Computing Quiz 2_1

April 26, 2024

[ ]:
```
import numpy as np
import random as rnd
import matplotlib.pyplot as plt
from scipy.stats import poisson
```

1. Exercise 14.1: Sampling the Poisson Distribution

[ ]:
```
# 1. Find numerically the smallest positive integer alpha and beta so that the␣
 ↪P(X(I) = 0) <= 0.0005 and P(Y(A) = 0) <= 0.0005.

lambda_param = 2  # rate parameter lambda

def find_smallest_alpha(lambda_param, threshold): # Calculates smallest␣
 ↪positive integer alpha
    alpha = 0.01
    while True:
        # The length of interval I is alpha
        if poisson.pmf(0, lambda_param * alpha) <= threshold:
            return int(np.ceil(alpha)) # Return ceiling of alpha
        alpha += 0.01  # Increment alpha

def find_smallest_beta(lambda_param, probability_threshold): # Calculates␣
 ↪smallest positive integer beta
    beta = 0.01
    while True:
        A = beta**2  # Area of A
        if poisson.pmf(0, lambda_param * A) <= probability_threshold:
            return int(np.ceil(beta))  # Return ceiling of beta
        beta += 0.01  # Increment beta

threshold = 0.0005 # From question

alpha = find_smallest_alpha(lambda_param, threshold) # Find smallest alpha for␣
 ↪X(I)
beta = find_smallest_beta(lambda_param, threshold) # Find smallest beta for Y(A)

print("Alpha = ", alpha)
```

```
print("Beta = ", beta)
```

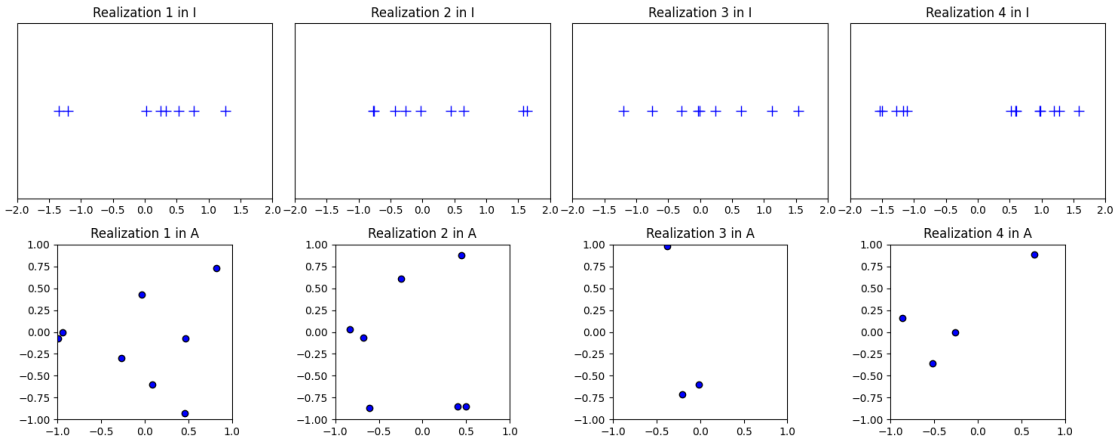
```
Alpha =  4
Beta =  2
```

[ ]:
```
# 2. What is the distributions of the two random variables X(I) and Y(A) with␣
 ↪the values of alpha and beta found in the previous section?
print("For X(I), since lambda = 2 and alpha = 4, |I| = alpha = 4 so that X(I)␣
 ↪is distributed as Poisson(lambda * |I|) = Poisson(2*4) = Poisson(8).\n")
print("Since Y(A) refers to the number of events in area A where Y is a Poisson␣
 ↪process with parameter rate 2 and beta = 2 so that \n|A| = beta^2 = 4, Y(A)␣
 ↪is distributed as Poisson(2*4) = Poisson(8).\n")
print("Thus, both random variables X(I) and Y(A) are poisson random variables␣
 ↪with parameter lambda = 8.")
```

```
For X(I), since lambda = 2 and alpha = 4, |I| = alpha = 4 so that X(I) is
distributed as Poisson(lambda * |I|) = Poisson(2*4) = Poisson(8).

Since Y(A) refers to the number of events in area A which is distributed as
Poisson(A) and beta = 2 so that
|A| = beta^2 = 4, Y(A) is distributed as Poisson(2*4) = Poisson(8).

Thus, both random variables X(I) and Y(A) are poisson random variables with
parameter lambda = 8.
```

[ ]:
```
# 3. Use the algorithm described in this section to generate a realizations of␣
 ↪a poisson random process in I and A. Recall one first draws the number of␣
 ↪events in I and A using X(I) and Y(A) and then places that number of points␣
 ↪uniformly at random in I and A. Plot 4 realizations of each process.

num_realizations = 4

# Generating the Poisson process in 1D for interval I and in 2d for area A
I_length = alpha # Interval I is [-alpha/2, alpha/2], so length is |-alpha/2| +␣
 ↪| alpha/2| = alpha
A_length = beta # Area A is a square with sides of length beta

# Plots for each realization
fig, axs = plt.subplots(2, num_realizations, figsize=(15, 6))

# For each realization, generate the number of events and plot
for i in range(num_realizations):
    # Generate the number of events for interval I, then randomly place in␣
 ↪interval
    events_I = np.random.poisson(lambda_param * I_length)
    points_I = np.random.uniform(-alpha/2, alpha/2, events_I)
```

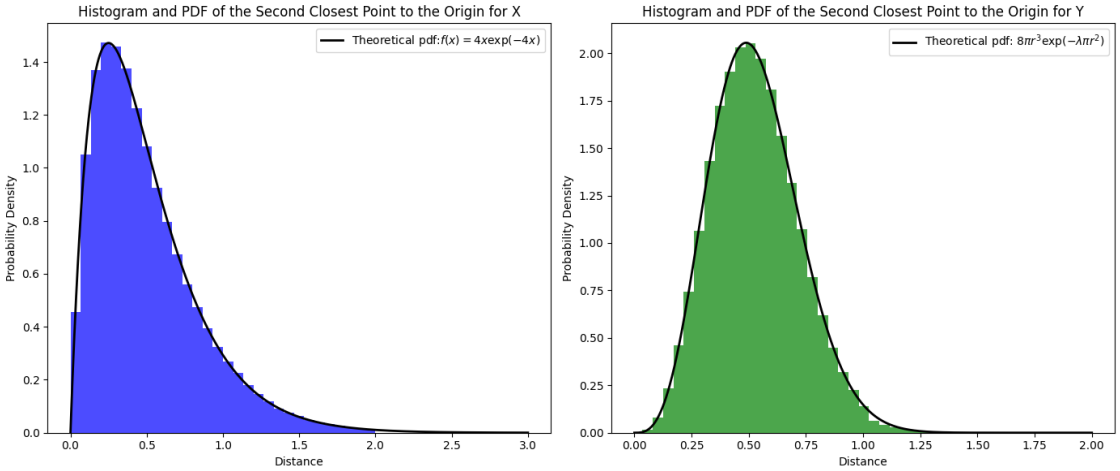
```
    # Plot the events for interval I
    axs[0, i].plot(points_I, np.zeros_like(points_I), 'b+', ms=10)
    axs[0, i].set_title(f"Realization {i+1} in I")
    axs[0, i].set_xlim(-alpha/2, alpha/2)
    axs[0, i].set_ylim(-1, 1)
    axs[0, i].axes.get_yaxis().set_visible(False)

    # Generate the number of events for area A, then randomly place in area A
    events_A = np.random.poisson(lambda_param * A_length**2)
    points_A_x = np.random.uniform(-beta/2, beta/2, events_A)
    points_A_y = np.random.uniform(-beta/2, beta/2, events_A)

    # Plot the events for area A
    axs[1, i].scatter(points_A_x, points_A_y, c='b', edgecolor='k')
    axs[1, i].set_title(f"Realization {i+1} in A")
    axs[1, i].set_xlim(-beta/2, beta/2)
    axs[1, i].set_ylim(-beta/2, beta/2)
    axs[1, i].set_aspect('equal')

plt.tight_layout()
plt.show()
```



[ ]:
```
#4. Using your code, estimate empirically the distribution of the second closes␣
 ↪point to the origin for both X and Y by producing a histogram. Use a enough␣
 ↪samples of the random process to make a nice histogram.

# Function to generate a realization of a Poisson process on the interval I =␣
 ↪[-alpha/2, alpha/2]
def generate_poisson_process_1d(lambda_param, alpha, num_realizations):
    second_closest_points = []  # Store the second closest points to the origin␣
 ↪for each realization
```

```
    for i in range(num_realizations):
        # Generate the number of points in realization
        num_points = np.random.poisson(lambda_param * alpha)
        # Generate uniform points within the interval I
        points = np.random.uniform(-alpha / 2, alpha / 2, num_points)
        # Sort the points by their absolute distance to the origin
        sorted_points = sorted(points, key=abs)
        # Get the second closest point to the origin if there are at least 2␣
 ↪points
        if len(sorted_points) >= 2:
            second_closest_points.append(abs(sorted_points[1]))

    return second_closest_points

# Function to generate a realization of a Poisson process on area A = [-beta/2,␣
 ↪beta/2]^2
def generate_poisson_process_2d(lambda_param, beta, num_realizations):
    second_closest_distances = []  # Store the distances of the second closest␣
 ↪points to the origin for each realization
    for i in range(num_realizations):
        # Generate the number of points in this realization
        num_points = np.random.poisson(lambda_param * beta**2)
        # Generate uniform points within the area A
        x_points = np.random.uniform(-beta/2, beta/2, num_points)
        y_points = np.random.uniform(-beta/2, beta/2, num_points)
        # Calculate distances from the origin
        distances = np.sqrt(x_points**2 + y_points**2)
        # Sort the distances
        sorted_distances = np.sort(distances)
        # Get the second closest distance to the origin if there are at least 2␣
 ↪points
        if len(sorted_distances) >= 2:
            second_closest_distances.append(sorted_distances[1])

    return second_closest_distances

num_realizations = 100_000

# Generate empirical data
second_closest_1d = generate_poisson_process_1d(lambda_param, alpha,␣
 ↪num_realizations)
second_closest_2d = generate_poisson_process_2d(lambda_param, beta,␣
 ↪num_realizations)

# Theoretical PDFs
theoretical_pdf_1d = lambda x: 4*x * np.exp(-4*x)
```

```
theoretical_pdf_2d = lambda r: 8*np.pi* r**3 * np.exp(-lambda_param*np.pi*r**2)

# Generate x values for plotting the theoretical PDFs
x_vals_1d = np.linspace(0, 3, 1000)
x_vals_2d = np.linspace(0, 2, 1000)

# Calculate the theoretical PDF values for each x value
pdf_vals_1d = theoretical_pdf_1d(x_vals_1d)
pdf_vals_2d = theoretical_pdf_2d(x_vals_2d)

# Normalize the theoretical PDFs
pdf_vals_1d /= np.trapz(pdf_vals_1d, x_vals_1d)
pdf_vals_2d *= (max(second_closest_2d) / 1000) * num_realizations

# Plotting the histograms for the 1D and 2D processes with theoretical PDFs
plt.figure(figsize=(14, 6))

# 1D histogram and theoretical PDF
plt.subplot(1, 2, 1)
plt.hist(second_closest_1d, bins=30, color='blue', alpha=0.7, density=True)
plt.plot(x_vals_1d, pdf_vals_1d, 'k-', linewidth=2, label = 'Theoretical pdf:
 ↪$f(x) = 4x \exp(-4x)$')
plt.legend()
plt.title('Histogram and PDF of the Second Closest Point to the Origin for X')
plt.xlabel('Distance')
plt.ylabel('Probability Density')

# 2D histogram and theoretical PDF
plt.subplot(1, 2, 2)
plt.hist(second_closest_2d, bins=30, color='green', alpha=0.7, density=True)
plt.plot(x_vals_2d, pdf_vals_2d / np.trapz(pdf_vals_2d, x_vals_2d), 'k-',␣
 ↪linewidth=2, label= "Theoretical pdf: $8 \pi r^3 \exp(-\lambda \pi r^2)$")
plt.legend()
plt.title('Histogram and PDF of the Second Closest Point to the Origin for Y')
plt.xlabel('Distance')
plt.ylabel('Probability Density')

plt.tight_layout()
plt.show()
```


Histogram and PDF of the Second Closest Point to the Origin for X
Theoretical pdf:$f(x) = 4x\exp(-4x)$
Probability Density
Distance
Histogram and PDF of the Second Closest Point to the Origin for Y
Theoretical pdf: $8\pi r^3\exp(-\lambda\pi r^2)$
Probability Density
Distance

[ ]:

1. Exercise 14.4: Sampling the Poisson Distribution

5.

- Distribution of second closest point to origin for X

$f(x) = \lambda e^{-\lambda x}, \quad x \geq 0 \quad (\text{otherwise } 0)$

since second closest point is the sum of independent exponential random variables, $f(x) = (\lambda e^{-\lambda x})(\lambda e^{-\lambda x}) = \lambda^2 e^{-2\lambda x} = (2)^2 e^{-2(2)x} = \boxed{4e^{-4x}}$

- Distribution of second closest point to origin for Y

since A is 2D, distance is from origin is radial $\Rightarrow$ # points within a circle of radius r follows $\text{Poisson}(\lambda \pi r^2)$

$f(r) = \frac{d}{dr}\left(1 - e^{-\lambda \pi r^2}\right) = 2\pi\lambda r e^{-\lambda \pi r^2}$ (pdf for closest point being at a distance r)

$\frac{d^2}{dr^2}\left(1 - e^{-\lambda \pi r^2}\right) = 2(\pi\lambda)^2 r^3 e^{-\lambda \pi r^2} = 8\pi r^3 e^{-\lambda \pi r^2}$ (pdf for second closest point being at a distance r) $\Rightarrow \boxed{f(x) = 8\pi x^3 e^{-2\pi x^2}}$

6.

- Distribution of mth closest point to origin for X

$f(x) = \lambda e^{-\lambda x}, \quad x \geq 0 \quad (\text{otherwise } 0)$

The distance to the mth-closest point is the sum of m independent exponential inter-arrival times (time between events)
Since dealing with the sum of exponential random variables with same rate, it follows a Gamma distribution

$\boxed{f(x) = \frac{2^m x^{m-1} e^{-2mx}}{(m-1)!}}$ From question 5: when $m = 2$, $f(x) = \frac{2^{(2)} x^{(2)-1} e^{-2(2)x}}{(2-1)!} = 4xe^{-4x}$

- Distribution of mth closest point to origin for Y

following the logic from question 5, differentiate to get pdf for mth-closest point: $\boxed{f_m(x) = \frac{d^m}{dx^m}\left(1 - e^{-2\pi x^2}\right)}$

Computing Quiz #2 — Due 4/24/24

Part 2:

1.) We want to find $L(X,\lambda)$ where $X = (X_1, X_2, \dots, X_n)$ given $\lambda > 0$

Poisson Dist is: $P(X_i = K) = \frac{e^{-\lambda}\lambda^K}{K!}$

Thus, $P(X_i = x_i \mid \lambda) = \frac{e^{-\lambda}\lambda^{x_i}}{x_i!}$

Since draws are indep (and order matters), we multiply to find $L(X,\lambda)$:

$$L(X,\lambda) = \prod_{i=1}^{n} P(X_i = x_i) = \prod_{i=1}^{n} \frac{e^{-\lambda}\lambda^{x_i}}{x_i!}$$

$$= e^{-n\lambda}\left(x_i!\right)^{-n}\lambda^{\sum x_i}$$

$$L(X,\lambda) = \left(\frac{e^{-\lambda}\lambda^{x_i}}{\Gamma(x_i+1)}\right)^{n}$$ ← where all x_i are same (given seq w/ one repeated x_i)

OR

In General: $$L(X,\lambda) = e^{-n\lambda}\lambda^{\sum_{i=1}^{n} x_i}\prod_{i=1}^{n}\frac{1}{x_i!}$$

2.) $\pi(\lambda) = \frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}$

Consider $\pi(\lambda \mid X) \propto P(X \mid \lambda)\cdot\pi(\lambda)$, but $P(X \mid \lambda) = L(X,\lambda)$

$$\pi(\lambda \mid X) \propto L(X,\lambda)\cdot\pi(\lambda)$$

$$\propto e^{-n\lambda}\lambda^{\sum_{i=1}^{n} x_i}\prod_{i=1}^{n}\frac{1}{x_i!}\cdot\frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}$$

$$\propto e^{-n\lambda-\beta\lambda}\lambda^{\sum_{i=1}^{n} x_i}\lambda^{\alpha-1}$$

$$\propto e^{-\lambda(\beta+n)}\lambda^{\left(\alpha+\sum_{i=1}^{n} x_i\right)-1}$$

This is a Gamma dist w/ $\alpha' = \alpha + \sum_{i=1}^{n} x_i$ and $\beta' = \beta + n$

3.) $\alpha' = 3948$ and $\beta' = 501$ ///

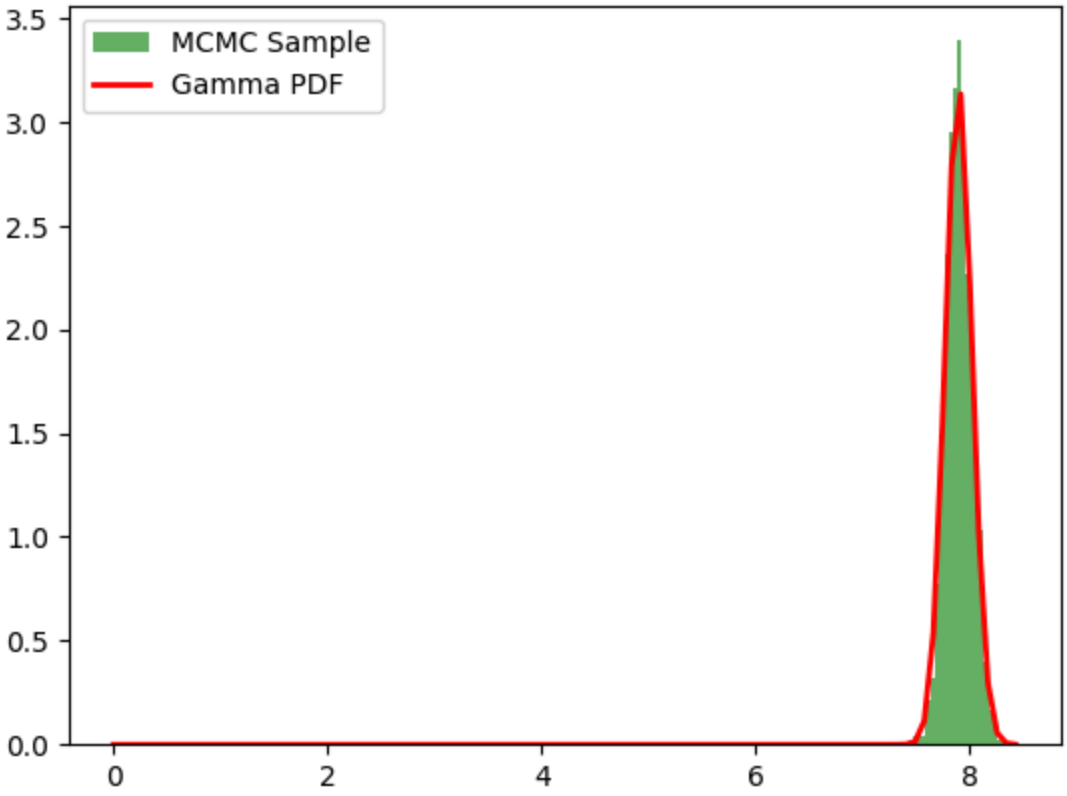
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
#QUESTION 2, Part 3:
X = np.loadtxt("poisson.txt")
#prior distrubution parameters:
a = 10
b = 1
#add up all the values in X
sum = 0
for i in X:
    sum += i
a_prime = a+sum
b_prime = b+500 #500 is the number of draws
print("Our 'a prime' is {:,} and our 'b prime' is {:,}".format(a_prime,b_prime))
```

```
Our 'a prime' is 3,948.0 and our 'b prime' is 501
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gamma

# Parameters for the prior Gamma distribution
alpha, beta = 10, 1

S = 3948 #the sum of the data (from part 3)
n = 500

#This is the same, as from part 3
alpha_prime = alpha + S
beta_prime = beta + n

# Initial lambda
lambda_current = S / n #our initial lambda
N = 10000 #number of samples
samples = []

for i in range(N): #each of these is one iteration of the algorithm
    lambda_proposal = np.random.normal(lambda_current, 0.5)  # Small variance
    if lambda_proposal <= 0:
        continue  # Reject non-positive proposals - we need lambda>0 for a poisson/

    # Calculate the acceptance ratio
    log_acceptance_ratio = (
        (alpha_prime - 1) * np.log(lambda_proposal / lambda_current)
        - (beta_prime * (lambda_proposal - lambda_current))
    )

    if np.log(np.random.random()) < log_acceptance_ratio: #we use np random as oppo
        lambda_current = lambda_proposal

    samples.append(lambda_current)

# Plotting the results
```

```
plt.hist(samples, bins=30, density=True, alpha=0.6, color='g', label='MCMC Sample')
x = np.linspace(0, max(samples), 100)
plt.plot(x, gamma.pdf(x, a=alpha_prime, scale=1/beta_prime), 'r-', lw=2, label='Gam
plt.legend()
plt.show()
```



Part 3 - Questions:

1) total possible codebooks = (number of letters)! = 26!

2) If we calculate pi(B), we would need to consider all possible permutations of the codebook. This would lead to a massive number of evaluations and become impractical for large codebooks or long texts. This would be extremely inefficient whereas using MCMC to take a random walk would be much faster.

3) The probability of selecting any two positions for swapping is the same because we randomly select positions. For example, when we propose to swap from codebook B to B', codebook B' is obtained by swapping two positions in codebook B. Similarly, when proposing to swap from B' to B, codebook B is obtained from swapping two positions in codebook B'. Therefore, the outcome of the swap is independent of the starting codebook B or the proposed B'.

4) $\text{acceptance prob} = \min\left(1, \frac{\pi(B')}{\pi(B)} \cdot \frac{Q(B', B))}{Q(B, B'))}\right)$ given symmetry: $Q(B', B) = Q(B, B')$

$= \min\left(1, \frac{\pi(B')}{\pi(B)}\right)$

In other words the acceptance rate is based on the ratio of the target distribution probabilities at the proposed B' and the current B. This ensures that the Markov chain converges to the correct stationary measure.

5)

C) To calculate the conditional probability, we found the total number of times the given pair occurred in the reference text and divided that by the total number of times the first character occurred in the reference text.

$\text{pair} = (c_1, c_2)$

$P(c_2 \mid c_1) = \frac{\#\text{ of } (c_1, c_2) \text{ pairs}}{\#\text{ of } c_1}$

Computing Quiz 2 Q3

April 26, 2024

3. MCMC to decode an encrypted text

[ ]:
```
import numpy as np
import random
from random import shuffle
import string
import math
import matplotlib.pyplot as plt
```

[ ]:
```
# Convert cipher to string
def cipher_string(cipher):
    cipher_st = ''
    for key in alphabet:
        if key in cipher:
            cipher_st = cipher_st + cipher[key]
    return cipher_st

# Use cipher to decode text
def string_cipher(in_string):
    cipher = {}
    for i in range(len(in_string)):
        cipher[list_alphabet[i]] = in_string[i]
    return cipher

# Create random cipher
def random_cipher():
    cipher = {}
    random_index = [[i] for i in range(len(alphabet))]
    shuffle(random_index)
    for i in range(len(alphabet)-1):
        cipher[list_alphabet[i]] = list_alphabet[random_index[i][0]]
    return cipher

# Encode text
def apply_cipher(text,ci):
    text = list(text)
    new_text = ''
    for char in text:
```

```
        if char.upper() in ci:
            new_text +=ci[char.upper()]
        else:
            new_text += char
    return new_text

# Counts the number of times each character appears in text
def create_single_count_dict(text):
    single_count = {}
    data = list(text.strip())
    for i in range(len(data) - 1):
        char = data[i].upper()
        if char not in alphabet_list and char != " ":
            char = " "
        if char in single_count:
            single_count[char] += 1
        else:
            single_count[char] = 1
    return single_count

# Counts the number of times a pair appears in text
def create_pair_count_dict(text):
    pair_count = {}
    data = list(text.strip())
    for i in range(len(data) - 1):
        char_1 = data[i].upper()
        char_2 = data[i + 1].upper()
        key = char_1 + char_2
        if char_1 not in alphabet_list and char_1 != " ":
            char_1 = " "
        if key in pair_count:
            pair_count[key] += 1
        else:
            pair_count[key] = 1
    return pair_count

# Create conditional prob dict
def create_pair_frequency_dict(text):
    frequency_dict = {}
    text_pair = create_pair_count_dict(text)
    text_single = create_single_count_dict(text)
    for i in range(len(list(text_pair.keys())) - 1):
        key = list(text_pair.keys())[i]
        if key[0] in text_single:
            frequency_dict[key] = text_pair[key]/text_single[key[0]]
    return frequency_dict
```

```
# Find log frequency
def create_pair_log_frequency_dict(text):
    frequency_dict = {}
    text_pair = create_pair_count_dict(text)
    text_single = create_single_count_dict(text)
    for i in range(len(list(text_pair.keys())) - 1):
        key = list(text_pair.keys())[i]
        if key[0] in text_single: #if char in text
            frequency_dict[key] = math.log(text_pair[key]) - math.
 ↪log(text_single[key[0]])
    return frequency_dict

# Find log likelihood of a given cipher
def get_cipher_log_likelihood(text, in_cipher):
    decrypted_text = apply_cipher(text, in_cipher)
    likelihood = 0
    # Loop through text pairs
    for i in range(len(decrypted_text)-1):
        char_1 = decrypted_text[i]
        char_2 = decrypted_text[i + 1]
        key = char_1 + char_2
        if key in likelihood_table: # in reference text
            likelihood = likelihood + likelihood_table[key] # adds the␣
 ↪conditional probabilities
        else: # doesn't appear in reference text
            likelihood = likelihood - 25 #decrease likelihood because this␣
 ↪pairing doesn't exist in reference
    return likelihood

# Generate proposal codebook
def generate_swap(cipher):
    pos1 = random.randint(0, len(list(cipher)) - 1)
    pos2 = random.randint(0, len(list(cipher)) - 1)
    if pos1 == pos2: # same codebook
        return generate_swap(cipher)
    else: # switch the letters in the substitution cipher
        cipher = list(cipher)
        pos1_alpha = cipher[pos1]
        pos2_alpha = cipher[pos2]
        cipher[pos1] = pos2_alpha
        cipher[pos2] = pos1_alpha
        return "".join(cipher)


def MCMC_sample_cipher(text, steps, log_likelihood):
    current_cipher_st = string.ascii_uppercase
    best_state = '' # holds the string of the best cipher
```

```
    switched = 0
    score = -1000000 #starts with a poor score because it is encoded
    for i in range(steps):
        proposed_cipher_st = generate_swap(current_cipher_st) #creates proposal␣
 ↪codebook
        # Convert ciphers to string
        current_cipher = string_cipher(current_cipher_st)
        proposed_cipher = string_cipher(proposed_cipher_st)
        # Find scores of current and proposal for acceptance
        score_current_cipher = get_cipher_log_likelihood(text, current_cipher)
        log_likelihood.append(score_current_cipher)
        score_proposed_cipher = get_cipher_log_likelihood(text, proposed_cipher)
        # Acceptance stage:
        acceptance_prob = score_proposed_cipher - score_current_cipher
        # Switch if uniform random variable is less than the acceptance prob
        if math.log(np.random.uniform(low=0,high=1,size=1)) <  acceptance_prob:
            current_cipher_st = proposed_cipher_st
            switched +=1
        # Don't switch because the current is better
        if score_current_cipher > score:
            best_state = current_cipher_st
            score = score_current_cipher
    #return best substitution (cipher that decodes best)
    return best_state, log_likelihood

# Load reference text
with open('/home/dknox/math231/Quizzes/walden.txt', 'r') as reference:
   reference_text=reference.read().replace('\n', '')

alphabet = string.ascii_uppercase
list_alphabet = list(alphabet)
alphabet_list = list_alphabet

reference_pair = create_pair_frequency_dict(reference_text)
likelihood_table = create_pair_log_frequency_dict(reference_text) # Create log␣
 ↪frequency table

# Load encoded text
directory="/home/dknox/math231/Quizzes/"
file_tag = open(directory+"encoded.txt", "r") # read in encoded message
text=file_tag.read()
file_tag.close()

# Create random cipher
random_cipher_list = list(string.ascii_uppercase)
random.shuffle(random_cipher_list)
test_cipher_st = "".join(random_cipher_list)
```

```
test_cipher = string_cipher(test_cipher_st)
inverse_test_cipher = {v: k for k, v in test_cipher.items()}
encrypted_text = apply_cipher(text,test_cipher)

# Track log-likelihood
log_likelihood = []

# Run MCMC
runs = 3000
MCMC, log_likelihood = MCMC_sample_cipher(encrypted_text,runs,log_likelihood) #␣
 ↪Motropolis-Hastings MCMC
print('The decrypted text is:')
print(apply_cipher(encrypted_text,string_cipher(MCMC)))
print('The best cipher found is:')
print(MCMC + '.')

plt.figure()
x_value = np.linspace(1,3000,3000)
plt.plot(x_value, log_likelihood)
plt.title('Log-Likelihood vs Step #')
plt.xlabel('Step')
plt.ylabel('Log-Likelihood')
plt.grid(True)
plt.show()
```

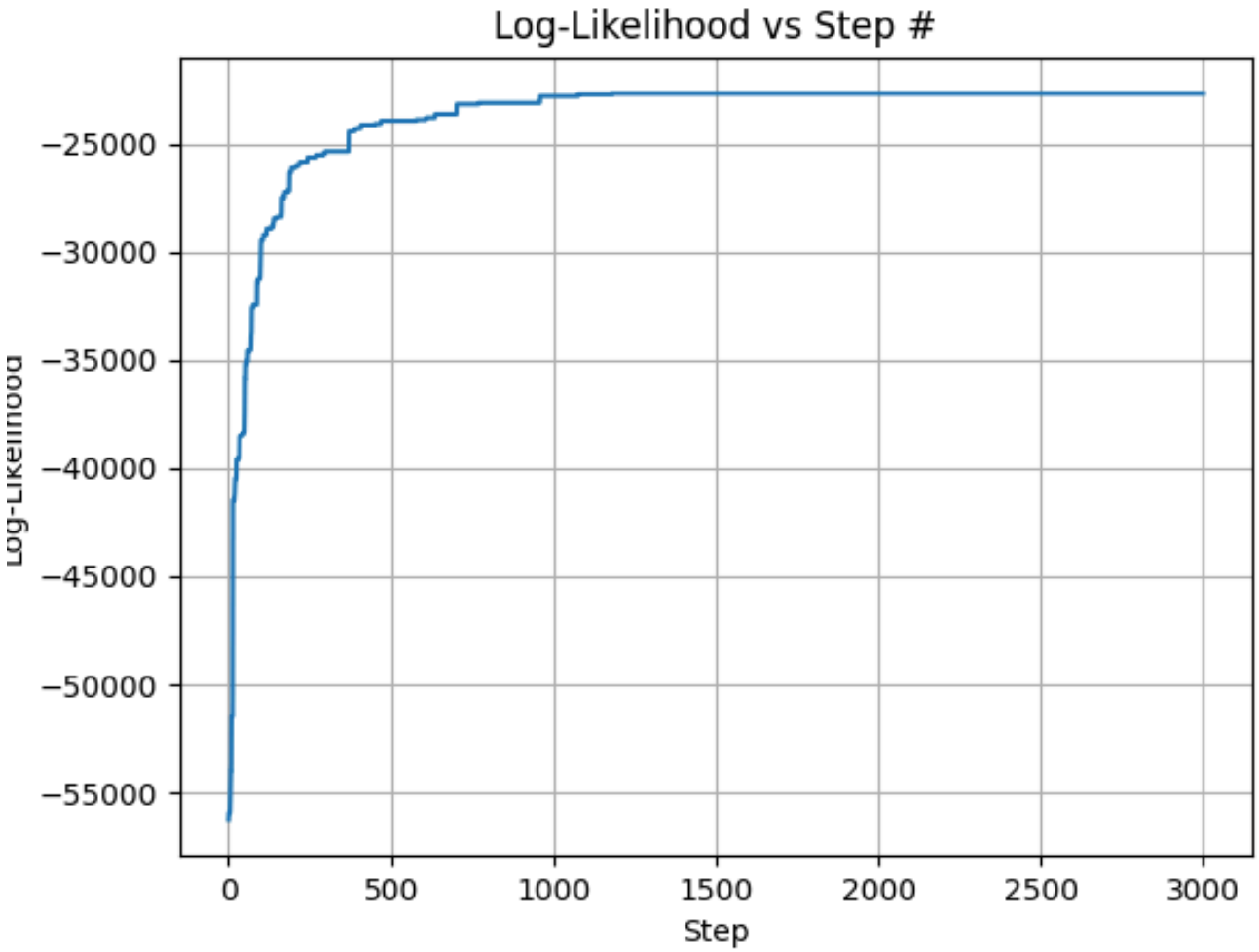
Output of code above is on the following page since it was not run locally in this notebook

The decrypted text is:
WHEN IN THE COURSE OF HUMAN EVENTS IT BECOMES NECESSARY FOR ONE PEOPLE TO DISSOLVE THE POLITICAL BANDS WHICH HAVE CONNECTED THEM WITH ANOTHER AND TO ASSUME AMONG THE POWERS OF THE EARTH THE SEPARATE AND EQUAL STATION TO WHICH THE LAWS OF NATURE AND OF NATURES GOD ENTITLE THEM A DECENT RESPECT TO THE OPINIONS OF MANKIND REQUIRES THAT THEY SHOULD DECLARE THE CAUSES WHICH IMPEL THEM TO THE SEPARATION WE HOLD THESE TRUTHS TO BE SELFEVIDENT THAT ALL MEN ARE CREATED EQUAL THAT THEY ARE ENDOWED BY THEIR CREATOR WITH CERTAIN UNALIENABLE RIGHTS THAT AMONG THESE ARE LIFE LIBERTY AND THE PURSUIT OF HAPPINESS THAT TO SECURE THESE RIGHTS GOVERNMENTS ARE INSTITUTED AMONG MEN DERIVING THEIR JUST POWERS FROM THE CONSENT OF THE GOVERNED THAT WHENEVER ANY FORM OF GOVERNMENT BECOMES DESTRUCTIVE OF THESE ENDS IT IS THE RIGHT OF THE PEOPLE TO ALTER OR TO ABOLISH IT AND TO INSTITUTE NEW GOVERNMENT LAYING ITS FOUNDATION ON SUCH PRINCIPLES AND ORGANIZING ITS POWERS IN SUCH FORM AS TO THEM SHALL SEEM MOST LIKELY TO EFFECT THEIR SAFETY AND HAPPINESS PRUDENCE INDEED WILL DICTATE THAT GOVERNMENTS LONG ESTABLISHED SHOULD NOT BE CHANGED FOR LIGHT AND TRANSIENT CAUSES AND ACCORDINGLY ALL EXPERIENCE HATH SHOWN THAT MANKIND ARE MORE DISPOSED TO SUFFER WHILE EVILS ARE SUFFERABLE THAN TO RIGHT THEMSELVES BY ABOLISHING THE FORMS TO WHICH THEY ARE ACCUSTOMED BUT WHEN A LONG TRAIN OF ABUSES AND USURPATIONS PURSUING INVARIABLY THE SAME OBJECT EVINCES A DESIGN TO REDUCE THEM UNDER ABSOLUTE DESPOTISM IT IS THEIR RIGHT IT IS THEIR DUTY TO THROW OFF SUCH GOVERNMENT AND TO PROVIDE NEW GUARDS FOR THEIR FUTURE SECURITY SUCH HAS BEEN THE PATIENT SUFFERANCE OF THESE COLONIES AND SUCH IS NOW THE NECESSITY WHICH CONSTRAINS THEM TO ALTER THEIR FORMER SYSTEMS OF GOVERNMENT THE HISTORY OF THE PRESENT KING OF GREAT BRITAIN IS A HISTORY OF REPEATED INJURIES AND USURPATIONS ALL HAVING IN DIRECT OBJECT THE ESTABLISHMENT OF AN ABSOLUTE TYRANNY OVER THESE STATES TO PROVE THIS LET FACTS BE SUBMITTED TO A CANDID WORLD HE HAS REFUSED HIS ASSENT TO LAWS THE MOST WHOLESOME AND NECESSARY FOR THE PUBLIC GOOD HE HAS FORBIDDEN HIS GOVERNORS TO PASS LAWS OF IMMEDIATE AND PRESSING IMPORTANCE UNLESS SUSPENDED IN THEIR OPERATIONS TILL HIS ASSENT SHOULD BE OBTAINED AND WHEN SO SUSPENDED HE HAS UTTERLY NEGLECTED TO ATTEND TO THEM HE HAS REFUSED TO PASS OTHER LAWS FOR THE ACCOMMODATION OF LARGE DISTRICTS OF PEOPLE UNLESS THOSE PEOPLE WOULD RELINQUISH THE RIGHT OF REPRESENTATION IN THE LEGISLATURE A RIGHT INESTIMABLE TO THEM AND FORMIDABLE TO TYRANTS ONLY HE HAS CALLED TOGETHER LEGISLATIVE BODIES AT PLACES UNUSUAL UNCOMFORTABLE AND DISTANT FROM THE DEPOSITORY OF THEIR PUBLIC RECORDS FOR THE SOLE PURPOSE OF FATIGUING THEM INTO COMPLIANCE WITH HIS MEASURES HE HAS DISSOLVED REPRESENTATIVE HOUSES REPEATEDLY FOR OPPOSING WITH MANLY FIRMNESS HIS INVASIONS ON THE RIGHTS OF THE PEOPLE HE HAS REFUSED FOR A LONG TIME AFTER SUCH DISSOLUTIONS TO CAUSE OTHERS TO BE ELECTED

WHEREBY THE LEGISLATIVE POWERS INCAPABLE OF ANNIHILATION HAVE RETURNED TO THE PEOPLE AT LARGE FOR THEIR EXERCISE THE STATE REMAINING IN THE MEANTIME EXPOSED TO ALL THE DANGERS OF INVASION FROM WITHOUT AND CONVULSIONS WITHIN HE HAS ENDEAVORED TO PREVENT THE POPULATION OF THESE STATES FOR THAT PURPOSE OBSTRUCTING THE LAWS FOR NATURALIZATION OF FOREIGNERS REFUSING TO PASS OTHERS TO ENCOURAGE THEIR MIGRATIONS HITHER AND RAISING THE CONDITIONS OF NEW APPROPRIATIONS OF LANDS HE HAS OBSTRUCTED THE ADMINISTRATION OF JUSTICE BY REFUSING HIS ASSENT TO LAWS FOR ESTABLISHING JUDICIARY POWERS HE HAS MADE JUDGES DEPENDENT ON HIS WILL ALONE FOR THE TENURE OF THEIR OFFICES AND THE AMOUNT AND PAYMENT OF THEIR SALARIES HE HAS ERECTED A MULTITUDE OF NEW OFFICES AND SENT HITHER SWARMS OF OFFICERS TO HARASS OUR PEOPLE AND EAT OUT THEIR SUBSTANCE HE HAS KEPT AMONG US IN TIMES OF PEACE STANDING ARMIES WITHOUT THE CONSENT OF OUR LEGISLATURES HE HAS AFFECTED TO RENDER THE MILITARY INDEPENDENT OF AND SUPERIOR TO THE CIVIL POWER HE HAS COMBINED WITH OTHERS TO SUBJECT US TO A JURISDICTION FOREIGN TO OUR CONSTITUTION AND UNACKNOWLEDGED BY OUR LAWS GIVING HIS ASSENT TO THEIR ACTS OF PRETENDED LEGISLATION FOR QUARTERING LARGE BODIES OF ARMED TROOPS AMONG US FOR PROTECTING THEM BY A MOCK TRIAL FROM PUNISHMENT FOR ANY MURDERS WHICH THEY SHOULD COMMIT ON THE INHABITANTS OF THESE STATES FOR CUTTING OFF OUR TRADE WITH ALL PARTS OF THE WORLD FOR IMPOSING TAXES ON US WITHOUT OUR CONSENT FOR DEPRIVING US IN MANY CASES OF THE BENEFITS OF TRIAL BY JURY FOR TRANSPORTING US BEYOND SEAS TO BE TRIED FOR PRETENDED OFFENCES FOR ABOLISHING THE FREE SYSTEM OF ENGLISH LAWS IN A NEIGHBORING PROVINCE ESTABLISHING THEREIN AN ARBITRARY GOVERNMENT AND ENLARGING ITS BOUNDARIES SO AS TO RENDER IT AT ONCE AN EXAMPLE AND FIT INSTRUMENT FOR INTRODUCING THE SAME ABSOLUTE RULE INTO THESE COLONIES FOR TAKING AWAY OUR CHARTERS ABOLISHING OUR MOST VALUABLE LAWS AND ALTERING FUNDAMENTALLY THE FORMS OF OUR GOVERNMENTS FOR SUSPENDING OUR OWN LEGISLATURES AND DECLARING THEMSELVES INVESTED WITH POWER TO LEGISLATE FOR US IN ALL CASES WHATSOEVER HE HAS ABDICATED GOVERNMENT HERE BY DECLARING US OUT OF HIS PROTECTION AND WAGING WAR AGAINST US HE HAS PLUNDERED OUR SEAS RAVAGED OUR COASTS BURNT OUR TOWNS AND DESTROYED THE LIVES OF OUR PEOPLE HE IS AT THIS TIME TRANSPORTING LARGE ARMIES OF FOREIGN MERCENARIES TO COMPLETE THE WORKS OF DEATH DESOLATION AND TYRANNY ALREADY BEGUN WITH CIRCUMSTANCES OF CRUELTY AND PERFIDY SCARCELY PARALLELED IN THE MOST BARBAROUS AGES AND TOTALLY UNWORTHY THE HEAD OF A CIVILIZED NATION HE HAS CONSTRAINED OUR FELLOWCITIZENS TAKEN CAPTIVE ON THE HIGH SEAS TO BEAR ARMS AGAINST THEIR COUNTRY TO BECOME THE EXECUTIONERS OF THEIR FRIENDS AND BRETHREN OR TO FALL THEMSELVES BY THEIR HANDS HE HAS EXCITED DOMESTIC INSURRECTIONS AMONGST US AND HAS ENDEAVORED TO BRING ON THE INHABITANTS OF OUR FRONTIERS THE MERCILESS INDIAN SAVAGES WHOSE KNOWN RULE OF WARFARE IS AN UNDISTINGUISHED DESTRUCTION OF ALL AGES

SEXES AND CONDITIONS IN EVERY STAGE OF THESE OPPRESSIONS WE HAVE PETITIONED FOR REDRESS IN THE MOST HUMBLE TERMS OUR REPEATED PETITIONS HAVE BEEN ANSWERED ONLY BY REPEATED INJURY A PRINCE WHOSE CHARACTER IS THUS MARKED BY EVERY ACT WHICH MAY DEFINE A TYRANT IS UNFIT TO BE THE RULER OF A FREE PEOPLE NOR HAVE WE BEEN WANTING IN ATTENTIONS TO OUR BRITISH BRETHREN WE HAVE WARNED THEM FROM TIME TO TIME OF ATTEMPTS BY THEIR LEGISLATURE TO EXTEND AN UNWARRANTABLE JURISDICTION OVER US WE HAVE REMINDED THEM OF THE CIRCUMSTANCES OF OUR EMIGRATION AND SETTLEMENT HERE WE HAVE APPEALED TO THEIR NATIVE JUSTICE AND MAGNANIMITY AND WE HAVE CONJURED THEM BY THE TIES OF OUR COMMON KINDRED TO DISAVOW THESE USURPATIONS WHICH WOULD INEVITABLY INTERRUPT OUR CONNECTIONS AND CORRESPONDENCE THEY TOO HAVE BEEN DEAF TO THE VOICE OF JUSTICE AND OF CONSANGUINITY WE MUST THEREFORE ACQUIESCE IN THE NECESSITY WHICH DENOUNCES OUR SEPARATION AND HOLD THEM AS WE HOLD THE REST OF MANKIND ENEMIES IN WAR IN PEACE FRIENDS WE THEREFORE THE REPRESENTATIVES OF THE UNITED STATES OF AMERICA IN GENERAL CONGRESS ASSEMBLED APPEALING TO THE SUPREME JUDGE OF THE WORLD FOR THE RECTITUDE OF OUR INTENTIONS DO IN THE NAME AND BY THE AUTHORITY OF THE GOOD PEOPLE OF THESE COLONIES SOLEMNLY PUBLISH AND DECLARE THAT THESE UNITED COLONIES ARE AND OF RIGHT OUGHT TO BE FREE AND INDEPENDENT STATES THAT THEY ARE ABSOLVED FROM ALL ALLEGIANCE TO THE BRITISH CROWN AND THAT ALL POLITICAL CONNECTION BETWEEN THEM AND THE STATE OF GREAT BRITAIN IS AND OUGHT TO BE TOTALLY DISSOLVED AND THAT AS FREE AND INDEPENDENT STATES THEY HAVE FULL POWER TO LEVY WAR CONCLUDE PEACE CONTRACT ALLIANCES ESTABLISH COMMERCE AND TO DO ALL OTHER ACTS AND THINGS WHICH INDEPENDENT STATES MAY OF RIGHT DO AND FOR THE SUPPORT OF THIS DECLARATION WITH A FIRM RELIANCE ON THE PROTECTION OF DIVINE PROVIDENCE WE MUTUALLY PLEDGE TO EACH OTHER OUR LIVES OUR FORTUNES AND OUR SACRED HONOUR

The best cipher found is:
KXZWOBAFPYHTDCEGQVRMUNILSJ.


Log-Likelihood vs Step #
−25000
−30000
−35000
−40000
−45000
−50000
−55000
Log-Likelihood
0
500
1000
1500
2000
2500
3000
Step