Conditional Huffman Encoding

核心代码包括以下三个函数：

```
%% Source generation function
function [x] = Markov_chain(space, px1, P, n)
    % Generate Markov chain of length n
    x = zeros(1, n);

    % Initial state sampling
    p = px1;
    cdf_px1 = cumsum(p);
    r = rand;
    idx_first = find(r <= cdf_px1, 1, 'first');
    x(1) = space(idx_first);

    % Subsequent states sampling
    for i = 2:n
        prev_state = x(i-1);
        row_idx = find(space == prev_state, 1, 'first');
        p_current = P(row_idx, :);
        cdf_p_current = cumsum(p_current);
        r = rand;
        idx_i = find(r <= cdf_p_current, 1, 'first');
        x(i) = space(idx_i);
    end
end
```

MATLAB 函数 `Markov_chain` 用于生成一个长度为 `n` 的马尔可夫链。

输入参数：

- `space`：状态空间，表示所有可能状态的集合。
- `px1`：初始状态的概率分布。
- `P`：转移概率矩阵，定义从一个状态转移到另一个状态的概率。
- `n`：所需的马尔可夫链长度。

输出： `x`：生成的马尔可夫链，一个长度为 `n` 的向量，包含依次采样得到的状态。

通过计算累积分布函数 `cumsum` 模拟马尔科夫链的概率分布。

```
%% Encoding process function
function [cx, R, Hx, z] = Conditional_Huffman_Encoding(x, space, px1, P)
    % Conditional Huffman encoding
    N = length(x);
    M = length(space);

    % Compute stationary distribution
    A = [P' - eye(M); ones(1,M)];
    b = [zeros(M,1); 1];
    z_ = A\b;
    z = z_.'/sum(z_);

    % Compute entropy rate
    Hi = zeros(1, M);
    for i = 1:M
        p_i_nonzero = P(i, P(i, :) > 0);
        Hi(i) = -sum(p_i_nonzero .* log2(p_i_nonzero));
    end
    Hx = sum(z .* Hi);

    % Encoding
    cx = cell(1, N);
    totalBits = 0;

    % Encode x(1) using px1
    [dict_init, ~] = huffmandict(space, px1);
```

```
    codeword = dict_init{x(1), 2};
    cx{1} = codeword;
    totalBits = totalBits + length(codeword);

    % Encode x(2) ~ x(N)
    for i = 2:N
        prev_symb = x(i-1);
        row_idx = find(space == prev_symb, 1, 'first');
        p_cond = P(row_idx, :);
        [dict_curr, ~] = huffmandict(space, p_cond);
        codeword = dict_curr{x(i), 2};
        cx{i} = codeword;
        totalBits = totalBits + length(codeword);
    end

    % Compute actual rate
    R = totalBits / N;
end
```

MATLAB 函数 `Conditional_Huffman_Encoding` 实现条件霍夫曼编码，用于编码基于马尔可夫链生成的数据序列。

输入参数:

- `x`：要编码的序列。
- `space`：状态空间（所有可能的状态集合）。
- `px1`：初始状态的概率分布。
- `P`：状态转移概率矩阵。

输出参数:

- `cx`：编码后的霍夫曼码字序列。
- `R`：实际编码速率（平均每个符号的比特数）。
- `Hx`：理论熵率。
- `z`：马尔可夫链的平稳分布。

在该函数中，首先通过矩阵方程 $A \cdot z = b$ 计算马尔可夫链的平稳分布 z，并归一化以确保总和为 1；随后，计算序列的熵率 H_x，对每个状态 i 计算其条件熵 $H(i)$（仅考虑非零转移概率），最终熵率通过平稳分布 z 加权条件熵得到。编码过程中，使用初始分布 p_{x1} 创建霍夫曼字典对序列的第一个符号编码；对于后续符号，依据前一个符号的转移概率分布生成条件霍夫曼字典并编码。所有符号的码字依次存入编码序列 cx，同时累加编码所需的比特数。最后，通过将总比特数除以序列长度 N，计算实际编码速率 R。

> 该函数的实现自动调用了MATLAB的 `huffmandict` 函数，可以直接得到输入概率分布的霍夫曼编码

```
function [x_dec] = Conditional_Huffman_Decoding(cx, px1, P)
    % Initialize variables
    N = length(cx);
    M = length(px1);
    space = 1:M;
    x_dec = zeros(1, N);

    % Decode the first symbol using the initial probability distribution
    [dict_init, ~] = huffmandict(space, px1);
    codeword_1 = cx{1};
    for k = 1:size(dict_init, 1)
        if isequal(dict_init{k, 2}, codeword_1)
            x_dec(1) = dict_init{k, 1};
            break;
        end
    end

    % Decode the remaining symbols based on conditional probabilities
    for i = 2:N
        prev_symb = x_dec(i-1); % Previous decoded symbol
        row_idx = find(space == prev_symb, 1); % Find corresponding row in P
        p_cond = P(row_idx, :); % Conditional probabilities
        [dict_curr, ~] = huffmandict(space, p_cond);
        codeword_i = cx{i};
        for k = 1:size(dict_curr, 1)
            if isequal(dict_curr{k, 2}, codeword_i)
                x_dec(i) = dict_curr{k, 1};
```

```
                break;
            end
        end
    end
end
```

函数 `Conditional\_Huffman\_Decoding` 用于解码条件霍夫曼编码生成的码字序列 cx 还原为原始序列 x_{dec}。解码过程分两部分：

1. **解码第一个符号**：根据初始概率分布 p_{x1} 构建霍夫曼字典，匹配第一个码字 $cx[1]$ 并解码。
2. **解码后续符号**：根据前一个解码符号，从转移概率矩阵 P 中获取条件概率分布，构建条件霍夫曼字典，匹配当前码字并解码。

整个过程依赖马尔可夫链的条件概率特性，逐步完成序列解码。

运行结果如下所示：

```
Generated Markov chain:
     3     4     2     4     3     1     1     2     4     4

Stationary distribution (z):
    0.2500    0.2500    0.2500    0.2500

Entropy rate (Hx): 1.7500
Encoding rate (R): 1.9000 bits/symbol
Decoded sequence:
     3     4     2     4     3     1     1     2     4     4

Decoding successful: no errors detected.
```