Vinh Vu

Dr. Chassiakos

EE 381-04

February 28th, 2019

Project 2: Communication through a Noisy Channel

**Problem 1: Probability of erroneous transmission**

**Introduction:** This problem transmits a one-bit message S and look at the received signal, R. Due to noise in the communication channel, the receive signal R has a small percentage of getting the wrong value from the transmit signal. If R = S, the experiment is considered a success, otherwise it is a failure. The experiment is then repeated N = 100,000 and count the number of failures. The failure probability that the transmitted bit will be received incorrectly is outputted.

**Methodology**: The function nSideDie(p) from Project 1 is used in this experiment to generate the S and R signal with inputted probability arrays. Symbol ‘0’ appears with probability p0 and ‘1’ appears with probability 1-p0 for transmit signal. The receiving signal R is ‘1’ if S = 0 with probability ε0 and ‘0’ with probability 1-ε0. Signal R is ‘1’ if S = 1 with probability 1-ε1 and ‘1’ with probability ε1. Probabilities p0,ε0, and ε1 are given as 0.6, 0.05, and 0.03. To generate S signal, nSidedDie(p) is called and the probability array passed into it is [p0, 1-p0] and S = S – 1 to keep it the range between 0 and 1. If S is 1, then R signal is generated with probability array [ε1, 1-ε1]. If S is 0, then R signal is generated with probability array [1-ε0, ε0]. S and R signals are compared to keep track of fail times and rate. The rate and count are outputted.

**Results and Conclusion:**

|  |  |
| --- | --- |
| Probability of transmission error |  |
| **Ans** | **p = 0.04153** |

**Appendix:**

# Vinh Vu

# EE 381 Project 2 Problem 1

import numpy as np

p\_0 = 0.6

e\_0 = 0.05

e\_1 = 0.03

fail = 0

def nSidedDie(p):

if sum(p) != 1:

print('Probability values are incorrect!')

sides = len(p)

cs = np.cumsum(p)

cp = np.append(0,cs)

r = np.random.rand()

for k in range(0,sides):

if r > cp[k] and r<= cp[k+1]:

sides = k+1

return sides

for i in range(100000):

S = nSidedDie([p\_0,1-p\_0])

S = S - 1

if (S == 1):

R = nSidedDie([e\_1,1-e\_1])

R = R - 1

elif (S == 0):

R = nSidedDie([1-e\_0,e\_0])

R = R - 1

if (S != R):

fail += 1

fail\_rate = fail/100000

print("The fail rate is:", fail\_rate)

**Problem 2: Conditional Probability: P(R=1 | S=1)**

**Introduction:** This problem focuses on when S and R signals match each other at ‘1’. The goal is to calculate the conditional probability P(R=1 | S=1). For all the events which the transmitted signal S is ‘1’, we look at the received signal R. If R matches with S at ‘1’, then it is counted as a success. The experiment is repeated N = 100,000 and count the number of successes. The probability is then calculated and outputted.

**Methodology:** Much similar to problem 1, the same methods are used to generate signals S and R with pre-determined probability for all cases. Function nSidedDie(p) is utilized to generate transmit one-bit value as well as receive one-bit value. Signals S and R are generated using the same method as Problem 1. In addition, to check the two conditions, a nested-if statement is used; first condition checks if S is ‘1’ then the next if-statement checks if R is ‘1’. If they match, then a variable is used to count. Success probability is calculated by dividing the count over the number of times S hits ‘1’. The result of calculated probability is outputted.

**Results and Conclusion:**

|  |  |
| --- | --- |
| Conditional probability P(R=1 | S=1) |  |
| **Ans** | **P = 0.969357** |

**Appendix:**

# Vinh Vu

# EE 381 Project 2 Problem 2

import numpy as np

p\_0 = 0.6

e\_0 = 0.05

e\_1 = 0.03

success = 0

count = 0

def nSidedDie(p):

if sum(p) != 1:

print('Probability values are incorrect!')

sides = len(p)

cs = np.cumsum(p)

cp = np.append(0,cs)

r = np.random.rand()

for k in range(0,sides):

if r > cp[k] and r<= cp[k+1]:

sides = k+1

return sides

for i in range(100000):

S = nSidedDie([p\_0,1-p\_0])

S = S - 1

if (S == 1):

R = nSidedDie([e\_1,1-e\_1])

R = R - 1

elif (S == 0):

R = nSidedDie([1-e\_0,e\_0])

R = R - 1

if (S == 1):

count +=1

if (R==1):

success += 1

success\_rate = success/count

print("The success rate is:", success\_rate)

**Problem 3: Conditional Probabilty P(S=1 | R=1)**

**Introduction:** This problem generates the transmit signal S as problem 1. The goal is to calculate the probability of P(S=1 | R=1). Opposite to problem 2, this experiment checks the received signal R first, to see if it is ‘1’. Once R is ‘1’, we check the transmitted signal S; if it is ‘1’, it is counted as a success. The experiment is repeated N = 100,000 times and count the number of successes. The probability is then calculated and outputted.

**Methodology:** The same method of generating signals S and R is used for this problem. Nothing changes beside the checking statement whether the conditional is true. It is a nested-if statement. Unlike number 2, problem 3 checks R value first. If R signal is ‘1’, then we to check transmit S signal. If S signal matches R, it is counted as one success. The experiment is repeated 100,000 times and success count is being tracked. At the end, the success rate is outputted.

**Results and Conclusion:**

|  |  |
| --- | --- |
| Conditional probability P(S=1 | R=1) |  |
| **Ans** | **P = 0.926754** |

**Appendix:**

# Vinh Vu

# EE 381 Project 2 Problem 3

import numpy as np

p\_0 = 0.6

e\_0 = 0.05

e\_1 = 0.03

success = 0

count = 0

def nSidedDie(p):

if sum(p) != 1:

print('Probability values are incorrect!')

sides = len(p)

cs = np.cumsum(p)

cp = np.append(0,cs)

r = np.random.rand()

for k in range(0,sides):

if r > cp[k] and r<= cp[k+1]:

sides = k+1

return sides

for i in range(100000):

S = nSidedDie([p\_0,1-p\_0])

S = S - 1

if (S == 1):

R = nSidedDie([e\_1,1-e\_1])

R = R - 1

elif (S == 0):

R = nSidedDie([1-e\_0,e\_0])

R = R - 1

if (R==1):

count += 1

if(S==1):

success +=1

success\_rate = success/count

print("The success rate is:", success\_rate)

**Problem 4: Enhanced transmission method**

**Introduction:** This experiment tries to improve reliability of the transmit and receive engine. Transmit signal S is create as before. The same bit S is transmitted three times (S S S) to create a three-bit wide signal. Received bits R are generated three times (R1 R2 R3). There are eight total combinations of (R1 R2 R3) = { (000), (001), (010), (011), (100), (101), (110), (111) }. When analyzing the received triples, we are to decode using voting and the majority rule. After decoding and get a one-bit value, we are to compare that bit to transmitted S bit. If they match, it counts as a success. The experiment is repeated N = 100,000 times and count the number of failures.

**Methodology:** To generate the initial one-bit S signal, method nSidedDie(p) is used and is called in the same manner as previous problems. Depending on S signal, whether it’s ‘1’ or ‘0’, (R1 R2 R3) are generated using pre-determined probabilities. There are two variables used decode R, one and zero. If there are more ‘1’ than ‘0’, decode value is 1 and vice versa. That decoded value is then compared to original S value. It is counted as a fail when they do not match. After repeating the experiment 100,000, the failing probability is calculated and outputted.

**Comment:** Compared to the failing rate in problem 1, this method provides a much accurate receiving engine. Failing rate in problem 1 is 0.0415 or around for every execution of the experiment. By generating a three-bit wide R signal, it lowers the failing rate and it is around 0.0057. With 3-bits transmitted signal, received signal R has less chance of being disturbed by noise in communication channel.

**Results and Conclusion:**

|  |  |
| --- | --- |
| Probability of error with enhanced transmission |  |
| **Ans** | **P = 0.00573** |

**Appendix:**

# Vinh Vu

# EE 381 Project 2 Problem 4

import numpy as np

p\_0 = 0.6

e\_0 = 0.05

e\_1 = 0.03

fail = 0

def nSidedDie(p):

if sum(p) != 1:

print('Probability values are incorrect!')

sides = len(p)

cs = np.cumsum(p)

cp = np.append(0,cs)

r = np.random.rand()

for k in range(0,sides):

if r > cp[k] and r<= cp[k+1]:

sides = k+1

return sides

for k in range(100000):

D = int()

one = int()

zero = int()

S = nSidedDie([p\_0,1-p\_0])

S = S - 1

for i in range(3):

if (S == 1):

R = nSidedDie([e\_1,1-e\_1])

R = R - 1

elif (S == 0):

R = nSidedDie([1-e\_0,e\_0])

R = R - 1

if (R == 0):

zero += 1

elif (R == 1):

one += 1

if (one > zero):

D = 1

elif (zero > one):

D = 0

if (D!=S):

fail += 1

fail\_rate = fail/100000

print("Fail rate:", fail\_rate)