Lecture 5 -- Vigenere Cipher, Index of Coincidence, and the Friedman Attack

In the last lecture we discussed the Kasiski attack, which finds the key length of a Vigenere text based on the pattern of repeated trigraphs (that is, 3 letter combinations). Today we discuss a more potent, modernized attack that ultimately uses letter frequency. Of course, the Vigenere cipher was designed to scramble letter frequency up, by encrypting each letter differently.

We began our discussion with an example of tossing a loaded coin, which lands heads (H) with probability p, and tails (T) with probability q = 1-p. The probability of getting two consecutive heads (HH) or two consecutive tails (TT) is $p^2 + q^2$, whereas the probability of getting mixed results (TH or HT) is 2pq. The difference between these two quantities is $p^2 - 2pq + q^2 = (p-q)^2$, which is never negative since it is a square of a real number. Thus it is more likely to get HH or TT than not. This is a bet you can win with a loaded coin, even if you don't know which way it is loaded: bet that consecutive tosses will come out the same. The worst you would do in that bet is break even, which happens when p = q = 1/2. This is because those two probabilies are equal only when (p-q) = 0, which means p=q.

Something is similar with rolling two loaded dice (e.g., in monopoly). If p_1, p_2, p_3, p_4, p_5, p_6 are the probabilities of getting 1, 2, 3, 4, 5, or 6 (respectively) with a particular loaded die, then the probability of "doubles" (two consecutive rolls coming out the same) is $p_1^2 + p_2^2 + p_3^2 + p_4^2 + p_5^2 + p_6^2$. This is in fact never less than 1/6 (we'll show this below), and that 1/6 happens only when it is a fair die (which means each probability is exactly 1/6).

Indeed, more generally, if there are n possible outcomes, with probabilities p_1, p_2, ..., p_n of happening, the odds of getting the same outcome in two consecutive outcomes is $p_1^2 + p_2^2 + \ldots + p_n^2$. We saw in class from the Cauchy-Schwarz inequality that this number is always at least $(p_1 + p_2 + \ldots + p_n)^2/n$. (Proof: apply Cauchy-Schwartz to the vector dot product $(p_1, p_2, \ldots, p_n).(1,1,...,1)$ and it pops right out).

We can apply this to English, which has uneven letter frequencies (e.g., "e" and "i" occur much more often than "q", "z", and "x"). The upshot is that two randomly chosen letters from the English language are equal with a rather large probability (about 6.8%), vs the probability 1/26 (roughtly 3.8%) that would happen if each letter occured exactly as often as any other. That's because "ee" and "ii", for example, occur much more often than other letter combinations.

This can be measured using the *Index of Coincidence*: what proportion of letters of a text a given spacing apart are the same. Define I(k) to be (the number of pairs of letters that are k spaces apart that are equal)/(length of the text - k), the denominator being the number of such pairs of letters. Friedman's attack is based on the idea that if k is a multiple of the key length, I(k) for a Vigenere message should be roughly the 6.8% we see in English. That's because those letters that are encrypted with the same part of the key are encrypted the same way. They may not be actual English (in that, say, a "z" may be much more common than "e"), but two letters are the same exactly when their unencrypted counterparts are. So we expect I(k) in that situation to be just like it is in English. On the other hand, if k is not a multiple of the key length, then Friedman pointed out that I(k) should be more like the 3.8% we see in English.

Let's illustrate this from the Gettysburg Address example we had last time, where the key was "ABE".

```
In[7]:= ciphertext = ToCharacterCode[
          "FPYRTGOSIAOHSFZEOCEBVSBKOPYRGETIIRTFRPYGIXFPVTISNULITGOOXIOINUENFANBXIPRCPRC␣
            FMVFHIOPICIRUCAOHDFHIDETFHTPXHFTRPTOTMTJSNULAUELMQEOERFGRFETFHERYAMROX␣
            AEBVEFRGBKEEMNBKRFETDMVJPWBVTFWTJRGXLEULESXHBXNBXIPROSENZRAUMOOWODSNDI␣
            IWIDBRDTSDFHIDETFHCBRLPRGFRDVVEXIASIMFXOOEGSIAUFAUXLFJIFPDPJTIETXERXIH␣
            BZEDSMFXOEIDJGAUIAQSRUMOOSFULAUJIFPDBWAGMNBPRFWTJRGQPADIFPVTISSFAHPLES␣
            IGBZEULEJVLJZETXHBXTIETOETJSNNMGIXLJZEJXITELUSGFXHFVFJXTJRGBRDQVOQIRUL␣
            AUAETLOVPDESTIMS"];
```

This table now lists the index of coincidence for all k up to 30. You can try this for your own example by replacing the ciphertext above.

```
In[15]:= Table[{shift, Sum[If[
              ciphertext[[k]] == ciphertext[[k + shift]], 1. / (Length[ciphertext] - shift), 0],
            {k, 1, Length[ciphertext] - shift}]}, {shift, 1, 30}] // TableForm
```

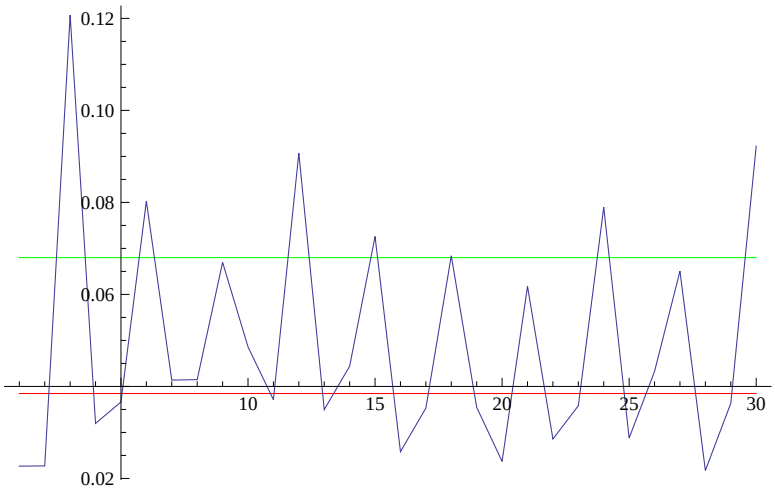
Out[15]//TableForm=

1	0.0226757
2	0.0227273
3	0.120729
4	0.0319635
5	0.0366133
6	0.0802752
7	0.0413793
8	0.0414747
9	0.0669746
10	0.0486111
11	0.037123
12	0.0906977
13	0.034965
14	0.0443925
15	0.0725995
16	0.0258216
17	0.0352941
18	0.0683962
19	0.035461
20	0.0236967
21	0.0617577
22	0.0285714
23	0.0357995
24	0.0789474
25	0.028777
26	0.0432692
27	0.0650602
28	0.0217391
29	0.0363196
30	0.092233

Here is a plot, in which we can see the peaks every 3. I'm plotting a red line at 3.8% and a green line at 6.8%, to see how the above theory fits this data sample.

In[22]:=
```
Show[Plot[{1 / 26, .068}, {x, 1, 30}, PlotStyle → {Red, Green}],
 ListPlot[Table[{shift, Sum[If[ciphertext[[k]] == ciphertext[[k + shift]],
      1. / (Length[ciphertext] - shift), 0], {k, 1, Length[ciphertext] - shift}]},
   {shift, 1, 30}], Joined → True], PlotRange → All]
```

Out[22]=



Indeed, the lines match pretty well. Here is the other example we did last time, whose key was “RADAR”. Because there are repeated letters in the key, it actually takes longer for the peaks to kick in -- it is muddled for smaller k due to this. For that reason I’ll draw these plots further than 30 out.

In[23]:=
```
ciphertext = ToCharacterCode[
   "KHHWVRTKEIWOUETRSWFFITRMFIRRWTRLOSWFRLCVRNGSCVEWJLJTOIBVTKEMZGHNVIEFIGYEUKVP\
     TKEWFRHCRJTUEGVAWSZKSHLWFVHRREDRVVIAQDFMEUAXRIQAKCEDSKWOUTYVTLMVSELNXW\
     OUTLEAWECPFRRLJSXMDVRZICCEYEEKUDLCPCRMV"];
```

In[26]:=
```
Table[{shift, Sum[If[
     ciphertext[[k]] == ciphertext[[k + shift]], 1. / (Length[ciphertext] - shift), 0],
    {k, 1, Length[ciphertext] - shift}]}, {shift, 1, 60}] // TableForm
```

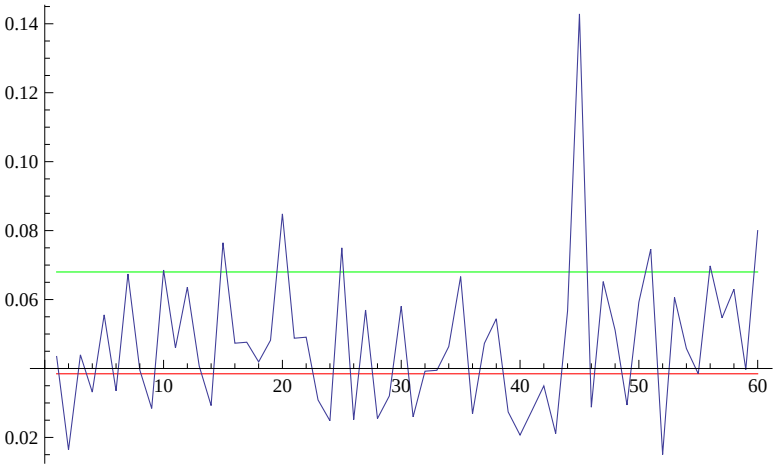
Out[26]//TableForm=

1	0.0434783
2	0.0163934
3	0.043956
4	0.0331492
5	0.0555556
6	0.0335196
7	0.0674157
8	0.039548
9	0.0284091
10	0.0685714
11	0.045977
12	0.0635838
13	0.0406977
14	0.0292398
15	0.0764706
16	0.0473373
17	0.047619
18	0.0419162
19	0.0481928

```
20    0.0848485
21    0.0487805
22    0.0490798
23    0.0308642
24    0.0248447
25    0.075
26    0.0251572
27    0.056962
28    0.0254777
29    0.0320513
30    0.0580645
31    0.025974
32    0.0392157
33    0.0394737
34    0.0463576
35    0.0666667
36    0.0268456
37    0.0472973
38    0.0544218
39    0.0273973
40    0.0206897
41    0.0277778
42    0.034965
43    0.0211268
44    0.0567376
45    0.142857
46    0.028777
47    0.0652174
48    0.0510949
49    0.0294118
50    0.0592593
51    0.0746269
52    0.0150376
53    0.0606061
54    0.0458015
55    0.0384615
56    0.0697674
57    0.0546875
58    0.0629921
59    0.0396825
60    0.08
```

In[25]:=
```
Show[Plot[{1 / 26, .068}, {x, 1, 60}, PlotStyle → {Red, Green}],
 ListPlot[Table[{shift, Sum[If[ciphertext[[k]] == ciphertext[[k + shift]],
       1. / (Length[ciphertext] - shift), 0], {k, 1, Length[ciphertext] - shift}]},
   {shift, 1, 60}], Joined → True], PlotRange → All]
```

Out[25]=



Indeed, the peaks eventually come about every 5, thought not always. The value at 40 is abnormally low, but the value at 45 is abnormally high. This happens with random data. The point is that with enough data, the spacing of 5 between these peaks is apparent, and that indicates the key length is 5.

The lesson is that even encrypting every letter differently doesn't necessarily shred letter frequency attacks.