ENGSCI 760 Assignment 2

Aiden Burgess - 600280511

1 Joint Distributions

1.1 Pairwise Independent

The events A and B are said to be independent iff $Pr(A \cap B) = Pr(A) \times Pr(B)$

1.1.1 Individual A

A is the event that the sum of the dice is 7

	1	**2**	**3**	**4**	**5**	**6**
1	2	3	4	5	6	**7**
2	3	4	5	6	**7**	8
3	4	5	6	**7**	8	9
4	5	6	**7**	8	9	10
5	6	**7**	8	9	10	11
6	**7**	8	9	10	11	12

The probability of A occurring is 6/36 => 1/6

1.1.2 Individual B

B is the event that die #1 is 2 or less

	1	**2**	**3**	**4**	**5**	**6**
1	**2**	**3**	4	5	6	7
2	**3**	**4**	5	6	7	8

	1	2	3	4	5	6
3	**4**	**5**	6	7	8	9
4	**5**	**6**	7	8	9	10
5	**6**	**7**	8	9	10	11
6	**7**	**8**	9	10	11	12

The probability of B occuring is 12/36 => 1/3

1.1.3 Individual C

B is the event that die #2 is odd

	1	2	3	4	5	6
1	**2**	**3**	**4**	**5**	**6**	**7**
2	3	4	5	6	7	8
3	**4**	**5**	**6**	**7**	**8**	**9**
4	5	6	7	8	9	10
5	**6**	**7**	**8**	**9**	**10**	**11**
6	7	8	9	10	11	12

The probability of C occuring is 18/36 => 1/2

1.1.4 A and B

A and B is the event that the sum of the dice is 7, and die #1 is 2 or less

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9

	1	2	3	4	5	6
4	5	6	7	8	9	10
5	6	**7**	8	9	10	11
6	**7**	8	9	10	11	12

$$Pr(A \cap B) = 2/36 = 1/18$$

$$Pr(A) \times Pr(B) = 1/6 \times 1/3 = 1/18$$

$$Pr(A \cap B) = Pr(A) \times Pr(B)$$

Therefore, A and B are independent

1.1.5 A and C

A and C is the event that the sum of the dice is 7, and die #2 is odd

	1	2	3	4	5	6
1	2	3	4	5	6	**7**
2	3	4	5	6	7	8
3	4	5	6	**7**	8	9
4	5	6	**7**	8	9	10
5	6	**7**	8	9	10	11
6	7	8	9	10	11	12

$$Pr(A \cap C) = 3/36 = 1/12$$

$$Pr(A) \times Pr(C) = 1/6 \times 1/2 = 1/3$$

$$Pr(A \cap C) = Pr(A) \times Pr(C)$$

Therefore, A and C are independent

1.1.6 B and C

A and B is the event that the die #1 is 2 or less, and die #2 is odd

	1	2	3	4	5	6

	1	2	3	4	5	6
1	**2**	**3**	4	5	6	7
2	3	4	5	6	7	8
3	**4**	**5**	6	7	8	9
4	5	6	7	8	9	10
5	**6**	**7**	8	9	10	11
6	7	8	9	10	11	12

$$Pr(B \cap C) = 6/36 = 1/6$$

$$Pr(B) \times Pr(C) = 1/3 \times 1/2 = 1/6$$

$$Pr(B \cap C) = Pr(B) \times Pr(C)$$

Therefore, B and C are independent

1.1.7 Explanation

The events A, B, C are pairwise independent as each pairing of A, B, and C is independent.

1.2 Mutually Independent

The events A*{1},A*{2},...A_{n} are said to be mutually independent iff the intersection of their probabilities is equal to the product of their individual probabilities.

Therefore, let us calculate the probability of all events occurring together, and the individual probabilities for each event.

1.2.1 Intersection of A, B, C

The intersection of A, B, C is the event where: sum of dices is 7, dice #1 is 2 or less, dice #2 is odd

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9

	1	2	3	4	5	6
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

The probability of this event occurring is 1/36

1.2.2 Explanation

$$Pr(A \cap B \cap C) = 1/36$$

$$Pr(A) \times Pr(B) \times Pr(C) = 1/36$$

$$Pr(A \cap B \cap C) = Pr(A) \times Pr(B) \times Pr(C)$$

Therefore, A, B, C are mutually independent events, as the product of individual probabilities is equal to the intersection of these events.

2 Markov Chains

2.1 Step One Markov Transition Matrix

Let us set the order of states to be: 0,1,2,3,4

$$\begin{matrix} 0.1 & 0.2 & 0.3 & 0.3 & 0.1 \\ 0.9 & 0.1 & 0 & 0 & 0 \\ 0.6 & 0.3 & 0.1 & 0 & 0 \\ 0.3 & 0.3 & 0.3 & 0.1 & 0 \\ 0.1 & 0.2 & 0.3 & 0.3 & 0.1 \end{matrix}$$

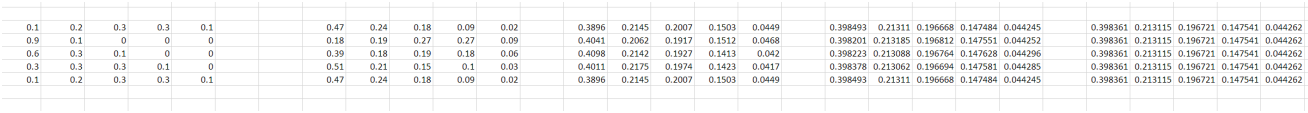
2.2 Limiting Distribution for number of Ohms

To find the limiting distribution, we can solve the following equation:

$$[v_1\ v_2\ v_3\ v_4\ v_5] = [v_1\ v_2\ v_3\ v_4\ v_5] \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.3 & 0.1 \\ 0.9 & 0.1 & 0 & 0 & 0 \\ 0.6 & 0.3 & 0.1 & 0 & 0 \\ 0.3 & 0.3 & 0.3 & 0.1 & 0 \\ 0.1 & 0.2 & 0.3 & 0.3 & 0.1 \end{bmatrix}$$

Alternatively, we may transition through increasing powers of the original state matrix until we reach the same values for every row.

Working here:



0.1	0.2	0.3	0.3	0.1		0.47	0.24	0.18	0.09	0.02		0.3896	0.2145	0.2007	0.1503	0.0449		0.398493	0.21311	0.196668	0.147484	0.044245		0.398361	0.213115	0.196721	0.147541	0.044262
0.9	0.1	0	0	0		0.18	0.19	0.27	0.27	0.09		0.4041	0.2062	0.1917	0.1512	0.0468		0.398201	0.213185	0.196812	0.147551	0.044252		0.398361	0.213115	0.196721	0.147541	0.044262
0.6	0.3	0.1	0	0		0.39	0.18	0.19	0.18	0.06		0.4098	0.2142	0.1927	0.1413	0.042		0.398223	0.213088	0.196764	0.147628	0.044296		0.398361	0.213115	0.196721	0.147541	0.044262
0.3	0.3	0.3	0.1	0		0.51	0.21	0.15	0.1	0.03		0.4011	0.2175	0.1974	0.1423	0.0417		0.398378	0.213062	0.196694	0.147581	0.044285		0.398361	0.213115	0.196721	0.147541	0.044262
0.1	0.2	0.3	0.3	0.1		0.47	0.24	0.18	0.09	0.02		0.3896	0.2145	0.2007	0.1503	0.0449		0.398493	0.21311	0.196668	0.147484	0.044245		0.398361	0.213115	0.196721	0.147541	0.044262

Therefore,

$$[v_1\ v_2\ v_3\ v_4\ v_5] = [0.398 \quad 0.213 \quad 0.197 \quad 0.148 \quad 0.044]$$

i.e. The expected amount of remaining stock of Ohms per week is as follows

# Ohms	Probability
0	0.398
1	0.213
2	0.197
3	0.147
4	0.044

2.3 Given 2 Ohms in stock, expected # weeks before 2 ohms again?

The mean return time is defined by the following equation, with i representing the starting state, and k representing an intermediate state.

$$r_i = 1 + \sum_{k=1}^{n} p_{ik} m_{ik}$$

Therefore if $i = 2$

$$r_2 = 1 + p_{20} m_{02} + p_{21} m_{21} + p_{22} M_{22} + p_{23} m_{32} + p_{24} m_{24}$$

2.4 Calculate Weekly Profit @ Steady State

From above we know the probability of the steady state at the end of each week.

# Ohms	Profit	Weighted Profit
0	$4 \times 5000 - 6000 = 14000$	5572

# Ohms	Profit	Weighted Profit
1	-100	-21.30
2	-200	-39.40
3	-300	-44.40
4	-400	-17.60

$$Expected\_weekly\_profit = 5572 - 21.30 - 39.40 - 44.40 - 17.60 = \$5449.30$$

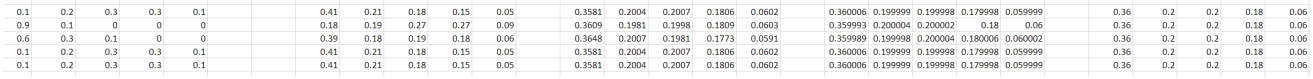
2.5 Better Reorder Policy

Reorder when there are either 0 or 1 Ohms in stock

Markov transition matrix:

$$\begin{matrix} 0.1 & 0.2 & 0.3 & 0.3 & 0.1 \\ 0.9 & 0.1 & 0 & 0 & 0 \\ 0.6 & 0.3 & 0.1 & 0 & 0 \\ 0.1 & 0.2 & 0.3 & 0.3 & 0.1 \\ 0.1 & 0.2 & 0.3 & 0.3 & 0.1 \end{matrix}$$

Steady state distribution working here:



# Ohms	Probability
0	0.36
1	0.2
2	0.2
3	0.18
4	0.06

# Ohms	Profit	Weighted Profit
0	$4 \times 5000 - 6000 = 14000$	5040

# Ohms	Profit	Weighted Profit
1	$3 \times 5000 - 6000 - 100 = 8900$	1780
2	-200	-40
3	-300	-54
4	-400	-24

Expected weekly profit = 5040+1780-40-54-24=$6702

3 Hidden Markov Model

3.1 constructEmissions()

```
def constructEmissions(pr_correct, adj):
    # This function takes in a matrix detailing the adjacent letters on
a keyboard, and the
    #  probability of hitting the correct key and outputs a matrix of
emission probabilities
    #
    # INPUT
    # pr_correct - the probability of correctly hitting the intended
key;
    # adj - a 26 x 26 matrix with adj[i][j] = 1 if the i'th letter in
the alphabet is adjacent
    # to the j'th letter.
    #
    # OUTPUT
    # b - a 26 x 26 matrix with b[i][j] being the probability of hitting
key j if you intended
    # to hit key i (the probabilities of hitting all adjacent keys are
identical).

    # Initialise empty 26x26 array
    b = [[0 for i in range(26)] for j in range(26)]

    for i, row in enumerate(adj):
        # Get number of keys by adding together all 1's
        num_keys = sum(row)
        # Calculate probability for remaining keys
        prob_rest = (1-pr_correct)/(num_keys)
        for j, letter in enumerate(row):
            if i == j:
                # Intended key has different probability
                b[i][j] = pr_correct
```

```
            elif letter == 1:
                b[i][j] = prob_rest
    return b
```

3.2 constructTransitions()

```
def constructTransitions(filename):
    # This function constructs tranisition matricies for lowercase
characters.
    # It is assumed that the file 'filename' only contains lowercase
characters
    # and whitespace.
    # INPUT
    #  filename is the file containing the text from which we wish to
develop a
    #  Markov process.
    #
    # OUTPUT
    #  p is a 26 x 26 matrix containing the probabilities of transition
from a
    #  state to another state, based on the frequencies observed in the
text.
    #  prior is a vector of prior probabilities based on how often each
character
    #  appears in the text

    # Read the file into a sting called text
    with open(filename, 'r') as myfile:
        text = myfile.read()
        # Your code goes here.
        import re
        # Initialise empty 26x26 array to store counts of each letter
transition
        counts = [[0 for i in range(26)] for j in range(26)]
        for word in text.split():
            # Remove any punctuation/whitespace
            word = re.sub(r'\W+', '', word)
            # Index by 0, i.e. a: 0, b: 1...
            prev = ord(word[0]) - 97
            for letter in word[1:]:
                letter = ord(letter) - 97
                counts[prev][letter] += 1
                prev = letter

    # Intialise empty 26x26 array to store percentages of each letter
transition
    p = [[0 for i in range(26)] for j in range(26)]
    for starting_letter, row in enumerate(counts):
        total_per_row = sum(row)
```

```
        for ending_letter, count_for_letter in enumerate(row):
            p[starting_letter][ending_letter] =
count_for_letter/total_per_row

    # Calculate priors
    from collections import Counter
    letter_counts = Counter(text)
    total_chars = sum(letter_counts[chr(i+97)] for i in range(26))
    prior = [letter_counts[chr(i+97)]/total_chars for i in range(26)]

    return (p, prior)
```

3.3 HMM

```
def HMM(p, pi, b, y):
    # This function implements the Viterbi algorithm, to find the most
likely
    # sequence of states given some set of observations.
    #
    # INPUT
    #  p is a matrix of transition probabilies for states x;
    #  pi is a vector of prior distributions for states x;
    #  b is a matrix of emission probabilities;
    #  y is a vector of observations.
    #
    # OUTPUT
    # x is the most likely sequence of states, given the inputs.

    n = len(y)
    m = len(pi)

    gamma = {}
    phi = {}

    # You must complete the code below
    for i in range(26):
        # Your code goes here (initialisation)
        y_1 = int(y[0])
        b_k = b[i][y_1]
        pi_k = pi[i]
        gamma_k = b_k * pi_k
        gamma[i, 0] = gamma_k

    for t in range(1, n):
        for k in range(26):
            gamma[k, t] = 0

            for j in range(26):
                # Your code goes here
```

```
                y_t = y[t]
                b_k = b[k][y_t]
                p_jk = p[j][k]
                gamma_j = gamma[j, t-1]
                new_prob = b_k*p_jk*gamma_j
                # Update gamma[k,t] to max
                if new_prob > gamma[k, t]:
                    gamma[k, t] = new_prob
                    # Update phi[k,t] to be letter which causes
transition
                    phi[k, t] = j

    best = 0
    x = []
    for t in range(n):
        x.append(0)

    # Find the final state in the most likely sequence x(n).
    for k in range(26):
        if best <= gamma[k, n-1]:
            best = gamma[k, n-1]
            x[n-1] = k

    for i in range(n-2, -1, -1):
        # Your code goes here
        x[i] = (phi[x[i+1], i+1])
        pass
    return x
```

3.4 main()

```
def main():
    # The text messages you have received.
    msgs = []
    # msgs.append('cljlx')
    msgs.append(
        'cljlx ypi ktxwf a pwfi psti vgicien aabdwucg vpd me and vtiex
voe zoicw')
    # could you
    msgs.append('qe qzby yii tl gp tp yhr cpozwdt fwstqurzby')
    # we want you to go to our
    msgs.append('qee ypi xfjvkjv ygetw ib ulur vae')
    # see you
    msgs.append('wgrrr zrw uiu')
    msgs.append('hpq fzr qee ypi vrpm grfw')
    # how far are you from
    msgs.append('qe zfr xtztvkmh')
    # we are
    msgs.append('wgzf tjmr will uiu xjoq jp ywfw')
```

```
    # The probability of hitting the intended key.
    pr_correct = 0.5

    # An adjacency matrix, adj(i,j) set to 1 if the i'th letter in the
alphabet is next
    # to the j'th letter in the alphabet on the keyboard.
    adj = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
0, 0, 1, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0], [0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], [0, 0, 1, 1, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0], [0, 1,
0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0],
[0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [
        0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1], [0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0], [0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0], [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]]

    # Call a function to construct the emission probabilities of hitting
a key
    # given you tried to hit a (potentially) different key.
    b = constructEmissions(pr_correct, adj)

    # Call a function to construct transmission probabilities and a
prior distribution
    # from the King James Bible.
    [p, prior] = constructTransitions('bible.txt')
```

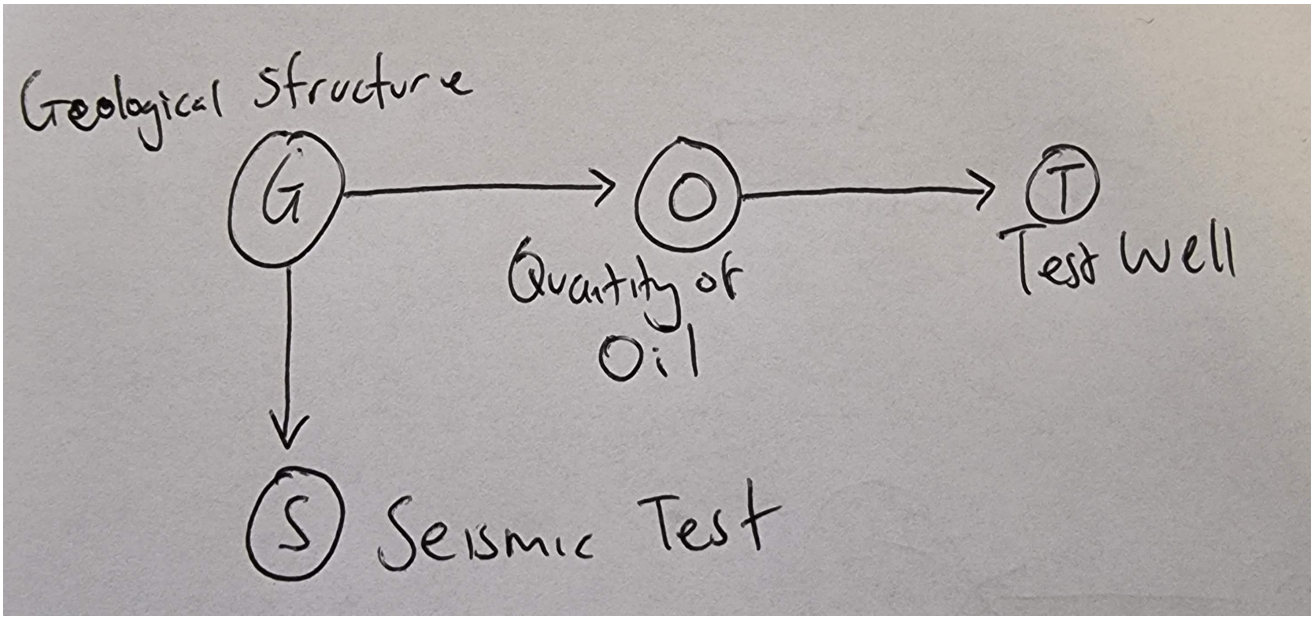
```
    # Run the Viterbi algorithm on each word of the messages to
determine the
    # most likely sequence of characters.
    for msg in msgs:
        s_in = msg.split(' ')  # divide each message into a list of
words

        decoded_message = ''

        for i in range(len(s_in)):
            y = []

            for j in range(len(s_in[i])):
                # convert the letters to numbers 0-25
                y.append(ord(s_in[i][j])-97)

            x = HMM(p, prior, b, y)  # perform the Viterbi algorithm

            output = ''
            for j in range(len(x)):
                # convert the states x back to letters
                output = output+chr(x[j]+97)

            if i != len(s_in):
                decoded_message += output+' '  # recreate the message

        print(msg)  # display received message
        print(decoded_message)  # display decoded message
        print('')
```

3.5 Intended Text Messages

```
could you order a peri peri chicken sandwich for me and fries for alice

we want you to go to the closest restaurant

are you driving there in your car

where are you

how far are you from here

we are starving

what time will you show up here
```

4 Bayesian Network

4.1 Drawing



4.2 Conditional Independence

Conditional independence is not the same as independence. The probability distribution for the quantity of oil is dependent on the seismic test, as the seismic test can provide information about the geological structure which then gives informs us of the probability distribution of the quantity of oil. However, if we already know the geological structure, then the information that the seismic test provides is irrelevant, which is the concept of conditional independence.

4.3 Probability Distributions

4.3.1 Seismic Positive, Test Well not Drilled

```
G: [0.25925926 0.59259259 0.14814815]
O: [0.54814815 0.26296296 0.18888889]
S: [0. 1.]
T: [0.36592593 0.63407407]
```

4.3.2 Seismic Negative, Test Well Positive

```
G: [0.1626506  0.18674699 0.65060241]
O: [0.55421687 0.35692771 0.08885542]
S: [1. 0.]
T: [0. 1.]
```

4.3.3 Test Well Negative, no Seismic Test

```
G: [0.14285714 0.3877551  0.46938776]
O: [0.24489796 0.30612245 0.44897959]
S: [0.49591837 0.50408163]
T: [1. 0.]
```