

Εργασία 2

Ψηφιακές Τηλεπικοινωνίες .

Όνομα/επώνυμο : Γρηγόρης Τζωρτζάκης

Αμ:1084538

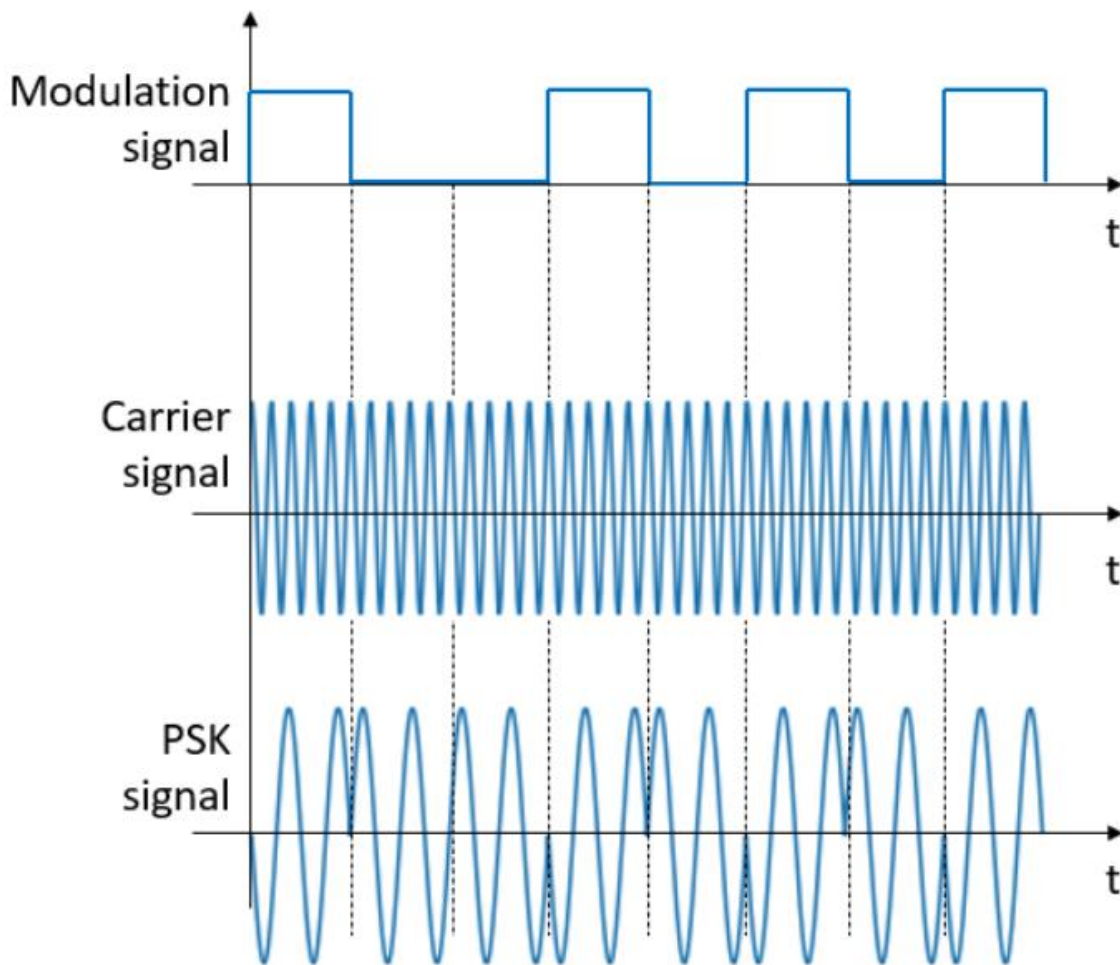
Περιεχόμενα

Εισαγωγή.....	2
Ερώτηση 1.....	4
A) Χωρίς κωδικοποίηση gray.....	4
Διαδική Ακολουθία Εισόδου.....	4
Αντιστοιχία Bits - Συμβόλων	4
Ορθογώνιος Παλμός	5
Διαμόρφωση M-PSK	5
AWGN Κανάλι	6
Αποδιαμορφωτής M-PSK.....	6
Φωρατής M-PSK	7
B) Με κωδικοποίηση gray	7
Ερώτηση 2	8
A) Χωρίς κωδικοποίηση gray.....	8
B) Με κωδικοποίηση gray	10
Ερώτηση 3	12
Ερώτηση 4	13
Ερώτηση 5.....	15
Κώδικας.....	17
1&2 χωρίς Gray).....	17
1&2 με Gray)	20
4).....	23
5)	26

Εισαγωγή

Αρχικά, η διαμόρφωση PSK είναι μια τεχνική όπου η φάση ενός φέροντος αναλογικού σήματος διαμορφώνεται για την κωδικοποίηση ενός διακριτού σήματος, ώστε να περάσει μέσα από αναλογικά κανάλια.

ΕΕΕ



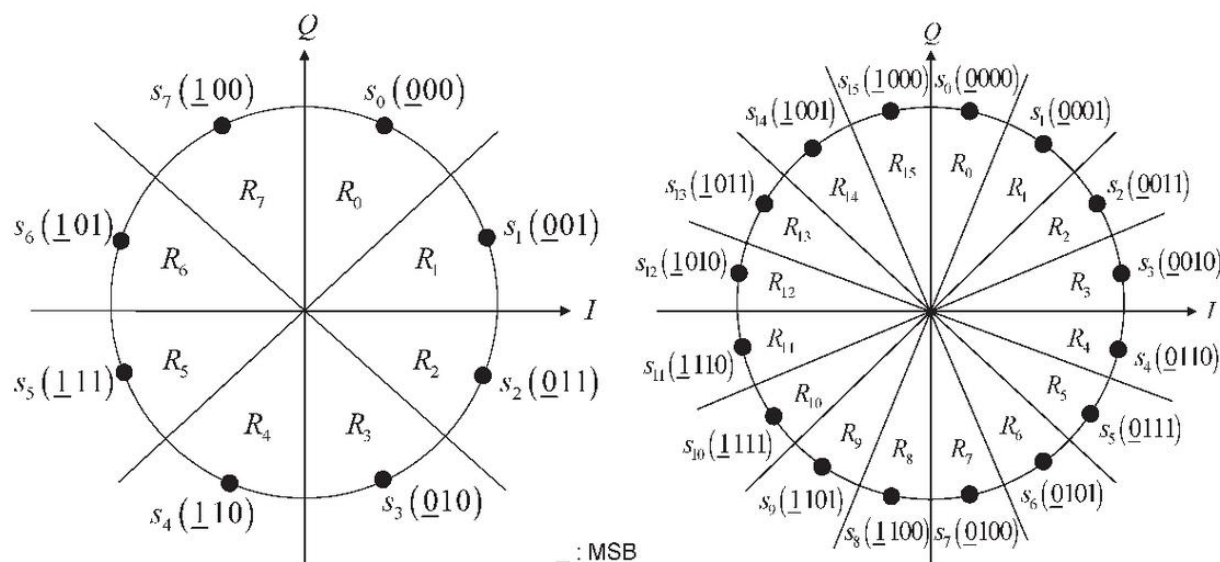
Εικόνα 1. Κλασσικό σύστημα PSK

Παρατηρούμε ότι η κλασσική μέθοδος έχει 2 επίπεδα φάσης, είτε 0 είτε 180 μοίρες. Δηλαδή, ο αποφασιστής είναι σχετικά απλός καθώς έχει μεγάλο περιθώριο για πιθανό λάθος λόγω της μεγάλης περιοχής μοιρών.

Συνεχίζοντας, όταν μιλάμε για M-PSK, το «M» αντιπροσωπεύει τον αριθμό των διαφορετικών διαθέσιμων μετατοπίσεων φάσης ή συμβόλων. Πλέον είναι αναγκαίος ο κύκλος των λεγόμενων αστερισμών, ώστε να διαχωρίσουμε τις μοίρες με βάση τον αριθμό των διαφορετικών φάσεων. Επίσης, το M δείχνει ποσά bit χρησιμοποιούνται

για την κωδικοποίηση των συμβολών (είναι $2^k \text{bits} = M \text{ symbols}$). Για παράδειγμα, η 8-PSK έχει οκτώ πιθανές καταστάσεις φάσης και 3 bit/σύμβολο, ενώ η 16-PSK έχει δεκαέξι καταστάσεις φάσης και 4bit/σύμβολο. Ο αριθμός των bits ανά σύμβολο αυξάνεται με τον αριθμό των μετατοπίσεων φάσης, με κάθε φάση να αντιπροσωπεύει έναν συγκεκριμένο συνδυασμό bits.

Το σύστημα M-PSK λειτουργεί με βάση την αρχή της διαίρεσης του κύκλου φάσης σε M διακριτές γωνίες φάσης. Κάθε γωνία αντιστοιχεί σε ένα μοναδικό σύμβολο. Για παράδειγμα, στο 8-PSK, ο κύκλος φάσης διαιρείται σε 8 ισαπέχοντα τμήματα των 45 μοιρών το καθένα, ενώ στο 16-PSK διαιρείται σε 16 τμήματα των 22,5 μοιρών το καθένα. Σε κάθε περίοδο συμβόλου, επιλέγεται μια συγκεκριμένη γωνία φάσης, που αντιστοιχεί στην ομάδα bit που μεταδίδεται.



Εικόνα 2. Κύκλος αστερισμών M-PSK

Τα πλεονεκτήματα του M-psk έναντι του κλασσικού είναι η καλύτερη απόδοση αποστολής δεδομένων, καθώς έχουμε περισσότερα bit/σύμβολο (δηλαδή αξιοποιεί καλύτερα το διαθέσιμο bandwidth). Όμως, όσο αυξάνουμε το M τόσο μεγαλώνει το BER (bit error rate) επειδή τα σύμβολα (δηλαδή οι φάσεις) είναι πιο κοντά μεταξύ τους και άρα ο θόρυβος είναι πιο πιθανό να μετατοπίσει το σήμα σε διαφορετική περιοχή του κύκλου και ο αποφασιστής να λάβει λανθασμένη απόφαση.

Ερώτηση 1

A) Χωρίς κωδικοποίηση gray

Διαδική Ακολουθία Εισόδου

Αρχικά, χρειάζεται να δημιουργήσουμε την ακολουθία bit εισόδου όπως μας ζητείτε. Αυτή θα είναι τυχαίου χαρακτήρα και το κάθε σύμβολο (0 ή 1 είναι το ίδιο πιθανό να εμφανιστεί.

```
%% Generate Binary Input Sequence  
bits = randi([0, 1], Lb, 1);
```

Συγκεκριμένα, ορίζουμε ότι θα παραχθούν 100.000 bit.

```
Lb = 1e5;
```

Επιπλέον, ορίζουμε ότι ο αριθμός των bit είναι διαιρέσιμος με των αριθμό bit/σύμβολο ώστε να μην μένουν περισσότερα bit από ότι χρειαζόμαστε ή αντίστοιχα να υπάρχει έλλειψη για την κωδικοποίηση όλων των συμβολών. Το k εξηγείτε παρακάτω πως υπολογίζεται.

```
if mod(Lb, k) ~= 0  
    Lb_adjusted = floor(Lb / k) * k;  
    Lb = Lb_adjusted;  
end
```

Αντιστοιχία Bits - Συμβόλων

Καλούμαστε να ομαδοποιήσουμε τα bit που δημιουργούνται σε ομάδες των k bit ώστε να κωδικοποιήσουμε τα σύμβολα. Αυτό γίνεται αφού υπολογίσουμε το K με βάση το M (βρίσκουμε τον λογάριθμο του M με βάση το 2) και μετρά χρησιμοποιούμε την reshape για την δημιουργία ομάδων.

```
M = 16; % Modulation order (8-PSK or 16-PSK)  
k = log2(M);
```

Παρακάτω, ενώ θα ήταν πολύ απλό απλά να παίρνουμε κατευθείαν από την ακολουθία τα νούμερα και να τα αναθέτουμε στα σωστά σύμβολα (πχ 0011 για 4-PSK παίρνουμε το 00 και το βάζουμε στο 00 και το 11 στο 11 απευθείας) προτιμάμε να τα μετατρέψουμε πρώτα τις ομάδες σε δεκαδικά νούμερα (πχ για 16-PSK έχουμε τιμές από 0-15) και μετά με βάση αυτές να τοποθετήσουμε την ομάδα στο σωστό σύμβολο.

```
binary_symbols = bi2de(bit_groups, 'left-msb');  
theta_m = 2 * pi * m / M;
```

Ορθογώνιος Παλμός

Αρχικοποιούμε όλα τα στοιχεία της εκφώνησης όπως ακριβώς δίνονται.

```
Rs = 250e3; % Symbol rate (symbols per second)
Tsym = 1 / Rs; % Symbol period (seconds)
Fs = 10e6; % Sampling frequency (Hz)
Tsample = 1 / Fs; % Sampling period (seconds)
Ns = Tsym / Tsample; % Number of samples per symbol
Ns = round(Ns); % Ensure Ns is an integer
```

Παράγουμε τον ορθογώνιο παλμό όπου αυτός εφαρμόζεται στις ενδοφασικές (I) και τετραγωνικές (Q) συνιστώσες του σήματος. Οι συνιστώσες αυτές αντιπροσωπεύουν τα δύο ορθογώνια μέρη του σήματος που μεταφέρουν τις πληροφορίες φάσης και πλάτους των συμβόλων.

```
pulse_shape = sqrt(2 / Ns) * ones(1, Ns); % Rectangular pulse with
unit energy
for idx = 1:num_symbols
    idx_range = (idx-1)*Ns + 1 : idx*Ns;
    m = binary_symbols(idx); % No Gray coding applied
    theta_m = 2 * pi * m / M;
    I_baseband_total(idx_range) = cos(theta_m) * pulse_shape;
    Q_baseband_total(idx_range) = sin(theta_m) * pulse_shape;
end
```

Διαμόρφωση M-PSK

Μετά την κατασκευή των συνιστωσών I/Q του σήματος βασικής ζώνης, το επόμενο βήμα είναι η διαμόρφωσή τους σε ένα φέρον κύμα υψηλής συχνότητας. Αυτό είναι απαραίτητο επειδή το ίδιο το σήμα βασικής ζώνης είναι χαμηλής συχνότητας και πρέπει να μετατοπιστεί σε υψηλότερη συχνότητα για μετάδοση. Η διαμόρφωση του φέροντος γίνεται με τη χρήση των συνημιτονικών και ημιτονικών κυμάτων της φέρουσας συχνότητας F_c .

```
carrier_cos_total = cos(2 * pi *  $F_c$  * t_total);
carrier_sin_total = sin(2 * pi *  $F_c$  * t_total);
```

Αφού καθοριστούν τα φέροντα σήματα, οι ενδοφασικές και τετραγωνικές συνιστώσες διαμορφώνονται στο φέρον.

```
s_t = I_baseband_total .* carrier_cos_total + Q_baseband_total .*
carrier_sin_total;
```

AWGN Κανάλι

Ορίζουμε την επιθυμητή τιμή SNR

```
SNR_dB = 20;  
SNR = 10^(SNR_dB / 10);
```

και προχωράμε στον υπολογισμό του θορύβου. Η φασματική πυκνότητα θορύβου (N_0) είναι ο λόγος της ενέργειας ανά bit (E_b) προς το SNR, δηλαδή αντιπροσωπεύει πόση ισχύς θορύβου υπάρχει στο κανάλι σε σχέση με την ενέργεια του bit. Επιπλέον, η διακύμανση του θορύβου (σ^2) υπολογίζεται διαιρώντας τη φασματική πυκνότητα του θορύβου με το 2. Αυτό συμβαίνει επειδή ο θόρυβος επηρεάζει τόσο τις πραγματικές όσο και τις φανταστικές συνιστώσες του σήματος (καθώς το σήμα είναι σύνθετο σε αυτή την περίπτωση λόγω της διαμόρφωσης).

```
N0 = Eb / SNR; % Noise spectral density  
sigma2 = N0 / 2; % Noise variance (for real and imaginary components)
```

Τα πραγματικά δείγματα θορύβου παράγονται χρησιμοποιώντας μια κανονική (γκουσσιανή) κατανομή, όπου η διακύμανση του θορύβου καθορίζει τη διασπορά των τιμών.

```
noise = sqrt(sigma2) * randn(1, length(s_t)); % Generate noise  
r_t = s_t + noise; % Add noise to the transmitted signal
```

Αποδιαμορφωτής M-PSK

Η αποδιαμόρφωση αρχίζει με το διαχωρισμό των συνιστωσών I (συμφασική) και Q (τετραγωνική) του λαμβανόμενου σήματος, χρησιμοποιώντας τις ίδιες φέρουσες συχνότητες που χρησιμοποιήθηκαν στη διαδικασία διαμόρφωσης.

```
r_I = r_t .* carrier_cos_total;  
r_Q = r_t .* carrier_sin_total;
```

Μετά το διαχωρισμό των συνιστωσών I και Q, το επόμενο βήμα είναι η μέση τιμή των δειγμάτων κατά τη διάρκεια κάθε συμβόλου. Το μεταδιδόμενο σήμα δειγματοληπτήθηκε πολλές φορές ανά σύμβολο (με N_s δείγματα/σύμβολο), και τώρα ο αποδιαμορφωτής υπολογίζει το μέσο όρο αυτών των δειγμάτων για να πάρει τις τελικές τιμές I και Q που αντιπροσωπεύουν το κάθε σύμβολο.

```
for idx = 1:num_symbols  
    idx_range = (idx-1)*Ns + 1 : idx*Ns;  
    I_r = sum(r_I(idx_range)) / Ns; % Normalize by Ns to average the  
values  
    Q_r = sum(r_Q(idx_range)) / Ns; % Normalize by Ns to average the  
values
```

```

received_symbols(idx, :) = [I_r, Q_r]; % Store the received I/Q
symbols
end

```

Φωρατής M-PSK

Ο αποδιαμορφωτής συγκρίνει τις λαμβανόμενες τιμές I/Q με τα σημεία αστερισμού (τις ιδανικές τιμές I/Q για κάθε σύμβολο) και επιλέγει το πλησιέστερο σημείο αστερισμού. Η διαδικασία αυτή γίνεται με τη χρήση ενός κριτηρίου ελάχιστης απόστασης.

```

for idx = 1:num_symbols
    r = received_symbols(idx, :);
    distances = sum((constellation - r).^2, 2);
    [~, min_idx] = min(distances);
    detected_symbols(idx) = min_idx - 1;
end

```

Αφού ανιχνευθούν τα σύμβολα, το τελικό βήμα είναι η αντιστοίχιση τους στις αντίστοιχες ακολουθίες bit. Αυτό είναι το αντίστροφο της διαδικασίας που πραγματοποιείται κατά τη διαμόρφωση.

```

binary_symbols_detected = detected_symbols; % No Gray-to-binary
conversion needed
detected_bits_matrix = de2bi(binary_symbols_detected, k, 'left-msb');
estimated_bits = reshape(detected_bits_matrix.', [], 1);

```

B) Με κωδικοποίηση gray

Τα μόνα τμήματα που αλλάζουν είναι η διαδικασία αντιστοίχισης bit σε σύμβολα.

```

binary_symbols = bi2de(bit_groups, 'left-msb'); % Convert to decimal
symbols
gray_symbols = bitxor(binary_symbols, floor(binary_symbols / 2)); %
Gray code

```

Αντίστοιχα αλλάζουμε όλα τα τμήματα που περιέχουν τα σύμβολα με τα gray symbols.

Ερώτηση 2

A) Χωρίς κωδικοποίηση gray

Μετά τη διαμόρφωση, το σήμα βασικής ζώνης διαμορφώνεται σε μια φέρουσα συχνότητα και μεταδίδεται. Τα λαμβανόμενα σύμβολα υπολογίζονται μετά την αποδιαμόρφωση και οι συνιστώσες I/Q συγκρίνονται με τα ιδανικά σημεία αστερισμού για την ανίχνευση των μεταδιδόμενων συμβόλων. Στο διάγραμμα αστερισμού, τα λαμβανόμενα σύμβολα (οι θορυβώδεις εκδοχές των μεταδιδόμενων συμβόλων) απεικονίζονται ως κόκκινα σημεία («r.»). Τα σημεία αυτά συμπίπτουν πλήρως με τα ιδανικά μόνο για πολύ μεγάλο SNR (άνω των 40db), ενώ σε άλλες περιπτώσεις διακρίνουμε ότι τα κόκκινα σημεία είναι μεγαλύτερα από τα μπλε λόγω του θορύβου (ο αποφασιστής κάνει λάθη και τα τοποθετεί σε διαφορετικά σημεία).

```
constellation = [cos(theta).' sin(theta).']; % Ideal constellation
figure;
plot(received_symbols(:,1), received_symbols(:,2), 'r.');
```

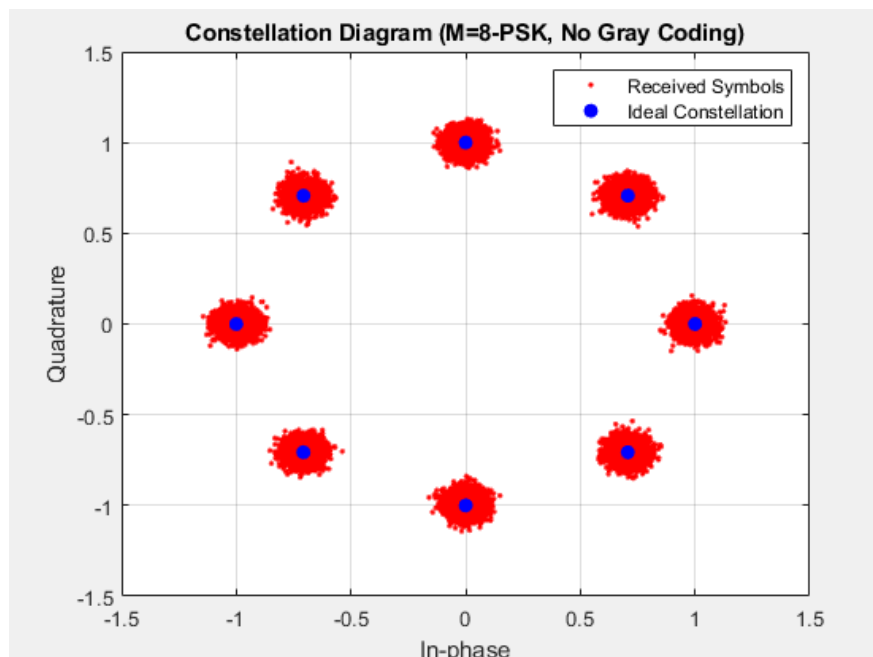
```
hold on;
plot(constellation(:,1), constellation(:,2), 'bo', 'MarkerFaceColor',
'b');
```

```
xlabel('In-phase');
ylabel('Quadrature');
```

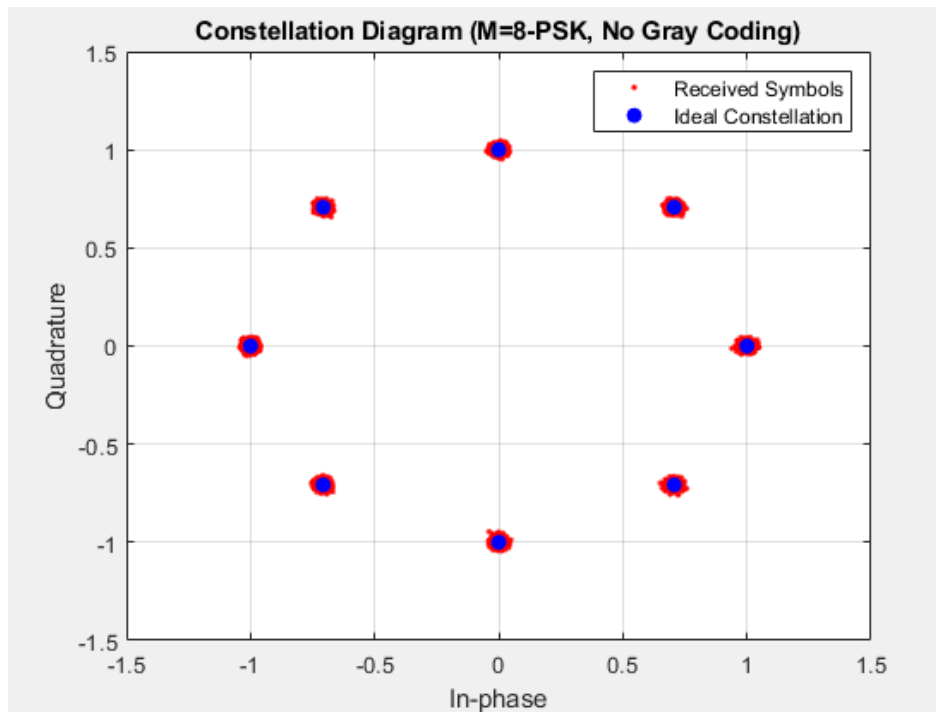
```
title(sprintf('Constellation Diagram (M=%d-PSK, No Gray Coding)', M));
legend('Received Symbols', 'Ideal Constellation');
```

```
grid on;
```

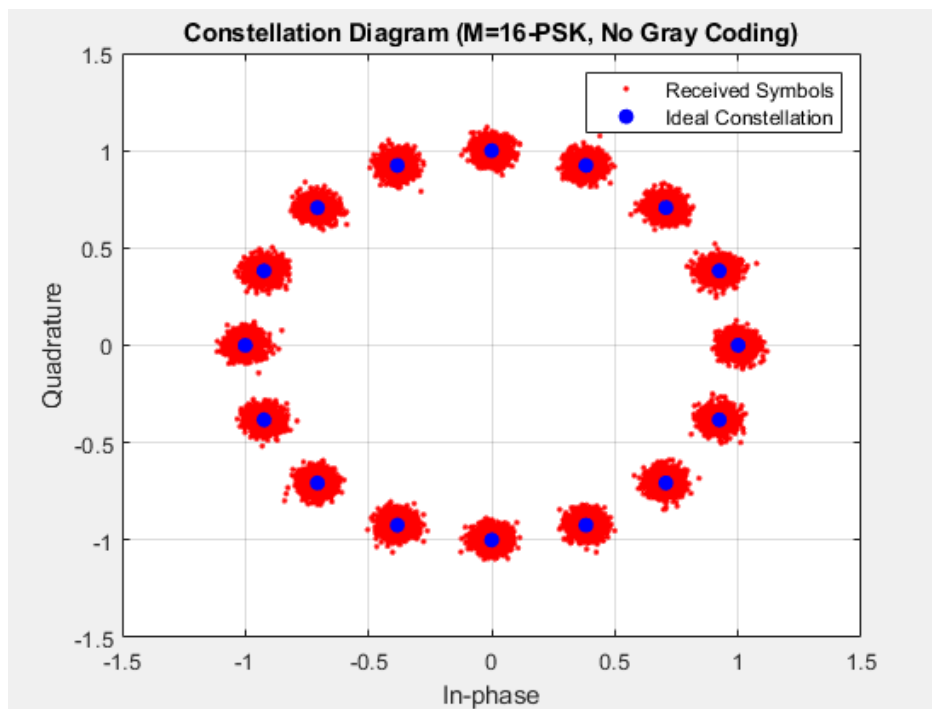
M=8: SNR 20 db



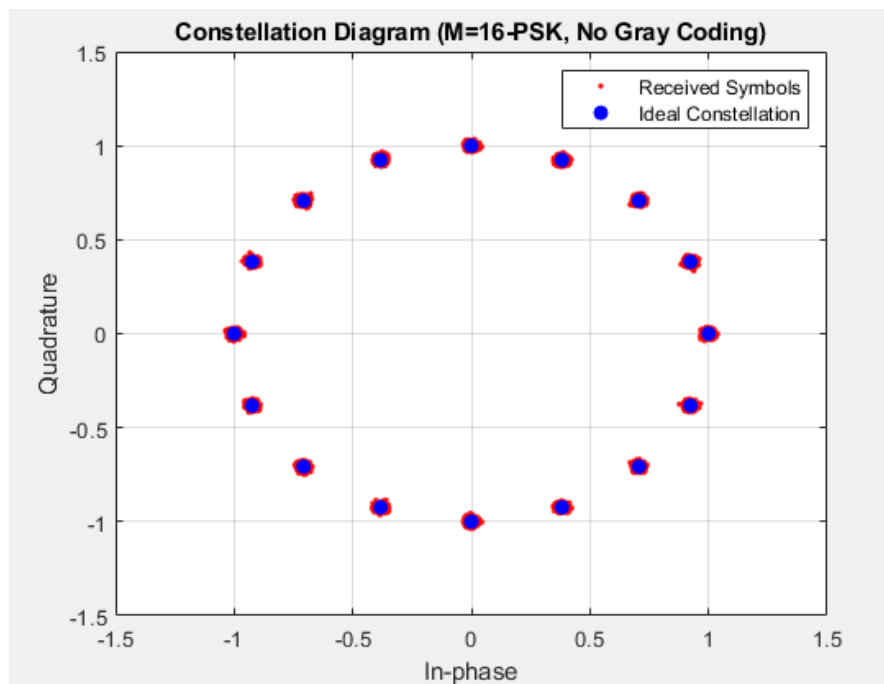
SNR 30 db



M=16: SNR 20 db

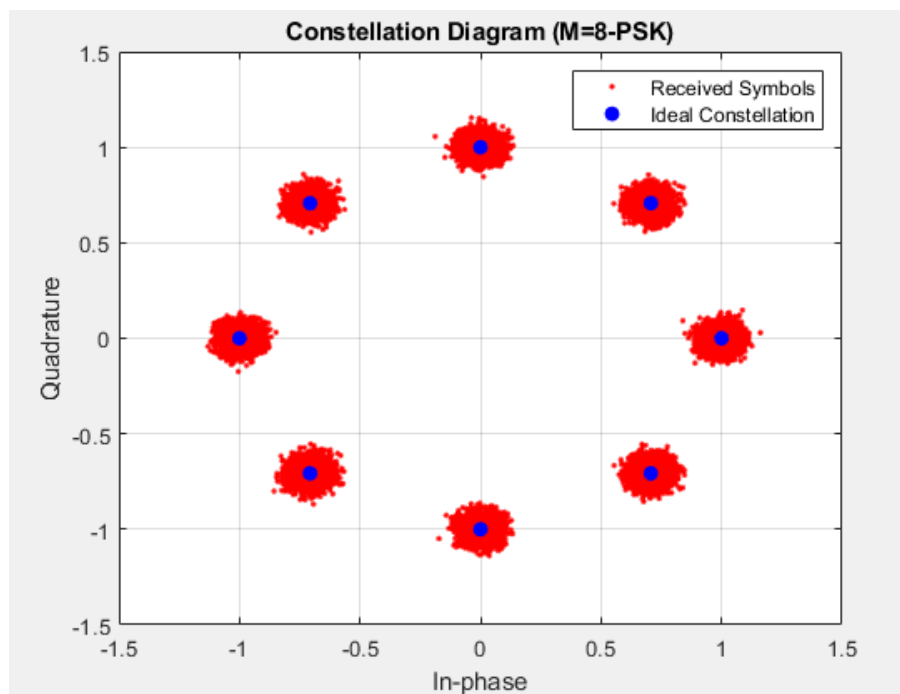


SNR 30db

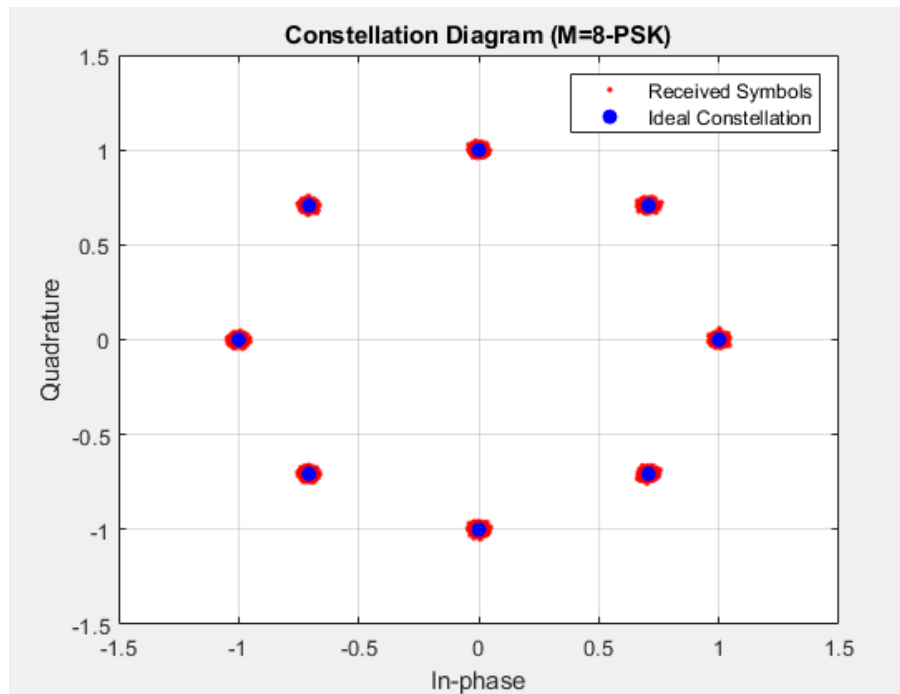


Β) Με κωδικοποίηση gray

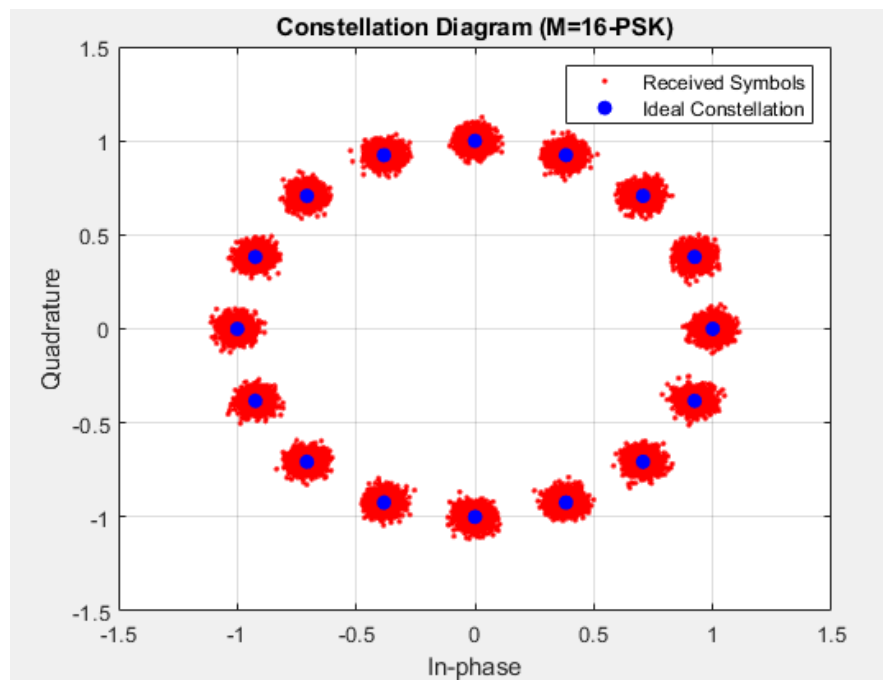
M=8: SNR 20 db



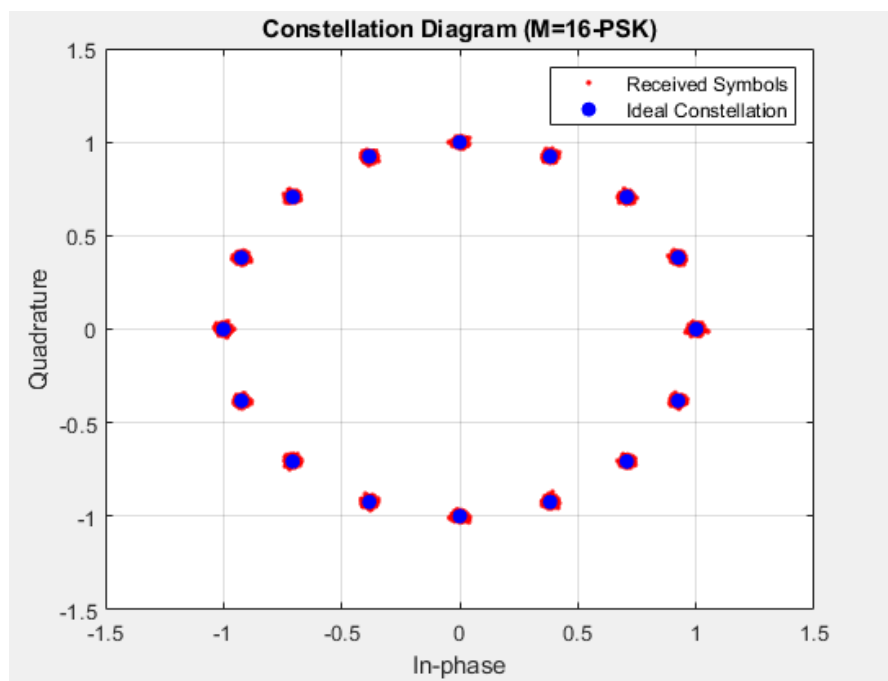
SNR 30 db



M=16: SNR 20 db



SNR 30 db



Ερώτηση 3

Σε ένα σύστημα M-PSK, η χρήση της κωδικοποίησης Gray παίζει καθοριστικό ρόλο στην ελαχιστοποίηση του BER. Η κωδικοποίηση αυτή είναι ένα δυαδικό σύστημα κωδικοποίησης όπου δύο διαδοχικά σύμβολα διαφέρουν μόνο κατά ένα bit. Η θεμελιώδης ιδέα πίσω από τη διαμόρφωση M-PSK είναι η αντιστοίχιση πολλαπλών bits σε διακριτά σύμβολα, όπου κάθε σύμβολο αντιστοιχεί σε μια συγκεκριμένη γωνία φάσης. Σε τέτοια συστήματα, ο θόρυβος ή οι παραμορφώσεις του σήματος κατά τη μετάδοση μπορούν να προκαλέσουν απόκλιση ενός λαμβανόμενου συμβόλου από την αρχική του φάση, οδηγώντας σε σφάλματα ανίχνευσης. Όταν συμβαίνει αυτό, το σύστημα μπορεί να ανιχνεύσει ένα γειτονικό σύμβολο αντί για το μεταδιδόμενο. Χωρίς κωδικοποίηση Gray, τα γειτονικά σύμβολα στο διάγραμμα αστερισμού θα μπορούσαν να έχουν εσφαλμένη αποκωδικοποίηση περισσότερων του ενός bit. Για παράδειγμα, σε ένα σύστημα 8-PSK, χωρίς κωδικοποίηση Gray, τα γειτονικά σύμβολα μπορεί να διαφέρουν κατά δύο ή περισσότερα bit. Εάν ένα σύμβολο ανιχνευθεί ως γειτονικό του λόγω θορύβου, το αποτέλεσμα θα μπορούσε να είναι ένα σφάλμα πολλών bit, αυξάνοντας σημαντικά Ber. Ωστόσο, όταν εφαρμόζεται κωδικοποίηση Gray, κάθε γειτονικό σύμβολο στον αστερισμό φάσης διαφέρει μόνο κατά ένα bit. Αυτό σημαίνει ότι εάν ο θόρυβος ή η

παρεμβολή αναγκάζει τον δέκτη να ανιχνεύσει ένα γειτονικό σύμβολο αντί για το σωστό, το σφάλμα που προκύπτει θα επηρεάσει μόνο ένα bit. Με τη μείωση των σφαλμάτων πολλαπλών bit σε σφάλματα ενός bit, η κωδικοποίηση Gray ελαχιστοποιεί την επίδραση του θορύβου, βελτιώνοντας έτσι την απόδοση του συστήματος. Επιπλέον, η κωδικοποίηση Gray είναι ιδιαίτερα επωφελής σε συστήματα M-PSK υψηλής τάξης (π.χ. 16-PSK ή 64-PSK) επειδή τα σημεία αστερισμού είναι πιο κοντά μεταξύ τους καθώς αυξάνεται η τάξη διαμόρφωσης. Σε τέτοια συστήματα, ακόμη και μια μικρή ποσότητα θορύβου μπορεί να προκαλέσει την εσφαλμένη ερμηνεία ενός συμβόλου ως ένα από τα γειτονικά του.

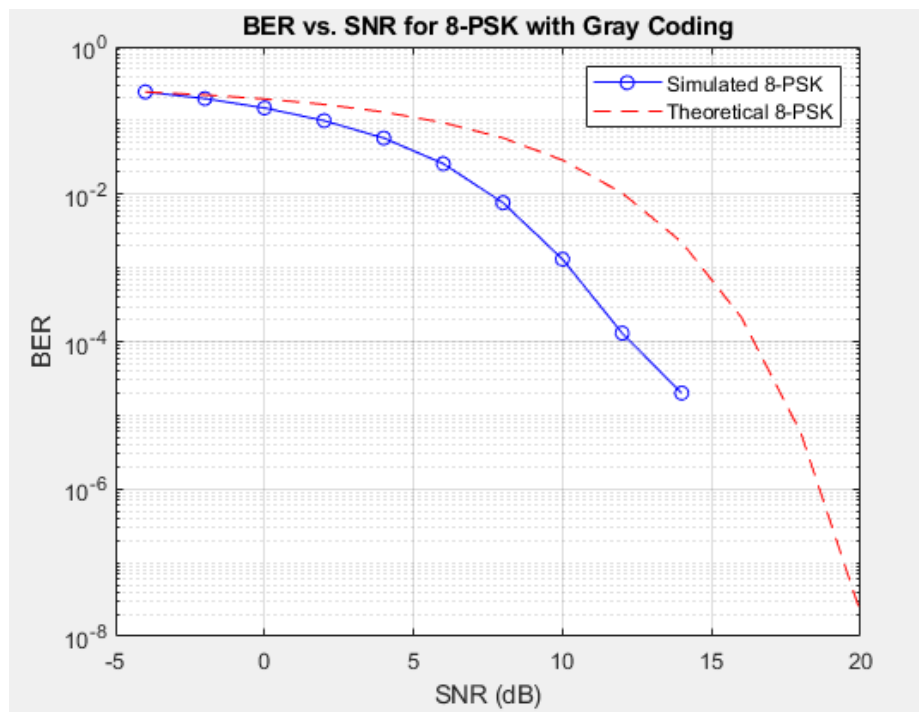
Ερώτηση 4

Μετά τη μετάδοση των συμβόλων μέσω του καναλιού, αποκωδικοποιούμε το λαμβανόμενο σήμα και το BER υπολογίζεται συγκρίνοντας τα εκτιμώμενα bit με τα αρχικά μεταδιδόμενα bit. Για κάθε τιμή SNR, υπολογίζεται το BER και αποθηκεύεται σε έναν πίνακα.

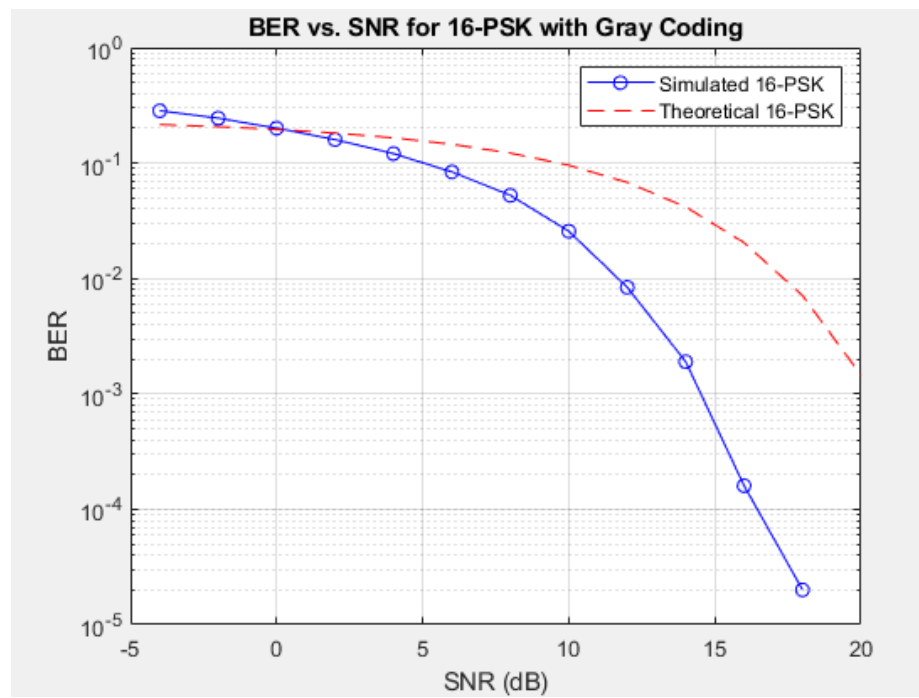
```
% Calculate Bit Error Rate (BER)
binary_symbols_detected = gray2binary(detected_symbols, k);
detected_bits_matrix = de2bi(binary_symbols_detected, k, 'left-
msb');
estimated_bits = reshape(detected_bits_matrix.', [], 1);
estimated_bits = estimated_bits(1:Lb);
num_errors = sum(bits ~= estimated_bits);
BER_simulated(snr_idx) = num_errors / Lb;

% Theoretical BER for M-PSK
BER_theoretical(snr_idx) = (2/k) * qfunc(sqrt(2*SNR) * sin(pi/M));
```

M=8:



M=16:



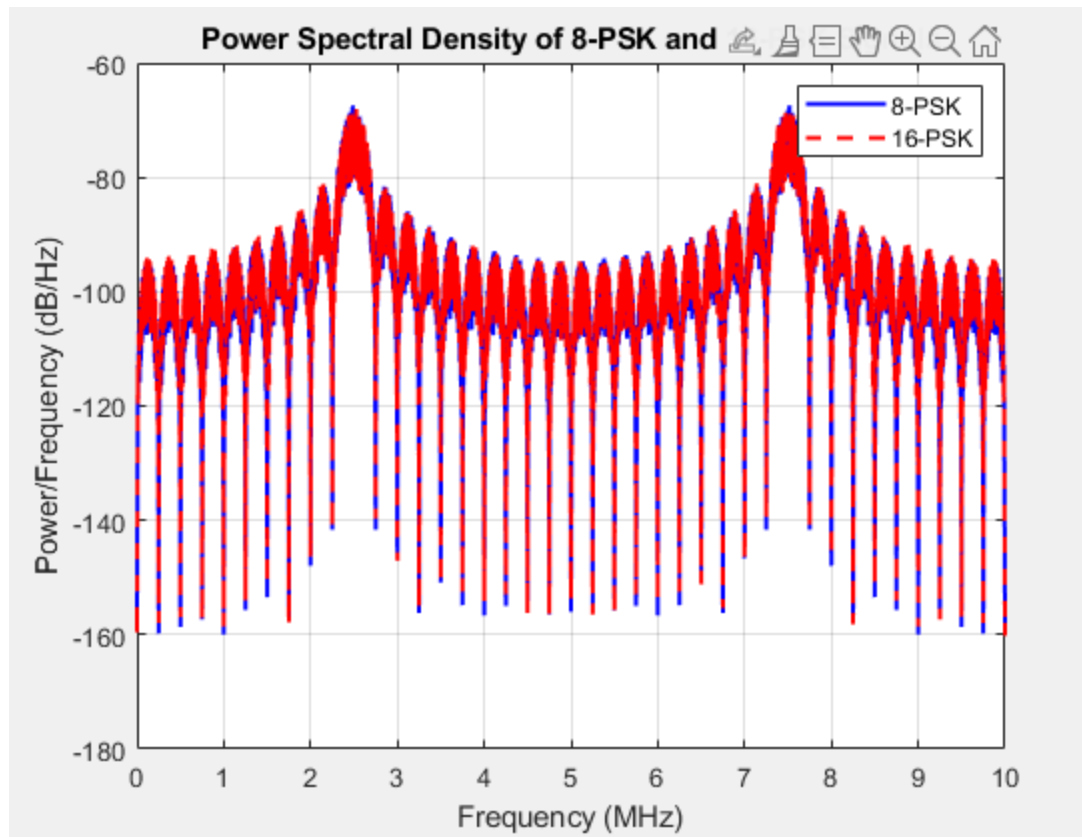
Όπως αναμέναμε και από την θεωρία, το 8-PSK έχει χαμηλότερο ber για το ίδιο SNR. Το 16-PSK χρειάζεται 3 db παραπάνω για να πέτυχει την ίδια πιθανότητα σφάλματος.

Ερώτηση 5

Αρχικά, μετά τη διαμόρφωση του σήματος, χρησιμοποιείται η συνάρτηση `pwelch` για τον υπολογισμό της φασματικής πυκνότητας ισχύος (PSD) του μεταδιδόμενου σήματος. Δηλαδή, βγάζει τις τιμές PSD σε dB/Hz και τις αντίστοιχες τιμές συχνότητας.

```
%% Power Spectral Density Calculation using pwelch
[psd, f] = pwelch(s_t, [], [], [], Fs, 'twosided'); % Estimate the
power spectrum
psd_dB = 10*log10(psd); % Convert to dB scale
```

Μια βασική παρατήρηση είναι το εύρος ζώνης που απαιτείται για κάθε σχήμα διαμόρφωσης. Τα σχήματα διαμόρφωσης υψηλότερης τάξης, όπως το 16-PSK, απαιτούν συνήθως μεγαλύτερο εύρος ζώνης από τα μικρότερα, όπως το 8-PSK. Αυτό οφείλεται στο γεγονός ότι έχουν περισσότερα σύμβολα προς αναπαράσταση, γεγονός που οδηγεί σε πυκνότερο αστερισμό συμβόλων και κατά συνέπεια, σε ευρύτερο φάσμα. Ένα άλλο σημαντικό χαρακτηριστικό είναι ο κύριος λοβός και οι πλευρικοί λοβοί. Ο κύριος αντιπροσωπεύει το πρωτογενές συχνοτικό περιεχόμενο του σήματος, ενώ οι πλευρικοί λοβοί περιέχουν δευτερογενείς φασματικές συνιστώσες. Γενικά, καθώς αυξάνεται η τάξη διαμόρφωσης, το πλάτος του κύριου λοβού μπορεί να διευρυνθεί και οι πλευρικοί λοβοί μπορεί να γίνουν πιο έντονοι. Επιπλέον, η κατανομή ισχύος διαφέρει μεταξύ των δύο συστημάτων, με το 16-PSK να παρουσιάζει γενικά ένα ελαφρώς πιο απλωμένο φάσμα ισχύος λόγω του μεγαλύτερου αριθμού πιθανών συμβόλων που πρέπει να μεταδοθούν. Το 8-PSK, ως απλούστερο σχήμα διαμόρφωσης, τείνει να έχει πιο συγκεντρωμένη κατανομή ισχύος, η οποία αντανακλάται σε στενότερη φασματική διασπορά.



Κώδικας

1&2 χωρίς Gray)

```
% M-PSK System Simulation without Gray Coding
clear all; close all; clc;

%% Parameters
M = 16; % Modulation order (8-PSK or 16-PSK)
k = log2(M); % Bits per symbol
Lb = 1e5; % Total number of bits

% Adjust Lb to be a multiple of k
if mod(Lb, k) ~= 0
    Lb_adjusted = floor(Lb / k) * k;
    Lb = Lb_adjusted;
end

Es = 1; % Symbol energy (normalized to 1)
Eb = Es / k; % Energy per bit
Rs = 250e3; % Symbol rate (symbols per second)
Tsym = 1 / Rs; % Symbol period (4 μs)
Fc = 2.5e6; % Carrier frequency (Hz)
Fs = 10e6; % Sampling frequency (Hz)
Tsample = 1 / Fs; % Sampling period (seconds)
Ns = Tsym / Tsample; % Number of samples per symbol
Ns = round(Ns); % Ensure Ns is an integer

% Display parameters for verification
fprintf('Sampling Frequency Fs = %.2f MHz\n', Fs / 1e6);
fprintf('Symbol Period Tsym = %.2f us\n', Tsym * 1e6);
fprintf('Number of Samples per Symbol Ns = %d\n', Ns);

%% Generate Binary Input Sequence
bits = randi([0, 1], Lb, 1);

%% Map Bits to Symbols (WITHOUT Gray Coding)
bit_groups = reshape(bits, k, []).'; % Reshape bits into k-bit
symbols
binary_symbols = bi2de(bit_groups, 'left-msb'); % Convert to decimal
symbols (no Gray coding)

%% Generate Baseband Signal
pulse_shape = sqrt(2 / Ns) * ones(1, Ns); % Rectangular pulse with
unit energy
num_symbols = length(binary_symbols);
I_baseband_total = zeros(1, num_symbols * Ns);
Q_baseband_total = zeros(1, num_symbols * Ns);
```

```

for idx = 1:num_symbols
    idx_range = (idx-1)*Ns + 1 : idx*Ns;
    m = binary_symbols(idx); % No Gray coding applied
    theta_m = 2 * pi * m / M;
    I_baseband_total(idx_range) = cos(theta_m) * pulse_shape;
    Q_baseband_total(idx_range) = sin(theta_m) * pulse_shape;
end

%% Modulate Baseband Signal onto Carrier (Corrected)
t_total = (0:(num_symbols * Ns - 1)) * Tsample; % Total time vector
carrier_cos_total = cos(2 * pi * Fc * t_total);
carrier_sin_total = sin(2 * pi * Fc * t_total);

% Modulate onto carrier (corrected sign)
s_t = I_baseband_total .* carrier_cos_total + Q_baseband_total .*
carrier_sin_total;

%% Add AWGN Noise
SNR_dB = 30;
SNR = 10^(SNR_dB / 10);
N0 = Eb / SNR;
sigma2 = N0 / 2;
noise = sqrt(sigma2) * randn(1, length(s_t)); % Generate noise
r_t = s_t + noise; % Add noise to the transmitted signal

%% Demodulate Received Signal (Corrected Scaling)
r_I = r_t .* carrier_cos_total;
r_Q = r_t .* carrier_sin_total;

received_symbols = zeros(num_symbols, 2);

for idx = 1:num_symbols
    idx_range = (idx-1)*Ns + 1 : idx*Ns;
    I_r = sum(r_I(idx_range)) / Ns; % Normalize by Ns to average the
values
    Q_r = sum(r_Q(idx_range)) / Ns; % Normalize by Ns to average the
values
    received_symbols(idx, :) = [I_r, Q_r]; % Store the received I/Q
symbols
end

%% Adjust Scaling of Received Symbols (Match Ideal Constellation)
received_symbols = received_symbols /
sqrt(mean(sum(received_symbols.^2, 2))); % Normalize symbol energy

%% Symbol Detection (Decision Device)
theta = (0:M-1) * (2 * pi / M);

```

```

constellation = [cos(theta).' sin(theta).']; % Ideal constellation

detected_symbols = zeros(num_symbols, 1);
for idx = 1:num_symbols
    r = received_symbols(idx, :);
    distances = sum((constellation - r).^2, 2);
    [~, min_idx] = min(distances);
    detected_symbols(idx) = min_idx - 1;
end

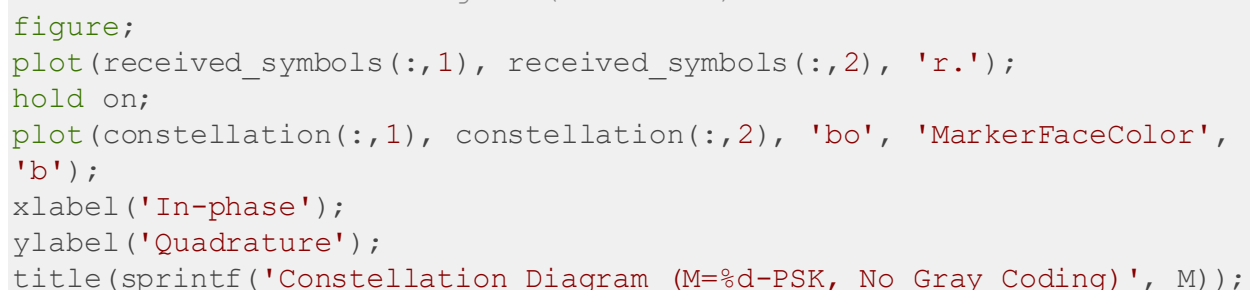
%% Phase Debugging: Check Phase Offset
received_phase = atan2(received_symbols(:, 2), received_symbols(:, 1)); % Phase of received symbols
ideal_phase = theta(binary_symbols + 1)'; % Ideal phase (non-Gray coded symbols)
phase_offset = received_phase - ideal_phase;

% Adjust phase_offset to be within [-pi, pi]
phase_offset = mod(phase_offset + pi, 2*pi) - pi;

% Debug output for phase comparison
fprintf('Phase comparison (first 5 symbols):\n');
for i = 1:5
    fprintf('Symbol %d: Ideal Phase = %f, Received Phase = %f, Phase Offset = %f\n', ...
        i, ideal_phase(i), received_phase(i), phase_offset(i));
end

%% Calculate Bit Error Rate (BER) -- Kept Untouched
binary_symbols_detected = detected_symbols; % No Gray-to-binary conversion needed
detected_bits_matrix = de2bi(binary_symbols_detected, k, 'left-msb');
estimated_bits = reshape(detected_bits_matrix.', [], 1);
estimated_bits = estimated_bits(1:Lb);
num_errors = sum(bits ~= estimated_bits);
BER = num_errors / Lb;
fprintf('BER at SNR = %d dB: %e\n', SNR_dB, BER);

%% Plot Constellation Diagram (Corrected)
figure;
plot(received_symbols(:,1), received_symbols(:,2), 'r.');
```



```

hold on;
plot(constellation(:,1), constellation(:,2), 'bo', 'MarkerFaceColor', 'b');
xlabel('In-phase');
ylabel('Quadrature');
title(sprintf('Constellation Diagram (M=%d-PSK, No Gray Coding)', M));

```

```

legend('Received Symbols', 'Ideal Constellation');
grid on;

```

1&2 $\mu\epsilon$ Gray)

```

% M-PSK System Simulation with Phase Debugging
clear all; close all; clc;

%% Parameters
M = 16; % Modulation order (8-PSK or 16-PSK)
k = log2(M); % Bits per symbol
Lb = 1e5; % Total number of bits

% Adjust Lb to be a multiple of k
if mod(Lb, k) ~= 0
    Lb_adjusted = floor(Lb / k) * k;
    Lb = Lb_adjusted;
end

Es = 1; % Symbol energy (normalized to 1)
Eb = Es / k; % Energy per bit
Rs = 250e3; % Symbol rate (symbols per second)
Tsym = 1 / Rs; % Symbol period (4  $\mu$ s)
Fc = 2.5e6; % Carrier frequency (Hz)
Fs = 10e6; % Sampling frequency (Hz)
Tsample = 1 / Fs; % Sampling period (seconds)
Ns = Tsym / Tsample; % Number of samples per symbol
Ns = round(Ns); % Ensure Ns is an integer

% Display parameters for verification
fprintf('Sampling Frequency Fs = %.2f MHz\n', Fs / 1e6);
fprintf('Symbol Period Tsym = %.2f  $\mu$ s\n', Tsym * 1e6);
fprintf('Number of Samples per Symbol Ns = %d\n', Ns);

%% Generate Binary Input Sequence
bits = randi([0, 1], Lb, 1);

%% Map Bits to Symbols (Gray Coding)
bit_groups = reshape(bits, k, []).'; % Reshape bits into k-bit
symbols
binary_symbols = bi2de(bit_groups, 'left-msb'); % Convert to decimal
symbols
gray_symbols = bitxor(binary_symbols, floor(binary_symbols / 2)); %
Gray code

```

```

%% Generate Baseband Signal
pulse_shape = sqrt(2 / Ns) * ones(1, Ns); % Rectangular pulse with
unit energy
num_symbols = length(gray_symbols);
I_baseband_total = zeros(1, num_symbols * Ns);
Q_baseband_total = zeros(1, num_symbols * Ns);

for idx = 1:num_symbols
    idx_range = (idx-1)*Ns + 1 : idx*Ns;
    m = gray_symbols(idx);
    theta_m = 2 * pi * m / M;
    I_baseband_total(idx_range) = cos(theta_m) * pulse_shape;
    Q_baseband_total(idx_range) = sin(theta_m) * pulse_shape;
end

%% Modulate Baseband Signal onto Carrier (Corrected)
t_total = (0:(num_symbols * Ns - 1)) * Tsample; % Total time vector
carrier_cos_total = cos(2 * pi * Fc * t_total);
carrier_sin_total = sin(2 * pi * Fc * t_total);

% Modulate onto carrier (corrected sign)
s_t = I_baseband_total .* carrier_cos_total + Q_baseband_total .*
carrier_sin_total;

%% Add AWGN Noise
SNR_dB = 30;
SNR = 10^(SNR_dB / 10);
N0 = Eb / SNR;
sigma2 = N0 / 2;
noise = sqrt(sigma2) * randn(1, length(s_t)); % Generate noise
r_t = s_t + noise; % Add noise to the transmitted signal

%% Demodulate Received Signal (Corrected Scaling)
r_I = r_t .* carrier_cos_total;
r_Q = r_t .* carrier_sin_total;

received_symbols = zeros(num_symbols, 2);

for idx = 1:num_symbols
    idx_range = (idx-1)*Ns + 1 : idx*Ns;
    I_r = sum(r_I(idx_range)) / Ns; % Normalize by Ns to average the
values
    Q_r = sum(r_Q(idx_range)) / Ns; % Normalize by Ns to average the
values
    received_symbols(idx, :) = [I_r, Q_r]; % Store the received I/Q
symbols
end

```

```

%% Adjust Scaling of Received Symbols (Match Ideal Constellation)
received_symbols = received_symbols /
sqrt(mean(sum(received_symbols.^2, 2))); % Normalize symbol energy

%% Symbol Detection (Decision Device)
theta = (0:M-1) * (2 * pi / M);
constellation = [cos(theta).' sin(theta).']; % Ideal constellation

detected_symbols = zeros(num_symbols, 1);
for idx = 1:num_symbols
    r = received_symbols(idx, :);
    distances = sum((constellation - r).^2, 2);
    [~, min_idx] = min(distances);
    detected_symbols(idx) = min_idx - 1;
end

%% Phase Debugging: Check Phase Offset
received_phase = atan2(received_symbols(:, 2), received_symbols(:,
1)); % Phase of received symbols
ideal_phase = theta(gray_symbols + 1)'; % Ideal phase from Gray-coded
symbols
phase_offset = received_phase - ideal_phase;

% Adjust phase_offset to be within [-pi, pi]
phase_offset = mod(phase_offset + pi, 2*pi) - pi;

% Debug output for phase comparison
fprintf('Phase comparison (first 5 symbols):\n');
for i = 1:5
    fprintf('Symbol %d: Ideal Phase = %f, Received Phase = %f, Phase
Offset = %f\n', ...
        i, ideal_phase(i), received_phase(i), phase_offset(i));
end

%% Calculate Bit Error Rate (BER) -- Kept Untouched
binary_symbols_detected = gray2binary(detected_symbols, k);
detected_bits_matrix = de2bi(binary_symbols_detected, k, 'left-msb');
estimated_bits = reshape(detected_bits_matrix.', [], 1);
estimated_bits = estimated_bits(1:Lb);
num_errors = sum(bits ~= estimated_bits);
BER = num_errors / Lb;
fprintf('BER at SNR = %d dB: %e\n', SNR_dB, BER);

%% Plot Constellation Diagram (Corrected)
figure;
plot(received_symbols(:,1), received_symbols(:,2), 'r.');
```

```

'b');
xlabel('In-phase');
ylabel('Quadrature');
title(sprintf('Constellation Diagram (M=%d-PSK)', M));
legend('Received Symbols', 'Ideal Constellation');
grid on;

%% Function for Gray to Binary Conversion
function binary_symbols = gray2binary(gray_symbols, k)
    gray_bits = de2bi(gray_symbols, k, 'left-msb');
    binary_bits = zeros(size(gray_bits));
    binary_bits(:,1) = gray_bits(:,1);
    for i = 2:k
        binary_bits(:,i) = xor(binary_bits(:,i-1), gray_bits(:,i));
    end
    binary_symbols = bi2de(binary_bits, 'left-msb');
end

```

4)

```

% M-PSK System Simulation with BER Curves for SNR=[-4:2:20]dB
clear all; close all; clc;

%% Parameters
M = 16; % Set this manually to either 8 or 16 for
8-PSK or 16-PSK
k = log2(M); % Bits per symbol
Lb = 1e5; % Total number of bits
SNR_range = -4:2:20; % SNR range in dB

% Preallocate BER array
BER_simulated = zeros(1, length(SNR_range));
BER_theoretical = zeros(1, length(SNR_range));

% Adjust Lb to be a multiple of k
if mod(Lb, k) ~= 0
    Lb_adjusted = floor(Lb / k) * k;
    Lb = Lb_adjusted;
end

Es = 1; % Symbol energy (normalized to 1)
Eb = Es / k; % Energy per bit
Rs = 250e3; % Symbol rate (symbols per second)
Ts = 1 / Rs; % Symbol period (seconds)
Fc = 2.5e6; % Carrier frequency (Hz)

```

```

Fs = 10e6; % Sampling frequency (Hz)
Tsample = 1 / Fs; % Sampling period (seconds)
Ns = Tsym / Tsample; % Number of samples per symbol
Ns = round(Ns); % Ensure Ns is an integer

%% Generate Binary Input Sequence
bits = randi([0, 1], Lb, 1);

%% Map Bits to Symbols (Gray Coding)
bit_groups = reshape(bits, k, []).';
binary_symbols = bi2de(bit_groups, 'left-msb');
gray_symbols = bitxor(binary_symbols, floor(binary_symbols / 2));

%% Generate Baseband Signal
pulse_shape = sqrt(2 / Ns) * ones(1, Ns);
num_symbols = length(gray_symbols);
I_baseband_total = zeros(1, num_symbols * Ns);
Q_baseband_total = zeros(1, num_symbols * Ns);

for idx = 1:num_symbols
    idx_range = (idx-1)*Ns + 1 : idx*Ns;
    m = gray_symbols(idx);
    theta_m = 2 * pi * m / M;
    I_baseband_total(idx_range) = cos(theta_m) * pulse_shape;
    Q_baseband_total(idx_range) = sin(theta_m) * pulse_shape;
end

% Modulate Baseband Signal onto Carrier
t_total = (0:(num_symbols * Ns - 1)) * Tsample;
carrier_cos_total = cos(2 * pi * Fc * t_total);
carrier_sin_total = sin(2 * pi * Fc * t_total);
s_t = I_baseband_total .* carrier_cos_total + Q_baseband_total .* carrier_sin_total;

% Loop over different SNR values to compute BER
for snr_idx = 1:length(SNR_range)
    SNR_dB = SNR_range(snr_idx);
    SNR = 10^(SNR_dB / 10);
    N0 = Eb / SNR;
    sigma2 = N0 / 2;

    % Add AWGN Noise
    noise = sqrt(sigma2) * randn(1, length(s_t));
    r_t = s_t + noise;

    % Demodulate Received Signal
    r_I = r_t .* carrier_cos_total;
    r_Q = r_t .* carrier_sin_total;

```



```

received_symbols = zeros(num_symbols, 2);
for idx = 1:num_symbols
    idx_range = (idx-1)*Ns + 1 : idx*Ns;
    I_r = sum(r_I(idx_range)) / Ns;
    Q_r = sum(r_Q(idx_range)) / Ns;
    received_symbols(idx, :) = [I_r, Q_r];
end

% Adjust Scaling of Received Symbols
received_symbols = received_symbols /
sqrt(mean(sum(received_symbols.^2, 2)));

% Symbol Detection (Decision Device)
theta = (0:M-1) * (2 * pi / M);
constellation = [cos(theta).' sin(theta).'];
detected_symbols = zeros(num_symbols, 1);
for idx = 1:num_symbols
    r = received_symbols(idx, :);
    distances = sum((constellation - r).^2, 2);
    [~, min_idx] = min(distances);
    detected_symbols(idx) = min_idx - 1;
end

% Calculate Bit Error Rate (BER)
binary_symbols_detected = gray2binary(detected_symbols, k);
detected_bits_matrix = de2bi(binary_symbols_detected, k, 'left-
msb');
estimated_bits = reshape(detected_bits_matrix.', [], 1);
estimated_bits = estimated_bits(1:Lb);
num_errors = sum(bits ~= estimated_bits);
BER_simulated(snr_idx) = num_errors / Lb;

% Theoretical BER for M-PSK
BER_theoretical(snr_idx) = (2/k) * qfunc(sqrt(2*SNR) * sin(pi/M));
end

%% Plot BER Curves
figure;
semilogy(SNR_range, BER_simulated, 'b-o', 'DisplayName',
sprintf('Simulated %d-PSK', M));
hold on;
semilogy(SNR_range, BER_theoretical, 'r--', 'DisplayName',
sprintf('Theoretical %d-PSK', M));
xlabel('SNR (dB)');
ylabel('BER');
title(sprintf('BER vs. SNR for %d-PSK with Gray Coding', M));
legend('show');

```

```

grid on;

%% Function for Gray to Binary Conversion
function binary_symbols = gray2binary(gray_symbols, k)
    gray_bits = de2bi(gray_symbols, k, 'left-msb');
    binary_bits = zeros(size(gray_bits));
    binary_bits(:,1) = gray_bits(:,1);
    for i = 2:k
        binary_bits(:,i) = xor(binary_bits(:,i-1), gray_bits(:,i));
    end
    binary_symbols = bi2de(binary_bits, 'left-msb');
end

```

5)

```

% M-PSK System Simulation with Phase Debugging
clear all; close all; clc;

%% Parameters
M_vals = [8, 16]; % Modulation orders for 8-PSK and 16-PSK
k_vals = log2(M_vals); % Bits per symbol for each M-PSK
Lb = 1e5; % Total number of bits
Rs = 250e3; % Symbol rate (symbols per second)
Fc = 2.5e6; % Carrier frequency (Hz)
Fs = 10e6; % Sampling frequency (Hz)
Tsample = 1 / Fs; % Sampling period (seconds)

% Prepare for power spectral densities
psd_8psk = [];
psd_16psk = [];
frequencies = [];

for M_idx = 1:2
    M = M_vals(M_idx); % Get M (8 or 16)
    k = k_vals(M_idx); % Get corresponding bits per symbol
    Tsym = 1 / Rs; % Symbol period (seconds)
    Ns = Tsym / Tsample; % Number of samples per symbol
    Ns = round(Ns); % Ensure Ns is an integer

    % Adjust Lb to be a multiple of k
    if mod(Lb, k) ~= 0
        Lb_adjusted = floor(Lb / k) * k;
        Lb = Lb_adjusted;
    end
end

```

```

Es = 1; % Symbol energy (normalized to 1)
Eb = Es / k; % Energy per bit

% Display parameters for verification
fprintf('Sampling Frequency Fs = %.2f MHz\n', Fs / 1e6);
fprintf('Symbol Period Tsym = %.2f us\n', Tsym * 1e6);
fprintf('Number of Samples per Symbol Ns = %d\n', Ns);

%% Generate Binary Input Sequence
bits = randi([0, 1], Lb, 1);

%% Map Bits to Symbols (Gray Coding)
bit_groups = reshape(bits, k, []).'; % Reshape bits into k-bit
symbols
binary_symbols = bi2de(bit_groups, 'left-msb'); % Convert to
decimal symbols
gray_symbols = bitxor(binary_symbols, floor(binary_symbols / 2));
% Gray code

%% Generate Baseband Signal
pulse_shape = sqrt(2 / Ns) * ones(1, Ns); % Rectangular pulse
with unit energy
num_symbols = length(gray_symbols);
I_baseband_total = zeros(1, num_symbols * Ns);
Q_baseband_total = zeros(1, num_symbols * Ns);

for idx = 1:num_symbols
    idx_range = (idx-1)*Ns + 1 : idx*Ns;
    m = gray_symbols(idx);
    theta_m = 2 * pi * m / M;
    I_baseband_total(idx_range) = cos(theta_m) * pulse_shape;
    Q_baseband_total(idx_range) = sin(theta_m) * pulse_shape;
end

%% Modulate Baseband Signal onto Carrier (Corrected)
t_total = (0:(num_symbols * Ns - 1)) * Tsample; % Total time
vector
carrier_cos_total = cos(2 * pi * Fc * t_total);
carrier_sin_total = sin(2 * pi * Fc * t_total);

% Modulate onto carrier (corrected sign)
s_t = I_baseband_total .* carrier_cos_total + Q_baseband_total .*
carrier_sin_total;

%% Power Spectral Density Calculation using pwelch
[psd, f] = pwelch(s_t, [], [], [], Fs, 'twosided'); % Estimate
the power spectrum
psd_dB = 10*log10(psd); % Convert to dB scale

```

```

    % Store PSD for plotting later, but ensure lengths match
    if isempty(frequencies)
        frequencies = f; % Use the first frequency vector as
reference
    else
        % Match lengths of PSDs by interpolation if needed
        if length(psd_dB) ~= length(frequencies)
            psd_dB = interp1(f, psd_dB, frequencies, 'linear',
'extrap');
        end
    end

    % Store PSD for respective modulation
    if M == 8
        psd_8psk = psd_dB;
    elseif M == 16
        psd_16psk = psd_dB;
    end
end

%% Plot both Power Spectra on the same graph
figure;
plot(frequencies / 1e6, psd_8psk, 'b-', 'LineWidth', 1.5); hold on;
plot(frequencies / 1e6, psd_16psk, 'r--', 'LineWidth', 1.5);
xlabel('Frequency (MHz)');
ylabel('Power/Frequency (dB/Hz)');
title('Power Spectral Density of 8-PSK and 16-PSK Signals');
legend('8-PSK', '16-PSK');
grid on;

%% Function for Gray to Binary Conversion (Kept Unchanged)
function binary_symbols = gray2binary(gray_symbols, k)
    gray_bits = de2bi(gray_symbols, k, 'left-msb');
    binary_bits = zeros(size(gray_bits));
    binary_bits(:,1) = gray_bits(:,1);
    for i = 2:k
        binary_bits(:,i) = xor(binary_bits(:,i-1), gray_bits(:,i));
    end
    binary_symbols = bi2de(binary_bits, 'left-msb');
end

```