23120135_cau2

May 31, 2025

1 FINAL PROJECT - CÂU 2

- Họ và tên: Trần Anh Khoa
- Lớp: 23CTT2
- Mã số sinh viên: 23120135
- Học phần: Toán ứng dụng và thống kê

1.1 (a) Hãy mô tả biến ngẫu nhiên X_n phù hợp cho bài toán trên mà có tính chất Markov. Từ đó, xác định ma trận chuyển trạng thái P và vectơ phân phối đầu π_0

Biến ngẫu nhiên X_n là phần dư của tổng các kết quả S_n thu được sau n lần tung xúc xắc khi chia cho 7

Ma trận chuyển trạng thái:

- Ban đầu, $X_n = 0$ vì chưa tung xúc xắc, tổng bằng 0.
- Khi tung xúc xắc, mỗi mặt từ 1 đến 6 có xác suất bằng 1/6, nên không thể giữ nguyên trạng thái vì không có mặt xúc xắc nào 0 chấm. Mỗi trạng thái mới khác đều có xác suất bằng 1/6.

=> Ma trận chuyển trạng thái:

$$P = \begin{pmatrix} 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 \end{pmatrix}$$

Vector phân phối đầu $\pi_0 = [1, 0, 0, 0, 0, 0, 0]$

1.2 (b) Viết hàm dùng để tính xác suất xuất hiện các giá trị phần dư của Sn khi chia cho 7

Ta có vector phân phối đầu $\pi_0 = [1, 0, 0, 0, 0, 0, 0]$ tức là khi n = 0 (chưa tung xúc xắc)

Gọi π_n là phân phối sau n lần tung xúc xắc, khi đó $\pi_n = \pi_0 \times P^n$

[1]:
```
# ========================= CÁC HÀM PHỤ TRỢ =========================

# Hàm tạo ma trận có kích thước row x col mà các số hạng đều là số 0
def create_zero_matrix(row, col):
    return [[0 for _ in range(col)] for _ in range(row)]

# Hàm tạo vector có kích thước size mà các phần tử đều là số 0
def create_zero_vector(size):
    return [0.0 for _ in range(size)]

# Hàm nhân vector với ma trận
def multiply_vector_matrix(vector, matrix):
    result = []
    num_columns = len(matrix[0])
    num_rows = len(matrix)

    for col in range(num_columns):
        total = 0.0
        for row in range(num_rows):
            total += vector[row] * matrix[row][col]
        result.append(total)
    return result

# Hàm nhân hai ma trận
def multiply_matrix(A_list, B_list):
    result_list = [[0 for _ in range(len(B_list[0]))] for _ in␣
 ↪range(len(A_list))]

    m_row_A = len(A_list)
    n_col_B = len(B_list[0])

    for i_row in range(m_row_A):
        for i_col in range(n_col_B):
            total = 0
            for i,a in enumerate(A_list[i_row]):
                total += a*B_list[i][i_col]
            result_list[i_row][i_col] = total

    return result_list

# Hàm tính lũy thừa ma trận
def matrix_power(matrix, power):
    size = len(matrix)

    # Tạo ma trận đơn vị
```

```python
    result = []
    for i in range(size):
        row = []
        for j in range(size):
            if i == j:
                row.append(1.0)
            else:
                row.append(0.0)
        result.append(row)

    # Nhân ma trận 'power' lần
    for _ in range(power):
        result = multiply_matrix(result, matrix)

    return result

# Hàm tạo ma trận chuyển trạng thái P
def create_transition_matrix():
    size = 7
    P = create_zero_matrix(7, 7)

    for current_state in range(size):
        for dice_value in range(1, 7):
            next_state = (current_state + dice_value) % size
            P[current_state][next_state] += 1/6
    return P

# ========================= CÁC HÀM CHÍNH =========================
import pandas as pd

# Hàm dùng để tính xác suất phân phối của Sn % 7 sau n lần tung xúc xắc
def calculate_distribution(n_throws):
    P = create_transition_matrix()
    initial_distribution = create_zero_vector(7)
    initial_distribution[0] = 1.0

    P_power_n = matrix_power(P, n_throws)

    final_distribution = multiply_vector_matrix(initial_distribution, P_power_n)

    return final_distribution

# Hàm tạo bảng phân phối xác suất
def create_table(max_throws):
    data = []

    for n in range(1, max_throws + 1):
```

```python
        distribution = calculate_distribution(n)
        row = []
        for prob in distribution:
            row.append(round(prob, 6))
        data.append(row)

    column_names = []
    for i in range(7):
        column_names.append(f"Sn%7={i}  ")

    row_names = []
    for n in range(1, max_throws + 1):
        row_names.append(f"n = {n}")

    df = pd.DataFrame(data, index=row_names, columns=column_names)

    pd.set_option('display.max_rows', None)
    pd.set_option('display.max_columns', None)
    pd.set_option('display.width', 1000)

    pd.set_option('display.float_format', '{:^9.6f}'.format)

    return df
```

[2]:
```python
if __name__ == "__main__":

    # Số lần tung xúc xắc
    max_throws = 10

    # Tạo bảng phân phối xác suất và in ra
    result_table = create_table(max_throws)
    print(result_table)
```

```
         Sn%7=0    Sn%7=1    Sn%7=2    Sn%7=3    Sn%7=4    Sn%7=5    Sn%7=6
n = 1   0.000000  0.166667  0.166667  0.166667  0.166667  0.166667  0.166667
n = 2   0.166667  0.138889  0.138889  0.138889  0.138889  0.138889  0.138889
n = 3   0.138889  0.143519  0.143519  0.143519  0.143519  0.143519  0.143519
n = 4   0.143519  0.142747  0.142747  0.142747  0.142747  0.142747  0.142747
n = 5   0.142747  0.142876  0.142876  0.142876  0.142876  0.142876  0.142876
n = 6   0.142876  0.142854  0.142854  0.142854  0.142854  0.142854  0.142854
n = 7   0.142854  0.142858  0.142858  0.142858  0.142858  0.142858  0.142858
n = 8   0.142858  0.142857  0.142857  0.142857  0.142857  0.142857  0.142857
n = 9   0.142857  0.142857  0.142857  0.142857  0.142857  0.142857  0.142857
n = 10  0.142857  0.142857  0.142857  0.142857  0.142857  0.142857  0.142857
```

1.3 (c) Viết hàm dùng để kiểm tra xích Markov đã cho có tồn tại phân phối dừng hay không. Nếu có, hãy tính phân phối dừng và chỉ ra thời điểm $t \in N$ sao cho phân phối xác suất π_t chính là phân phối dừng

Thiết lập hệ phương trình:

Từ ma trận chuyển trạng thái P, thiết lập hệ phương trình tuyến tính:

$$\pi P = \pi, \quad \sum_i \pi_i = 1$$

Tương đương:

$$(P^T - I) \cdot \pi^T = 0, \quad \sum_i \pi_i = 1$$

Giải hệ phương trình bằng phương pháp Gauss:

- Chuyển ma trận mở rộng thành dạng bậc thang bằng thuật toán khử Gauss (`Gauss_elimination`).
- Tìm nghiệm bằng thế ngược (`back_substitution`).

Kiểm tra tính hợp lệ của phân phối dừng:

- Tính phân phối sau t bước xuất phát từ phân phối ban đầu.
- Nếu sau một số bước t, phân phối tiệm cận phân phối dừng (sai số nhỏ hơn `epsilon`), thì phân phối dừng tồn tại và tìm được t.

Kết quả:

- Trả về phân phối dừng π, thời điểm hội tụ t (nếu có), và cờ kiểm tra hợp lệ.

[3]:

```python
# ========================= HÀM PHỤ TRỢ =========================
# Hàm chuyển ma trận về dạng bậc thang
def gauss_elimination(matrix):
    rows = len(matrix)
    cols = len(matrix[0])
    row_index = 0

    for col in range(cols - 1):  # Tránh cột hệ số tự do
        pivot_row = -1
        for i in range(row_index, rows):
            if abs(matrix[i][col]) > 1e-9:
                pivot_row = i
                break

        if pivot_row == -1:
            continue
        matrix[row_index], matrix[pivot_row] = matrix[pivot_row],␣
↪matrix[row_index]
        pivot = matrix[row_index][col]
```

```
        if abs(pivot) > 1e-9:
            matrix[row_index] = [value / pivot for value in matrix[row_index]]

        for i in range(row_index + 1, rows):
            factor = matrix[i][col]
            for j in range(cols):
                matrix[i][j] -= factor * matrix[row_index][j]
        row_index += 1

    for i in range(rows):
        for j in range(cols):
            if abs(matrix[i][j]) < 1e-9:
                matrix[i][j] = 0.0
    return matrix

# Hàm giải hệ phương trình từ ma trận đã có dạng bậc thang bằng phương pháp thế␣
↪ngược
def back_substitution(A):
    rows, cols = len(A), len(A[0])
    n = cols - 1
    x = [0] * n

    for i in range(n - 1, -1, -1):
        if abs(A[i][i]) < 1e-9:
            if abs(A[i][-1]) > 1e-9:
                return "Hệ phương trình vô nghiệm"
            continue
        x[i] = A[i][-1]
        for j in range(i + 1, n):
            x[i] -= A[i][j] * x[j]
        x[i] /= A[i][i]
    return x

# ========================= HÀM CHÍNH =========================
# Hàm kiểm tra và tìm phân phối dừng của xích Markov
def check_and_find_stationary_distribution(P, epsilon=1e-9, max_iter=100):
    size = len(P)
    A = []

    # Tạo hệ phương trình để tìm phân phối dừng
    for i in range(size):
        row = [P[j][i] - (1.0 if i == j else 0.0) for j in range(size)]
        row.append(0.0)
        A.append(row)
    normalization_row = [1.0 for _ in range(size)] + [1.0]
    A.append(normalization_row)
```

```
    # Tạo bản sao của A để tránh thay đổi A gốc
    A_copy = [row[:] for row in A]

    # Giải hệ phương trình bằng phương pháp Gauss
    reduced_matrix = gauss_elimination(A_copy)
    solution = back_substitution(reduced_matrix)

    # Xác định kết quả tìm được
    if isinstance(solution, str):
        return None, None, False
    else:
        pi_stationary = solution

        # Kiểm tra xem phân phối dừng có hợp lệ không
        initial_distribution = [1.0] + [0.0] * (size - 1)
        pi_t = initial_distribution[:]

        for t in range(1, max_iter + 1):
            pi_t = multiply_vector_matrix(pi_t, P)
            diff = sum(abs(pi_t[i] - pi_stationary[i]) for i in range(size))
            if diff < epsilon:
                return pi_stationary, t, True
        return pi_stationary, None, True
```

[4]:
```
if __name__ == "__main__":
    P = create_transition_matrix()
    pi_stationary, t_value, has_stationary =␣
↪check_and_find_stationary_distribution(P)

    if not has_stationary:
        print("Xích Markov không tồn tại phân phối dừng.")
    else:
        print("Xích Markov có tồn tại phân phối dừng.")

        print("Phân phối dừng tìm được: pi =", end=" [")
        for i, x in enumerate(pi_stationary):
            print(f"{x:.8f}", end=", " if i < len(pi_stationary)-1 else "")
        print("]")

        if t_value is None:
            print("Không tìm được t sao cho pi_t gần bằng phân phối dừng trong␣
↪số lần lặp cho trước.")
        else:
            print(f"Phân phối dừng đạt được tại t = {t_value}.")
```

```
Xích Markov có tồn tại phân phối dừng.
Phân phối dừng tìm được: pi = [0.14285714, 0.14285714, 0.14285714, 0.14285714,
```

```
0.14285714, 0.14285714, 0.14285714]
Phân phối dừng đạt được tại t = 12.
```

1.4 (d) Quá trình tung xúc xắc được diễn ra cho đến khi tồn tại $i \in N*$ sao cho giá trị S_i chia hết cho 7 thì dừng. Viết hàm tính xác suất tung xúc xắc không quá n lần với giá trị n là một trong những đầu vào của hàm

1.4.1 Ý tưởng triển khai chi tiết cho câu (d)

Xây dựng ma trận chuyển trạng thái hấp thụ

- Xét không gian trạng thái $\{0, 1, 2, 3, 4, 5, 6\}$ tương ứng với $S_n \mod 7$.
- Trạng thái 0 là **trạng thái hấp thụ**, vì khi tổng chia hết cho 7 thì quá trình dừng lại.

Ma trận chuyển trạng thái P:

- $P[0][0] = 1$ (khi đã vào trạng thái 0 thì không thoát ra).
- Với trạng thái $i \neq 0$:

$$P[i][(i+d) \mod 7] += \frac{1}{6} \quad \text{với } d = 1, 2, 3, 4, 5, 6$$

- Và hơn nữa các mặt xúc xắc có giá trị từ 1 đến 6, nên không thể ở lại trạng thái i sau một lần tung (tức là $P[i][i] = 0$ cho $i \neq 0$).

Cụ thể, ma trận P sẽ có dạng:

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 \end{pmatrix}$$

Phân phối xác suất ban đầu

- Xác suất khởi đầu không nằm ở trạng thái hấp thụ:

$$\pi_0 = [0, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}]$$

Tính phân phối xác suất sau mỗi bước

- Với mỗi bước t, phân phối xác suất được tính bằng:

$$\pi_t = \pi_{t-1} \cdot P$$

- Xác suất tổng dừng đúng tại bước t:

$$P(\text{dừng tại bước t}) = \pi_t[0] - \pi_{t-1}[0]$$

- Tổng xác suất dừng trong khoảng từ 2 đến n:

$$P(\text{dừng trong [2,n]}) = \sum_{t=2}^{n} (\pi_t[0] - \pi_{t-1}[0])$$

[5]:
```python
# Hàm tạo ma trận chuyển trạng thái cho câu d
def create_absorbing_matrix():
    size = 7
    P = create_zero_matrix(size, size)

    # Khi ở trạng thái 0, xác suất chuyển sang trạng thái 0 là 1
    P[0][0] = 1.0

    # Các trạng thái khác vẫn chuyển bình thường
    for i in range(1, size):
        for dice in range(1, 7):
            next_state = (i + dice) % size
            P[i][next_state] += 1/6

    return P

# Hàm tính xác suất dừng trong vòng n lần tung xúc xắc
def prob_stop_within_n(n):
    P = create_absorbing_matrix()

    # Bắt đầu từ trạng thái 0 và phân phối đều cho các trạng thái 1 đến 6
    current = [0.0, 1/6, 1/6, 1/6, 1/6, 1/6, 1/6]
    total_prob = 0.0

    # Không thể dừng ở lần 1 vì từ 0 không thể về 0 nên ta sẽ xét từ lần tung␣
↪thứ 2 trở đi
    step = 2
    while step <= n:
        # Tính phân phối xác suất mới sau khi tung xúc xắc
        next_dist = multiply_vector_matrix(current, P)

        # Tính xác suất dừng tại lần tung này
        prob_new_0 = next_dist[0] - current[0]
        total_prob += prob_new_0

        current = next_dist
        step += 1

    return total_prob
```

[6]:

```python
if __name__ == "__main__":
    for n in range(1, 21):
        prob = prob_stop_within_n(n)
        print(f"Xác suất tung xúc xắc không quá {n} lần: {prob:.6f}")
```

```
Xác suất tung xúc xắc không quá 1 lần: 0.000000
Xác suất tung xúc xắc không quá 2 lần: 0.166667
Xác suất tung xúc xắc không quá 3 lần: 0.305556
Xác suất tung xúc xắc không quá 4 lần: 0.421296
Xác suất tung xúc xắc không quá 5 lần: 0.517747
Xác suất tung xúc xắc không quá 6 lần: 0.598122
Xác suất tung xúc xắc không quá 7 lần: 0.665102
Xác suất tung xúc xắc không quá 8 lần: 0.720918
Xác suất tung xúc xắc không quá 9 lần: 0.767432
Xác suất tung xúc xắc không quá 10 lần: 0.806193
Xác suất tung xúc xắc không quá 11 lần: 0.838494
Xác suất tung xúc xắc không quá 12 lần: 0.865412
Xác suất tung xúc xắc không quá 13 lần: 0.887843
Xác suất tung xúc xắc không quá 14 lần: 0.906536
Xác suất tung xúc xắc không quá 15 lần: 0.922113
Xác suất tung xúc xắc không quá 16 lần: 0.935095
Xác suất tung xúc xắc không quá 17 lần: 0.945912
Xác suất tung xúc xắc không quá 18 lần: 0.954927
Xác suất tung xúc xắc không quá 19 lần: 0.962439
Xác suất tung xúc xắc không quá 20 lần: 0.968699
```