TOÁN ỨNG DỤNG VÀ THỐNG KÊ

Final Project - Câu 2

Thông tin sinh viên

- Họ và tên: Nguyễn Thái Bảo
- Mã số sinh viên: 23120023
- Lớp: 23CTT1

Sử dụng thư viện

- Tính toán trên số thực: math

In [1]:
```
import math
```

Các hàm tiện ích

In [2]:
```
def create_matrix(rows, cols, default_value = 0):
    """Tạo ma trận với rows hàng và cols cột"""
    return [[default_value for _ in range(cols)] for _ in range(rows)]


def create_identity_matrix(n):
    """Tạo ma trận đơn vị"""
    identity = create_matrix(n, n)
    for i in range(n):
        identity[i][i] = 1
    return identity
```

```python
def matrix_copy(A):
    """Copy ma trận"""
    return [row[:] for row in A]


def matrix_multiply(A, B):
    """Nhân ma trận A và B"""

    # Trường hợp B là vector
    if not isinstance(B[0], list):
        # Chuyển B thành ma trận cột
        B = [[b] for b in B]

    # Lấy kích thước của ma trận
    rows_A = len(A)
    cols_A = len(A[0])
    rows_B = len(B)
    cols_B = len(B[0]) if isinstance(B[0], list) else 1

    # Kiểm tra xem có thể nhân ma trận không
    if cols_A != rows_B:
        raise ValueError(f"Kích thước không phù hợp: ({rows_A}x{cols_A}) và ({rows_B}x{cols_B})")

    # Khởi tạo ma trận kết quả C kích thước rows_A x cols_B
    C = create_matrix(rows_A, cols_B)
    for i in range(rows_A):
        for j in range(cols_B):
            for k in range(cols_A):
                C[i][j] += A[i][k] * B[k][j]

    # Nếu ma trận kết quả chỉ có 1 cột, trả về vector
    if cols_B == 1:
        return [row[0] for row in C]

    return C


def print_matrix(A):
    """Hàm in ma trận"""
    for row in A:
        print(f"{'  '.join(f'{x:.6f}' for x in row)}")
```

```python
def inverse(P):
    """Tính ma trận nghịch đảo bằng phương pháp Gauss - Jordan"""
    n = len(P)

    # Tạo ma trận mở rộng [P | I]
    P_augmented = []
    for i in range(n):
        row = P[i][:]
        for j in range(n):
            if i == j:
                row.append(1.0)
            else:
                row.append(0.0)
        P_augmented.append(row)

    # Biến đổi Gauss - Jordan
    for i in range(n):
        # Tìm pivot
        max_val = abs(P_augmented[i][i])
        max_row = i
        for k in range(i + 1, n):
            if abs(P_augmented[k][i]) > max_val:
                max_val = abs(P_augmented[k][i])
                max_row = k

        # Hoán đổi hàng
        if max_row != i:
            P_augmented[i], P_augmented[max_row] = P_augmented[max_row], P_augmented[i]

        # Chuẩn hóa hàng pivot
        pivot = P_augmented[i][i]
        for j in range(i, 2*n):
            P_augmented[i][j] /= pivot

        # Khử tất cả các hàng khác
        for k in range(n):
            if k != i:
                factor = P_augmented[k][i]
                for j in range(i, 2*n):
                    P_augmented[k][j] -= factor * P_augmented[i][j]
```

```python
    # Lấy ma trận nghịch đảo
    P_inverse = []
    for i in range(n):
        P_inverse.append(P_augmented[i][n:])

    return P_inverse


def transpose(A):
    """Tính ma trận chuyển vị"""

    # Kiểm tra nếu A là vector
    if not isinstance(A[0], list):
        return [[a] for a in A]  # Chuyển vector hàng thành vector cột

    # Tính ma trận chuyển vị
    rows = len(A)
    cols = len(A[0])
    A_T = [[0 for _ in range(rows)] for _ in range(cols)]

    for i in range(rows):
        for j in range(cols):
            A_T[j][i] = A[i][j]

    return A_T


def matrix_vector_multiply(A, v):
    """
    Nhân ma trận A với vector v
    """
    rows = len(A)
    cols = len(A[0])

    # Kiểm tra điều kiện
    if cols != len(v):
        raise ValueError("Số cột của ma trận phải bằng độ dài của vector")

    result = [0 for _ in range(rows)]

    for i in range(rows):
```

```python
        for j in range(cols):
            result[i] += A[i][j] * v[j]

    return result


def matrix_power(A, n):
    """
    Tính lũy thừa bậc n của ma trận A
    """
    size = len(A)

    # Ma trận đơn vị
    result = create_identity_matrix(size)

    if n == 0:
        return result  # A^0 = I

    if n == 1:
        return matrix_copy(A)

    for _ in range(abs(n)):
        result = matrix_multiply(result, A)

    return result


def gauss_elimination(A, b):
    """
    Giải hệ phương trình tuyến tính Ax = b bằng phương pháp Gauss
    Trả về nghiệm hoặc None nếu không có nghiệm duy nhất
    """
    n = len(A)

    # Tạo ma trận mở rộng [A|b]
    augmented = []
    for i in range(n):
        row = [float(x) for x in A[i]] + [float(b[i])]
        augmented.append(row)

    # Khử Gauss
    for i in range(n):
```

```python
        # Tìm pivot
        max_row = i
        for k in range(i + 1, n):
            if abs(augmented[k][i]) > abs(augmented[max_row][i]):
                max_row = k

        # Hoán đổi hàng
        augmented[i], augmented[max_row] = augmented[max_row], augmented[i]

        # Kiểm tra pivot = 0
        if abs(augmented[i][i]) < 1e-10:
            continue

        # Khử cột
        for k in range(i + 1, n):
            factor = augmented[k][i] / augmented[i][i]
            for j in range(i, n + 1):
                augmented[k][j] -= factor * augmented[i][j]

    # Thế ngược
    x = [0.0] * n
    for i in range(n - 1, -1, -1):
        x[i] = augmented[i][n]
        for j in range(i + 1, n):
            x[i] -= augmented[i][j] * x[j]

        if abs(augmented[i][i]) < 1e-10:
            return None  # Không có nghiệm duy nhất

        x[i] /= augmented[i][i]

    return x


def vector_distance(v1, v2):
    """Tính khoảng cách Euclid giữa hai vector"""

    if len(v1) != len(v2):
        raise ValueError("Hai vector phải có cùng độ dài")

    return math.sqrt(sum((a - b) ** 2 for a, b in zip(v1, v2)))
```

Giải quyết các yêu cầu

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 xích Markov biểu diễn phần dư của S_n khi chia cho 7 (S_n mod 7), với S_n là tổng các kết quả sau khi tung xúc xắc n lần đầu tiên.

Không gian trạng thái:

$$S = \{0, 1, 2, 3, 4, 5, 6\} \tag{1}$$

Ma trận chuyển trạng thái P (7×7):

$$P = \begin{pmatrix} 0 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 0 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 0 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 0 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 0 \end{pmatrix} \tag{2}$$

Khi $n = 0$ (chưa tung lần nào), ta có $S_0 = 0 \Rightarrow X_0 = 0 \bmod 7 = 0$.

Vector phân phối đầu π_0:

$$\pi_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{3}$$

In [3]:
```python
# Ma trận chuyển trạng thái P
P = [
        [0, 1/6, 1/6, 1/6, 1/6, 1/6, 1/6],
        [1/6, 0, 1/6, 1/6, 1/6, 1/6, 1/6],
        [1/6, 1/6, 0, 1/6, 1/6, 1/6, 1/6],
        [1/6, 1/6, 1/6, 0, 1/6, 1/6, 1/6],
        [1/6, 1/6, 1/6, 1/6, 0, 1/6, 1/6],
        [1/6, 1/6, 1/6, 1/6, 1/6, 0, 1/6],
        [1/6, 1/6, 1/6, 1/6, 1/6, 1/6, 0]
    ]

# Phân phối đầu
pi_0 = [1, 0, 0, 0, 0, 0, 0]
```

b. Viết hàm dùng để tính xác suất xuất hiện các giá trị phần dư của Sn khi chia cho 7 theo bảng đã cho.

Hàm phụ trợ

In [4]:
```python
def verify_results(results):
    """
    Kiểm tra xem tổng xác suất của mỗi hàng có bằng 1 không
    """
    valid = True
    for row in results:
        total = sum(row[1:])
        if abs(total - 1.0) > 1e-6:
            valid = False
            print(f"n = {row[0]}: Tổng xác suất = {total:.6f} (không bằng 1)")
            break

    return valid


def print_results_table(results):
    """
    In bảng kết quả
    """
    # Tạo header bảng
    headers = ['n'] + [f"Sn % 7 = {i}" for i in range(7)]
    print(" | ".join([f"{headers[0]:^3}"] + [f"{h:^12}" for h in headers[1:]]) + " |")
```

```python
    print("- " * 55)

    # In bảng kết quả
    for row in results:
        n = row[0]
        probs = row[1:]
        print(f"{n:^3} | ", end="")
        for prob in probs:
            print(f"{prob:^12.8f} | ", end="")

        print()
        print("- " * 55)
```

Hàm chính

```python
In [5]: def calculate_remainder_probabilities(P, pi_0, max_n=10):
    """
    Tính xác suất xuất hiện các giá trị phần dư của Sn khi chia cho 7
    """

    # Khởi tạo danh sách kết quả
    results = []

    for n in range(1, max_n + 1):
        # Tính P^n
        P_n = matrix_power(P, n)

        # Tính pi_n = P^n * pi_0
        pi_n = matrix_vector_multiply(P_n, pi_0)

        # Thêm vào kết quả
        row = [n] + pi_n
        results.append(row)

    # Kiểm tra tổng xác suất
    if not verify_results(results):
        print("Cảnh báo: Tổng xác suất không bằng 1 cho một số hàng.")

    # In bảng kết quả
    print("Bảng xác suất xuất hiện các giá trị phần dư của Sn khi chia cho 7:\n")
    print_results_table(results)
```

```python
    return results
```

Tính xác suất cần tìm với n = 10 (theo bảng)

In [6]:
```python
# Tính xác suất phần dư của Sn khi chia cho 7
result_2b = calculate_remainder_probabilities(P, pi_0, max_n=10)
```

Bảng xác suất xuất hiện các giá trị phần dư của Sn khi chia cho 7:

n	Sn % 7 = 0	Sn % 7 = 1	Sn % 7 = 2	Sn % 7 = 3	Sn % 7 = 4	Sn % 7 = 5	Sn % 7 = 6
1	0.00000000	0.16666667	0.16666667	0.16666667	0.16666667	0.16666667	0.16666667
2	0.16666667	0.13888889	0.13888889	0.13888889	0.13888889	0.13888889	0.13888889
3	0.13888889	0.14351852	0.14351852	0.14351852	0.14351852	0.14351852	0.14351852
4	0.14351852	0.14274691	0.14274691	0.14274691	0.14274691	0.14274691	0.14274691
5	0.14274691	0.14287551	0.14287551	0.14287551	0.14287551	0.14287551	0.14287551
6	0.14287551	0.14285408	0.14285408	0.14285408	0.14285408	0.14285408	0.14285408
7	0.14285408	0.14285765	0.14285765	0.14285765	0.14285765	0.14285765	0.14285765
8	0.14285765	0.14285706	0.14285706	0.14285706	0.14285706	0.14285706	0.14285706
9	0.14285706	0.14285716	0.14285716	0.14285716	0.14285716	0.14285716	0.14285716
10	0.14285716	0.14285714	0.14285714	0.14285714	0.14285714	0.14285714	0.14285714

Nhận xét:

Khi n càng lớn, các giá trị xác suất càng tiến về cùng một giá trị ($\sim \frac{1}{7}$).

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 $t \in \mathbb{N}$ sao cho phân phối xác suất π_t chính là phân phối dừng.

Các hàm phụ trợ

In [7]:
```python
def is_valid_matrix(P):
    """
    Kiểm tra tính hợp lệ của ma trận (tổng mỗi cột = 1 và không có phần tử âm)
    """
    n = len(P)

    for i in range(n):
        row_sum = sum(P[i])
        if row_sum < 0 or abs(row_sum - 1.0) > 1e-10:
            return False, f"Tổng hàng {i} = {row_sum:.6f} không bằng 1"

        for j in range(n):
            if P[i][j] < 0:
                return False, f"P[{i}][{j}] = {P[i][j]} < 0"

    return True, "Ma trận hợp lệ: Tổng mỗi hàng = 1 và không có phần tử âm"


def is_regular_matrix(P):
    """
    Kiểm tra ma trận chính quy (có lũy thừa m sao cho tất cả phần tử > 0)
    """
    n = len(P)

    max_power = 100  # Giới hạn lũy thừa để tránh vòng lặp vô hạn

    # Lặp qua các lũy thừa từ 1 đến n
    for m in range(1, max_power + 1):

        P_m = matrix_power(P, m)

        # Kiểm tra tất cả phần tử của P^m có dương không
        if all(P_m[i][j] > 0 for i in range(n) for j in range(n)):
            return True, f"Ma trận chính quy với lũy thừa m = {m}"

    # Nếu không tìm thấy lũy thừa nào
    return False, "Ma trận không phải là ma trận chính quy"
```

```python
def find_stationary_distribution(P):
    """
    Tìm phân phối dừng của xích Markov với ma trận chuyển P
    Phân phối dừng pi là nghiệm của hệ (P - I) x pi = 0
    """
    n = len(P)

    # Tạo hệ phương trình (P - I) x pi = 0
    A = []
    b = []

    for i in range(n - 1):
        row = []
        for j in range(n):
            if i == j:
                row.append(P[i][j] - 1)
            else:
                row.append(P[i][j])
        A.append(row)
        b.append(0)

    # Thêm phương trình phụ để đảm bảo tổng xác suất = 1
    A.append([1] * n)
    b.append(1)

    # Giải hệ phương trình
    pi = gauss_elimination(A, b)

    if pi is None:
        return None, "Không thể giải hệ phương trình cho phân phối dừng"

    # Kiểm tra nghiệm hợp lệ
    for i, p in enumerate(pi):
        if p < -1e-6:
            return None, f"Phân phối dừng có thành phần âm: pi[{i}] = {p:.6f}"

    # Chuẩn hóa để đảm bảo tổng = 1 vì hệ phương trình là dấu xấp xỉ
    total = sum(pi)
    if abs(total - 1.0) > 1e-10:
        pi = [p / total for p in pi]

    return pi, "Tìm thấy phân phối dừng"
```

```python
def find_stationary_time(P, pi_0, stationary_pi, tolerance=1e-6, max_iterations=1000):
    """
    Tìm thời điểm t sao cho phân phối xác suất pi_t chính là phân phối dừng
    """

    pi_t = [float(x) for x in pi_0]

    for t in range(1, max_iterations + 1):

        # Tính pi_t = P^t * pi_0
        pi_t = matrix_vector_multiply(P, pi_t)

        # Kiểm tra xấp xỉ với phân phối dừng
        distance = vector_distance(pi_t, stationary_pi)

        if distance < tolerance:
            return t, pi_t, distance    # Thời điểm t, phân phối pi_t, khoảng cách đến phân phối dừng

    # Nếu không tìm được t sau số bước tối đa
    print(f"Không tìm được t sau {max_iterations} bước, khoảng cách cuối: {vector_distance(pi_t, stationary_pi):.6f}'
    return None, pi_t, vector_distance(pi_t, stationary_pi)
```

Hàm chính:

In [8]:
```python
def check_stationary_distribution(P, pi_0=None, tolerance=1e-6, max_iterations=1000):
    """
    Hàm chính để kiểm tra phân phối dừng của xích Markov theo yêu cầu.
    """
    n = len(P)

    # Phân phối ban đầu mặc định là phân phối đều
    if pi_0 is None:
        pi_0 = [1/n] * n

    print("KIỂM TRA PHÂN PHỐI DỪNG\n")

    # KIỂM TRA MA TRẬN CHUYỂN
    print("1. Kiểm tra ma trận chuyển:")
```

```python
    # Kiểm tra tính hợp lệ
    is_valid, msg = is_valid_matrix(P)
    if is_valid:
        print(msg)
    else:
        print(f"Lỗi: {msg}")
        return {"error": msg}

    # Kiểm tra ma trận chính quy
    is_regular, msg = is_regular_matrix(P)
    if is_regular:
        print(msg)
    else:
        print(f"Lỗi: {msg}")
        return {"error": msg}

    print()


    # TÍNH PHÂN PHỐI DỪNG
    print("2. Tính phân phối dừng:")

    stationary_pi, msg = find_stationary_distribution(P)
    print(f"{msg}")

    if stationary_pi is None:
        return {"error": msg}

    print(f"Phân phối dừng pi: [{', '.join(f'{p:.6f}' for p in stationary_pi)}]")
    # for i, p in enumerate(stationary_pi):
    #     print(f"pi[{i}] = {p:.6f}")
    print()


    # KIỂM TRA PHÂN PHỐI DỪNG
    print("3. Kiểm tra phân phối dừng: P x pi = pi:")

    P_pi = matrix_vector_multiply(P, stationary_pi)
    max_error = max(abs(a - b) for a, b in zip(stationary_pi, P_pi))

    print(f"Sai số tối đa: {max_error:.2e}")
    if max_error < tolerance:
```

```
        print("Kết luận: P x pi = pi (phân phối dừng hợp lệ)")
    else:
        print("Kết luận: P x pi khác pi (phân phối dừng không hợp lệ)")
    print()


    # 4. CHỈ RA THỜI ĐIỂM XÁC SUẤT PI_T LÀ PHÂN PHỐI DỪNG
    print("4. Chỉ ra thời điểm t sao cho pi_t là phân phối dừng:")
    print(f"Độ chính xác: {tolerance}")

    # Tính thời điểm t
    convergence_time, final_pi, final_distance = find_stationary_time(P, pi_0, stationary_pi, tolerance, max_iteratio

    if convergence_time is not None:
        print(f"Thời điểm: t = {convergence_time}")
        print(f"Khoảng cách (xấp xỉ): {final_distance:.6f}")
        print(f"Phân phối tại t = {convergence_time}: [{', '.join(f'{p:.6f}' for p in final_pi)}]")
        # for i, p in enumerate(final_pi):
        #     print(f"pi_{convergence_time}[{i}] = {p:.6f}")
    else:
        print(f"Chưa tìm thấy thời điểm t trong {max_iterations} bước.")
        print(f"Khoảng cách cuối: {final_distance:.6f}")

    return {
        "stationary_distribution": stationary_pi,
        "convergence_time": convergence_time,
        "final_distribution": final_pi,
        "final_distance": final_distance
    }
```

Kiểm tra kết quả

In [9]:

```
result_2c = check_stationary_distribution(P, pi_0)
```

```
KIỂM TRA PHÂN PHỐI DỪNG

1. Kiểm tra ma trận chuyển:
Ma trận hợp lệ: Tổng mỗi hàng = 1 và không có phần tử âm
Ma trận chính quy với lũy thừa m = 2

2. Tính phân phối dừng:
Tìm thấy phân phối dừng
Phân phối dừng pi: [0.142857, 0.142857, 0.142857, 0.142857, 0.142857, 0.142857, 0.142857]

3. Kiểm tra phân phối dừng: P x pi = pi:
Sai số tối đa: 8.33e-17
Kết luận: P x pi = pi (phân phối dừng hợp lệ)

4. Chỉ ra thời điểm t sao cho pi_t là phân phối dừng:
Độ chính xác: 1e-06
Thời điểm: t = 8
Khoảng cách (xấp xỉ): 0.000001
Phân phối tại t = 8: [0.142858, 0.142857, 0.142857, 0.142857, 0.142857, 0.142857, 0.142857]
```

Nhận xét:

Phân phối dừng cho thấy tất cả các phần dư có xác suất xuất hiện gần bằng nhau (~1/7), có thể giải thích là vì mỗi trạng thái đều có thể chuyển đến các trạng thái khác với xác suất như nhau.

d. Quá trình tung xúc xắc được diễn ra cho đến khi tồn tại $i \in \mathbb{N}^*$ 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.

Khác với ý a, b, c: Mọi trạng thái đều có thể chuyển qua lại giữa các trạng thái khác, thì ở ý d này, nếu $X_i = 0$ thì S_i đã chia hết cho 7, dừng việc tung xúc xắc. Do đó, ta sẽ có ma trận chuyển mới P_2 và vector phân phối đầu π_1 phù hợp với yêu cầu bài toán.

Ma trận chuyển trạng thái P_2 (7×7):

$$P_2 = \begin{pmatrix} 1 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 1/6 & 0 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 1/6 & 1/6 & 0 & 1/6 & 1/6 & 1/6 \\ 0 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 \\ 0 & 1/6 & 1/6 & 1/6 & 1/6 & 0 & 1/6 \\ 0 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 0 \end{pmatrix} \tag{4}$$

Vector phân phối đầu π_1:

$$\pi_1 = \begin{pmatrix} 0 \\ 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \\ 1/6 \end{pmatrix} \tag{5}$$

Vì ban đầu X_0 đã bằng 0, nhưng quá trình tung xúc xắc chưa diễn ra ($i \in \mathbb{N}^*$), nên ta tạm bỏ qua trường hợp chưa tung xúc xắc và bắt đầu tính phân phối đầu từ lần tung đầu tiên (vì xúc xắc chỉ có 6 giá trị từ 1 đến 6 nên sau lần tung đầu, X_1 chưa thể = 0)

In [10]:
```
# Khởi tạo ma trận chuyển P_2
P_2 = create_matrix(7, 7, 0.0)

# 0: chia hết cho 7 --> dừng
# 1-6: các phần dư khi chia cho 7

# S_i chia hết cho 7 thì dừng
P_2[0][0] = 1.0

# Cập nhật ma trận chuyển trạng thái P
for current_state in range(1, 7):
    for dice_value in range(1, 7):
        next_state = (current_state + dice_value) % 7
        P_2[next_state][current_state] += 1/6
```

```python
# Phân phối đầu: xét sau lần tung đầu tiên
# Từ S_0 = 0, tung lần đầu có thể được 1, 2, 3, 4, 5, 6
# X_1 có thể là 1, 2, 3, 4, 5, 6 với xác suất 1/6 mỗi trạng thái
pi = [0.0, 1/6, 1/6, 1/6, 1/6, 1/6, 1/6]

print(f"Ma trận P_2:")
print_matrix(P_2)

print("\nPhân phối đầu pi:")
print(f"[{' '.join(f'{x:.6f}' for x in pi)}]\n")
```

```
Ma trận P_2:
1.000000  0.166667  0.166667  0.166667  0.166667  0.166667  0.166667
0.000000  0.000000  0.166667  0.166667  0.166667  0.166667  0.166667
0.000000  0.166667  0.000000  0.166667  0.166667  0.166667  0.166667
0.000000  0.166667  0.166667  0.000000  0.166667  0.166667  0.166667
0.000000  0.166667  0.166667  0.166667  0.000000  0.166667  0.166667
0.000000  0.166667  0.166667  0.166667  0.166667  0.000000  0.166667
0.000000  0.166667  0.166667  0.166667  0.166667  0.166667  0.000000

Phân phối đầu pi:
[0.000000 0.166667 0.166667 0.166667 0.166667 0.166667 0.166667]
```

Hàm tính xác suất

In [11]:
```python
def calculate_dice_probability(n, P=P_2, pi=pi):
    """
    Tính xác suất tung xúc xắc không quá n lần để tổng chia hết cho 7
    """

    # Kiểm tra
    if n < 1:
        raise ValueError("Số lần tung n phải lớn hơn hoặc bằng 1")

    if len(P) != len(pi):
        raise ValueError("Kích thước ma trận P và vector pi không khớp")

    # Khởi tạo xác suất ban đầu
    current_prob_end = pi[0]

    # Kiểm tra xác suất kết thúc ngay ở lần tung đầu tiên (chắc chắn là 0)
```

```python
    # print(f"Sau 1 lần tung: pi_1 = [{' '.join(f'{x:.6f}' for x in pi)}]")
    # print(f"Xác suất kết thúc sau 1 lần tung: P(kết thúc ngay lần tung đầu) = {current_prob_end:.6f}\n")

    # Lặp từ 2 đến n để tính xác suất kết thúc
    for step in range(2, n + 1):

        # Tính pi_step
        pi = matrix_multiply(P, pi)

        # Cập nhật xác suất kết thúc
        current_prob_end = pi[0]

        # In kết quả từng bước (để tiện kiểm tra)
        # print(f"Sau {step} lần tung: pi_{step} = [{' '.join(f'{x:.6f}' for x in pi)}]")
        # print(f"Xác suất kết thúc sau {step} lần tung: P(kết thúc <= {step} lần) = {current_prob_end:.6f}\n")

    return current_prob_end
```

Áp dụng hàm tính xác suất trên cho các giá trị n

In [12]:
```python
for n in range(1, 11):
    result_2d = calculate_dice_probability(n, P_2, pi)
    print(f"Xác suất tung xúc xắc không quá {n} lần")
    print(f"P(kết thúc <= {n} lần) = {result_2d:.6f}\n")
```

```
Xác suất tung xúc xắc không quá 1 lần
P(kết thúc <= 1 lần) = 0.000000

Xác suất tung xúc xắc không quá 2 lần
P(kết thúc <= 2 lần) = 0.166667

Xác suất tung xúc xắc không quá 3 lần
P(kết thúc <= 3 lần) = 0.305556

Xác suất tung xúc xắc không quá 4 lần
P(kết thúc <= 4 lần) = 0.421296

Xác suất tung xúc xắc không quá 5 lần
P(kết thúc <= 5 lần) = 0.517747

Xác suất tung xúc xắc không quá 6 lần
P(kết thúc <= 6 lần) = 0.598122

Xác suất tung xúc xắc không quá 7 lần
P(kết thúc <= 7 lần) = 0.665102

Xác suất tung xúc xắc không quá 8 lần
P(kết thúc <= 8 lần) = 0.720918

Xác suất tung xúc xắc không quá 9 lần
P(kết thúc <= 9 lần) = 0.767432

Xác suất tung xúc xắc không quá 10 lần
P(kết thúc <= 10 lần) = 0.806193
```

Nhận xét:

Xác suất tăng và tiến dần về 1 khi n tăng.