64QAM 不同 mapping 及程式比較

謝以恩(411086035)

國立臺北大學 通訊工程學系

內容摘要

藉由本學期所學習之內容，本專案的目的是通過比較兩種不同的映射(mapping)方式來研究 64QAM 的性能，還會從原本的程式優化，並從錯誤率和執行時間的角度，對於四種版本的程式進行分析。我們將研究的兩種映射方式分別是，第一，映射数组方式，會使用預先定義的信號點映射表；第二，手動計算方式，根據公式手動計算信號點。

關鍵字: *映射、時間效能、程式優化*

1. 背景介紹

正交振幅調變(QAM) 是一種廣泛使用的調變方法，其中多種振幅和相位組合用以表示不同的符號。QAM 特別適合用在頻寬受限的環境，因为它能在有限的頻寬中傳輸更多的數據等資料。

然而，隨着調變之位元的增加，例如從上課學的 16QAM 到本次做的 64QAM，系統對於信噪比 (SNR) 的要求也隨之提高，位元錯誤率也隨之增加。

在本次期末專案中，我以 16QAM 的習得知識為基礎，並進一步探討 64QAM 的性能。通過比較不同映射(mapping)方法，對 64QAM 系统的影響，特別是錯誤率率和執行時間的性能，來理解高階 QAM 系统實際性能和計算複雜度。

2. 方法與理論說明

2.1 流程:

大致介紹整體程式之流程，首先是生成 64QAM 信號，這部分就會用到兩種方式，分別為映射數組方式及手動計算方式。前者是預先定義的映射表將隨機生成的數據映射到信號點，這部分是之前所學習的方式。而後者，會利用到星座圖(constellation)(圖一)的概念，根據公式動態計算出每個數據點對應的信號點。

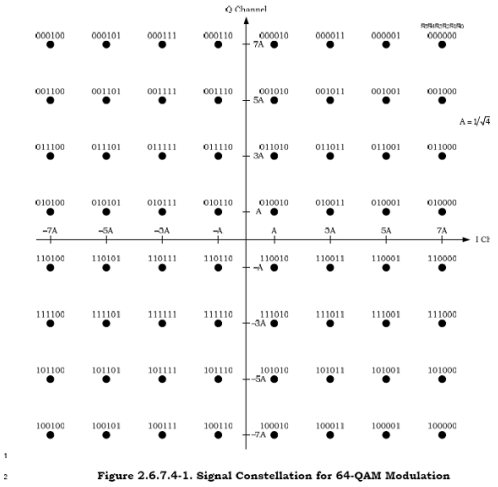


Figure 2.6.7.4-1. Signal Constellation for 64-QAM Modulation

圖一:64QAM 之星座圖

第二步，是添加雜訊，在生成的信號上加入噪聲以模擬接收信號。

第三步，檢測和解碼，我們在 QAM 中，由於每個訊號的能量不會一致，計算兩點之間最小歐幾里得距離。對比 PSK 模擬，PSK 由於能量分布均勻，故可使用內積去計算。這部分也會在程式內做修改，優化解碼之迴圈，最後判定接收信號對應的信號點。

第三，計算錯誤率，通過比較發送和接收的信號點來計算錯誤率。

最後計算執行時間：使用 MATLAB 內置的計時功能(tic、toc)來測量每種方法的運行時間。綜合比較出錯誤率的偏差及效率的程度。

2.2 錯誤率等計算

模擬值的部分，使用的是蒙地卡羅法，仿真方式是一種通過大量隨機試驗來估計結果的數值。在通信系統中，符號錯誤率的精確分析非常複雜且難以直接計算。所以通過類比大量的隨機符號傳輸過程，我們可以得到誤碼率的統計估計值，這種方法對於複雜的調製方式（如本次的64QAM）特別有效。

理論值的話，可透過圖二的式子，並將M=64、$\log_2 M$=6 帶入，計算結果如圖三。

$$P_s \approx 4\left(1-\frac{1}{\sqrt{M}}\right)Q\left(\sqrt{\frac{3\log_2 M \cdot SNR}{M-1}}\right)$$

圖二:理論值計算錯誤率

$$P_s \approx \frac{3}{2}Q\left(\sqrt{\frac{4\cdot SNR}{7}}\right)$$

圖三:錯誤率計算結果

換算成 Bit error rate(BER)的話，可以用圖四的方式換算。

$$P_b \approx \frac{P_s}{\log_2 M}$$

圖四:Bit error rate 跟 symbol error rate 於 QAM 之關係簡化

3. 實驗結果與與討論

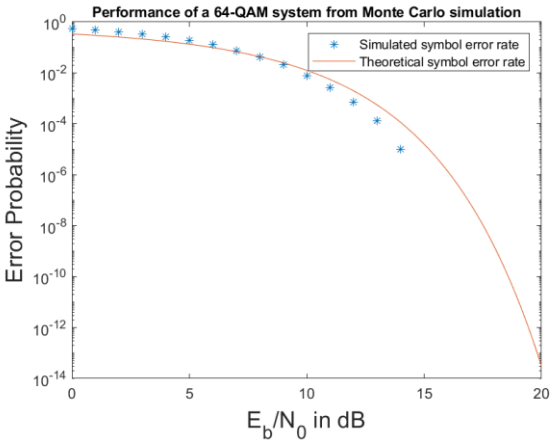
先直接按照之前的作業去調整，改變的部分即是理論值跟映射表。

3.1 使用先定義映射表，且未優化解碼迴圈

```
% Mapping to the signal constellation follows.
mapping = [
    -7*d  7*d; -5*d  7*d; -3*d  7*d; -1*d  7*d;  1*d  7*d;  3*d  7*d;  5*d  7*d;  7*d  7*d;
    -7*d  5*d; -5*d  5*d; -3*d  5*d; -1*d  5*d;  1*d  5*d;  3*d  5*d;  5*d  5*d;  7*d  5*d;
    -7*d  3*d; -5*d  3*d; -3*d  3*d; -1*d  3*d;  1*d  3*d;  3*d  3*d;  5*d  3*d;  7*d  3*d;
    -7*d  1*d; -5*d  1*d; -3*d  1*d; -1*d  1*d;  1*d  1*d;  3*d  1*d;  5*d  1*d;  7*d  1*d;
    -7*d -1*d; -5*d -1*d; -3*d -1*d; -1*d -1*d;  1*d -1*d;  3*d -1*d;  5*d -1*d;  7*d -1*d;
    -7*d -3*d; -5*d -3*d; -3*d -3*d; -1*d -3*d;  1*d -3*d;  3*d -3*d;  5*d -3*d;  7*d -3*d;
    -7*d -5*d; -5*d -5*d; -3*d -5*d; -1*d -5*d;  1*d -5*d;  3*d -5*d;  5*d -5*d;  7*d -5*d;
    -7*d -7*d; -5*d -7*d; -3*d -7*d; -1*d -7*d;  1*d -7*d;  3*d -7*d;  5*d -7*d;  7*d -7*d;
];
for i=1:N
   qam_sig(i,:)=mapping(dsource(i),:);
end
% received signal
for i=1:N
   n  = [sgma*randn sgma*randn];
   %[n(1) n(2)]=gngauss(sgma);
   r(i,:) = qam_sig(i,:) + n;
end
```

圖五:預先定義之映射表

輸出結果如圖六，時間戳記請見圖七。



圖六:使用映射表，未優化之輸出圖形

```
>> prob_07_10
Elapsed time is 95.066112 seconds.
```

圖七:其執行時間

3.2 使用先定義映射表，且優化解碼迴圈

一樣使用映射表，但這次在迴圈上進行優化。

```
numoferr=0;
for i=1:N
    min_dist = Inf;
    min_index = 0;
    % Calculate distance between received signal and each constellation point
    for j = 1:M
        dist = norm(r(i,:) - mapping(j,:)); % Euclidean distance
        if dist < min_dist
            min_dist = dist;
            min_index = j;
        end
    end
    % Decoded symbol is the index of the closest constellation point
    decis = min_index;

  if (decis~=dsource(i))
    numoferr=numoferr+1;
  end
end
p=numoferr/(N);
```

圖八:3.1 方法的迴圈

```
% Detection and error probability calculation
numoferr = 0;

% Vectorized detection
for i = 1:N
    dist = sum((mapping - r(i, :)).^2, 2); % Calculate squared Euclidean distance
    [~, decis] = min(dist); % Find the index of the minimum distance
    if decis ~= dsource(i)
        numoferr = numoferr + 1;
    end
end

p = numoferr / N;
```

圖九:優化的迴圈

嘗試盡量減少迴圈層數後，以下修改效率大提升。

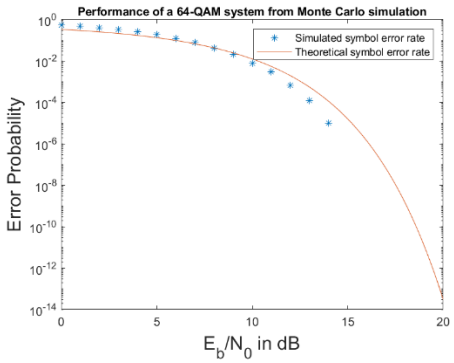
```
% Vectorized detection
for i = 1:N
    dist = sum((mapping - r(i, :)).^2, 2); % Calculate squared Euclidean distance
    [~, decis] = min(dist); % Find the index of the minimum distance
    if decis ~= dsource(i)
        numoferr = numoferr + 1;
    end
end

p = numoferr / N;
```

圖十:修改後的解碼迴圈

```
>> prob_07_10
Elapsed time is 6.125129 seconds.
```

圖十一:執行時間

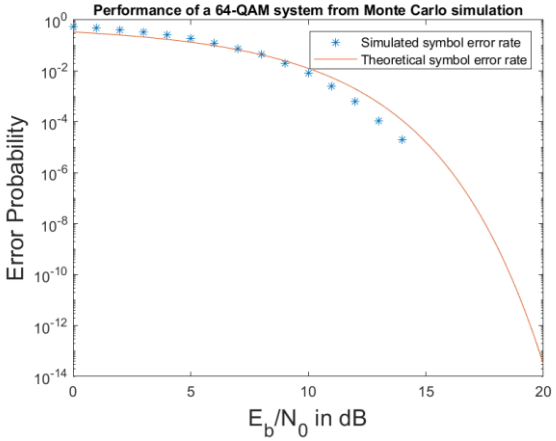


圖十二:執行結果圖，跟圖六相比，幾乎一樣

從圖十一及圖七相比，兩者相差了十幾倍，可見運算的複雜度也是需要留意的地方。

3.3 使用公式動態計算，且未優化解碼迴圈

使用動態運算進行，得出結果如圖十三，執行時間請見圖十四。



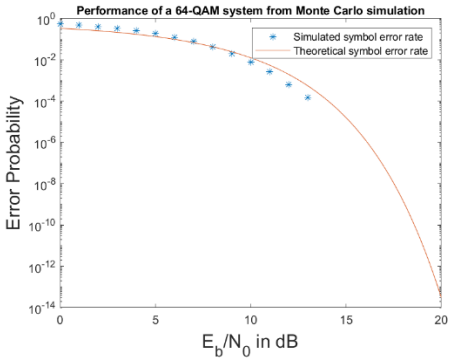
圖十三:動態運算之輸出結果，得出的值與前面相當

```
>> notmap
Elapsed time is 22.119272 seconds.
```

圖十四:執行時間，相對於使用映射表，此情況能有更佳的速度

3.4 使用公式動態計算，且優化解碼迴圈

跟 3.2 的修改方法一樣，去觀察輸出圖形及效率。



圖十五:輸出圖形

```
>> notmap
Elapsed time is 5.210660 seconds.
```

圖十六:執行時間

從圖十四及圖十相比，兩者相差了四倍，可見運算的複雜度也是需要留意的地方。而且，在此專案，使用動態運算的方式，計算速度有明顯的差距。

4. 結論與心得

通過本專案，我們學習並掌握了 64QAM 的調製和解調技術，並了解了不同映射方式對錯誤率和運行效率的影響。

照理來說，映射數組方式由於預先定義了所有信號點，能夠在解碼時快速查找，但需要額外的儲存空間。而手動計算方式則在運行效率上稍遜一籌，但在某些特定應用中可能更為靈活，像是在此專案。

另外，藉由優化解碼端，也能大大減少重複執行之時間，我想未來若有機會再製作相關專案，這也是可以精進的地方。

我認為，本專案的實現和分析為我們在實際通信系統中選擇合適的調製和解調方法提供了寶貴的參考。

5. 參考資料

5.1 16QAM: [Bit error probability of M-ary quadrature amplitude modulation | IEEE Conference Publication | IEEE Xplore]

5.2 Proakis, J. G. (2001). Digital Communications. McGraw-Hill.

5.3 Sklar, B. (2001). Digital Communications: Fundamentals and Applications. Prentice Hall.

附錄(如程式碼)

使用映射表法的主程式

```
clear all; close all;
tic
SNRindB1 = 0:1:20; SNRindB2 = 0:0.1:20; M = 64; k = log2(M);
for i = 1:length(SNRindB1)
    smld_err_prb(i) = cm_sm41(SNRindB1(i)); % simulated error rate
end
for i = 1:length(SNRindB2)
    SNR = exp(SNRindB2(i) * log(10) / 10); % signal-to-noise ratio
    % theoretical symbol error rate for 64-QAM
    theo_err_prb(i) = (3/2) * qfunc(sqrt((4/7) * SNR));
end
% Plotting commands follow.
semilogy(SNRindB1, smld_err_prb, '*');
hold on
semilogy(SNRindB2, theo_err_prb);
legend('Simulated symbol error rate', 'Theoretical symbol error rate');
xlabel('E_b/N_0 in dB', 'fontsize', 16, 'fontname', 'Helvetica');
ylabel('Error Probability', 'fontsize', 16, 'fontname', 'Helvetica');
title('Performance of a 64-QAM system from Monte Carlo simulation', 'fontsize', 10, 'fontname', 'Helvetica');
fname = 'index.png'; print(fname, '-dpng');
toc
```

使用映射表函數(未優化)

```
function [p]=cm_sm41(snr_in_dB)
N=100000;
d=1;                    % min. distance between symbols
Eav=10*d^2;             % energy per symbol
snr=10^(snr_in_dB/10);          % SNR per bit (given)
sgma=sqrt(Eav/(8*snr));         % noise variance
M = 64; % Number of symbols in 64-QAM

% Generation of the data source
dsource = zeros(1, N);
for i = 1:N
    temp = rand; % Generates a uniform random variable between 0 and 1
    dsource(i) = 1 + floor(M * temp); % Generates a number between 1 and 64 uniformly
end
% Mapping to the signal constellation follows.
mapping = [
    -7*d  7*d; -5*d  7*d; -3*d  7*d; -1*d  7*d;  1*d  7*d;  3*d  7*d;  5*d  7*d;  7*d  7*d;
    -7*d  5*d; -5*d  5*d; -3*d  5*d; -1*d  5*d;  1*d  5*d;  3*d  5*d;  5*d  5*d;  7*d  5*d;
    -7*d  3*d; -5*d  3*d; -3*d  3*d; -1*d  3*d;  1*d  3*d;  3*d  3*d;  5*d  3*d;  7*d  3*d;
    -7*d  1*d; -5*d  1*d; -3*d  1*d; -1*d  1*d;  1*d  1*d;  3*d  1*d;  5*d  1*d;  7*d  1*d;
     -7*d -1*d; -5*d -1*d; -3*d -1*d; -1*d -1*d;  1*d -1*d;  3*d -1*d;  5*d -1*d;  7*d -1*d;
     -7*d -3*d; -5*d -3*d; -3*d -3*d; -1*d -3*d;  1*d -3*d;  3*d -3*d;  5*d -3*d;  7*d -3*d;
     -7*d -5*d; -5*d -5*d; -3*d -5*d; -1*d -5*d;  1*d -5*d;  3*d -5*d;  5*d -5*d;  7*d -5*d;
     -7*d -7*d; -5*d -7*d; -3*d -7*d; -1*d -7*d;  1*d -7*d;  3*d -7*d;  5*d -7*d;  7*d -7*d;
];
for i=1:N
    qam_sig(i,:)=mapping(dsource(i),:);
end
% received signal
for i=1:N
    n  = [sgma*randn sgma*randn];
    %[n(1) n(2)]=gngauss(sgma);
    r(i,:) = qam_sig(i,:) + n;
end
% detection and error probability calculation
numoferr=0;
for i=1:N
    min_dist = Inf;
    min_index = 0;
    % Calculate distance between received signal and each constellation point
    for j = 1:M
        dist = norm(r(i,:) - mapping(j,:)); % Euclidean distance
        if dist < min_dist
            min_dist = dist;
            min_index = j;
        end
    end
    % Decoded symbol is the index of the closest constellation point
    decis = min_index;

  if (decis~=dsource(i))
    numoferr=numoferr+1;
  end
end
p=numoferr/(N);
```

使用映射表函數(優化)

```
function [p] = cm_sm41(snr_in_dB)
    N = 100000;
    d = 1;
    Eav = 10 * d^2;
    snr = 10^(snr_in_dB / 10);
    sgma = sqrt(Eav / (8 * snr));
    M = 64;

    % Generate the data source
    dsource = randi([1, M], 1, N);

    % Precompute the mapping array
    mapping = [
        -7*d  7*d; -5*d  7*d; -3*d  7*d; -1*d  7*d;  1*d  7*d;  3*d  7*d;  5*d  7*d;  7*d  7*d;
        -7*d  5*d; -5*d  5*d; -3*d  5*d; -1*d  5*d;  1*d  5*d;  3*d  5*d;  5*d  5*d;  7*d  5*d;
        -7*d  3*d; -5*d  3*d; -3*d  3*d; -1*d  3*d;  1*d  3*d;  3*d  3*d;  5*d  3*d;  7*d  3*d;
        -7*d  1*d; -5*d  1*d; -3*d  1*d; -1*d  1*d;  1*d  1*d;  3*d  1*d;  5*d  1*d;  7*d  1*d;
        -7*d -1*d; -5*d -1*d; -3*d -1*d; -1*d -1*d;  1*d -1*d;  3*d -1*d;  5*d -1*d;  7*d -1*d;
        -7*d -3*d; -5*d -3*d; -3*d -3*d; -1*d -3*d;  1*d -3*d;  3*d -3*d;  5*d -3*d;  7*d -3*d;
        -7*d -5*d; -5*d -5*d; -3*d -5*d; -1*d -5*d;  1*d -5*d;  3*d -5*d;  5*d -5*d;  7*d -5*d;
        -7*d -7*d; -5*d -7*d; -3*d -7*d; -1*d -7*d;  1*d -7*d;  3*d -7*d;  5*d -7*d;  7*d -7*d;
    ];

    % Generate QAM signal using precomputed mapping
    qam_sig = mapping(dsource, :);

    % Add Gaussian noise to the QAM signal
    r = qam_sig + sgma * randn(N, 2);

    % Detection and error probability calculation
    numoferr = 0;

    % Vectorized detection
    for i = 1:N
        dist = sum((mapping - r(i, :)).^2, 2); % Calculate squared Euclidean distance
        [~, decis] = min(dist); % Find the index of the minimum distance
        if decis ~= dsource(i)
            numoferr = numoferr + 1;
        end
    end

    p = numoferr / N;
end
```

使用映射動態運算的主程式

```
clear all; close all;
tic
SNRindB1 = 0:1:20; SNRindB2 = 0:0.1:20;
M = 64; k = log2(M);
for i = 1:length(SNRindB1)
    smld_err_prb(i) = nptmapfunc(SNRindB1(i)); % simulated error rate
end
for i = 1:length(SNRindB2)
    SNR = exp(SNRindB2(i) * log(10) / 10); % signal-to-noise ratio
    % theoretical symbol error rate for 64-QAM
    theo_err_prb(i) = (3/2) * qfunc(sqrt((4/7) * SNR));
end
semilogy(SNRindB1, smld_err_prb, '*');% Plotting commands follow.
hold on
semilogy(SNRindB2, theo_err_prb);
legend('Simulated symbol error rate', 'Theoretical symbol error rate');
xlabel('E_b/N_0 in dB', 'fontsize', 16, 'fontname', 'Helvetica');
ylabel('Error Probability', 'fontsize', 16, 'fontname', 'Helvetica');
title('Performance of a 64-QAM system from Monte Carlo simulation', 'fontsize', 10, 'fontname', 'Helvetica');
fname = 'notmaparray.png'; print(fname, '-dpng');
toc
```

```
function [p] = nptmapfunc(snr_in_dB)
    % CM_SM41 finds the probability of error for the given value of snr_in_dB, SNR in dB.
    N = 100000;
    d = 1; % min. distance between symbols
    Eav = 10 * d^2; % energy per symbol
    snr = 10^(snr_in_dB / 10); % SNR per bit (given)
    sgma = sqrt(Eav / (8 * snr)); % noise variance
    M = 64; % Number of symbols in 64-QAM

    % Generation of the data source
    dsource = randi([1, M], 1, N); % Generates random integers between 1 and 64
    % Mapping to the signal constellation without explicit mapping array
    qam_sig = zeros(N, 2);
    for i = 1:N
        re = 2 * (mod(dsource(i) - 1, sqrt(M)) - (sqrt(M) / 2 - 1/2)) * d;
        im = 2 * (floor((dsource(i) - 1) / sqrt(M)) - (sqrt(M) / 2 - 1/2)) * d;
        qam_sig(i, :) = [re, im];
    end

    % received signal
    r = qam_sig + sgma * randn(N, 2);
    % detection and error probability calculation
    numoferr = 0;
    for i = 1:N
        min_dist = Inf;
        for j = 1:M
            re = 2 * (mod(j - 1, sqrt(M)) - (sqrt(M) / 2 - 1/2)) * d;
            im = 2 * (floor((j - 1) / sqrt(M)) - (sqrt(M) / 2 - 1/2)) * d;
            dist = norm(r(i, :) - [re, im]); % Euclidean distance
            if dist < min_dist
                min_dist = dist;
                decis = j;
            end
        end
        if decis ~= dsource(i)
            numoferr = numoferr + 1;
        end
    end
    p = numoferr / N;
end
```

使用映射動態運算函數(未優化)

```
function [p] = nptmapfunc(snr_in_dB)
    % CM_SM41 finds the probability of error for the given value of snr_in_dB, SNR in dB.
    N = 100000;
    d = 1; % min. distance between symbols
    Eav = 10 * d^2; % energy per symbol
    snr = 10^(snr_in_dB / 10); % SNR per bit (given)
    sgma = sqrt(Eav / (8 * snr)); % noise variance
    M = 64; % Number of symbols in 64-QAM
    sqrtM = sqrt(M); % Calculate the square root of M once

    % Generation of the data source
    dsource = randi([1, M], 1, N); % Generates random integers between 1 and 64

    % Precompute the signal constellation points
    [I, Q] = meshgrid(0:sqrtM-1, 0:sqrtM-1);
    re_points = 2 * (I(:) - (sqrtM / 2 - 1/2)) * d;
    im_points = 2 * (Q(:) - (sqrtM / 2 - 1/2)) * d;
    constellation = [re_points, im_points];

    % Map data source to signal constellation
    % Map data source to signal constellation
    qam_sig = constellation(dsource, :);

    % Add noise to the signal
    noise = sgma * randn(N, 2);
    r = qam_sig + noise;
    % Detection and error probability calculation
    numoferr = 0;
    % Calculate distances between received signal and each constellation point
    for i = 1:N
        distances = sum((constellation - r(i, :)).^2, 2);
        [~, decis] = min(distances);
        if decis ~= dsource(i)
            numoferr = numoferr + 1;
        end
    end
    % Compute error probability
    p = numoferr / N;
end
```

使用映射動態運算函數(優化)