297
1687100 5.326819159677908
1687200 5.356110927700864
1687300 5.473277999792691
1687400 5.209652087586081
1687500 5.238943855609038
1687600 5.326819159677908
1687700 4.653108495149903
1687800 4.184440206782596
1687900 3.3935624701627662

1688900 3.2471036300479827
1689000 3.7450636864382467
1689100 4.155148438759639
1689200 4.037981366667814
1689300 4.33089904689738
1689400 4.389482582943293
1689500 4.77027556724173
1689600 4.740983799218774
1689700 4.68240026317286
1689800 4.77027556724173
1689900 4.799567335264687
1690000 4.975317943402427
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
1691500 4.916734407356514
1691600 4.799567335264687
1691700 4.50664965503512
1691800 3.71577191841529
1691900 3.3349789341168528
1692000 3.2178118620250262
1801700 3.2178118620250262
1801800 4.155148438759639
1801900 4.096564902713726
1802000 4.360190814920336
1802100 5.092485015494254
1802200 5.297527391654951
1802300 5.209652087586081
1802400 5.531861535838605
1802500 5.121776783517211
1802600 5.297527391654951
1802700 5.297527391654951
1802800 4.59452495910399
1802900 4.096564902713726
1803000 3.3349789341168528
1804000 3.2763953980709397
1804100 3.9501060625989437
1804200 4.184440206782596
1804300 4.096564902713726
1804400 4.360190814920336
1804500 4.389482582943293
1804600 4.799567335264687
1804700 4.68240026317286
1804800 4.740983799218774
1804900 4.740983799218774

1805000 4.799567335264687
1805100 4.916734407356514
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
1806400 4.59452495910399
1806500 4.360190814920336
1806600 3.9501060625989437
1806700 3.5986048463234632
1806800 3.2763953980709397
1975100 3.9793978306219007
1975200 4.828859103287644
1975300 5.297527391654951
1975400 5.326819159677908
1975500 5.502569767815648
1975600 5.238943855609038
1975700 5.033901479448341
1975800 5.268235623631994
1975900 4.77027556724173
1976000 4.77027556724173
1976100 4.887442639333557
1976200 4.067273134690771
1976300 3.422854238185723
1976400 3.2763953980709397
1976500 3.0713530219102427
1976600 3.2178118620250262
1976900 3.2178118620250262
1977000 3.1592283259791127
1977100 3.0127694858643292
1977200 3.5693130783005067
1977300 4.272315510851466
1977400 4.184440206782596
1977500 4.33089904689738
1977600 4.301607278874423
1977700 4.887442639333557
1977800 4.828859103287644
1977900 4.711692031195817
1978000 4.77027556724173
1978100 4.8581508713106
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
1979600 4.799567335264687
1979700 4.887442639333557
1979800 4.94602617537947
1979900 4.711692031195817
1980000 4.50664965503512
1980100 4.50664965503512
1980200 4.740983799218774
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
1981600 4.916734407356514
1981700 4.50664965503512
1981800 4.448066118989206
1981900 4.243023742828509
1982000 3.5986048463234632
1982100 3.0420612538872858
2579500 -3.109210030933617
2748000 -3.0799182629106605
2764000 -3.021334726864747
2764100 -4.075838375691188
2764200 -5.0424667204487585
2764300 -6.067678601252243
2764400 -6.946431641940943
2764500 -7.913059986698514
```

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 evolutio. 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

- 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 [7]:
```
import regex as re

f=open('files/Staaur.fa', 'r')

seq = ""
k=2
kmers={}

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

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

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

Out[7]:
```
{'GT': 0.052417641124766205,
 'TA': 0.09542485456825499,
 'AT': 0.11214421533772755,
 'TT': 0.12211817633673684,
 'AC': 0.0527258121046812,
 'CT': 0.04827324782914885,
 'TC': 0.05299347234545244,
 'CA': 0.06572144020018092,
 'AA': 0.12267628410904767,
 'CG': 0.02563642552046353,
 'TG': 0.06441677737793518,
 'GG': 0.02547908470325341,
 'CC': 0.025711298047273862,
 'AG': 0.04831231175618033,
 'GA': 0.05203640613318122,
 'GC': 0.033911829099659674}
```

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 [8]:
```
import regex as re

f=open('files/Staaur.fa', 'r')

seq = ""
k=2
kmers={}

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

pnuc={}
for i in range(len(seq)):
    nuc=seq[i]
    if nuc not in pnuc.keys():
        pnuc[nuc]=1
    else:
        pnuc[nuc]+=1

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

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

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

rkmers=kmers
```

```
for kmer in kmers.keys():
    nuc1=list(kmer)[0]
    nuc2=list(kmer)[1]
    rkmers[kmer]=round(kmers[kmer]/(pnuc[nuc1]*pnuc[nuc2]),3)

print(rkmers)
```

```
{'GT': 0.955, 'TA': 0.848, 'AT': 0.997, 'TT': 1.088, 'AC': 0.949, 'CT': 0.8
72, 'TC': 0.957, 'CA': 1.183, 'AA': 1.088, 'CG': 0.946, 'TG': 1.174, 'GG':
0.949, 'CC': 0.94, 'AG': 0.878, 'GA': 0.946, 'GC': 1.252}
```

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

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 *phase 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 [9]:
```
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)

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")
        print(i*step,"\t",i*step+window,"\t",float(A-T)/(A+T))
```

0	2716200	-0.016058394160583942
2715300	2716300	-0.03074670571010249
2715400	2716400	-0.03468208092485549
2715500	2716500	-0.04885057471264368
2715600	2716600	-0.04775687409551375
2715700	2716700	-0.043227665706051875
2715800	2716800	-0.035868005738880916
2715900	2716900	-0.04885057471264368
2716000	2717000	-0.047619047619047616
2716100	2717100	-0.021645021645021644
2716200	2717200	-0.013024602026049204
2716300	2717300	-0.02170767004341534
2716400	2717400	-0.023529411764705882
2716500	2717500	-0.008928571428571428
2716600	2717600	-0.008928571428571428
2716700	2717700	-0.013657056145675266
2716800	2717800	-0.01984732824427481
2716900	2717900	0.006191950464396285
2717000	2718000	-0.020030816640986132
2717100	2718100	-0.05443234836702955
2717200	2718200	-0.059748427672955975
2717300	2718300	-0.05750798722044728
2717400	2718400	-0.0784
2717500	2718500	-0.10112359550561797
2717600	2718600	-0.10223642172523961
2717700	2718700	-0.11145996860282574
2717800	2718800	-0.09233176838810642
2717900	2718900	-0.13538461538461538
2718000	2719000	-0.12074303405572756
2718100	2719100	-0.13003095975232198
2718200	2719200	-0.15527950310559005
2718300	2719300	-0.1529051987767584
2718400	2719400	-0.13109756097560976
2718500	2719500	-0.15789473684210525
2718600	2719600	-0.17664670658682635
2718700	2719700	-0.16890881913303438
2718800	2719800	-0.1963470319634703
2718900	2719900	-0.19142419601837674
2719000	2720000	-0.2115677321156773
2719100	2720100	-0.20060331825037708
2719200	2720200	-0.1880597014925373
2719300	2720300	-0.208955223880597
2719400	2720400	-0.20592592592592593
2719500	2720500	-0.16790490341753342
2719600	2720600	-0.15805022156573117
2719700	2720700	-0.16790490341753342
2719800	2720800	-0.1495601173020528
2719900	2720900	-0.1316931982633864
2720000	2721000	-0.09482758620689655
2720100	2721100	-0.07514450867052024
2720200	2721200	-0.04610951008645533
2720300	2721300	-0.017142857142857144
2720400	2721400	-0.00727802037845706
2720500	2721500	-0.008797653958944282
2720600	2721600	-0.013254786450662739
2720700	2721700	-0.036603221083455345

2720800	2721800	-0.07917888563049853
2720900	2721900	-0.07462686567164178
2721000	2722000	-0.09580838323353294
2721100	2722100	-0.10911808669656203
2721200	2722200	-0.13213213213213212
2721300	2722300	-0.1345565749235474
2721400	2722400	-0.16463414634146342
2721500	2722500	-0.20059880239520958
2721600	2722600	-0.18796992481203006
2721700	2722700	-0.15568862275449102
2721800	2722800	-0.12094395280235988
2721900	2722900	-0.12389380530973451
2722000	2723000	-0.09896602658788774
2722100	2723100	-0.09821428571428571
2722200	2723200	-0.08902077151335312
2722300	2723300	-0.11209439528023599
2722400	2723400	-0.09798270893371758
2722500	2723500	-0.06086956521739131
2722600	2723600	-0.06990014265335236
2722700	2723700	-0.10473457675753228
2722800	2723800	-0.10495626822157435
2722900	2723900	-0.0953757225433526
2723000	2724000	-0.0967741935483871
2723100	2724100	-0.08100147275405008
2723200	2724200	-0.08358208955223881
2723300	2724300	-0.07807807807807808
2723400	2724400	-0.06646058732612056
2723500	2724500	-0.0670926517571885
2723600	2724600	-0.08469055374592833
2723700	2724700	-0.05140961857379768
2723800	2724800	-0.03908794788273615
2723900	2724900	-0.10289389067524116
2724000	2725000	-0.15639810426540285
2724100	2725100	-0.15968992248062017
2724200	2725200	-0.16946564885496182
2724300	2725300	-0.15022761760242792
2724400	2725400	-0.1566265060240964
2724500	2725500	-0.16272189349112426
2724600	2725600	-0.1263001485884101
2724700	2725700	-0.11370262390670553
2724800	2725800	-0.12481644640234948
2724900	2725900	-0.0962962962962963
2725000	2726000	-0.05216095380029806
2725100	2726100	-0.0832072617246596
2725200	2726200	-0.0834597875569044
2725300	2726300	-0.0783132530120482
2725400	2726400	-0.12
2725500	2726500	-0.12071535022354694
2725600	2726600	-0.13569321533923304
2725700	2726700	-0.17138599105812222
2725800	2726800	-0.16272189349112426
2725900	2726900	-0.16936671575846834
2726000	2727000	-0.18594436310395315
2726100	2727100	-0.14532374100719425
2726200	2727200	-0.1856115107913669
2726300	2727300	-0.23823529411764705

2726400	2727400	-0.20236336779911374
2726500	2727500	-0.2178932178932179
2726600	2727600	-0.25936599423631124
2726700	2727700	-0.22988505747126436
2726800	2727800	-0.2314410480349345
2726900	2727900	-0.2088888888888889
2727000	2728000	-0.20239880059970014
2727100	2728100	-0.22038980509745126
2727200	2728200	-0.14586466165413534
2727300	2728300	-0.10914454277286136
2727400	2728400	-0.13323572474377746
2727500	2728500	-0.09925925925925926
2727600	2728600	-0.0638930163447251
2727700	2728700	-0.06666666666666667
2727800	2728800	-0.06901615271659324
2727900	2728900	-0.06861313868613139
2728000	2729000	-0.06801736613603473
2728100	2729100	-0.04283604135893648
2728200	2729200	-0.04525547445255475
2728300	2729300	-0.03529411764705882
2728400	2729400	0.029940119760479042
2728500	2729500	0.04932735426008968
2728600	2729600	0.09063893016344725
2728700	2729700	0.10456553755522828
2728800	2729800	0.13274336283185842
2728900	2729900	0.11078286558345643
2729000	2730000	0.12941176470588237
2729100	2730100	0.11942446043165468
2729200	2730200	0.10215827338129496
2729300	2730300	0.07183908045977011
2729400	2730400	0.015691868758915834
2729500	2730500	-0.017241379310344827
2729600	2730600	-0.06666666666666667
2729700	2730700	-0.10395314787701318
2729800	2730800	-0.14782608695652175
2729900	2730900	-0.12790697674418605
2730000	2731000	-0.15350223546944858
2730100	2731100	-0.2
2730200	2731200	-0.20367534456355282
2730300	2731300	-0.1926040061633282
2730400	2731400	-0.15789473684210525
2730500	2731500	-0.15348837209302327
2730600	2731600	-0.14152410575427682
2730700	2731700	-0.12539184952978055
2730800	2731800	-0.11428571428571428
2730900	2731900	-0.10204081632653061
2731000	2732000	-0.08978328173374613
2731100	2732100	-0.08864696734059098
2731200	2732200	-0.11455108359133127
2731300	2732300	-0.13713405238828968
2731400	2732400	-0.1354642313546423
2731500	2732500	-0.14674735249621784
2731600	2732600	-0.16314199395770393
2731700	2732700	-0.15555555555555556
2731800	2732800	-0.16395864106351551
2731900	2732900	-0.19174041297935104

2732000	2733000	-0.22666666666666666
2732100	2733100	-0.19880418535127056
2732200	2733200	-0.1874062968515742
2732300	2733300	-0.19398496240601504
2732400	2733400	-0.21036585365853658
2732500	2733500	-0.2260061919504644
2732600	2733600	-0.20245398773006135
2732700	2733700	-0.2172573189522342
2732800	2733800	-0.19937694704049844
2732900	2733900	-0.17209302325581396
2733000	2734000	-0.1490015360983103
2733100	2734100	-0.16257668711656442
2733200	2734200	-0.13671274961597543
2733300	2734300	-0.11145038167938931
2733400	2734400	-0.0889894419306184
2733500	2734500	-0.04918032786885246
2733600	2734600	-0.07761194029850746
2733700	2734700	-0.0712166172106825
2733800	2734800	-0.09278350515463918
2733900	2734900	-0.10355029585798817
2734000	2735000	-0.09063893016344725
2734100	2735100	-0.10364963503649635
2734200	2735200	-0.11366906474820145
2734300	2735300	-0.11976911976911978
2734400	2735400	-0.13069016152716592
2734500	2735500	-0.15373352855051245
2734600	2735600	-0.12812960235640647
2734700	2735700	-0.12035661218424963
2734800	2735800	-0.1044776119402985
2734900	2735900	-0.11641791044776119
2735000	2736000	-0.14032496307237813
2735100	2736100	-0.11773472429210134
2735200	2736200	-0.0962962962962963
2735300	2736300	-0.05154639175257732
2735400	2736400	-0.04538799414348463
2735500	2736500	-0.021961932650073207
2735600	2736600	-0.01744186046511628
2735700	2736700	-0.0029154518950437317
2735800	2736800	-0.005698005698005698
2735900	2736900	0.022922636103151862
2736000	2737000	0.041726618705035974
2736100	2737100	0.02865329512893983
2736200	2737200	0.005797101449275362
2736300	2737300	-0.036284470246734396
2736400	2737400	-0.04460431654676259
2736500	2737500	-0.0659025787965616
2736600	2737600	-0.04667609618104668
2736700	2737700	-0.07317073170731707
2736800	2737800	-0.052478134110787174
2736900	2737900	-0.08163265306122448
2737000	2738000	-0.0824891461649783
2737100	2738100	-0.04899135446685879
2737200	2738200	-0.045584045584045586
2737300	2738300	-0.03862660944206009
2737400	2738400	-0.0513950073421439
2737500	2738500	-0.02373887240356083

2737600	2738600	-0.05421686746987952
2737700	2738700	-0.04552129221732746
2737800	2738800	-0.0513950073421439
2737900	2738900	-0.06358381502890173
2738000	2739000	-0.09302325581395349
2738100	2739100	-0.11208151382823872
2738200	2739200	-0.10850439882697947
2738300	2739300	-0.07488986784140969
2738400	2739400	-0.08751793400286945
2738500	2739500	-0.11331444759206799
2738600	2739600	-0.11360448807854137
2738700	2739700	-0.13229018492176386
2738800	2739800	-0.14693295292439373
2738900	2739900	-0.1308139534883721
2739000	2740000	-0.11527377521613832
2739100	2740100	-0.12992700729927006
2739200	2740200	-0.12536443148688048
2739300	2740300	-0.14739884393063585
2739400	2740400	-0.11304347826086956
2739500	2740500	-0.09169054441260745
2739600	2740600	-0.08645533141210375
2739700	2740700	-0.042735042735042736
2739800	2740800	-0.03546099290780142
2739900	2740900	-0.006915629322268326
2740000	2741000	0.012517385257301807
2740100	2741100	0.042876901798063624
2740200	2741200	0.05337078651685393
2740300	2741300	0.06723891273247497
2740400	2741400	0.08806818181818182
2740500	2741500	0.10404624277456648
2740600	2741600	0.12372634643377002
2740700	2741700	0.11634756995581738
2740800	2741800	0.1130690161527166
2740900	2741900	0.13505311077389984
2741000	2742000	0.16049382716049382
2741100	2742100	0.16329704510108864
2741200	2742200	0.15707620528771385
2741300	2742300	0.11764705882352941
2741400	2742400	0.05180533751962323
2741500	2742500	0.033386327503974564
2741600	2742600	-0.0031446540880503146
2741700	2742700	0.0
2741800	2742800	0.017214397496087636
2741900	2742900	-0.04294478527607362
2742000	2743000	-0.09145427286356822
2742100	2743100	-0.10714285714285714
2742200	2743200	-0.11669128508124077
2742300	2743300	-0.11143695014662756
2742400	2743400	-0.09224011713030747
2742500	2743500	-0.10980966325036604
2742600	2743600	-0.09495548961424333
2742700	2743700	-0.12667660208643816
2742800	2743800	-0.1518796992481203
2742900	2743900	-0.1346444780635401
2743000	2744000	-0.13201820940819423
2743100	2744100	-0.13538461538461538

2743200	2744200	-0.13761467889908258
2743300	2744300	-0.1171993911719939
2743400	2744400	-0.11179173047473201
2743500	2744500	-0.11450381679389313
2743600	2744600	-0.1232876712328767
2743700	2744700	-0.1043872919818457
2743800	2744800	-0.08444444444444445
2743900	2744900	-0.06647398843930635
2744000	2745000	-0.043227665706051875
2744100	2745100	-0.05070422535211268
2744200	2745200	-0.042134831460674156
2744300	2745300	-0.012693935119887164
2744400	2745400	-0.0014104372355430183
2744500	2745500	0.025210084033613446
2744600	2745600	0.03506311360448808
2744700	2745700	0.0584958217270195
2744800	2745800	0.09065550906555091
2744900	2745900	0.09971509971509972
2745000	2746000	0.12056737588652482
2745100	2746100	0.14122681883024252
2745200	2746200	0.1563845050215208
2745300	2746300	0.12446351931330472
2745400	2746400	0.13105413105413105
2745500	2746500	0.12624113475177304
2745600	2746600	0.16550764951321278
2745700	2746700	0.17663421418636996
2745800	2746800	0.1420534458509142
2745900	2746900	0.13807531380753138
2746000	2747000	0.1279887482419128
2746100	2747100	0.12305516265912306
2746200	2747200	0.1111111111111111
2746300	2747300	0.1383737517831669
2746400	2747400	0.15655853314527504
2746500	2747500	0.1622002820874471
2746600	2747600	0.11522048364153627
2746700	2747700	0.07692307692307693
2746800	2747800	0.04046242774566474
2746900	2747900	0.005952380952380952
2747000	2748000	-0.01925925925925926
2747100	2748100	-0.041176470588235294
2747200	2748200	-0.050946142649199416
2747300	2748300	-0.08069164265129683
2747400	2748400	-0.09943181818181818
2747500	2748500	-0.11608391608391608
2747600	2748600	-0.11543810848400557
2747700	2748700	-0.1307901907356948
2747800	2748800	-0.09066666666666667
2747900	2748900	-0.06217616580310881
2748000	2749000	-0.041237113402061855
2748100	2749100	-0.015665796344647518
2748200	2749200	0.018276762402088774
2748300	2749300	0.03800786369593709
2748400	2749400	0.03504043126684636
2748500	2749500	0.04
2748600	2749600	0.06424581005586592
2748700	2749700	0.1037463976945245

2748800	2749800	0.11045655375552282
2748900	2749900	0.13343328335832083
2749000	2750000	0.1518796992481203
2749100	2750100	0.1263001485884101
2749200	2750200	0.06110283159463487
2749300	2750300	0.032640949554896145
2749400	2750400	0.0029154518950437317
2749500	2750500	0.0
2749600	2750600	-0.02023121387283237
2749700	2750700	-0.052781740370898715
2749800	2750800	-0.04261363636363636
2749900	2750900	-0.07909604519774012
2750000	2751000	-0.10256410256410256
2750100	2751100	-0.10760401721664276
2750200	2751200	-0.07471264367816093
2750300	2751300	-0.07111756168359942
2750400	2751400	-0.06588579795021962
2750500	2751500	-0.08905109489051095
2750600	2751600	-0.0858806404657933
2750700	2751700	-0.06956521739130435
2750800	2751800	-0.09090909090909091
2750900	2751900	-0.0670391061452514
2751000	2752000	-0.06536856745479833
2751100	2752100	-0.04299583911234397
2751200	2752200	-0.0440771349862259
2751300	2752300	-0.01084010840108401
2751400	2752400	0.016
2751500	2752500	0.023936170212765957
2751600	2752600	0.023809523809523808
2751700	2752700	0.031746031746031744
2751800	2752800	0.026385224274406333
2751900	2752900	0.058666666666666666
2752000	2753000	0.024128686327077747
2752100	2753100	0.02796271637816245
2752200	2753200	0.028037383177570093
2752300	2753300	-0.023709902370990237
2752400	2753400	-0.06821480406386067
2752500	2753500	-0.0728862973760933
2752600	2753600	-0.04525547445255475
2752700	2753700	-0.06881405563689605
2752800	2753800	-0.07488986784140969
2752900	2753900	-0.14666666666666667
2753000	2754000	-0.1288888888888889
2753100	2754100	-0.16292974588938713
2753200	2754200	-0.16691285081240767
2753300	2754300	-0.1276297335203366
2753400	2754400	-0.12275862068965518
2753500	2754500	-0.1361760660247593
2753600	2754600	-0.1696551724137931
2753700	2754700	-0.18028169014084508
2753800	2754800	-0.18361581920903955
2753900	2754900	-0.16182572614107885
2754000	2755000	-0.12465373961218837
2754100	2755100	-0.11691884456671252
2754200	2755200	-0.10679611650485436
2754300	2755300	-0.0970464135021097

2754400	2755400	-0.038781163434903045
2754500	2755500	0.019337016574585635
2754600	2755600	0.027472527472527472
2754700	2755700	0.04054054054054054
2754800	2755800	0.06291834002677377
2754900	2755900	0.06166219839142091
2755000	2756000	0.05053191489361702
2755100	2756100	0.06860158311345646
2755200	2756200	0.06613756613756613
2755300	2756300	0.0610079575596817
2755400	2756400	0.064
2755500	2756500	0.049664429530201344
2755600	2756600	0.06267029972752043
2755700	2756700	0.06938775510204082
2755800	2756800	0.05131761442441054
2755900	2756900	0.06064880112834979
2756000	2757000	0.05898876404494382
2756100	2757100	0.019943019943019943
2756200	2757200	-0.019886363636363636
2756300	2757300	-0.03428571428571429
2756400	2757400	-0.0546984572230014
2756500	2757500	-0.07862068965517241
2756600	2757600	-0.09633649932157395
2756700	2757700	-0.08943089430894309
2756800	2757800	-0.05263157894736842
2756900	2757900	-0.07442489851150202
2757000	2758000	-0.07250341997264022
2757100	2758100	-0.0546448087431694
2757200	2758200	-0.023383768913342505
2757300	2758300	-0.024657534246575342
2757400	2758400	-0.057342657342657345
2757500	2758500	-0.049786628733997154
2757600	2758600	-0.0436046511627907
2757700	2758700	-0.051698670605613
2757800	2758800	-0.09658246656760773
2757900	2758900	-0.09333333333333334
2758000	2759000	-0.10714285714285714
2758100	2759100	-0.08682634730538923
2758200	2759200	-0.07530120481927711
2758300	2759300	-0.0781010719754977
2758400	2759400	-0.08307692307692308
2758500	2759500	-0.07219662058371736
2758600	2759600	-0.08448540706605223
2758700	2759700	-0.0973724884080371
2758800	2759800	-0.10606060606060606
2758900	2759900	-0.09531013615733737
2759000	2760000	-0.10708898944193061
2759100	2760100	-0.11212121212121212
2759200	2760200	-0.11782477341389729
2759300	2760300	-0.11613876319758673
2759400	2760400	-0.09531013615733737
2759500	2760500	-0.08536585365853659
2759600	2760600	-0.0790273556231003
2759700	2760700	-0.09955423476968796
2759800	2760800	-0.10060975609756098
2759900	2760900	-0.11490683229813664

```
2760000         2761000         -0.09921259842519685
2760100         2761100         -0.1277258566978193
2760200         2761200         -0.1321928460342146
2760300         2761300         -0.14021571648690292
2760400         2761400         -0.12597200622083982
2760500         2761500         -0.14152410575427682
2760600         2761600         -0.14064914992272023
2760700         2761700         -0.10172143974960876
2760800         2761800         -0.07765451664025357
2760900         2761900         -0.06289308176100629
2761000         2762000         -0.07086614173228346
2761100         2762100         -0.044444444444444446
2761200         2762200         -0.0592
2761300         2762300         -0.048701298701298704
2761400         2762400         -0.05228758169934641
2761500         2762500         -0.051070840197693576
2761600         2762600         -0.023026315789473683
2761700         2762700         -0.0016474464579901153
2761800         2762800         0.019417475728155338
2761900         2762900         0.033707865168539325
2762000         2763000         0.05345911949685535
2762100         2763100         0.06269592476489028
2762200         2763200         0.08722741433021806
2762300         2763300         0.0900763358778626
2762400         2763400         0.08383233532934131
2762500         2763500         0.07917888563049853
2762600         2763600         0.038461538461538464
2762700         2763700         0.0014727540500736377
2762800         2763800         -0.011869436201780416
2762900         2763900         -0.037481259370314844
2763000         2764000         -0.06505295007564296
2763100         2764100         -0.09559939301972686
2763200         2764200         -0.12859304084720122
2763300         2764300         -0.14634146341463414
2763400         2764400         -0.1484848484848485
2763500         2764500         -0.16463414634146342
2763600         2764600         -0.14630467571644043
2763700         2764700         -0.14803625377643503
2763800         2764800         -0.16
2763900         2764900         -0.15985130111524162
2764000         2765000         -0.16701902748414377
2764100         2765100         -0.15403422982885084
2764200         2765200         -0.13450292397660818
2764300         2765300         -0.11913357400722022
2764400         2765400         -0.15942028985507245
2764500         2765500         -0.14285714285714285
```

Problem 4: Plotting the Values

In [10]:

```
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)
    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=5.0)
#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()

# 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

- 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. How then can we locate that point?
- The answer is given by iterative splits of [k,m] in intervals that are always have the size of m-k. (That is [i=k, j=(k+m)/2] or [i=(k+m)/2, j=m]) and checking if the condition of f(i)f(j)<0 holds. If it does, we choose that interval and repeat.
- Question 1: What do we need to consider before starting?

- Question 2: How do we stop?

Exercises: To think about

- 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?
- Write a program to locate the origin of replication for a given bacterial genome using the parity rules described in the lecture.
- The approach of the Genomic Signatures for k=2 works rather well because the k-mers are 16 but what about larger numbers of k (k=7 or more)? Would you use the same approach?

In [ ]: