# EE 381 Project 2
# Conditional Probabilities:
# Communication through a Noisy Channel

by Alexandra McBride

October 2, 2019

---

California State University, Long Beach
Dr. Anastasios Chassiakos

## Problem 1

### Introduction

This problem involves a digital message sent across a noisy communication channel where the message is transmitted as bits, which consists of only zeros and ones. In this scenario, we are calling the sent message $S$ and the received message $R$, where both are one-bit messages. We want to find the probability that the message was received incorrectly, meaning that the corresponding $S$ and $R$ messages did not match, and the transmission was a failure. Through this experiment, we will see how often the message is received incorrectly, and overall how the probability of it being incorrect will affect the outcome.

### Methodology

For this problem, I will be using Python in the PyCharm IDE. The tools used will include methods from the `numpy` library and a method called `nSidedDie`. In the program, $S$ and $R$ will both be generated as lists containing 100,000 elements in order to simulate the repetition of the experiment for $N$= 100,000 times.

To generate the values for $S$, we will use probabilities $p_0$ and $p_1$, where $p_0$ is the probability of S being 0 and $p_1$ is the probability of $S$ being 1. By using the `nSidedDie` function, we will pass in the probabilities list $[p_0, p_1]$ and generate either a 0 or 1.

To generate the values for $R$, we will use probabilities $e_0$ and $e_1$, where $e_0$ is the probability of error when transferring $R$ if $S$ is 0 and $e_1$ is the probability of error when transferring $R$ if $S$ is 1. By using the `nSidedDie` function, we will pass in the probabilities and generate either a 0 or 1. It will assign the R bit in this way:

$$R = \begin{cases} 1 \; if \;\; S = 0 \; and \;\; t \leq \; e_0 \\ 0 \; if \;\; S = 0 \; and \;\; t > \; e_0 \\ 1 \; if \;\; S = 1 \; and \;\; t > \; e_1 \\ 0 \; if \;\; S = 1 \; and \;\; t \leq \; e_1 \end{cases}$$

If $S$ = 0, we pass in the list [1-$e_0, e_0$] into `nSidedDie`() and if $S$ =1 we pass in the list $[e_1, 1 - e_1]$ into `nSidedDie`().

Once we have generated $S$ and $R$, we will then compare each corresponding $S$ and $R$ value and see if they are the same. We will gather our results by keeping a counter for how many times the transmission failed and divide it by $N$ to find the probability of failure.

### Results and Conclusion

The values for each of the probabilities in the test case are the following:

$$p_0 = 0.35 \;\; p_1 = 1 - p_0 = 0.65 \quad e_0 = 0.04 \;\; e_1 = 0.07$$

After running the program to simulate this, we get the following numerical result:

| Probability of Transmission Error | |
|---|---|
| Answer | $p$ =0.05987 |

This result is based on the probabilities $e_0$ and $e_1$ that were set for generating the $R$ values, and we see that it is a correct result from those given probabilities. To conclude, we see that in this case that the sent message has a very low probability of being received incorrectly, which means that the most transmissions will be successful.

## Appendix

Here are the programs used for this problem:

problem1.py

```python
import generatorForSAndR as gen

def problem1():
    N=100000

    #Generating S,R
    S,R = gen.generatorForSAndR()
    numOfErrors = 0

    #Comparing S and R values
    for k in range(N):
        if S[k] != R[k]:
            numOfErrors += 1

     #Displaying probability
    print("Probability of failure during transmission: ",numOfErrors/N)

problem1()
```

generatorForSAndR.py

```python
import numpy as np
import nSidedDie as nsd

def generatorForSAndR():
    # Setting N, p0, p1
    N = 100000
    p0 = 0.35
    p1 = 1 - p0

    # Creating S - Sent bits
    S = np.zeros((N, 1))

    # Creating list of probablities of bit being 0 or 1 and cumulative sums list
    probs01 = [p0, p1]


    # Creating R - Received bits
    R = np.zeros((N, 1))
    # Setting e0, e1
    e0 = 0.04
    e1 = 0.07

    # Generating 100,000 bits for S and R
    for i in range(N):
        sBit = (nsd.nSidedDie(probs01)) - 1
        S[i] = sBit
        if S[i] == 0:
            rBit = nsd.nSidedDie([1 - e0,e0])-1
            R[i]=rBit
        elif S[i] == 1:
            rBit = nsd.nSidedDie([e1, 1 - e1])-1
            R[i] = rBit
        #print(R[i])
    return S,R
```

nSidedDie.py

```
mport numpy as np
def nSidedDie(p):
    #Getting n (number of sides)
    n=len(p)

    #Setting up the cumulative sum
    cumulProbs = np.cumsum(p)  # array of sums
    cumulProbs0 = np.append(0, cumulProbs)

    #random number between 0 and 1
    randNum = np.random.rand()
    for j in range(0, len(p)):
        # checks for the range that the dieRoll is in and get the probability for that
range
        if randNum > cumulProbs0[j] and randNum <= cumulProbs0[j + 1]:
            side = j+1
            return side
    return 0
```

## Problem 2

### Introduction

This problem involves the same scenario as mentioned in the Problem 1 Introduction. For this second experiment, we want to find the conditional probability ($S$ =1| $R$ =1), or that the received message $R$ is one given that the sent message $S$ is one. Through the experiment, we will see how often the message is correctly received as one and allow us to better understand the rate of successful transmissions.

### Methodology

For this problem, I will be using Python in the PyCharm IDE. The tools used will include methods from the `numpy` library and a method called `nSidedDie`. The methodology will be the same for generating $S$ and $R$ as in the Problem 1 Methodology. To find the probability that ($R$ =1| $S$ =1), we will check every element in $S$ and see if it equals one. If it does, it will add to a counter to keep count of the number of messages that were sent as one. The program will then check that the corresponding received message is one and add this to another counter that keeps track of the number of messages that were received as one. The results will be gathered by dividing the number of times that $R$ is one by the number of times that $S$ is one.

### Results and Conclusion

We will use the same test case values for the probabilities as in the Problem 1 Results and Conclusion section.

After running the program to simulate this, we get the following numerical result:

| Conditional Probability $p(R =1 \mid S=1)$ | |
|---:|---|
| Answer | $p$ = 0.929504560019719 |

In these results, if a message is sent as one, then the probability that it will be received as one is very high. To conclude, the rate of successful transmission is high, just as we concluded in the Problem 1 with the given probabilities.

### Appendix

Here are the programs used for this problem:

problem2.py

```
import generatorForSAndR as gen

#PROBLEM 2:  #Probability (S=1 | R=1)
```

```python
def problem2():

    N = 100000

    #Generating S,R
    S, R = gen.generatorForSAndR()

    #Keeping track of S=1 and R=1
    numberOfS1 = 0
    numberOfSuccessesforR = 0

    #Comparing S,R
    for k in range(N):
     if (S[k] == 1):
        numberOfS1 += 1
        if (R[k] == 1):
            numberOfSuccessesforR += 1

     #Displaying Probability
    print("Probability of successful transmission when the S bit is 1: ",
numberOfSuccessesforR / numberOfS1)

problem2()
```

generatorForSAndR.py

```python
import numpy as np
import nSidedDie as nsd

def generatorForSAndR():
    # Setting N, p0, p1
    N = 100000
    p0 = 0.35
    p1 = 1 - p0

    # Creating S - Sent bits
    S = np.zeros((N, 1))

    # Creating list of probablities of bit being 0 or 1 and cumulative sums list
    probs01 = [p0, p1]


    # Creating R - Received bits
    R = np.zeros((N, 1))

    # Setting e0, e1
    e0 = 0.04
    e1 = 0.07

    # Generating 100,000 bits for S and R
    for i in range(N):
        sBit = (nsd.nSidedDie(probs01)) - 1
        S[i] = sBit
        if S[i] == 0:
            rBit = nsd.nSidedDie([1 - e0,e0])-1
            R[i]=rBit
        elif S[i] == 1:
            rBit = nsd.nSidedDie([e1, 1 - e1])-1
            R[i] = rBit
        #print(R[i])
    return S,R
```

nSidedDie.py

```python
mport numpy as np
def nSidedDie(p):
    #Getting n (number of sides)
    n=len(p)

    #Setting up the cumulative sum
    cumulProbs = np.cumsum(p)  # array of sums
    cumulProbs0 = np.append(0, cumulProbs)

    #random number between 0 and 1
    randNum = np.random.rand()
    for j in range(0, len(p)):
        # checks for the range that the dieRoll is in and get the probability for that
range
        if randNum > cumulProbs0[j] and randNum <= cumulProbs0[j + 1]:
            side = j+1
            return side
    return 0
```

## Problem 3

### Introduction

This problem involves the same scenario as mentioned in the Problem 1 Introduction. For this third experiment, we want now to find the conditional probability that ($S$ =1| $R$ =1), or that the sent message $S$ is one given that the received message $R$ is one. Through the experiment, we will see how often the message is correctly received as one and allow us to better understand the rate of successful transmissions.

### Methodology

For this problem, I will be using Python in the PyCharm IDE. The tools used will include methods from the `numpy` library and a method called `nSidedDie`. The methodology will be the same for generating $S$ and $R$ as in the Problem 1 Methodology. To find the probability that ($S$ =1| $R$ =1), we will check every element in $R$ and see if it equals one. If it does, it will add to a counter to keep count of the number of messages that were received as one. The program will then check that the corresponding sent message is one and add this to another counter that keeps track of the number of messages that were sent as one. The results will be gathered by dividing the number of times that $S$ is one by the number of times that $R$ is one.

### Results and Conclusion

We will use the same test case values for the probabilities as in the Problem 1 Results and Conclusion section.

After running the program to simulate this, we get the following numerical result:

| Conditional Probability $p(S=1 \mid R=1)$ | |
|---|---|
| Answer | $p$ =0.9771250282906011 |

In these results, if the message was received as one, then the probability that it was sent as one is very high. Compared to Problem 2 results, when looking first at all received messages that were one, then comparing the message sent, it seems that is it is more likely that when it is received as one, then the message matched the sent one more often. To conclude, the rate of successful transmission is high, just as we concluded in the Problem 1 and 2 with given probabilities.

### Appendix

Here are the programs used for this problem:

problem3.py

```python
import generatorForSAndR as gen

#PROBLEM 3:  #Probability (R=1 | S=1)
def problem3():

    N=100000

    #Generating S,R
    S,R=gen.generatorForSAndR()

    #Keeping track of R=1 and S=1
    numberOfR1=0
    numberOfSuccessesforS=0

    #Comparing S and R
    for k in range(N):
        if (R[k] == 1):
            numberOfR1 += 1
            if (S[k] == 1):
                numberOfSuccessesforS += 1

    #Displaying probability
    print("Probability of successful transmission when the R bit is 1: ",
numberOfSuccessesforS / numberOfR1)
```

generatorForSAndR.py

```python
import numpy as np
import nSidedDie as nsd

def generatorForSAndR():
    # Setting N, p0, p1
    N = 100000
    p0 = 0.35
    p1 = 1 - p0

    # Creating S - Sent bits
    S = np.zeros((N, 1))

    # Creating list of probablities of bit being 0 or 1 and cumulative sums list
    probs01 = [p0, p1]


    # Creating R - Received bits
    R = np.zeros((N, 1))

    # Setting e0, e1
    e0 = 0.04
    e1 = 0.07

    # Generating 100,000 bits for S and R
    for i in range(N):
        sBit = (nsd.nSidedDie(probs01)) - 1
        S[i] = sBit
        if S[i] == 0:
            rBit = nsd.nSidedDie([1 - e0,e0])-1
            R[i]=rBit
        elif S[i] == 1:
            rBit = nsd.nSidedDie([e1, 1 - e1])-1
            R[i] = rBit
        #print(R[i])
    return S,R
```

nSidedDie.py

```
mport numpy as np
def nSidedDie(p):
    #Getting n (number of sides)
    n=len(p)

    #Setting up the cumulative sum
    cumulProbs = np.cumsum(p)   # array of sums
    cumulProbs0 = np.append(0, cumulProbs)

    #random number between 0 and 1
    randNum = np.random.rand()
    for j in range(0, len(p)):
        # checks for the range that the dieRoll is in and get the probability for that 
range
        if randNum > cumulProbs0[j] and randNum <= cumulProbs0[j + 1]:
            side = j+1
            return side
    return 0
```

## Problem 4

### Introduction

This problem involves the same scenario as mentioned in the Problem 1 Introduction. Now, in order to try and ensure a better transmission of the sent message, we will send the bit over three times. So, if $S$ =0, then (0,0,0) is sent, and if $S$ =1, then (1,1,1) is sent. We will determine the received bit by using majority rule, meaning that if the received message has more zeros than ones, it will be seen as zero, and if it has more ones than zeros, then it will be seen as one. In this experiment, we want to find the probability that the message was sent incorrectly. From this, we will see if this improves the result from the first problem by sending over more bits.

### Methodology

For this problem, I will be using Python in the PyCharm IDE. The tools used will include methods from the `numpy` library and a method called `nSidedDie`. The methodology will be the same for generating $S$ as in the Problem 1 Methodology, but our method for generating $R$ will be different. Since $R$ will consist of three bits, it could have eight options which are (000), (001), (010), (100), (011), (101), (110), (111) . To create the tuple, we will generate 3 bits by using the `nSidedDie.` We then get add up the three bits in the tuple, and if it is greater than or equal to 2, then $R$ is 1, otherwise $R$ is 0. We will then compare each bit and see if they are the same. We will gather our results by keeping a counter for how many times the transmission failed and divide it by $N$ to find the probability of failure.

### Results and Conclusion

We will use the same test case values for the probabilities as in the Problem 1 Results and Conclusion section.

After running the program to simulate this, we get the following numerical result:

| Probability of error with enhanced transmission | |
|---|---|
| Answer | $p$ = 0.01052 |

In this result, we see that the probability of the message being received incorrectly has gone down compared to the result in Problem 1 where we only sent over one bit. To conclude, we see that a way to increase the rate of successful transmission is by sending over multiple copies of that message.

### Appendix

Here are the programs used for this problem:

problem4.py

```
import numpy as np
import nSidedDie as nsd


#PROBLEM 4:  #Probability (R=1 | S=1)
def problem4():
    # Setting N, p0, p1
    N = 100000
    p0 = 0.35
    p1 = 1 - p0

    # Creating S - Sent bits
    S = np.zeros((N, 1))

    # Creating list of probablities of bit being 0 or 1 and cumulative sums list
    probs01 = [p0, p1]

    # Generating 100,000 bits for S
    for i in range(N):
        sBit = (nsd.nSidedDie(probs01)) - 1
        S[i] = sBit

    #Creating R
    R = np.zeros((N, 1))

    # Setting e0, e1
    e0 = 0.04
    e1 = 0.07


    #Generating S and 3 bits for R
    numIncorrect=0
    for j in range(N):

        if S[j] == 0:
            rBit1 = nsd.nSidedDie([1 - e0,e0])-1
            rBit2 = nsd.nSidedDie([1 - e0,e0])-1
            rBit3 = nsd.nSidedDie([1 - e0,e0])-1
            threeBits=[rBit1,rBit2,rBit3]
            if sum(threeBits)>=2:
                R[j]=1
            else:
                R[j]=0
        elif S[j] == 1:
            rBit1 = nsd.nSidedDie([e1, 1 - e1])-1
            rBit2 = nsd.nSidedDie([e1, 1 - e1])-1
            rBit3 = nsd.nSidedDie([e1, 1 - e1])-1
            threeBits = [rBit1, rBit2, rBit3]
            if sum(threeBits) >= 2:
                R[j] = 1
            else:
                R[j] = 0

        #Comparing S and R values
        if S[j]!=R[j]:
            numIncorrect+=1

     #Displaying probability
    print("Probability of a message being incorrectly received " ,numIncorrect/N)

problem4()
```

nSidedDie.py

```
mport numpy as np
def nSidedDie(p):
    #Getting n (number of sides)
    n=len(p)

    #Setting up the cumulative sum
    cumulProbs = np.cumsum(p)   # array of sums
    cumulProbs0 = np.append(0, cumulProbs)

    #random number between 0 and 1
    randNum = np.random.rand()
    for j in range(0, len(p)):
        # checks for the range that the dieRoll is in and get the probability for that
range
        if randNum > cumulProbs0[j] and randNum <= cumulProbs0[j + 1]:
            side = j+1
            return side
    return 0
```