Lab 1: Modeling Packet Losses

Zhipeng Ye

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

April 17, 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.

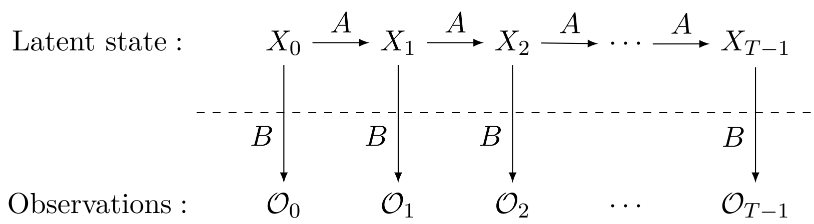

Figure 1: Hidden Markov Model

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. Furtheremore, HMM also has three parameters - initial probability vector π, 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 $\boldsymbol{P} = \begin{bmatrix} (1-p) & q \\ p & (1-q) \end{bmatrix}$ and emission probability vector $\begin{bmatrix} 1-k & 1-h \end{bmatrix}^T$.

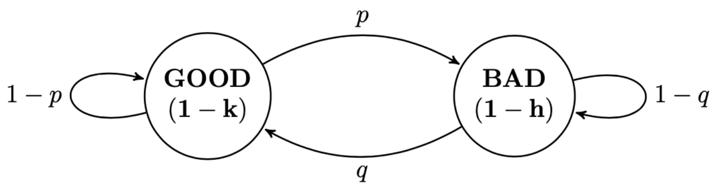

Figure 2: Gilbert-Elliott Model

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

$$\boldsymbol{\pi} = \boldsymbol{P}\boldsymbol{\pi}, \mathbf{1}^T\boldsymbol{\pi} \tag{1}$$

where

$$\boldsymbol{\pi} = \begin{bmatrix} \pi_G \\ \pi_B \end{bmatrix} \tag{2}$$

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

$$\pi_G = \frac{q}{p+q}, \pi_B = \frac{p}{p+q} \tag{3}$$

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

$$p_L = (1-k)\pi_G + (1-h)\pi_B \tag{4}$$

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.,

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

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*,

$$\begin{aligned} p = P(1|0) = \tfrac{P(01)}{P(0)} = \tfrac{n_{01}}{n_0} \\ q = P(0|1) = \tfrac{P(10)}{P(1)} = \tfrac{n_{10}}{n_1} \end{aligned} \tag{6}$$

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:

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

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

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

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

$$\begin{aligned} p &= 0.14727215389467044 \\ q &= 0.2550574084199016 \end{aligned} \tag{9}$$

The steady vector $\boldsymbol{\pi}$:

$$\pi_G = \frac{q}{p+q} = 0.6339514475460747, \pi_B = \frac{p}{p+q} = 0.3660485524539253 \tag{10}$$

Packet loss probability p_L as follows:

$$p_L = \pi_B = 0.3660485524539253 \tag{11}$$

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

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

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
```

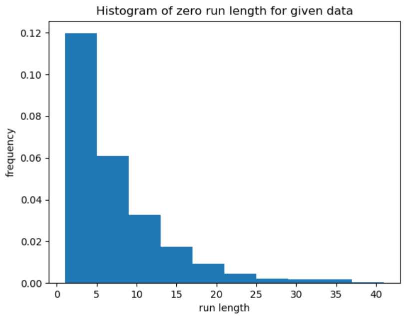

Figure 3: SGM zero length for given data

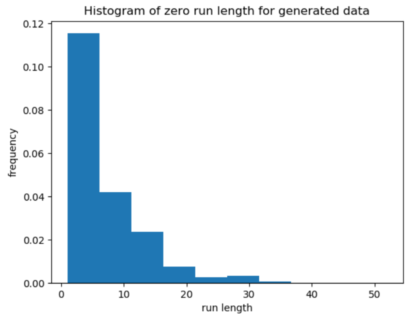

Figure 4: SGM zero length for generated data

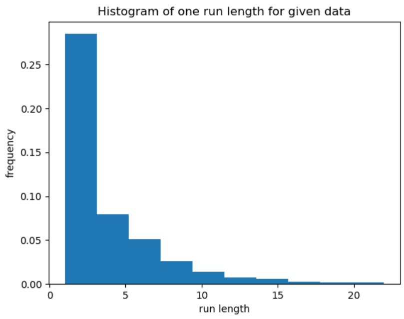

Figure 5: SGM one length for given data

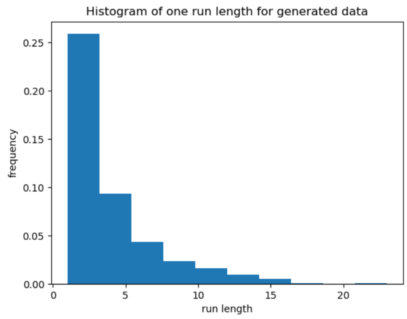

Figure 6: SGM one length for generated data

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 Histogram, **figure 3-6**, 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 12. 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 generating theoretical curve is attached at appendix D.

Furtheremore, 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.

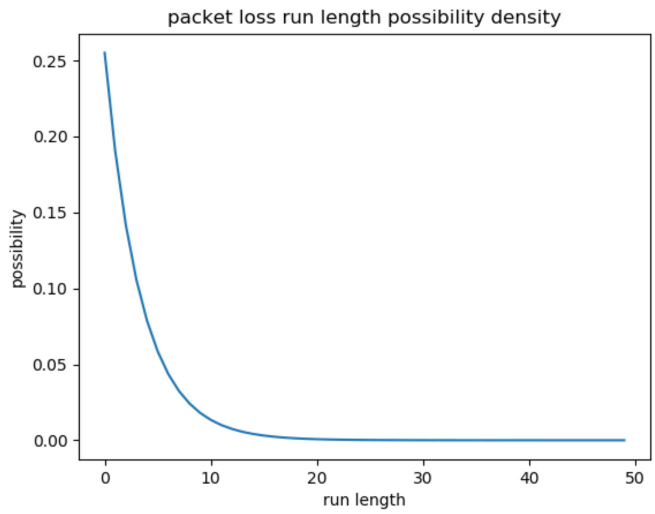

Figure 7: SGM packet loss run length theoretical possibility density

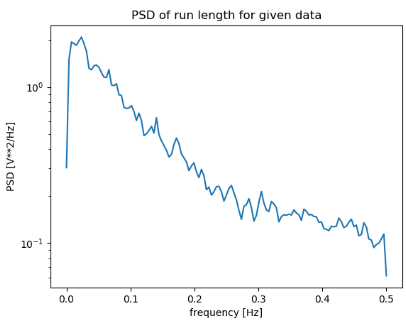

Figure 8: SGM PSD of given data

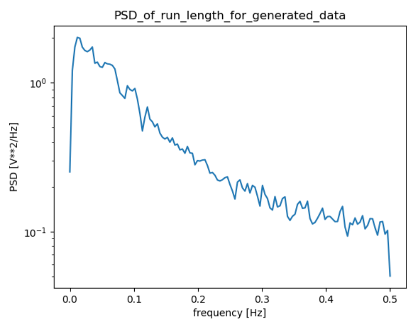

Figure 9: SGM PSD of generated data

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:

```
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
    # if latent state is 1 (Bad state), we must consider the emission probability to
    # generate sequence
    if state == 1:
        randomvalue = random.random()
        if randomvalue <= emission_probability:
            loss_state = 1

    # add a binary value to output
    seq[i] = loss_state

    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.

$$\begin{aligned}&a = 0.3658, b = 0.7446692181519955, c = 0.93013698630136999\\&p = 0.1449793136870664, q = 0.2469390489546801, h = 0.01114349759030242\end{aligned} \tag{13}$$

The steady vector $\boldsymbol{\pi}$:

$$\pi_G = \frac{q}{p+q} = 0.6300777725498095, \pi_B = \frac{p}{p+q} = 0.36992222745019054 \tag{14}$$

Packet loss probability p_L as follows:

$$p_L = (1-h)\pi_B = 0.3658 \tag{15}$$

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

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

4.2.3 Comparison and discussion

Based on the parameters we observed, I run the GM 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. Furtheremore, 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.* Furtheremore, based

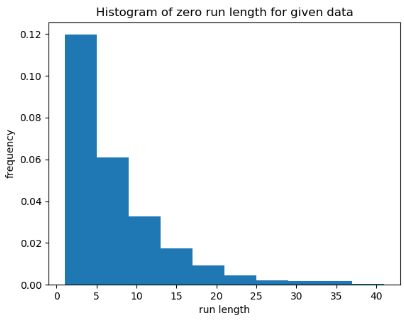

Figure 10: GM zero length for given data

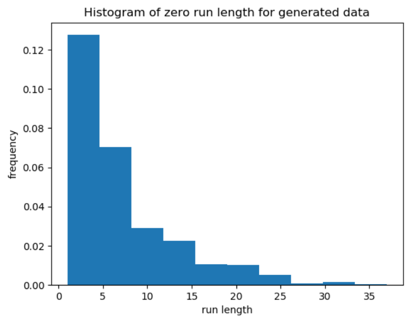

Figure 11: GM zero length for generated data

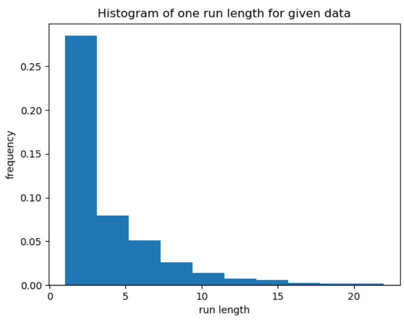

Figure 12: GM one length for given data

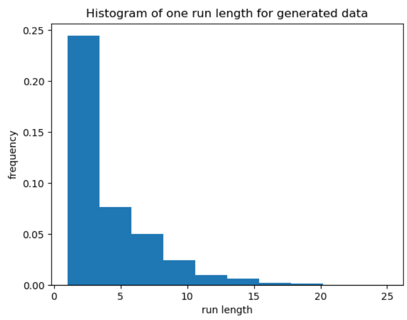

Figure 13: GM one length for generated data

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

Furtheremore, 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.

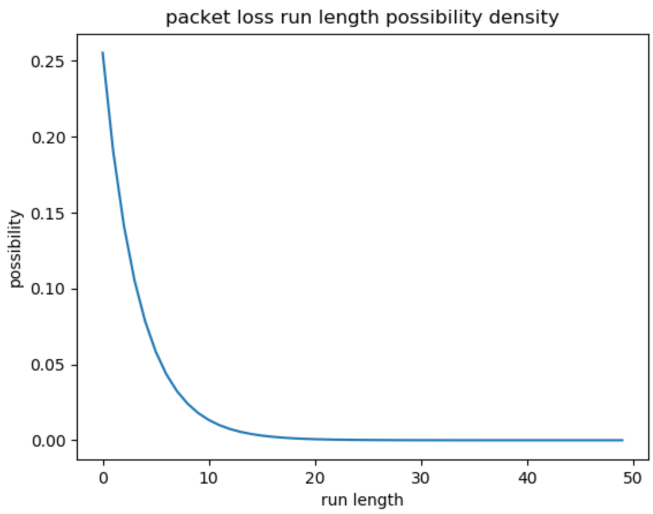

Figure 14: GM packet loss run length theoretical possibility density

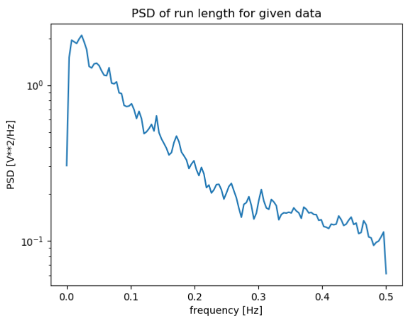

Figure 15: GM PSD of given data

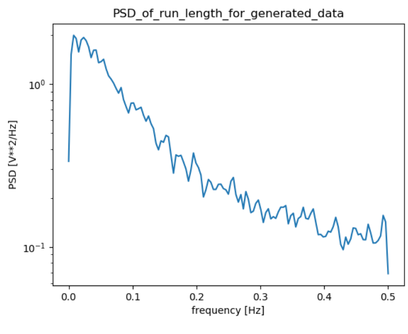

Figure 16: GM PSD of generated data

Detailed Python Script Appendix

A Construct Simple Gilbert Model (SGM)

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##
# @file     sgm_generate.py
# @author   Kyeong Soo (Joseph) Kim <kyeongsoo.kim@gmail.com>
# @date     2020-03-25
#
# @brief    A function for generating loss pattern based on simple Guilbert model.
#

import numpy as np
import sys


def sgm_generate(len, tr):
    """
    Generates a binary sequence of 0 (GOOD) and 1 (BAD) of length
    len from an SGM specified by a 2x2 transition probability matrix
    tr; tr[i, j] is the probability of transition from state i to
    state j.

    This function always starts the model in GOOD (0) state.

    Examples:

    import numpy as np

    tr = np.array([[0.95, 0.10],
                   [0.05, 0.90]])
    seq = sgm_generate(100, tr)
    """

    seq = np.zeros(len)

    # tr must be 2x2 matrix
    tr = np.asarray(tr)  # make sure seq is numpy 2D array
    if tr.shape != (2, 2):
        sys.exit("size of transition matrix is not 2x2")

    # create a random sequence for state changes
    statechange = np.random.rand(len)

    # Assume that we start in GOOD state (0).
    state = 0

    # 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

```

```python
    return seq


if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-L",
        "--length",
        help="the length of the loss pattern to be generated; default is 10",
        default=10,
        type=int)
    parser.add_argument(
        "-T",
        "--transition",
        help="transition matrix in row-major order; default is \"0.95,0.10,0.05,0.90\"",
        default="0.95,0.10,0.05,0.90",
        type=str)
    args = parser.parse_args()
    len = args.length
    tr = np.reshape(np.fromstring(args.transition, sep=','), (2, 2))
    print(sgm_generate(len, tr))
```

B Estimate Simple Gilbert Model

```python
#!/usr/bin/env python
# encoding: utf-8

# @author: Zhipeng Ye
# @contact: Zhipeng.ye19@xjtlu.edu.cn
# @file: estimateparams.py
# @time: 2020-04-03 19:41
# @desc:
import scipy.io as sio
import numpy as np

if __name__ == "__main__":
    matfile = sio.loadmat('loss_seq.mat')
    binary_seq = matfile['seq'][0].tolist()
    length_seq = len(binary_seq)

    n1 = np.count_nonzero(binary_seq)
    n0 = length_seq - n1
    n01 = 0
    n10 = 0

    for index in range(len(binary_seq) - 2):
        if binary_seq[index:index + 2] == [0, 1]:
            n01 += 1

        if binary_seq[index:index + 2] == [1, 0]:
            n10 += 1

    p = n01 / n0

    q = n10 / n1

    print("p:{},q:{}".format(p, q))
```

C Plot Histogram and PSD of SGM

```python
#!/usr/bin/env python
# encoding: utf-8

# @author: Zhipeng Ye
# @contact: Zhipeng.ye19@xjtlu.edu.cn
# @file: main.py
# @time: 2020-04-03 14:16
# @desc:
import scipy.io as sio
import binary_runlengths as brl
import numpy as np
import matplotlib.pyplot as plt
import json
from scipy import signal

if __name__ == '__main__':
    mat_contents = sio.loadmat('loss_seq.mat')
    binary_seq = mat_contents['seq']

    given_zerorl, given_onerl = brl.binary_runlengths(binary_seq)

    # plot histogram
    plt.hist(given_zerorl, normed=True)
    plt.title("Histogram of zero run length for given data")
    plt.xlabel("run length")
    plt.ylabel('frequency')
    plt.savefig('Histogram of zero run length for given data')
    plt.show()

    plt.hist(given_onerl, normed=True)
    plt.title("Histogram of one run length for given data")
    plt.xlabel("run length")
    plt.ylabel('frequency')
    plt.savefig('Histogram of one run length for given data')
    plt.show()

    with open('sgmseq.out') as file:
        generated_data = json.load(file)

    generated_zerorl, generated_onerl = brl.binary_runlengths(generated_data)

    # plot histogram
    plt.hist(generated_zerorl, normed=True)
    plt.title("Histogram of zero run length for generated data")
    plt.xlabel("run length")
    plt.ylabel('frequency')
    plt.savefig('Histogram of zero run length for generated data')
    plt.show()
    plt.hist(generated_onerl, normed=True)
    plt.title("Histogram of one run length for generated data")
    plt.xlabel("run length")
    plt.ylabel('frequency')
    plt.savefig('Histogram of one run length for generated data')
    plt.show()

    frequency_given, psd_given = signal.welch(binary_seq[0])
    plt.semilogy(frequency_given, psd_given)
    plt.title("PSD of run length for given data")
    plt.xlabel("frequency [Hz]")
```

```
    plt.ylabel('PSD [V**2/Hz]')
    plt.savefig("PSD of run length for given data")
    plt.show()

    frequency_generated, psd_generated = signal.welch(generated_data)
    plt.semilogy(frequency_generated, psd_generated)
    plt.title("PSD of run length for generated data")
    plt.xlabel("frequency [Hz]")
    plt.ylabel('PSD [V**2/Hz]')
    plt.savefig("PSD of run length for generated data")
    plt.show()
```

D Plot Theoretical SGM Packet Loss Curve

```
#!/usr/bin/env python
# encoding: utf-8

# @author: Zhipeng Ye
# @contact: Zhipeng.ye19@xjtlu.edu.cn
# @file: generatetheoreticalcurve.py
# @time: 2020-04-05 16:15
# @desc:

import matplotlib.pyplot as plt

k = 50

x = [i for i in range(k)]
seq = [0.2550574084199016*0.7449425915800985**(i) for i in range(k)]

plt.plot(x,seq)
plt.title('packet loss run length possibility density')
plt.xlabel('run length')
plt.ylabel('possibility')
plt.savefig("packet_loss_run_length_possibility_density")
plt.show()
```

E Construct Gilbert Model (GM)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##
# @file     sgm_generate.py
# @author   Kyeong Soo (Joseph) Kim <kyeongsoo.kim@gmail.com>
# @date     2020-03-25
#
# @brief    A function for generating loss pattern based on simple Guilbert model.
#

import numpy as np
import sys
import json
import random

def gm_generate(len, tr, emission_probability):
    """
    Generates a binary sequence of 0 (GOOD) and 1 (BAD) of length
    len from an SGM specified by a 2x2 transition probability matrix
    tr; tr[i, j] is the probability of transition from state i to
    state j.
```

```

    This function always starts the model in GOOD (0) state.

    Examples:

    import numpy as np

    tr = np.array([[0.95, 0.10],
                   [0.05, 0.90]])
    seq = sgm_generate(100, tr)
    """

    seq = np.zeros(len)

    # tr must be 2x2 matrix
    tr = np.asarray(tr)  # make sure seq is numpy 2D array
    if tr.shape != (2, 2):
        sys.exit("size of transition matrix is not 2x2")

    # create a random sequence for state changes
    statechange = np.random.rand(len)

    # Assume that we start in GOOD state (0).
    state = 0
    loss_state = 0

    # 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
        # if latent state is 1 (Bad state), we must consider the emission probability to
        # generate sequence
        if state == 1:
            randomvalue = random.random()
            if randomvalue <= emission_probability:
                loss_state = 1

        # add a binary value to output
        seq[i] = loss_state

        loss_state = 0

    return seq


if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-L",
        "--length",
        help="the length of the loss pattern to be generated; default is 10",
        default=10,
        type=int)
    parser.add_argument(
        "-H",
        "--H",
```

```
        help="emission probability; default is 0.0",
        default=0.0,
        type=float)
    parser.add_argument(
        "-T",
        "--transition",
        help="transition matrix in row-major order; default is \"0.95,0.10,0.05,0.90\"",
        default="0.95,0.10,0.05,0.90",
        type=str)
    args = parser.parse_args()
    len = args.length
    emission_probability = 1 - args.H
    tr = np.reshape(np.fromstring(args.transition, sep=','), (2, 2))
    seq = gm_generate(len, tr, emission_probability).tolist()

    with open('gmseq.out','w') as file:
        json.dump(seq,file)
```

F Estimate Gilbert Model

```
#!/usr/bin/env python
# encoding: utf-8

# @author: Zhipeng Ye
# @contact: Zhipeng.ye19@xjtlu.edu.cn
# @file: estimateGMparams.py
# @time: 2020-04-06 14:12
# @desc:
import scipy.io as sio
import numpy as np

if __name__ == "__main__":
    matfile = sio.loadmat('loss_seq.mat')
    binary_seq = matfile['seq'][0].tolist()
    length_seq = len(binary_seq)

    n1 = np.count_nonzero(binary_seq)

    p1 = n1 / length_seq
    a = p1

    n11 = 0
    for index in range(length_seq - 1):
        if binary_seq[index:index + 2] == [1, 1]:
            n11 += 1

    p11 = n11 / length_seq
    b = p11 / p1

    n111 = 0
    n101 = 0
    for index in range(length_seq - 2):
        if binary_seq[index:index + 3] == [1, 1, 1]:
            n111 += 1
        if binary_seq[index:index + 3] == [1, 0, 1]:
            n101 += 1

    p111 = n111 / length_seq
    p101 = n101 / length_seq

    c = p111 / (p111 + p101)
```

```python

    q = 1 - (a * c - b ** 2) / (2 * a * c - b * (a + c))

    h = 1 - b / (1 - q)

    p = (a * q) / (1 - h - a)

    print("a:{},b:{},c:{}".format(a, b, c))
    print("p:{},q:{},h:{}".format(p, q, h))
```

G Plot Histogram and PSD of GM

```python

# !/usr/bin/env python
# encoding: utf-8

# @author: Zhipeng Ye
# @contact: Zhipeng.ye19@xjtlu.edu.cn
# @file: GMHitPSD.py
# @time: 2020-04-03 14:16
# @desc:
import scipy.io as sio
import binary_runlengths as brl
import numpy as np
import matplotlib.pyplot as plt
import json
from scipy import signal

if __name__ == '__main__':
    mat_contents = sio.loadmat('loss_seq.mat')
    binary_seq = mat_contents['seq']

    given_zerorl, given_onerl = brl.binary_runlengths(binary_seq)

    # plot histogram
    plt.hist(given_zerorl, normed=True)
    plt.title("Histogram of zero run length for given data")
    plt.xlabel("run length")
    plt.ylabel('frequency')
    plt.savefig('GM_Histogram_of_zero_run_length_for_given_data')
    plt.show()

    plt.hist(given_onerl, normed=True)
    plt.title("Histogram of one run length for given data")
    plt.xlabel("run length")
    plt.ylabel('frequency')
    plt.savefig('GM_Histogram_of_one_run_length_for_given_data')
    plt.show()

    with open('gmseq.out') as file:
        generated_data = json.load(file)

    generated_zerorl, generated_onerl = brl.binary_runlengths(generated_data)

    # plot histogram
    plt.hist(generated_zerorl, normed=True)
    plt.title("Histogram of zero run length for generated data")
    plt.xlabel("run length")
    plt.ylabel('frequency')
    plt.savefig('GM_Histogram_of_zero_run_length_for_generated_data')
    plt.show()
```

```
    plt.hist(generated_onerl, normed=True)
    plt.title("Histogram of one run length for generated data")
    plt.xlabel("run length")
    plt.ylabel('frequency')
    plt.savefig('GM_Histogram_of_one_run_length_for_generated_data')
    plt.show()

    frequency_given, psd_given = signal.welch(binary_seq[0])
    plt.semilogy(frequency_given, psd_given)
    plt.title("PSD of run length for given data")
    plt.xlabel("frequency [Hz]")
    plt.ylabel('PSD [V**2/Hz]')
    plt.savefig("GM_PSD_of_run_length_for_given_data")
    plt.show()

    frequency_generated, psd_generated = signal.welch(generated_data)
    plt.semilogy(frequency_generated, psd_generated)
    plt.title("PSD_of_run_length_for_generated_data")
    plt.xlabel("frequency [Hz]")
    plt.ylabel('PSD [V**2/Hz]')
    plt.savefig("GM_PSD_of_run_length_for_generated_data")
    plt.show()
```

H Plot Theoretical GM Packet Loss Curve

```
#!/usr/bin/env python
# encoding: utf-8

# @author: Zhipeng Ye
# @contact: Zhipeng.ye19@xjtlu.edu.cn
# @file: generateGMtheoreticalcurve.py
# @time: 2020-04-06 21:30
# @desc:

import matplotlib.pyplot as plt

k = 50

x = [i for i in range(k)]
seq = [0.2553307818480045*0.7446692181519955**(i) for i in range(k)]
plt.plot(x,seq)
plt.title('packet loss run length possibility density')
plt.xlabel('run length')
plt.ylabel('possibility')
plt.savefig("GM_packet_loss_run_length_possibility_density")
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.