GLYPH_ANALYSIS

December 31, 2025

This is a companion notebook to Noita Eye Glyphs: Analytical Overview.

This is work the community has done, and should not be confused with my own.

1 Eye message text

The text of all nine messages in their “trigram” form.

[2]:
```
eye_messages = [
    [50, 66, 5, 48, 62, 13, 75, 29, 24, 61, 42, 70, 66, 62, 32, 14, 81, 8, 15,␣
 ↪78, 2, 29, 13, 49, 1, 80, 82, 40, 63, 81, 21, 19, 0, 40, 51, 65, 26, 14, 21,␣
 ↪70, 47, 44, 48, 42, 19, 48, 13, 47, 19, 49, 72, 31, 5, 24, 3, 43, 59, 67,␣
 ↪33, 49, 41, 60, 21, 26, 30, 5, 25, 20, 71, 11, 74, 56, 4, 74, 19, 71, 4, 51,␣
 ↪41, 43, 80, 72, 54, 63, 79, 81, 15, 16, 44, 31, 30, 12, 33, 57, 28, 13, 64,␣
 ↪43, 48],
    [80, 66, 5, 48, 62, 13, 75, 29, 24, 61, 42, 70, 66, 62, 32, 14, 81, 8, 15,␣
 ↪78, 2, 29, 13, 49, 1, 29, 11, 30, 52, 81, 21, 19, 0, 25, 26, 54, 20, 14, 21,␣
 ↪70, 47, 44, 48, 42, 19, 48, 13, 47, 19, 49, 44, 26, 59, 77, 64, 43, 79, 28,␣
 ↪72, 64, 1, 30, 73, 23, 67, 6, 33, 25, 64, 81, 68, 46, 17, 36, 13, 17, 21,␣
 ↪68, 13, 9, 46, 67, 57, 34, 62, 82, 15, 10, 73, 62, 2, 11, 65, 72, 37, 44,␣
 ↪10, 43, 68, 62, 9, 34, 18],
    [36, 66, 5, 48, 62, 13, 75, 29, 24, 61, 42, 70, 66, 62, 32, 14, 81, 8, 15,␣
 ↪78, 2, 29, 13, 49, 1, 69, 76, 52, 9, 48, 66, 80, 22, 64, 57, 40, 49, 78, 3,␣
 ↪16, 56, 19, 47, 40, 80, 6, 13, 64, 29, 49, 64, 63, 6, 49, 31, 13, 16, 10,␣
 ↪45, 24, 26, 77, 10, 60, 81, 61, 34, 54, 70, 21, 15, 4, 66, 77, 42, 37, 30,␣
 ↪22, 0, 11, 41, 72, 57, 20, 23, 57, 65, 41, 23, 18, 72, 42, 5, 3, 26, 78, 8,␣
 ↪5, 54, 45, 77, 25, 64, 61, 16, 44, 54, 51, 20, 63, 25, 11, 26, 45, 53, 60,␣
 ↪38, 34],
    [76, 66, 5, 49, 75, 54, 69, 46, 32, 1, 42, 60, 26, 48, 50, 80, 32, 24, 55,␣
 ↪61, 47, 12, 21, 12, 49, 54, 34, 25, 36, 15, 56, 55, 20, 9, 8, 62, 13, 82, 9,␣
 ↪44, 29, 60, 53, 82, 42, 80, 5, 43, 71, 3, 80, 77, 47, 78, 34, 25, 62, 18,␣
 ↪10, 49, 62, 64, 52, 81, 11, 66, 62, 13, 47, 17, 52, 70, 26, 23, 32, 31, 64,␣
 ↪23, 35, 32, 50, 6, 1, 25, 8, 37, 47, 43, 26, 76, 65, 68, 80, 17, 7, 45, 63,␣
 ↪14, 53, 63, 60, 16],
   [63, 66, 5, 49, 75, 54, 2, 60, 29, 40, 78, 47, 60, 75, 67, 71, 60, 2, 65,␣
 ↪7, 47, 14, 45, 74, 59, 41, 80, 13, 60, 13, 81, 22, 35, 50, 40, 39, 2, 59,␣
 ↪48, 31, 76, 2, 80, 75, 1, 56, 67, 11, 21, 8, 40, 65, 45, 75, 55, 39, 60, 42,␣
 ↪13, 3, 22, 57, 2, 6, 58, 9, 70, 1, 58, 56, 63, 68, 25, 79, 7, 20, 19, 64, 2,␣
 ↪66, 73, 30, 71, 16, 12, 30, 65, 37, 20, 13, 22, 63, 18, 46, 64, 59, 41, 81,␣
 ↪82, 22, 78, 36, 47, 17, 4, 6, 17, 5, 36, 79, 63, 1, 64, 69, 15, 43, 4, 58,␣
 ↪56, 31, 14, 64, 58, 18, 44, 78, 69, 1, 0, 46, 20, 71, 73, 25, 35, 8, 24],
   [34, 66, 5, 49, 75, 54, 23, 74, 11, 13, 28, 26, 19, 48, 67, 57, 37, 60, 34,␣
 ↪28, 74, 10, 17, 32, 11, 18, 19, 43, 19, 81, 42, 4, 62, 9, 46, 49, 32, 51,␣
 ↪76, 58, 4, 43, 47, 17, 67, 79, 21, 32, 44, 16, 30, 37, 26, 28, 41, 68, 57,␣
 ↪34, 51, 10, 69, 70, 8, 6, 46, 43, 18, 39, 47, 43, 15, 13, 33, 30, 35, 62,␣
 ↪37, 0, 37, 5, 38, 55, 37, 13, 40, 25, 9, 21, 11, 64, 5, 79, 42, 68, 11, 71,␣
 ↪11, 48, 3, 67, 61, 40, 22, 14, 35, 50, 61, 39, 11, 2, 66, 49, 51, 53, 17,␣
 ↪73, 36, 75, 74, 54, 24, 30, 54, 70],
   [27, 66, 5, 49, 75, 54, 2, 60, 29, 40, 2, 55, 9, 15, 59, 18, 68, 3, 36, 5,␣
 ↪47, 77, 44, 38, 1, 18, 28, 76, 4, 34, 60, 63, 58, 80, 17, 54, 79, 75, 48,␣
 ↪54, 55, 19, 62, 64, 14, 47, 51, 70, 75, 5, 11, 47, 45, 58, 68, 69, 79, 25,␣
 ↪38, 45, 73, 47, 68, 50, 34, 45, 78, 26, 79, 57, 4, 56, 22, 60, 18, 75, 43,␣
 ↪60, 59, 67, 63, 42, 49, 33, 40, 65, 79, 77, 7, 3, 26, 62, 31, 78, 26, 57,␣
 ↪69, 40, 4, 23, 26, 13, 67, 42, 38, 72, 11, 39, 65, 60, 25, 6, 80, 66, 68,␣
 ↪77, 59, 78, 19],
   [77, 66, 5, 49, 75, 54, 2, 60, 29, 40, 2, 55, 9, 15, 59, 18, 68, 3, 36, 5,␣
 ↪47, 60, 21, 80, 1, 72, 55, 16, 82, 35, 57, 19, 1, 66, 18, 27, 39, 17, 74,␣
 ↪81, 39, 14, 78, 0, 25, 65, 43, 66, 64, 38, 81, 23, 24, 50, 57, 30, 71, 75,␣
 ↪26, 68, 54, 57, 56, 50, 71, 73, 14, 21, 8, 32, 26, 63, 5, 37, 19, 43, 66,␣
 ↪47, 53, 34, 66, 23, 73, 31, 54, 38, 77, 67, 11, 63, 79, 6, 22, 21, 51, 69,␣
 ↪74, 21, 5, 17, 67, 37, 29, 21, 60, 14, 82, 44, 30, 4, 20, 42, 35, 1, 31, 54,␣
 ↪46, 20, 40, 30],
   [33, 66, 5, 49, 75, 54, 2, 60, 29, 40, 2, 55, 9, 15, 59, 18, 68, 3, 36, 5,␣
 ↪47, 33, 21, 59, 44, 18, 28, 76, 59, 34, 60, 63, 79, 27, 12, 54, 5, 49, 48,␣
 ↪54, 55, 52, 62, 72, 69, 10, 57, 22, 58, 48, 67, 53, 7, 34, 32, 30, 31, 19,␣
 ↪26, 8, 34, 46, 7, 30, 71, 55, 34, 75, 54, 9, 6, 60, 5, 23, 25, 45, 42, 80,␣
 ↪25, 12, 22, 76, 20, 51, 62, 21, 40, 9, 41, 10, 44, 73, 8, 33, 70, 73, 6, 31,␣
 ↪21, 72, 5, 40, 61, 51, 42, 66, 64, 74, 61, 25, 63, 42, 24, 41]
]
```

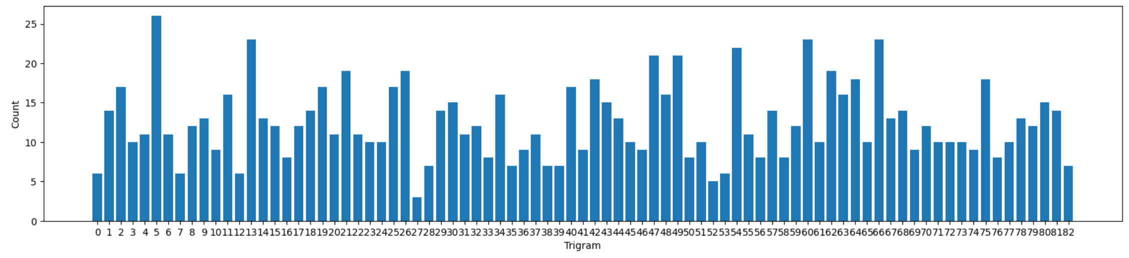
2 Single-letter frequency analysis

Count the number of times each trigram occurs across all messages. Plot the results, and print the most and least common trigrams and the mean and median frequencies.

[3]:
```python
from collections import Counter
from statistics import mean, median
from matplotlib import pyplot as plt

counts = Counter()
```

```
for msg in eye_messages:
  counts += Counter(msg)


print(f"5 most common letters: {[let for let, cnt in counts.most_common()[:
  ↪5]]}")
print(f"5 least common letters: {[let for let, cnt in counts.most_common()[-5:
  ↪]]}")
print(f"Median frequency: {median(counts.values())}")
print(f"Mean frequency: {mean(counts.values())}")

x = range(83)
y = [counts[a] for a in x]
plt.figure(0, (20, 4))
bars = plt.bar(x, y, tick_label=x)
plt.xlabel("Trigram")
plt.ylabel("Count")
plt.show()
```

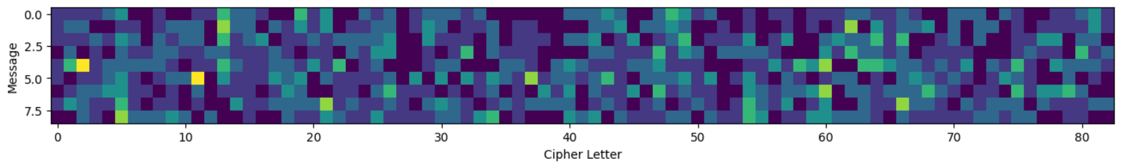
```
5 most common letters: [5, 66, 13, 60, 54]
5 least common letters: [12, 53, 7, 52, 27]
Median frequency: 12
Mean frequency: 12.481927710843374
```



2.1 By message

Use a bitmap image to show how frequently each letter occurs in each message.

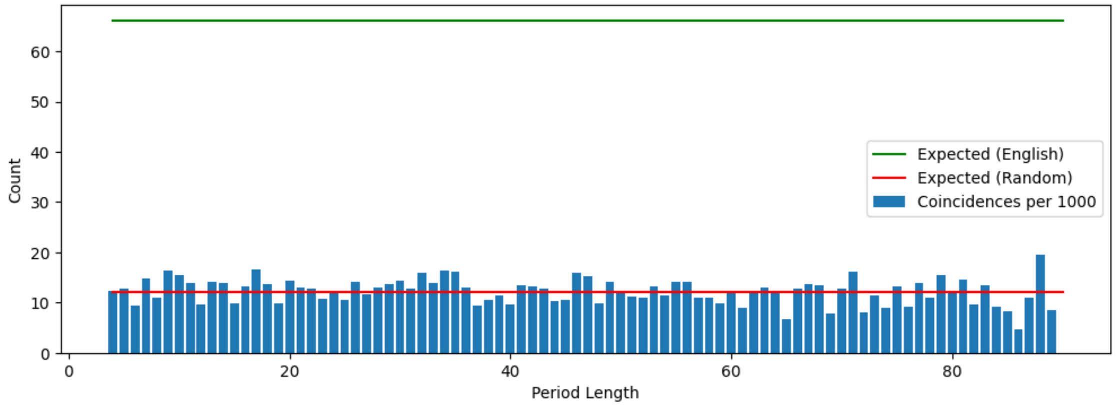
[4]:
```
import numpy as np
im = np.zeros((9, 83), 'i4')
for y, msg in enumerate(eye_messages):
  for let in msg:
    im[y, let] += 1

plt.figure(0, (16, 4))
plt.xlabel("Cipher Letter")
plt.ylabel("Message")
plt.imshow(im)
None
```



3 Kappa test (Periodic)

Superimpose each message against each message and count the number of times the same letter is in the same position in both messages. This gives the rate of coincidence between the two. Repeat the process across a range of offsets. If the cipher is periodic, one of the offsets will show the same rate of coincidence as language, indicating the periods are aligned at that offset.

[5]:
```
bounds = (4, 90)

def kappa_test(msg1, msg2, offset):
    """
    Kappa test for rate of coincidence between distinct ciphertexts.
    Test the two ciphertexts and return an (N, D) tuple, where:
        N is the number of coincidences
        D is the number of tests performed
    """
    m1 = msg1[offset:]
    m2 = msg2[:len(m1)]
    m1 = m1[:len(m2)]
    return sum(a == b for a,b in zip(m1, m2)), len(m1)

x = list(range(*bounds))
results_y = []
for offset in x:
    matches = 0
    checks = 0
    for msg1 in eye_messages:
        for msg2 in eye_messages:
            match, check = kappa_test(msg1, msg2, offset)
            matches += match
            checks += check

    results_y.append(1000 * matches / checks)

plt.figure(0, (12, 4))
plt.bar(x, results_y, 0.8, label="Coincidences per 1000")
plt.plot(bounds, (66, 66), 'g', label="Expected (English)")
plt.plot(bounds, (12, 12), 'r', label="Expected (Random)")
plt.xlabel("Period Length")
plt.ylabel("Count")
plt.legend()
plt.show()
```



4 Kappa test (Positional alphabets)

Use the kappa test again, but without offsets. Test each message against each *other* message, but not itself. If the cipher is polyalphabetic, and always uses the same alphabet at the same position, then the same plain letter will always encipher to the same cipher letter at the same position, which would cause the rate of coincidence from this test to be the same as language.

[6]:
```
def do_test(start=0):
    matches = 0
    checks = 0
    for i in range(9):
        for j in range(i+1, 9):
            match, check = kappa_test(eye_messages[i][start:],␣
 ↪eye_messages[j][start:], 0)
            matches += match
            checks += check
    return checks, matches

checks, matches = do_test()
print(f"== Full messages ==\n"
      f"Tests:            {checks:>5}\n"
      f"Matches:          {matches:>5}\n"
      f"Coincidence rate: {(matches * 1000 // checks):>4} per thousand")
checks, matches = do_test(25)
print(f"== Messages[25:] ==\n"
      f"Tests:           {checks:>5}\n"
      f"Matches:         {matches:>5}\n"
      f"Coincidence rate: {(matches * 1000 // checks):>4} per thousand")
checks, matches = do_test(50)
print(f"== Messages[50:] ==\n"
      f"Tests:           {checks:>5}\n"
      f"Matches:         {matches:>5}\n"
      f"Coincidence rate: {(matches * 1000 // checks):>4} per thousand")
```

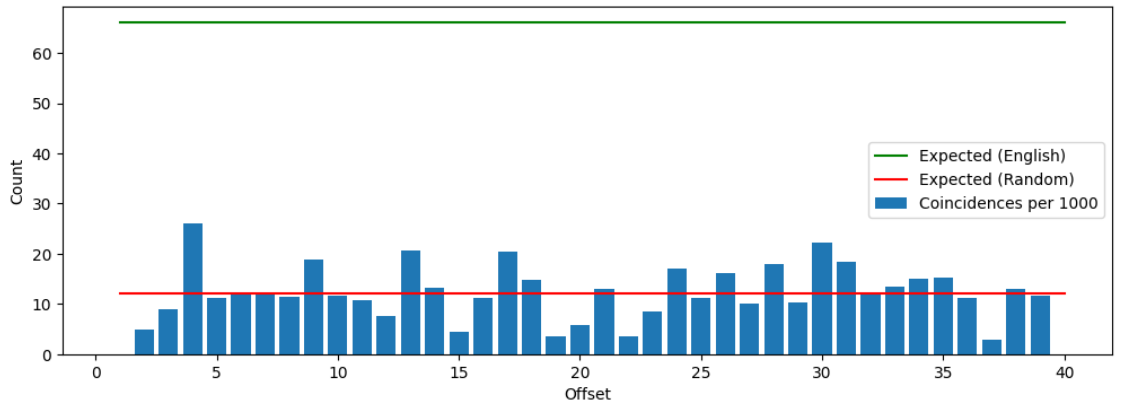
```
== Full messages ==
Tests:             3887
Matches:            338
Coincidence rate:   86 per thousand
== Messages[25:] ==
Tests:             2987
Matches:             76
Coincidence rate:   25 per thousand
== Messages[50:] ==
Tests:             2087
Matches:             31
Coincidence rate:   14 per thousand
```

5 Kappa test (Autocorrelation)

Test each message against itself, across a range of offsets. If this is an autokeyed cipher, the highest test result will give the size of the initial key.

It also happens to count how many times single letters repeat after each offset.

[7]:
```
bounds = (1, 40)
x = list(range(*bounds))
results_y = []
for w in x:
    matches = 0
    checks = 0
    for i in range(9):
        match, check = kappa_test(eye_messages[i], eye_messages[i], w)
        matches += match
        checks += check

    results_y.append(1000 * matches / checks)

plt.figure(0, (12, 4))
plt.bar(x, results_y, 0.8, label="Coincidences per 1000")
plt.plot(bounds, (66, 66), 'g', label="Expected (English)")
plt.plot(bounds, (12, 12), 'r', label="Expected (Random)")
plt.xlabel("Offset")
plt.ylabel("Count")
plt.legend()
plt.show()
```


Count
Offset
Expected (English)
Expected (Random)
Coincidences per 1000