In [77]:
```
import math
import numpy as np
import matplotlib.pyplot as plt
```

Problem 1

{(at, 0.028), (she, 0.030), (that, 0.033), (be, 0.034), (it, 0.035), (was, 0.035), (a, 0.044), (had, 0.076), (he, 0.079), (to, 0.074), (in, 0.094), (of, 0.16), (the, 0.278)}

{((of, the), 0.22), ((in, the), 0.186), ((he, had), 0.092), ((to, the), 0.09), ((of, a), 0.088), ((it, was), 0.07), ((to, be), 0.069), ((that, he), 0.067), ((she, had), 0.061), ((at, the), 0.057))}

(A)

$H[P(X)] = -\sum_x P(x)\,\log_2 P(x)$

In [78]:
```
single_word_freqs = [0.028, 0.030, 0.033, 0.034, 0.035, 0.035, 0.044, 0.076, 0.079, 0.074,
print(sum(single_word_freqs))
entropy = -sum([x*math.log2(x) for x in single_word_freqs])
print(entropy)
```

```
1.0
3.2682218993876324
```

$H[P(X)] = 3.27 \text{ bits}$

$L(x, c) \geq H[P(X)] = 3.27 \text{ bits}$

(B)

The most frequent element should be the shortest and least frequent the longest, so "the" should have the shortest encoding and "at" should have the longest encoding.

(C)

In [79]:
```
single_word_code_lengths = [math.ceil(-math.log2(x)) for x in single_word_freqs]
print(single_word_code_lengths)
```

```
[6, 6, 5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 2]
```

Code lengths: (at, 6), (she, 6), (that, 5), (be, 5), (it, 5), (was, 5), (a, 5), (had, 4), (he, 4), (to, 4), (in, 4), (of, 3), (the, 2)

(D)

$H_2 = H[P(X_2)] = -\sum_{x_2} P(x_2)\,\log_2 P(x_2)$

In [80]:
```
word_pair_freqs = [0.22, 0.186, 0.092, 0.09, 0.088, 0.07, 0.069, 0.067, 0.061, 0.057]
print(sum(word_pair_freqs))
entropy2 = -sum([x2*math.log2(x2) for x2 in word_pair_freqs])
print(entropy2)
```

```
0.9999999999999999
```

```
3.147521464890332
```

$H_2 = H[P(X_2)] = 3.15 \text{ bits}$

$L(x, c) \geq H[P(X_2)] = 3.15 \text{ bits}$

(E)

$2H_1 = 2 * 3.27 \text{ bits} = 6.54 \text{ bits}$

$H_2 = 3.15 \text{ bits}$

$H_2 < 2H_1$, so we should use the word pair code.

The word pair encoding is on average $6.54 - 3.15 = 3.39$ bits shorter per encoding because there are half the amount of encodings.

Problem 2

(A)

There are a total of 3 × 3 = 9 possible positions each able to be 0 or 1 for a total of $2^9 = 512$ possible input patterns.

(B)

$P(x_1) = P(x_2) = P(x_3) = .3$, $x_j = 0.1/509$

$H[P(X)] = -\sum_x P(x) \log_2 P(x)$

In [81]:

```
visual_input_freqs = [0.3,0.3,0.3] + [0.1/509]*509
print(sum(visual_input_freqs))
visual_entropy = -sum([x*math.log2(x) for x in visual_input_freqs])
print(visual_entropy)
```

```
1.00000000000001
2.794614028845932
```

$H[P(X)] = 2.79 \text{ bits}$

(C)

$L(x, c) \geq H[P(X)] = 2.79 \text{ bits}$

(D)

$l(x_i) = -\lceil \log P(x_i) \rceil$

In [82]:

```
visual_input_code_lengths = [math.ceil(-math.log2(x)) for x in visual_input_freqs]
print(visual_input_code_lengths)
# for i in range(len(visual_input_code_lengths)):
#   print("x_"+str(i+1)+";"+str(visual_input_freqs[i]))
```

```
[2, 2, 2, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
```

```
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13]
```

The first three inputs, x_1, x_2, x_3, of probability 0.3 have code lengths of 2, and the rest of the inputs have code lengths of 13.

(E)

Huffman Codes: $x_1 = 01, x_2 = 10, x_3 = 11,$ representitive code $x_j = 00011011011$

If neural codings utilize similar strategies, this implies that neurons may be able to very efficiently code sensory signals that they see commonly relative to other inputs.

Problem 3

Binary symmetric channel with bit-flip probability p

(A)

$$I[X,Y] = \sum_x \sum_y P(X,Y) log \frac{P(X,Y)}{P(X)P(Y)}$$

$$P(X,Y) = P(X)P(Y|X)$$

x	y	P(x)	P(y)	P(y\	x)	P(x,y)
0	0	$\frac{1}{2}$	$\frac{1}{2}$	1-p	$\frac{1-p}{2}$	
0	1	$\frac{1}{2}$	$\frac{1}{2}$	p	$\frac{p}{2}$	
1	0	$\frac{1}{2}$	$\frac{1}{2}$	p	$\frac{p}{2}$	
1	1	$\frac{1}{2}$	$\frac{1}{2}$	1-p	$\frac{1-p}{2}$	

$$
\begin{aligned}
I[X,Y] &= \sum_x \sum_y P(X,Y) log \frac{P(X,Y)}{P(X)P(Y)} && (1)\\
&= \frac{1-p}{2} log \frac{\frac{1-p}{2}}{\frac{1}{2} * \frac{1}{2}} + \frac{p}{2} log \frac{\frac{p}{2}}{\frac{1}{2} * \frac{1}{2}} + \frac{p}{2} log \frac{\frac{p}{2}}{\frac{1}{2} * \frac{1}{2}} + \frac{1-p}{2} log \frac{\frac{1-p}{2}}{\frac{1}{2} * \frac{1}{2}} && (2)\\
&= \frac{1-p}{2} log(2-2p) + \frac{p}{2} log(2p) + \frac{p}{2} log(2p) + \frac{1-p}{2} log(2-2p) && (3)\\
&= (1-p)\ log(2-2p) + p\ log(2p) && (4)
\end{aligned}
$$

(B)

$P(Y|X) =$ product over the bits $x \in X$ and their corresponding bits $y \in Y$ of $P(y|x)$

(C)

Given an observed output Y, we would calculated $P(Y|X)$ for all possible inputs X and choose the binary word X with the highest $P(Y|X)$.

(D)

$p = .2,\ N = 5$

The probability that a code word will experience 0 or 1 bit flips is $P($0 bits flips$)+P($1 bit flip$)$.

$P($0 bits flips$)+P($1 bit flip$)= (1-0.2)^5 + 5(1-0.2)^4(0.2) = 0.73728$

(E)

For a length of 5, to be able to have all single bit flips corrected, we must have no overlaps between any single bit flips of any input words. This means there must be 3 bits different between any two input words, so we can construct a 4 word code but not a 5 word code.

$00000, 00111, 11100, 11011$

Problem 4

(A)

$P(y) = 1/5$ for all y

$H[P(Y)] = -\sum_y P(y)\ \log P(y) = -5\frac{1}{5} log \frac{1}{5} = log\ 5$

(B)

$P(z|y) = \frac{(p_c y)^z e^{(-p_c y)}}{z!}$, where p_c is the receptor counting efficiency

$P(z,y) = P(y)P(z|y) = \frac{1}{5}\frac{(p_c y)^z e^{(-p_c y)}}{z!}$

$P(z) = \sum_{y \in Y} P(z,y) = \sum_{y \in Y} \frac{1}{5}\frac{(p_c y)^z e^{(-p_c y)}}{z!}$

(C)/(D)

In [119…

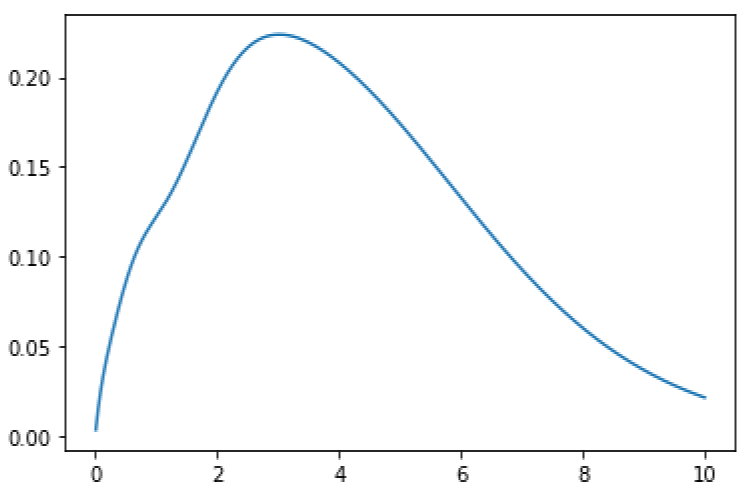
```
def Py(y):
  return 1/5

def poisson(z,y,pc):
  return (pc*y)**z*math.exp(-pc*y)/math.factorial(z)

def P(z,y,pc):
  return Py(y)*poisson(z,y,pc)

def Pz(z,Y,pc):
  total = 0
  for y in Y:
    total += P(z,y,pc)
  return total

def mutual(Z,Y,pc):
  total = 0
  for z in Z:
    for y in Y:
      total += P(z,y,pc)*math.log(P(z,y,pc)/(Pz(z,Y,pc)*Py(y)))
  return total
```

In [120…

```
Z=[1,2,3,4,5]
Y=[1,2,3,4,5]
pcArray = np.linspace(0.01,10, num = 500)
mutualArray = []
for pc in pcArray:
  mutualArray.append(mutual(Z,Y,pc))
plt.plot(pcArray, mutualArray)
```

Out[120…

```
[<matplotlib.lines.Line2D at 0x7fb91291f250>]
```



(E)

If we increase the distance between different signals then we can increase the mutual information between Z and Y as they are more likely to be the same number.

In [135…

```
Z= range(1,42)
Y=[1,16,31,46,61]
pcArray = np.linspace(0.01,10, num = 500)
mutualArray = []
for pc in pcArray:
  mutualArray.append(mutual(Z,Y,pc))
```

```python
print(max(mutualArray))
plt.plot(pcArray, mutualArray)
```

```
0.913790390232016
```

Out[135…

```
[<matplotlib.lines.Line2D at 0x7fb90754cd10>]
```