Project Report

Spread Spectrum Communication & Jamming

Autumn 2015

Anirud Thyagharajan [12EE35011], Ashish Ranjan Karn [12EE35010]

Introduction

The performance of any CDMA system is largely determined by the choice of the spreading codes which are used as a signature code for the users in such a system. However, depending upon the system requirement, the selection of spreading code has been the most important task on which a proper attention is required.

The choice of spreading codes in CDMA mobile systems is mainly governed by the following desirable characteristics that include (i) availability of large number of codes (ii) impulsive auto-correlation function (iii) zero cross-correlation values (iv) randomness (v) ease of generation and (vi) support for variable data rates.

This project focuses on the performance of the communication system using Gold Codes over an AWGN channel. In this project, first BER vs SNR comparison was done for BPSK and QPSK with Gaussian noise but without spreading. Next, similar comparison was carried out with spreading using 31 bit Gold Code. Finally the performance comparison is done with a 60 GHz AWGN channel with given channel coefficients. In order to account for ISI (Inter Symbol Interference) and fast fading frequency selective channel distortion, MMSE Equalizer has been used.

Motivation

Multi-user codes are widely used in transmitting user data from base station to mobile station and vice versa. In wireless communications, a signal received by a mobile station goes through different fading channels resulting in multi-path delays and attenuation at the mobile station. The autocorrelation and cross correlation properties of the Multi-user codes also play a significant role in extracting the user data from the signal corrupted by multipath fading channel.

This project aims to determine the optimum length of spreading code (Gold code) and to evaluate its performance based on BER vs SNR for communication over the given channel (7 meters/60 GHz AWGN channel). So that a comparative study of performance of Gold Code vs other codes can be done at the end.

This project motivates students to understand several components of the wireless communication link and undertake design steps from the perspective of a system engineer, from the selection of appropriate codes in order to reduce the cross correlation, to the use of an appropriate equalizer for mitigating the channel fading effect.

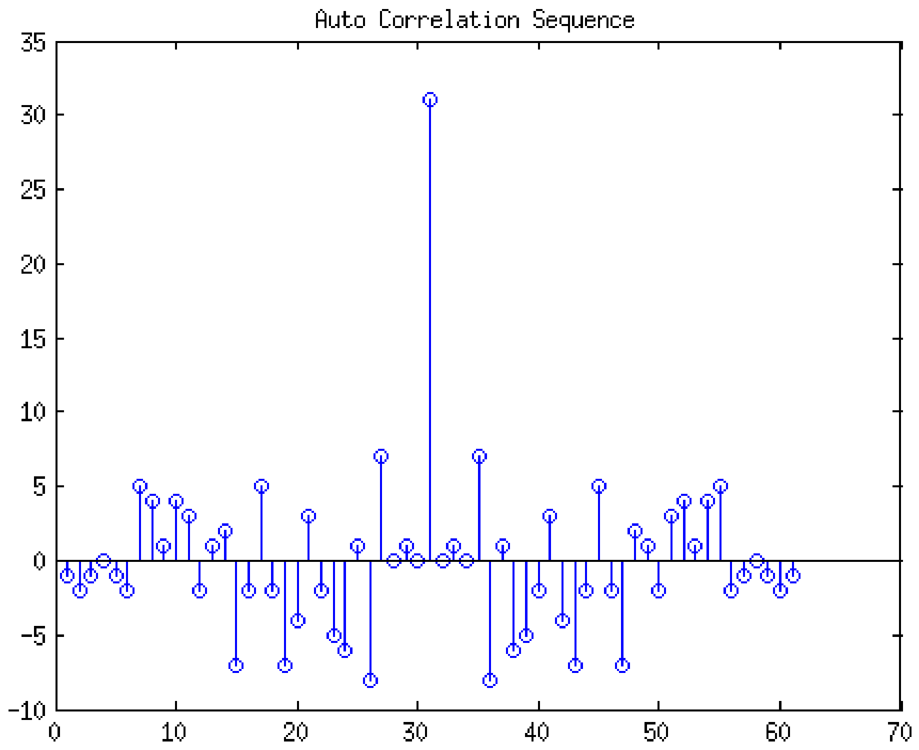
Gold Code Generation

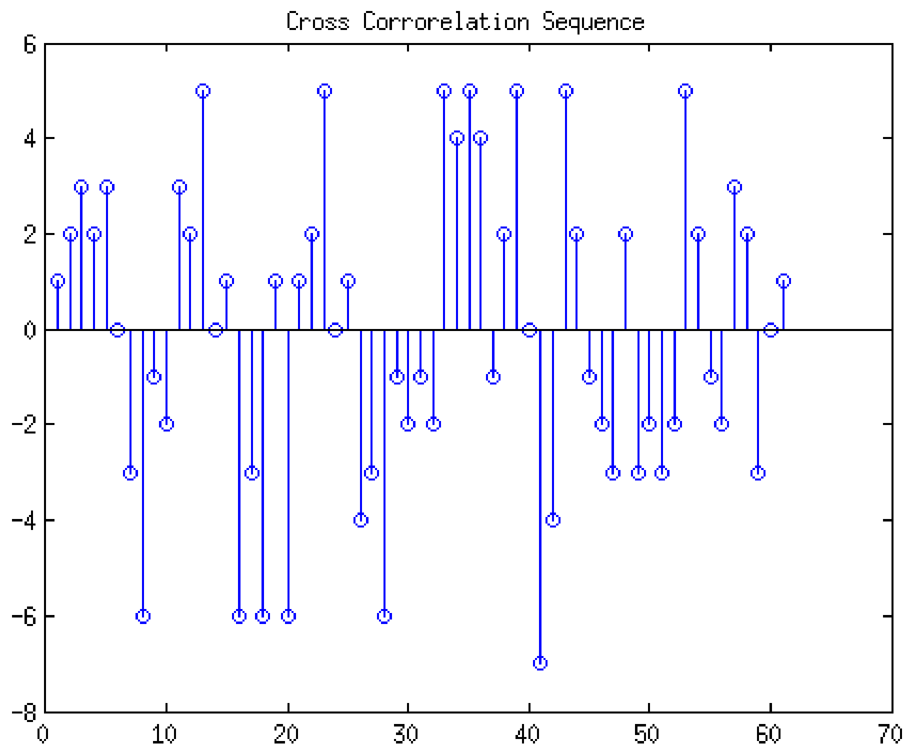
Gold code sequences are useful because a large number of codes can be generated with controlled cross correlation and with same length.

These codes can be created using a shift-register with feedback-taps. By using a single shift-register, maximum length sequences (M-sequences) can be obtained. Such sequences can be created by applying a single shift-register with a number of specially selected feedback-taps. If the shift-register size is n, then the length of the code is equal to 2n-1. The number of possible codes is dependent on the number of possible sets of feedback-taps that produce an M-sequence. Here we generate a new gold sequence by using two M-sequences. The actual operation can be performed by EX-OR or modulo-2 addition of two M-sequences of same length. As the length of sequences is same, they can maintain the same phase relationship. Every change in phase position between two M-sequences cause a new sequence to be generated.

The two M sequences are called the preferable pairs, which have to be suitably chosen so that the circular shifting and XORing operation yields codes with minimal cross correlation. The code attached, uses such preferable pairs available in literature for code lengths of 31, 63 and 127. The preferable pairs were available for lengths of 511, 1023 and 2047 as well, but seemed computationally heavy to carry out.

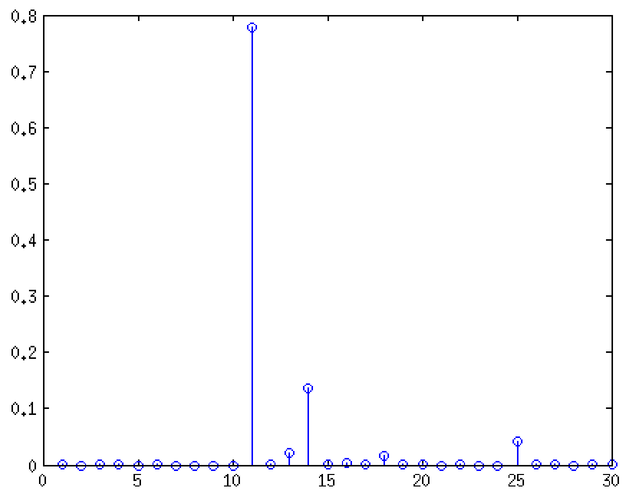
Here are a few plots showing the autocorrelation and the cross-correlation of the gold sequences thus generated. You may refer to the *gold.m* file for seeing the Matlab code associated with it.





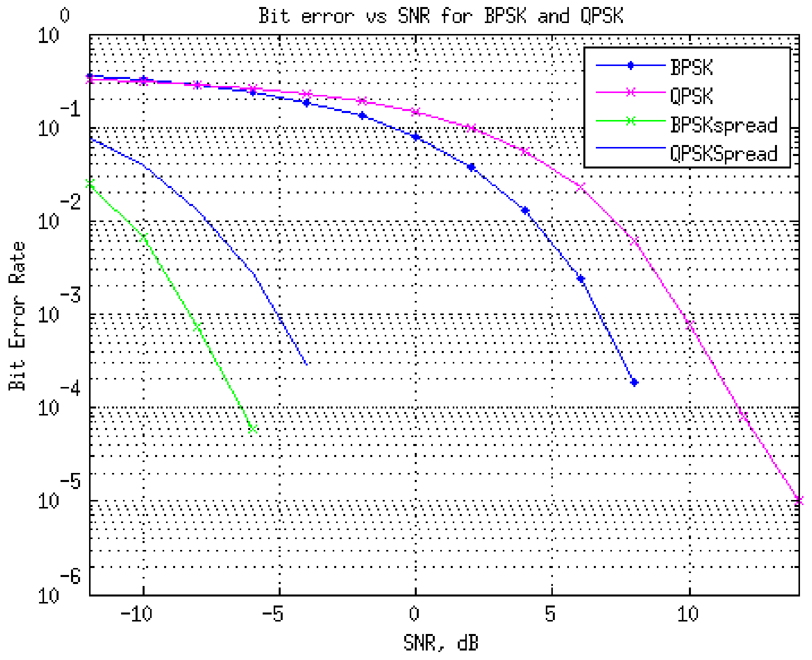
An algorithm is followed for picking up the best code out of the 33 codes. For ever pair of codes, we compute the cross-correlation peak, and find the pair for which the peak is the least. Thus, this pair is bound to give minimum cross correlation and hence can be used in conjunction with each other. The **above plot** gives us the least correlation, and hence one out of them is chosen for the spread spectrum communication link design.

Power Delay profile of the channel

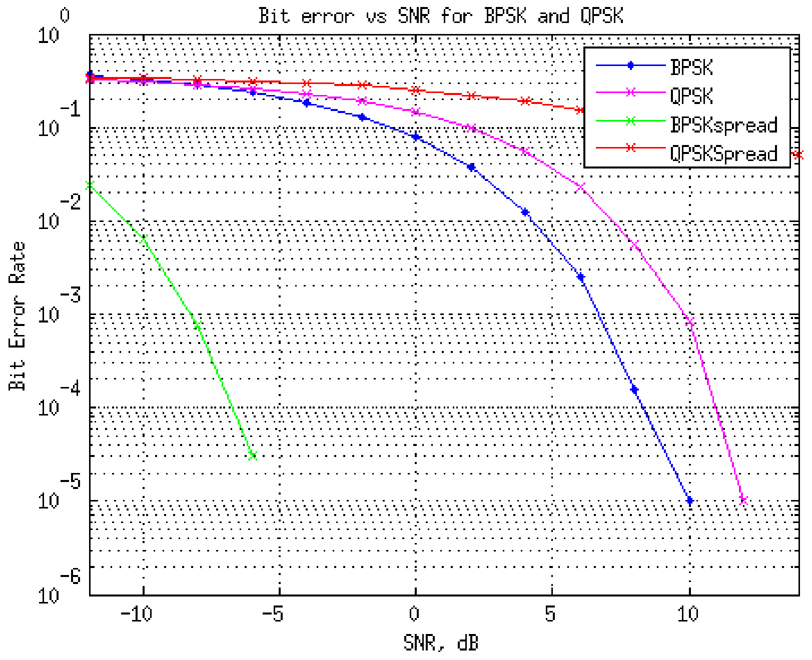


The power delay profile suggests that the peak occurs only after a delay of 10 time units in the channel. Thus, our *MMSE_eq.m* does the synchronization and the equalization both.

The following plot describes the relation between BER and SNR for a spread spectrum communication link over an AWGN channel.



The following plot describes the relation between BER and SNR for a spread spectrum communication link over the given 60GHz / 7m channel without equalization.



Discussions and Conclusions

The above program was run for 3 epochs of code length {31, 63, 127}, to study the effect of code length on the BER. We conclude that we obtain the least bit error rate for a high length code, i.e. for *code_len = 127*. However, it will also be noticed that acquisition and synchronization may become more complicated as the code length increases. Thus, we have to find an optimal length according to our application.

```
clear all;
close all;
load ch_coeff.mat;

n = 10^5;
K = length(ch_coeff);
L = length(ch_coeff);

eb_n0_db = -12:2:14;
snr_db = -12:2:14;

tx_data_bp = rand(1,n);
tx_data_bpsk = 2 * (tx_data_bp > 0.5) - 1;
tx_data_qpr = rand(1,n/2);
tx_data_qpi = rand(1,n/2);
tx_data_qpsk = (2*(tx_data_qpr>0.5)-1) + j*(2*(tx_data_qpi>0.5)-1);

eb_n0_lin = 10.^(eb_n0_db/10);
snr_lin = 10.^(snr_db/10);
codes = [31, 63, 127];

bpsk1 = zeros(length(codes), length(eb_n0_db));
qpsk1 = zeros(length(codes), length(eb_n0_db));
bpsk_spread1 = zeros(length(codes), length(eb_n0_db));

bpsk2 = zeros(length(codes), length(eb_n0_db));
qpsk2 = zeros(length(codes), length(eb_n0_db));
bpsk_spread2 = zeros(length(codes), length(eb_n0_db));

% Iterating over all code lengths, to find the optimal code sequence
for k = 1:length(codes)
  code = gold(k);
  spread_data_bpsk = kron(tx_data_bpsk, code');
  spread_data_qpsk = kron(tx_data_qpsk, code');
  code_len = codes(k);

  for i = 1:length(eb_n0_db)
    clear decoded_sig1
    clear decoded_sig2
    % Adding noise
    fprintf('Epoch %d \n', i);
    noise_bpsk = randn(1, n)/(sqrt(2*eb_n0_lin(i)));
    noise_bpsk2 = randn(1,n)/sqrt(2*snr_lin(i));
    noise_qpsk = (randn(1, n/2)+j*randn(1, n/2))/sqrt(2*(2*eb_n0_lin(i)));
    noise_qpsk2 = (randn(1,n/2)+j*randn(1,n/2))/sqrt(2*2*0.5*snr_lin(i));

    % Added noise for spreading
    noise_bpsk_spread = randn(1,code_len*n)/(sqrt(2*eb_n0_lin(i)/code_len));
    noise_bpsk_spread2 = randn(1, code_len*n)/sqrt(2*snr_lin(i));
    noise_qpsk_spread = (randn(1,code_len*n/2) + j*randn(1,code_len*n/2))/(sqrt(2*2*
eb_n0_lin(i))/code_len);
    noise_qpsk_spread2 = (randn(1,n/2*code_len)+j*randn(1,n/2*code_len))/sqrt(2*2*0.
5*snr_lin(i));
    sigma = var(noise_qpsk_spread2);

    % Transmitted Data stream, unspread, power normalized for QPSK
    rx_bpsk = tx_data_bpsk + noise_bpsk;
    rx_bpsk2 = tx_data_bpsk + noise_bpsk2;
    rx_qpsk = tx_data_qpsk/sqrt(2) + noise_qpsk;
    rx_qpsk2 = tx_data_qpsk/sqrt(2) + noise_qpsk2;

    % Transmitted Data stream, spread
    rx_bpsk_spread = spread_data_bpsk + noise_bpsk_spread;
    rx_bpsk_spread2 = spread_data_bpsk + noise_bpsk_spread2;
    rx_qpsk_spread = spread_data_qpsk/sqrt(2) + noise_qpsk_spread2;

    % Applying the channel and equalization followed by synchronization, for QPSK
    rx_qpsk_spread = filter(ch_coeff, 1, rx_qpsk_spread);
    [rx_qpsk_spread, a] = MMSE_eq(rx_qpsk_spread, ch_coeff, sigma);

    % Triggered to see effect of channel and equalization on simple unspread QPSK [O
ptional]
    %rx_qpsk2 = filter(ch_coeff, 1, rx_qpsk2);
    %[rx_qpsk2, a] = MMSE_eq(rx_qpsk2, ch_coeff, sigma);

    rx_bpsk = 2 * (rx_bpsk > 0) - 1;
    rx_bpsk2 = 2 * (rx_bpsk2 > 0) - 1;

    % Dealing with I, Q components
    rx_qpsk_i = 2 * (real(rx_qpsk) > 0) - 1;
    rx_qpsk_q = 2 * (imag(rx_qpsk) > 0) - 1;
    rx_qpsk = rx_qpsk_i + j*rx_qpsk_q;
    rx_qpsk2_i = 2 * (real(rx_qpsk2) > 0) - 1;
    rx_qpsk2_q = 2 * (imag(rx_qpsk2) > 0) - 1;
    rx_qpsk2 = rx_qpsk2_i + j*rx_qpsk2_q;

    % BPSK Spreading, 1
    temp_sig1 = rx_bpsk_spread';
    temp_sig2 = reshape(temp_sig1, code_len, n);
    temp_sig3 = kron(ones(n,1), code');
    temp_sig4 = temp_sig2'.*temp_sig3;
    despread_sig = (sum(temp_sig4'))/code_len;
    decoded_sig1 = 2 * (despread_sig>0) - 1;

    % BPSK Spreading, 2
    temp_sig1 = rx_bpsk_spread2';
    temp_sig2 = reshape(temp_sig1, code_len, n);
    temp_sig4 = temp_sig2'.*temp_sig3;
    despread_sig = (sum(temp_sig4'))/code_len;
    decoded_sig2 = 2 * (despread_sig>0) - 1;

    rx_qpsk_spread_i = real(rx_qpsk_spread);
    rx_qpsk_spread_q = imag(rx_qpsk_spread);

    %channel i despreading
    temp_sig1 = rx_qpsk_spread_i';
    temp_sig2 = reshape(temp_sig1, code_len, n/2);
    temp_sig3 = kron(ones(n/2,1), code');
    temp_sig4 = temp_sig2'.*temp_sig3;
    despread_sig_qpsk_i = (sum(temp_sig4'))/code_len;
    decoded_sig1_qpsk_i = 2 * (despread_sig_qpsk_i>0) - 1;

    %channel q despreading
    temp_sig1 = rx_qpsk_spread_q';
    temp_sig2 = reshape(temp_sig1, code_len, n/2);
    temp_sig3 = kron(ones(n/2,1), code');
    temp_sig4 = temp_sig2'.*temp_sig3;
    despread_sig_qpsk_q = (sum(temp_sig4'))/code_len;
    decoded_sig1_qpsk_q = 2 * (despread_sig_qpsk_q>0) - 1;

    %merging decoded_sig1_i&q into 1 matrix
    decoded_sig1_qpsk = decoded_sig1_qpsk_i + j*decoded_sig1_qpsk_q;

    % Decision Time!
    bpsk1(k,i) = sum(rx_bpsk ~= tx_data_bpsk)/n;
    bpsk2(k,i) = sum(rx_bpsk2 ~= tx_data_bpsk)/n;
    bpsk_spread1(k,i) = sum(decoded_sig1 ~= tx_data_bpsk)/n;
    bpsk_spread2(k,i) = sum(decoded_sig2 ~= tx_data_bpsk)/n;

    qpsk1(k,i) = 0.5*(sum(rx_qpsk ~= tx_data_qpsk)*2/n);
    qpsk2(k,i) = 0.5*(sum(rx_qpsk2 ~= tx_data_qpsk)*2/n);
    qpsk_spread1(k,i) = 0.5*sum(decoded_sig1_qpsk ~= tx_data_qpsk)*2/n;

end
end


% For spread spectrum, we use only Ec : chip energy or we use SNR.
figure;
semilogy(snr_db,bpsk2(1,:),'b.-', snr_db,qpsk2(1,:), 'mx-', eb_n0_db,bpsk_spread2(1,
:), 'gx-', snr_db, qpsk_spread1(1,:), 'rx-');
grid on;
legend('BPSK', 'QPSK', 'BPSKspread', 'QPSKSpread');
axis([-12 14 1e-6 1]);

xlabel('SNR, dB');
```

```
ylabel('Bit Error Rate');
title('Bit error vs SNR for BPSK and QPSK');


% Selection of Optimal Code Length from qpsk spreading
[ii, jj] = min(sum(qpsk_spread1, 2));
codes(jj)

% Plot to select Optimal code
figure;
semilogy(snr_db,qpsk_spread1(1,:),'b.-', snr_db,qpsk_spread1(2,:), 'mx-', eb_n0_db,q
psk_spread1(3, :), 'gx-');
grid on;
legend('len: 31', 'len: 63', 'len: 127');
axis([-12 14 1e-6 1]);
xlabel('SNR, dB');
ylabel('Bit Error Rate');
title('BER vs SNR for varying code lengths');
```