Matlab code for Implementing SS-OFDM

Mohammad Alhasnawi and Ron Addie

June 19, 2018

Abstract
This report contains the matlab code used to implement an SS-OFDM system.

1 Introduction
SS-OFDM stands for Spread Spectrum Orthogonal Frequency-Division Multiplexing of wireless communication system. An SS-OFDM system has been implemented in Matlab and the code for it is included in this report. A number of experiments have been carried out, for different choices of the Galois field size (f), background noise power (\(\eta\)) and the size of the constellation.

In [1] an experiment using the code in this report, in which \(f=1023\), was described. This experiment is sufficient to convey the key features of the system. In this system it was found that if the number of users is less than or equal to 1000, and the constellation size was 60, all users were able to communicate simultaneously with a low error rate (usually without error). The system implemented did not include error correction. The background noise of this system has a standard deviation of 0.05, so the Shannon capacity is approximately 8.65 bits/s. The implemented system was transmitting at \(\approx 5 \text{ bps}\).

2 Description of Code
Table 1 provides a list of all the matlab files in the system, together with a description of the purpose of that particular script or function. All but two of the files define matlab functions. The script `wholesystem.m` performs an experiment in which `NU` users send and receive messages of length `ML` simultaneously.
The script TotalBR.m creates graphs showing the total capacity of a collection of SS-OFDM systems operating simultaneously, either in the same location, or at locations separated in space geographically sufficiently to have a certain signal power loss relative to each other.

For an explanation of how the system works, and how to interpret the experiments conducted by means of wholesystem.m, and how to interpret the graphs produced by TotalBR.m, please see [1].

Table 1: Codes clarification

<table>
<thead>
<tr>
<th>Name of function / file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate_codes</td>
<td>Generate codes for each user.</td>
</tr>
<tr>
<td>findConst</td>
<td>Find a constellation with ( f ) as the prime, ( n ) the number of different circles, and ( \phi ) is the number of symbols in the outer circle.</td>
</tr>
<tr>
<td>constSep</td>
<td>find the minimum separation of constellation symbols</td>
</tr>
<tr>
<td>Primitive</td>
<td>Find a primitive element of the group.</td>
</tr>
<tr>
<td>decodeFromConst</td>
<td>Decode a message using a certain constellation of symbols.</td>
</tr>
<tr>
<td>encodeToConst</td>
<td>Encode a message using a certain constellation of symbols.</td>
</tr>
<tr>
<td>findLattice</td>
<td>construct a constellation as a lattice, with a specific number of symbols per row.</td>
</tr>
<tr>
<td>TotalBR</td>
<td>Calculate the total bandwidth per user of the whole system.</td>
</tr>
<tr>
<td>Matlab main program</td>
<td>The whole system, with ( NU ) users sending different messages and each message is coded using DSSS, then all messages are combined together, then each receiver decodes their own message from the aggregate signal.</td>
</tr>
</tbody>
</table>
function codes = generate_codes(f, p)
gs = f - 1;
codes = zeros(gs, gs);
ppowerforkloop = 1;
for k = 1:gs
    ppowerforjloop = ppowerforkloop;
    for j = 1:gs
        codes(j, k) = ppowerforjloop;
        % mod(p^mod(j+k-2,gs),f); this method leads to overflow
        ppowerforjloop = mod(ppowerforjloop * p, f);
    end
    ppowerforkloop = mod(ppowerforkloop * p, f);
end
end

Listing 1: Generate_codes

function signal = dsss_modulate_ho(code, msg, NoOfBits, f)
    chiplen = length(code);
    siglen = ceil(chiplen * length(msg) / NoOfBits);
    signal = zeros(siglen, 1);
    msgInSymbols = symbolcodemsg(msg, f);
    for k = 1:length(msgInSymbols);
        for j = 1:chiplen
            % signal(chiplen*(k-1)+j,1) =
            signal(chiplen*(k-1)+j, 1) =
            % gfdiv(msgInSymbols(k),code(j),f);
            signal(chiplen*(k-1)+j, 1) = mod((code(j) +
            % msgInSymbols(k)), f);
        end
    end
end

Listing 2: findConst
function msginsymbols = symbolcodemsg(msg,f)
    j = ceil(log2(f-1)); % j is the maximum number of bits encoded in each symbol
    % sometimes only j-1 bits can be encoded
    msginsymbols = zeros(1,ceil(length(msg)/(j-1)));
    k = 1; % where we are up to in the msg
    ks = 1; % where we are up to in the symbolmsg
    thresh = f - 2^(j-1) - 1;
    while (k<=length(msg))
        % calculate the part of the next symbol due to the first j-1 binary digits
        lowbitspart = 0;
        for bit = (k+(j-2)) : -1 : k
            if bit<=length(msg)
                lowbitspart = 2*lowbitspart + msg(bit);
            else
                lowbitspart = 2*lowbitspart;
            end
        end
        if lowbitspart < thresh && ((k+j-1) <= length(msg))
            msginsymbols(ks) = msg(k+j-1)*2^(j-1) + lowbitspart ;
            k = k + j;
        else
            msginsymbols(ks) = lowbitspart;
            k = k + j - 1;
        end
        ks = ks+1;
    end
    ks = ks - 1;
    msginsymbols = msginsymbols(1:ks) + ones(1,ks);
end

Listing 3: constSep
function p = primitive(f)
    for k=2:(f-1)
        success = true;
        test = k;
        for j=1:(f-3)
            test = mod (test*k,f);
            if test==1
                success = false;
                break;
            end
        end
        if success
            p = k;
            return;
        end
    end
end

Listing 4: Primitive
function message = dsss_demodulate_ho(code, signal, f, NU)
    chiplen = length(code);
    symbol = zeros(1, length(signal));  % for some reason we are using rows
    for k=1:(length(signal)/chiplen)
        W = 0;
        for j=1:chiplen
            thisest = signal(chiplen*(k-1)+j)*exp(-2*pi*i*code(j)/f);
            % signalmag = abs(signal(chiplen*(k-1)+j));
            % signalarg = myangle(signal(chiplen*(k-1)+j));
            % signalsymbol =
            %    mod(round(f*signalarg/(2*pi)),f);
            % signalsymbol = mod(signalsymbol * code(j),f);
            % thisest = signalmag *
            %    exp(2*pi*i*signalsymbol/f);
            W = W + thisest;
            % chipcode = mod((primitive*chipcode), fs);  % rotation (not used)
        end
        symbol(k) = mod(round(f*myangle(W)/(2*pi)),f);
    end
    message = bitcodemsg(symbol,f);
end

Listing 5: decodeFromConst
function message = dsss_demodulate_ho(code, signal, f, NU)
  chiplen = length(code);
  symbol = zeros(1,length(signal));  % for some reason we are
  % using rows
  for k=1:(length(signal)/chiplen)
    W = 0;
    for j=1:chiplen
      thisest = signal(chiplen*(k-1)+j)*exp(-2*pi*i*code(j)/f);
      % signalmag = abs(signal(chiplen*(k-1)+j));
      % signalarg = myangle(signal(chiplen*(k-1)+j));
      % signalsymbol =
      % mod(round(f*signalarg/(2*pi)),f);
      % signalsymbol = mod(signalsymbol * code(j),f);
      % thisest = signalmag *
      % exp(2*pi*i*signalsymbol/f);
      W = W + thisest;
      % chipcode = mod((primitive*chipcode), fs);  %
      % rotation (not
      % used)
    end
    symbol(k) = mod(round(f*myangle(W)/(2*pi)),f);
  end
  message = bitcodemsg(symbol,f);
end

Listing 6: encodeToConst
function message = dsss_demodulate_ho(code, signal, f, NU)
chiplen = length(code);
symbol = zeros(1,length(signal));  % for some reason we are
               % using rows
    for k=1:(length(signal)/chiplen)
        W = 0;
            for j=1:chiplen
                thisest = signal(chiplen*(k-1)+j)*exp(-2*pi*i*code(j)/f);
                % signalmag = abs(signal(chiplen*(k-1)+j));
                % signalarg = myangle(signal(chiplen*(k-1)+j));
                % signalsymbol =
                % mod(round(f*signalarg/(2*pi)),f);
                % signalsymbol = mod(signalsymbol * code(j),f);
                % thisest = signalmag *
                % exp(2*pi*i*signalsymbol/f);
                W = W + thisest;
                % chipcode = mod((primitive*chipcode), fs);  %
                % rotation (not
                % used)
            end
        symbol(k) = mod(round(f*myangle(W)/(2*pi)),f);
    end
message = bitcodemsg(symbol,f);
end

Listing 7: findLattice
% Calculate the total bandwidth per user of the whole system

NU = 100;
Z = 1: NU;
chiplen = Z;
f = chiplen+1;
B = 1000000;
kappa = 1022;  % an alternative notation for chip length
phi = 28;      % constellation size
eta2 = 0.0025;
alpha = 0.5;
logbit = \log_2\left(1 + \frac{\sin(\pi \cdot \text{ones}(1,NU) \cdot f)^2 \cdot \text{chiplen}^2}{\alpha \cdot (Z-1) + \eta2}\right);

% correction for the case n=1
logbit(1) = \log_2\left(1 + \frac{1}{\eta2}\right);

totalBRcircperim = B \cdot Z \cdot \logbit \div \text{chiplen};
totalBRcircarea = B \cdot Z \cdot \logbit \div \pi^2;
singleuserBR = B \cdot \log_2\left(1 + \frac{1}{\eta2}\right) \cdot \text{ones}(1,NU);

% Shannon formula
newsingleuserB = B \cdot \left(\log_2(\phi) \cdot \log_2\left(1 + \frac{\pi}{4 \cdot \phi \cdot \eta2}\right)\right) \cdot \text{ones}(1,NU);

totalBRnew = B \cdot \left(\log_2(\phi) \div \kappa\right) \cdot Z \cdot \log_2\left(\text{ones}(1,NU) + \kappa \cdot \pi \cdot \text{Z}^{1+\alpha}\right) \div \left(4 \cdot \phi \cdot \eta2 \cdot \left(\kappa \cdot \eta2 \cdot \text{ones}(1,NU) + \alpha \cdot \kappa \cdot (Z - \text{ones}(1,NU))\right)\right);

totalBRnew2 = B \cdot \log_2(\phi) \cdot \log_2\left(\text{ones}(1,NU) + \pi \cdot \text{Z}^{1+\alpha}\right) \div \left(4 \cdot \phi \cdot \eta2 \cdot \left(\text{Z} + \text{Z}^{1+\alpha}\right)\right);

figure();
subplot(2,1,1);
plot(Z,newsingleuserB, Z, totalBRnew2);
title('Total system throughput')
ylabel('Throughput')
xlabel('Number of users / chiplength')
legend({'No sharing', 'Sharing by DSSS-OFDM'},'Location','east')
return;

% plot a diagram with different alphas
figure();
subplot(2,1,1)
alpha = 0.1;
logbit = \log_2\left(1 + \frac{\pi^2 \cdot \text{chiplen}^2}{\left(\alpha \cdot (Z-1) + \eta2\right)}\right);

logbit(1) = \log_2\left(1 + \frac{1}{\eta2}\right);  % correction for the case n=1

totalBRcircarea0_1 = B \cdot Z \cdot \logbit \div \left(2 \cdot \pi\right);
alpha = 0.2;
logbit = log2 (1 + pi^2 * chiplen.^2 ./ (f.^2 * (alpha*(Z-1)+eta2)));  
logbit(1) = log2(1+1/eta2); % correction for the case n=1  
totalBRcircarea0_2 = B * Z.* logbit /(2*pi);  
alpha = 0.4;  
logbit = log2 (1 + pi^2 * chiplen.^2 ./ (f.^2 * (alpha*(Z-1)+eta2)));  
logbit(1) = log2(1+1/eta2); % correction for the case n=1  
totalBRcircarea0_4 = B * Z.* logbit /(2*pi);  
alpha = 1;  
logbit = log2 (1 + pi^2 * chiplen.^2 ./ (f.^2 * (alpha*(Z-1)+eta2)));  
logbit(1) = log2(1+1/eta2); % correction for the case n=1  
totalBRcircarea1 = B * Z.* logbit /(2*pi);  
plot(Z, totalBRcircarea0_1, Z, totalBRcircarea0_2, Z, totalBRcircarea0_4, Z, totalBRcircarea1);  
title('Total system bandwidth')  
ylabel('BR')  
xlabel('Number of users')  
legend({'alpha=0.1', 'alpha=0.2', 'alpha=0.4', 'alpha=1'},'Location','east')  

% Plot the Correlation between variance of user noise vs number of users  
Z = 1:NU  
figure();  
subplot(2,1,1)  
correlation = plot(Z,varianceusernoise);  
title('Correlation between variance of user noise vs number of users')  
ylabel('variance of user noise')  
xlabel('Number of users')

Listing 8: TotalBR

% The whole system, with N users sending different messages and each  
% message is coded using DSSS, then all messages are combined together,  
% then each receiver decodes their own message from the aggregate signal  
% parameters: NU: number of users  
global usess;  
NU = 1000 ;
ML = 200; % ML: message lengths
bkgndnoisestdev = 0.05;
usess = true;
if (usess)
    f = 1031; % prime number, which is the size of Galois Field
    chiplen = f-1; % length of chips
    p = primitive(f); % the primitive element of the group
    code = generate_codes(f, p); % codes for each user
    phi = f/3
else
    f = 1031;
    ML = ML*NU;
    NU = 1;
    n = 3
    phi = 16;
    code = ones(1,1);
end
beta = 1/sqrt(chiplen); % we reduce signal power to keep within regulations
rowcount = 10;
[const, constlength] = findLattice(rowcount);
[outerconst, fullcount] = findConst(f, 1, f, true);
constlength
bps = floor(log2(constlength)+0.001); % bits per symbol (only approximate -- actually its bps sometimes and bps-1
if (usess)
    siglen = chiplen*ceil(ML/bps);
else
    siglen = ceil(ML/bps);
end
minsep = constSep(const, constlength);
h = 0; % h = scatterplot(const,1,0,’b’); %
scatterplot(outerconst,1,0,’b’);
messages = round(rand(ML,NU)); % Generate messages
signals = zeros(siglen,NU); % Modulate message to create N signals
for k=1:NU
    [x, lengthinsymbols, msgInSymbols] = encodeToConst(const, outerconst, code(k,:), messages(:,k), bps, f);
signals(1:length(x),k) = beta * x;
end
% signals(1,1), Add signals together
aggregatesignal = zeros(length(signals(:,1)),1);
aggregatesignal(1:length(signals(:,k))) = signals(:,1);
for k=2:NU
    vec = aggregatesignal(1:length(signals(:,k))) + signals(:,k);
    aggregatesignal(1:length(signals(:,k))) = vec;
end
bkgndnoise =
    random('norm',0,bkgndnoisestdev,[length(signals(:,1)),1]);
phase = random('uniform',0,2*pi,[length(signals(:,1)),1]);
bkgndnoise = bkgndnoise .* exp(i*phase);
bkgndnoise_actual_stdev = rms(bkgndnoise)
aggregatesignal = aggregatesignal + bkgndnoise;
% Demodulate and decode N messages from the combined signal
decodedmessages = messages;
for k=1:NU
    [dm,tnoise] = decodeFromConst(const, outerconst, code(k,:),
    aggregatesignal, f, h, msgInSymbols, beta);
    decodedmessages(:,k) = dm(1:ML);
end
totalnoise = tnoise(1:lengthinsymbols);
% Check error rate
errorcount = sum(xor(messages(1:(ML-1)),decodedmessages(1:(ML-1))))
% Estimate the standard deviation of user-noise and mean of
aggregatesignal
meanusernoise = mean(totalnoise)
variancetotalnoise = var(abs(totalnoise))
stdevtotalnoise = sqrt(variancetotalnoise)
throughput = NU * log2(constlength)
symbolstddev = rms(const)
rmsnoise = rms(totalnoise)

Listing 9: Matlab main program

3 Conclusion

A communication system which combines spread-spectrum codes and OFDM
with the potential to operate at optimal efficiency has been defined, implemented
and tested. All the code for this system is included in this report.
References