

# Lab 1: Modeling Packet Losses

Zhipeng Ye

Department of Electrical and Electronic Engineering  
Xi'an Jiaotong-Liverpool University

April 6, 2020

## 1 Hidden Markov Model

Gilbert-Elliott Model is packet loss model which is based on two-state Hidden Markov Model. In other words, Gilbert-Elliott Model is a simply version of Hidden Markov Model. Therefore, It's necessary to learn more about Hidden Markov Model to understand Gilbert-Elliott Model completely.

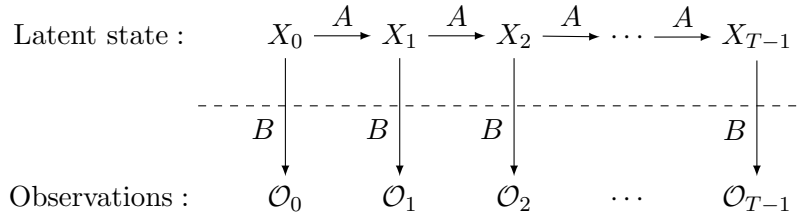


Figure 1: Hidden Markov Model

Hidden Markov Model introduce two random variable - Latent state and Observations. Latent state is a variable that we can't see. Observations is a variable that we can observe at the experiment. Furthermore, HMM also has three parameters - initial probability vector  $\pi$ , transition probability matrix  $A$ , emission probability matrix  $B$ . So if we can compute three parameters  $\{A, B, \pi\}$ , we will reconstruct HMM. Researchers have developed numerous approaches to estimate parameters of Hidden Markov Model (HMM) such as Forward/Backward Algorithm.

Markov Model simplifies the Hidden Markov Model (HMM). It only uses two parameters  $\{A, \pi\}$  and removes the Latent state. Maximum likelihood estimation (MLE) is a good way to estimate these parameters. The relationship between Hidden Markov Model and Markov Model is similar to the relation between Gilbert-Elliott Model (GM) and Simple Gilbert-Elliott Model (SGM). Section two of this article will demonstrate the reason.

## 2 Gilbert-Elliott Model

Gilbert-Elliott Model resembles Hidden Markov Model. To be specific, Gilbert-Elliott Model has two state -  $\{Good, Bad\}$ , transition matrix  $\mathbf{P} = \begin{bmatrix} (1-p) & q \\ p & (1-q) \end{bmatrix}$  and emission probability vector  $\begin{bmatrix} 1-k & 1-h \end{bmatrix}^T$ .

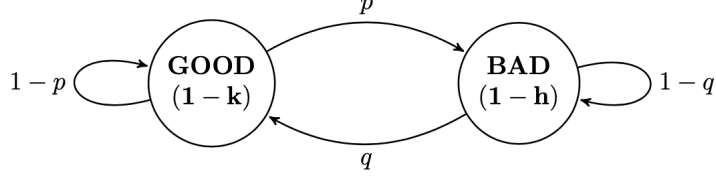


Figure 2: Gilbert-Elliott Model

Furthermore, we can obtain the other parameters of Gilbert-Elliott Model such as the steady state probability vector  $\pi$ , packet loss probability  $p_L$  and the probability distribution of loss run length  $p_k$ . The steady state probability vector  $\pi$  satisfies the following conditions:

$$\pi = P\pi, \mathbf{1}^T \pi \quad (1)$$

where

$$\pi = \begin{bmatrix} \pi_G \\ \pi_B \end{bmatrix} \quad (2)$$

The steady state probabilities exist for  $0 < p, q < 1$  and are given by

$$\pi_G = \frac{q}{p+q}, \pi_B = \frac{p}{p+q} \quad (3)$$

From equation (3), we can get packet loss probability  $p_L$  as follows:

$$p_L = (1-k)\pi_G + (1-h)\pi_B \quad (4)$$

The special cases of  $k = 1$  and  $k = 1, h = 0$  are called a Gilbert Model (GM) and a simple Gilbert Model (SGM), respectively. Note that in case of the SGM, the probability distribution of loss run length has a geometric distribution, i.e.,

$$p_k \triangleq \text{Prob} \{ \text{loss run length} = k \} = q(1-q)^{k-1} \text{ for } k = 1, 2, \dots, \infty \quad (5)$$

### 3 Estimation of parameters

#### 3.1 Simple Gilbert Model (SGM)

Because 0, 1 represent  $\{Good, Bad\}$  respectively. The probabilities from *Good* to *Bad* is equal to  $P(1|0)$  and the probabilities from *Bad* to *Good* is equal to  $P(0|1)$ . According to *Bayes theorem*,

$$p = P(1|0) = \frac{P(01)}{P(0)} = \frac{n_{01}}{n_0} \quad (6)$$

$$q = P(0|1) = \frac{P(10)}{P(1)} = \frac{n_{10}}{n_1}$$

$n_{10}$  means the times 0 follows 1.  $n_{01}$  means the times 1 follows 0.  $n_0$  represents the times of 0.  $n_1$  represents the times of 1.

#### 3.2 Gilbert Model (GM)

Gilbert applied a method to decide the parameters of GM [1]. Firstly, the parameters of  $p, q, h$  can be estimated from another parameters of  $a, b, c$ . The expression equation of  $a, b, c$  is as follow:

$$a = P(1), b = P(1|1), c = \frac{P(111)}{P(101) + P(111)} \quad (7)$$

Finally, we can get parameters of Gilbert Model by the following relation between these variables.

$$1 - q = \frac{ac - b^2}{2ac - b(a + c)}, h = 1 - \frac{b}{1 - q}, p = \frac{aq}{1 - h - a} \quad (8)$$

## 4 Experiment Task

In this experiment, we simulate the Simple Gilbert Model (SGM) and Gilbert Model (GM) separately.

### 4.1 Simple Gilbert Model (SGM)

#### 4.1.1 Construct model

The script and detailed comments for Constructing Simple Gilbert Model is attached at the appendix A. The main implement is here.

```
# main loop
for i in range(len):
    # if the random sequence is large than the transition probability
    # we must change the state for example 0 -> 1 or 1 -> 0
    if statechange[i] > tr[state, state]:
        # transition into the other state
        state ^= 1
    # add a binary value to output
    seq[i] = state
```

As we can see from the code segment, if the sequence[i] is large than the transition probability, the start will flip from 0 to 1 or from 1 to 0.

#### 4.1.2 Estimate model parameters

To determine the parameters of  $p, q$ , I write the script which is in Appendix B to count the number of times where binary sequence occurs. Consequently, I observe the value of  $p, q$  is

$$\begin{aligned} p &= 0.14727215389467044 \\ q &= 0.2550574084199016 \end{aligned} \quad (9)$$

The steady vector  $\pi$ :

$$\pi_G = \frac{q}{p + q} = 0.6339514475460747, \pi_B = \frac{p}{p + q} = 0.3660485524539253 \quad (10)$$

Packet loss probability  $p_L$  as follows:

$$p_L = \pi_B = 0.3660485524539253 \quad (11)$$

The probability distribution of loss run length has a geometric distribution:

$$p_k = 0.2550574084199016(0.7449425915800985)^{k-1} \text{ for } k = 1, 2, \dots, \infty \quad (12)$$

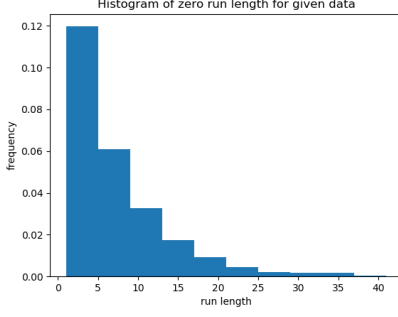


Figure 3: SGM zero length for given data

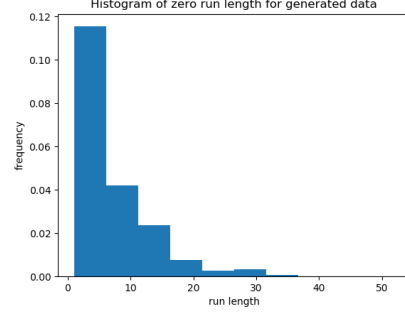


Figure 4: SGM zero length for generated data

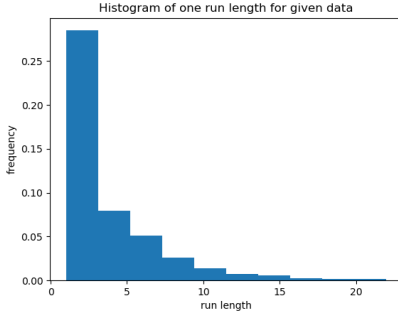


Figure 5: SGM one length for given data

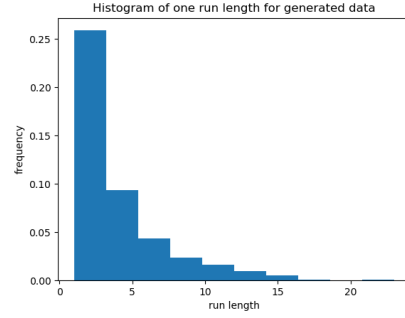


Figure 6: SGM one length for generated data

#### 4.1.3 Comparison and discussion

Based on the parameters we observed, I run the SGM script to generate the sequence which has the same length as the supplied sequence. Here is the shell command:

```
python sgm_generate.py -L 10000 -T 0.8527278461053296,0.2550574084199016,
0.14727215389467044,0.7449425915800985
```

Next, I compare the histogram and power spectrum density between given data and generated data. The plot script can be seen in the appendix C. As we can see from the pictures, the given data is similar to the generated data, because random variable *submit the same stochastic process*. Interestingly, the whole process of parameters estimation is **similar to the learning process in Machine learning** and Hidden Markov Model is a significant model of Machine learning. In addition, we have known the distribution of loss run length from equation 16. Therefore we can generate theoretical curve. **From figure 7**, we can learn that the one length histogram submit to the theoretical cure. The script for generate theoretical curve is attached at appendix D.

Furthermore, I compare the power spectrum density PSD for given sequence and generated sequence. we can see from **Figure 8** and **Figure 9**, they have similar distribution. In general, the estimated parameters is correct.

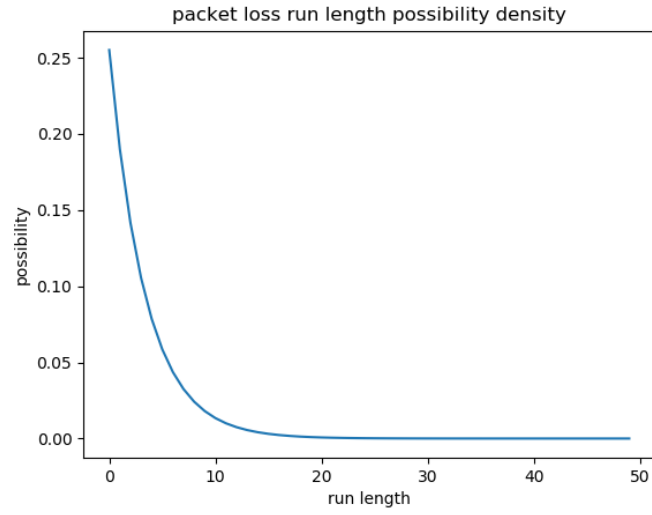


Figure 7: SGM packet loss run length theoretical possibility density

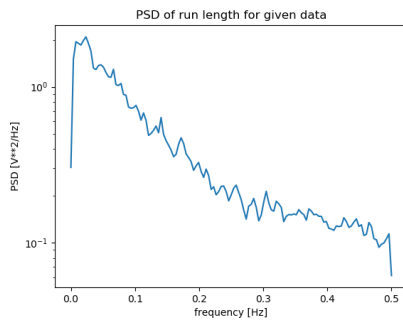


Figure 8: SGM PSD of given data

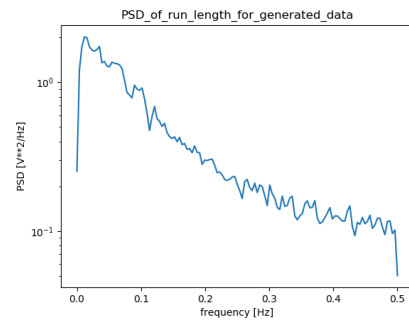


Figure 9: SGM PSD of generated data

## 4.2 Gilbert Model (GM)

Gilbert Model has one more parameters  $H$  than Simple Gilbert Model (SGM). Therefore following steps must be around parameter  $H$ .

### 4.2.1 Construct model

The script and detailed comments for Constructing Gilbert Model is attached at the appendix E. The main implement is here:

```
1  for i in range(len):
2      # if the random sequence is large than the transition probability
3      # we must change the state for example 0 -> 1 or 1 -> 0
4      if statechange[i] > tr[state, state]:
5          # transition into the other state
6          state ^= 1
7      # if latent state is 1 (Bad state), we must consider the emission probability to
8      # generate sequence
9      if state == 1:
10         randomvalue = random.random()
11         if randomvalue <= emission_probability:
12             loss_state = 1
13
14     # add a binary value to output
15     seq[i] = loss_state
16
17     loss_state = 0
```

To create Gilbert Model, I introduce loss state to consider emission probability. if the random value is large than  $1 - H$ , the Bad state will cause packet loss.

### 4.2.2 Estimate model parameters

To determine the parameters of  $a, b, c$  and  $p, q, h$ , I write the script to estimated them at appendix F. According to equation 7 and 8, we can get these parameters of Gilbert Model. Finally, I obtain the parameters of Gilbert Model.

$$\begin{aligned} a &= 0.3658, b = 0.7446692181519955, c = 0.9301369863013699 \\ p &= 0.1449793136870664, q = 0.2469390489546801, h = 0.01114349759030242 \end{aligned} \quad (13)$$

The steady vector  $\pi$ :

$$\pi_G = \frac{q}{p+q} = 0.6300777725498095, \pi_B = \frac{p}{p+q} = 0.36992222745019054 \quad (14)$$

Packet loss probability  $p_L$  as follows:

$$p_L = (1 - h)\pi_B = 0.3658 \quad (15)$$

The probability distribution of loss run length has a geometric distribution:

$$\begin{aligned} p_k &= (1 - q)^{k-1}(1 - h)^{k-1}[1 - (1 - q)(1 - h)] \\ &= 0.7530609510453199^{k-1} * 0.9888565024096976^{k-1} * 0.2553307818480045 \\ &= 0.2553307818480045 * 0.7446692181519955^{k-1} \text{ for } k = 1, 2, \dots, \infty \end{aligned} \quad (16)$$

### 4.2.3 Comparison and discussion

Based on the parameters we observed, I run the SGM script to generate the sequence which has the same length as the supplied sequence. Here is the shell command:

```
python gm_generate.py -L 10000 -T 0.8550206863129336,0.2469390489546801,  
0.1449793136870664,0.7530609510453199 -H 0.01114349759030242
```

Next, I compare the histogram and power spectrum density between given data and generated data. The plot script can be seen in the appendix G. As we can see from the Figure 10 - 13, the given data is **similar** to the generated data. Furthermore, GM has a **better performance to simulate the given data than SGM**, when we compare GM histogram and SGM histogram. In my opinion, Maybe GM has one more parameters  $H$  which can improve the performance of model. **This concept can be easily seen in Machine learning and Deep learning**. To some extent, *the more parameters we consider in the model, the better the model fit the training set*. Furthermore, based

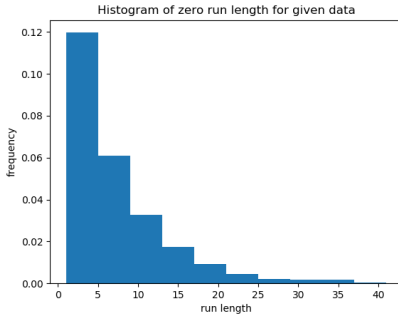


Figure 10: GM zero length for given data

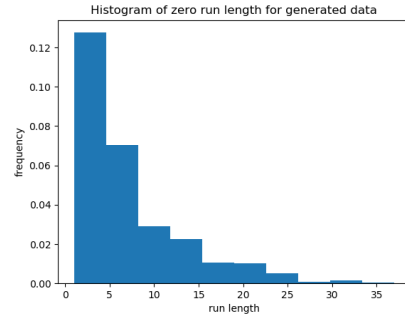


Figure 11: GM zero length for generated data

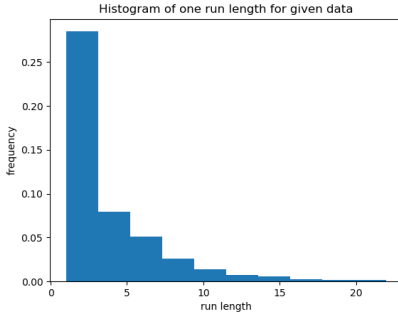


Figure 12: GM one length for given data

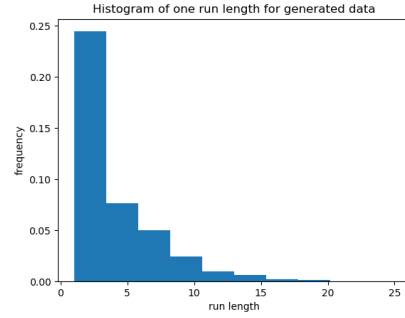


Figure 13: GM one length for generated data

on the theoretical packet loss distribution, I write script to plot the theoretical curve. The script for generate theoreticalcurve is attached at appendix H. **From figure 14**, we can learn that the one length histogram submit to the theoretical cure.

Furthermore, I compare the power spectrum density PSD for given sequence and generated sequence. we can see from **Figure 15** and **Figure 16**, they have similar distribution. In general, the estimated parameters is correct.

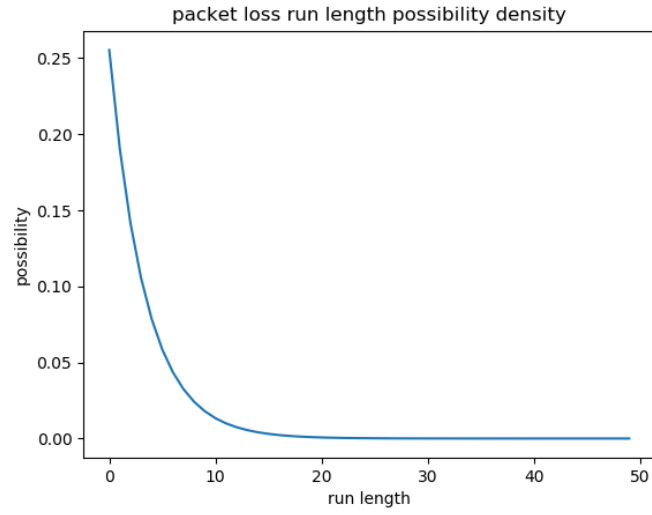


Figure 14: SGM packet loss run length theoretical possibility density

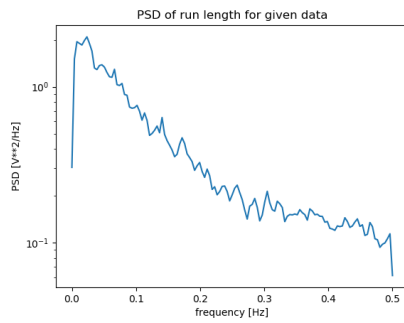


Figure 15: GM PSD of given data

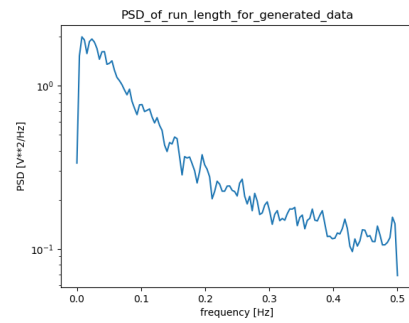


Figure 16: GM PSD of generated data



## A Construct Simple Gilbert Model (SGM)

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 ##
4 # @file      sgm_generate.py
5 # @author    Kyeong Soo (Joseph) Kim <kyeongsoo.kim@gmail.com>
6 # @date      2020-03-25
7 #
8 # @brief     A function for generating loss pattern based on simple Guilbert model.
9 #
10
11 import numpy as np
12 import sys
13
14
15 def sgm_generate(len, tr):
16     """
17     Generates a binary sequence of 0 (GOOD) and 1 (BAD) of length
18     len from an SGM specified by a 2x2 transition probability matrix
19     tr; tr[i, j] is the probability of transition from state i to
20     state j.
21
22     This function always starts the model in GOOD (0) state.
23
24     Examples:
25
26     import numpy as np
27
28     tr = np.array([[0.95, 0.10],
29                   [0.05, 0.90]])
30     seq = sgm_generate(100, tr)
31     """
32
33     seq = np.zeros(len)
34
35     # tr must be 2x2 matrix
36     tr = np.asarray(tr) # make sure seq is numpy 2D array
37     if tr.shape != (2, 2):
38         sys.exit("size of transition matrix is not 2x2")
39
40     # create a random sequence for state changes
41     statechange = np.random.rand(len)
42
43     # Assume that we start in GOOD state (0).
44     state = 0
45
46     # main loop
47     for i in range(len):
48         # if the random sequence is large than the transition probability
49         # we must change the state for example 0 -> 1 or 1 -> 0
50         if statechange[i] > tr[state, state]:
51             # transition into the other state
52             state ^= 1
53         # add a binary value to output
54         seq[i] = state
55
```

```

56     return seq
57
58
59 if __name__ == "__main__":
60     import argparse
61
62     parser = argparse.ArgumentParser()
63     parser.add_argument(
64         "-L",
65         "--length",
66         help="the length of the loss pattern to be generated; default is 10",
67         default=10,
68         type=int)
69     parser.add_argument(
70         "-T",
71         "--transition",
72         help="transition matrix in row-major order; default is
73         \"0.95,0.10,0.05,0.90\",
74         default="0.95,0.10,0.05,0.90",
75         type=str)
76     args = parser.parse_args()
77     len = args.length
78     tr = np.reshape(np.fromstring(args.transition, sep=','), (2, 2))
79     print(sgm_generate(len, tr))

```

## B Estimate Simple Gilbert Model

```

1     #!/usr/bin/env python
2     # encoding: utf-8
3
4     # @author: Zhipeng Ye
5     # @contact: Zhipeng.ye19@xjtlu.edu.cn
6     # @file: estimateparams.py
7     # @time: 2020-04-03 19:41
8     # @desc:
9     import scipy.io as sio
10    import numpy as np
11
12    if __name__ == "__main__":
13        matfile = sio.loadmat('loss_seq.mat')
14        binary_seq = matfile['seq'][0].tolist()
15        length_seq = len(binary_seq)
16
17        n1 = np.count_nonzero(binary_seq)
18        n0 = length_seq - n1
19        n01 = 0
20        n10 = 0
21
22        for index in range(len(binary_seq) - 2):
23            if binary_seq[index:index + 2] == [0, 1]:
24                n01 += 1
25
26            if binary_seq[index:index + 2] == [1, 0]:
27                n10 += 1
28
29        p = n01 / n0
30
31        q = n10 / n1
32
33        print("p:{},q:{}".format(p, q))

```

## C Plot Histogram and PSD of SGM

```
1 #!/usr/bin/env python
2 # encoding: utf-8
3
4 # @author: Zhipeng Ye
5 # @contact: Zhipeng.ye19@xjtlu.edu.cn
6 # @file: main.py
7 # @time: 2020-04-03 14:16
8 # @desc:
9 import scipy.io as sio
10 import binary_runlengths as brl
11 import numpy as np
12 import matplotlib.pyplot as plt
13 import json
14 from scipy import signal
15
16 if __name__ == '__main__':
17     mat_contents = sio.loadmat('loss_seq.mat')
18     binary_seq = mat_contents['seq']
19
20     given_zerorl, given_onerl = brl.binary_runlengths(binary_seq)
21
22     # plot histogram
23     plt.hist(given_zerorl, normed=True)
24     plt.title("Histogram of zero run length for given data")
25     plt.xlabel("run length")
26     plt.ylabel('frequency')
27     plt.savefig('Histogram of zero run length for given data')
28     plt.show()
29
30     plt.hist(given_onerl, normed=True)
31     plt.title("Histogram of one run length for given data")
32     plt.xlabel("run length")
33     plt.ylabel('frequency')
34     plt.savefig('Histogram of one run length for given data')
35     plt.show()
36
37     with open('sgmseq.out') as file:
38         generated_data = json.load(file)
39
40     generated_zerorl, generated_onerl = brl.binary_runlengths(generated_data)
41
42     # plot histogram
43     plt.hist(generated_zerorl, normed=True)
44     plt.title("Histogram of zero run length for generated data")
45     plt.xlabel("run length")
46     plt.ylabel('frequency')
47     plt.savefig('Histogram of zero run length for generated data')
48     plt.show()
49     plt.hist(generated_onerl, normed=True)
50     plt.title("Histogram of one run length for generated data")
51     plt.xlabel("run length")
52     plt.ylabel('frequency')
53     plt.savefig('Histogram of one run length for generated data')
54     plt.show()
55
56     frequency_given, psd_given = signal.welch(binary_seq[0])
57     plt.semilogy(frequency_given, psd_given)
58     plt.title("PSD of run length for given data")
59     plt.xlabel("frequency [Hz]")
```

```

60 plt.ylabel('PSD [V**2/Hz]')
61 plt.savefig("PSD of run length for given data")
62 plt.show()
63
64 frequency_generated, psd_generated = signal.welch(generated_data)
65 plt.semilogy(frequency_generated, psd_generated)
66 plt.title("PSD of run length for generated data")
67 plt.xlabel("frequency [Hz]")
68 plt.ylabel('PSD [V**2/Hz]')
69 plt.savefig("PSD of run length for generated data")
70 plt.show()

```

## D Plot Theoretical SGM Packet Loss Curve

```

1  #!/usr/bin/env python
2  # encoding: utf-8
3
4  # @author: Zhipeng Ye
5  # @contact: Zhipeng.ye19@xjtlu.edu.cn
6  # @file: generatetheoreticalcurve.py
7  # @time: 2020-04-05 16:15
8  # @desc:
9
10 import matplotlib.pyplot as plt
11
12 k = 50
13
14 x = [i for i in range(k)]
15 seq = [0.2550574084199016*0.7449425915800985**(i) for i in range(k)]
16
17 plt.plot(x, seq)
18 plt.title('packet loss run length possibility density')
19 plt.xlabel('run length')
20 plt.ylabel('possibility')
21 plt.savefig("packet_loss_run_length_possibility_density")
22 plt.show()

```

## E Construct Gilbert Model (GM)

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  ##
4  # @file      sgm_generate.py
5  # @author    Kyeong Soo (Joseph) Kim <kyeongsoo.kim@gmail.com>
6  # @date      2020-03-25
7  #
8  # @brief     A function for generating loss pattern based on simple Gilbert model.
9  #
10
11 import numpy as np
12 import sys
13 import json
14 import random
15
16 def gm_generate(len, tr, emission_probability):
17     """
18     Generates a binary sequence of 0 (GOOD) and 1 (BAD) of length
19     len from an SGM specified by a 2x2 transition probability matrix
20     tr; tr[i, j] is the probability of transition from state i to
21     state j.

```

```

22
23     This function always starts the model in GOOD (0) state.
24
25     Examples:
26
27     import numpy as np
28
29     tr = np.array([[0.95, 0.10],
30                    [0.05, 0.90]])
31     seq = sgm_generate(100, tr)
32     """
33
34     seq = np.zeros(len)
35
36     # tr must be 2x2 matrix
37     tr = np.asarray(tr) # make sure seq is numpy 2D array
38     if tr.shape != (2, 2):
39         sys.exit("size of transition matrix is not 2x2")
40
41     # create a random sequence for state changes
42     statechange = np.random.rand(len)
43
44     # Assume that we start in GOOD state (0).
45     state = 0
46     loss_state = 0
47
48     # main loop
49     for i in range(len):
50         # if the random sequence is large than the transition probability
51         # we must change the state for example 0 -> 1 or 1 -> 0
52         if statechange[i] > tr[state, state]:
53             # transition into the other state
54             state ^= 1
55         # if latent state is 1 (Bad state), we must consider the emission
56         # generate sequence
57         if state == 1:
58             randomvalue = random.random()
59             if randomvalue <= emission_probability:
60                 loss_state = 1
61
62         # add a binary value to output
63         seq[i] = loss_state
64
65         loss_state = 0
66
67     return seq
68
69
70 if __name__ == "__main__":
71     import argparse
72
73     parser = argparse.ArgumentParser()
74     parser.add_argument(
75         "-L",
76         "--length",
77         help="the length of the loss pattern to be generated; default is 10",
78         default=10,
79         type=int)
80     parser.add_argument(
81         "-H",

```

```

82     "--H",
83     help="emission probability; default is 0.0",
84     default=0.0,
85     type=float)
86 parser.add_argument(
87     "-T",
88     "--transition",
89     help="transition matrix in row-major order; default is
90     \"0.95,0.10,0.05,0.90\"",
91     default="0.95,0.10,0.05,0.90",
92     type=str)
93 args = parser.parse_args()
94 len = args.length
95 emission_probability = 1 - args.H
96 tr = np.reshape(np.fromstring(args.transition, sep=','), (2, 2))
97 seq = gm_generate(len, tr, emission_probability).tolist()
98
99 with open('gmseq.out', 'w') as file:
    json.dump(seq, file)

```

## F Estimate Gilbert Model

```

1  #!/usr/bin/env python
2  # encoding: utf-8
3
4  # @author: Zhipeng Ye
5  # @contact: Zhipeng.ye19@xjtlu.edu.cn
6  # @file: estimateGMparams.py
7  # @time: 2020-04-06 14:12
8  # @desc:
9  import scipy.io as sio
10 import numpy as np
11
12 if __name__ == "__main__":
13     matfile = sio.loadmat('loss_seq.mat')
14     binary_seq = matfile['seq'][0].tolist()
15     length_seq = len(binary_seq)
16
17     n1 = np.count_nonzero(binary_seq)
18
19     p1 = n1 / length_seq
20     a = p1
21
22     n11 = 0
23     for index in range(length_seq - 1):
24         if binary_seq[index:index + 2] == [1, 1]:
25             n11 += 1
26
27     p11 = n11 / length_seq
28     b = p11 / p1
29
30     n111 = 0
31     n101 = 0
32     for index in range(length_seq - 2):
33         if binary_seq[index:index + 3] == [1, 1, 1]:
34             n111 += 1
35         if binary_seq[index:index + 3] == [1, 0, 1]:
36             n101 += 1
37
38     p111 = n111 / length_seq
39     p101 = n101 / length_seq

```

```

40
41     c = p111 / (p111 + p101)
42
43     q = 1 - (a * c - b ** 2) / (2 * a * c - b * (a + c))
44
45     h = 1 - b / (1 - q)
46
47     p = (a * q) / (1 - h - a)
48
49     print("a:{},b:{},c:{}".format(a, b, c))
50     print("p:{},q:{},h:{}".format(p, q, h))

```

## G Plot Histogram and PSD of GM

```

1
2 # !/usr/bin/env python
3 # encoding: utf-8
4
5 # @author: Zhipeng Ye
6 # @contact: Zhipeng.ye19@xjtlu.edu.cn
7 # @file: GMHitPSD.py
8 # @time: 2020-04-03 14:16
9 # @desc:
10 import scipy.io as sio
11 import binary_runlengths as brl
12 import numpy as np
13 import matplotlib.pyplot as plt
14 import json
15 from scipy import signal
16
17 if __name__ == '__main__':
18     mat_contents = sio.loadmat('loss_seq.mat')
19     binary_seq = mat_contents['seq']
20
21     given_zerorl, given_onerl = brl.binary_runlengths(binary_seq)
22
23     # plot histogram
24     plt.hist(given_zerorl, normed=True)
25     plt.title("Histogram of zero run length for given data")
26     plt.xlabel("run length")
27     plt.ylabel('frequency')
28     plt.savefig('GM-Histogram_of_zero_run_length_for_given_data')
29     plt.show()
30
31     plt.hist(given_onerl, normed=True)
32     plt.title("Histogram of one run length for given data")
33     plt.xlabel("run length")
34     plt.ylabel('frequency')
35     plt.savefig('GM-Histogram_of_one_run_length_for_given_data')
36     plt.show()
37
38     with open('gmseq.out') as file:
39         generated_data = json.load(file)
40
41     generated_zerorl, generated_onerl = brl.binary_runlengths(generated_data)
42
43     # plot histogram
44     plt.hist(generated_zerorl, normed=True)
45     plt.title("Histogram of zero run length for generated data")
46     plt.xlabel("run length")
47     plt.ylabel('frequency')

```

```

48 plt.savefig('GM_Histogram_of_zero_run_length_for_generated_data')
49 plt.show()
50 plt.hist(generated_onerl, normed=True)
51 plt.title("Histogram of one run length for generated data")
52 plt.xlabel("run length")
53 plt.ylabel('frequency')
54 plt.savefig('GM_Histogram_of_one_run_length_for_generated_data')
55 plt.show()
56
57 frequency_given, psd_given = signal.welch(binary_seq[0])
58 plt.semilogy(frequency_given, psd_given)
59 plt.title("PSD of run length for given data")
60 plt.xlabel("frequency [Hz]")
61 plt.ylabel('PSD [V**2/Hz]')
62 plt.savefig("GM_PSD_of_run_length_for_given_data")
63 plt.show()
64
65 frequency_generated, psd_generated = signal.welch(generated_data)
66 plt.semilogy(frequency_generated, psd_generated)
67 plt.title("PSD of run length for generated data")
68 plt.xlabel("frequency [Hz]")
69 plt.ylabel('PSD [V**2/Hz]')
70 plt.savefig("GM_PSD_of_run_length_for_generated_data")
71 plt.show()

```

## H Plot Theoretical GM Packet Loss Curve

```

1  #!/usr/bin/env python
2  # encoding: utf-8
3
4  # @author: Zhipeng Ye
5  # @contact: Zhipeng.ye19@xjtlu.edu.cn
6  # @file: generateGMtheoreticalcurve.py
7  # @time: 2020-04-06 21:30
8  # @desc:
9
10 import matplotlib.pyplot as plt
11
12 k = 50
13
14 x = [i for i in range(k)]
15 seq = [0.2553307818480045*0.7446692181519955**(i) for i in range(k)]
16 plt.plot(x, seq)
17 plt.title('packet loss run length possibility density')
18 plt.xlabel('run length')
19 plt.ylabel('possibility')
20 plt.savefig("GM_packet_loss_run_length_possibility_density")
21 plt.show()

```



## References

- [1] E. N. Gilbert, “Capacity of a burst-noise channel,” *Bell System Technical Journal*, vol. 39, no. 5, pp. 1253–1265, Sep. 1960.