Contents

Correlate for LTS

```
clear;

LTS_CORR_THRESH=.8;
DO_APPLY_CFO_CORRECTION=1;
DO_APPLY_SFO_CORRECTION=1;
DO_APPLY_PHASE_ERR_CORRECTION=1;

MIMO_FLAG = 0; % 0 as default. Do not change

num_sim_snr = 11;
num_sim_mc = 50;

ber1 = zeros(num_sim_snr, num_sim_mc);
ber2 = zeros(num_sim_snr, num_sim_mc);

%for snr_i = 1:num_sim_snr

snrdB = 9+1;

%for j = 1:num_sim_mc

MIMO_OFDM_TX;

% For simplicity, we'll only use RFA for LTS correlation and peak
% discovery. A straightforward addition would be to repeat this process for
% RFB and combine the results for detection diversity.
if (MIMO_FLAG == 0)
    raw_rx_dec_1A = rx_vec_dec_1A;
    raw_rx_dec_1B = rx_vec_dec_1B;
    raw_rx_dec_1C = rx_vec_dec_1C;
    raw_rx_dec_1D = rx_vec_dec_1D;
end


% Complex cross correlation of Rx waveform with time-domain LTS
lts_corr_A = abs(conv(conj(fliplr(lts_t)), sign(raw_rx_dec_A)));
lts_corr_B = abs(conv(conj(fliplr(lts_t)), sign(raw_rx_dec_B)));

% Skip early and late samples - avoids occasional false positives from pre-AGC samples
lts_corr_A = lts_corr_A(32:end-32);
lts_corr_B = lts_corr_B(32:end-32);

% Find all correlation peaks
lts_peaks_A = find(lts_corr_A > LTS_CORR_THRESH*max(lts_corr_A));
lts_peaks_B = find(lts_corr_B > LTS_CORR_THRESH*max(lts_corr_B));


if(MIMO_FLAG == 0)
    lts_corr_1A = abs(conv(conj(fliplr(lts_t)), sign(raw_rx_dec_1A)));
    lts_corr_1B = abs(conv(conj(fliplr(lts_t)), sign(raw_rx_dec_1B)));

    lts_corr_1A = lts_corr_1A(32:end-32);
    lts_corr_1B = lts_corr_1B(32:end-32);

    lts_peaks_1A = find(lts_corr_1A > LTS_CORR_THRESH*max(lts_corr_1A));
    lts_peaks_1B = find(lts_corr_1B > LTS_CORR_THRESH*max(lts_corr_1B));

    [LTS11A, LTS21A] = meshgrid(lts_peaks_1A,lts_peaks_1A);
    [lts_last_peak_index_1A,y] = find(LTS21A-LTS11A == length(lts_t));
```

```
    [LTS11B, LTS21B] = meshgrid(lts_peaks_1B,lts_peaks_1B);
    [lts_last_peak_index_1B,y] = find(LTS21B-LTS11B == length(lts_t));

    lts_corr_1C = abs(conv(conj(fliplr(lts_t)), sign(raw_rx_dec_1C)));
    lts_corr_1D = abs(conv(conj(fliplr(lts_t)), sign(raw_rx_dec_1D)));

    lts_corr_1C = lts_corr_1C(32:end-32);
    lts_corr_1D = lts_corr_1D(32:end-32);

    lts_peaks_1C = find(lts_corr_1C > LTS_CORR_THRESH*max(lts_corr_1C));
    lts_peaks_1D = find(lts_corr_1D > LTS_CORR_THRESH*max(lts_corr_1D));

    [LTS11C, LTS21C] = meshgrid(lts_peaks_1C,lts_peaks_1C);
    [lts_last_peak_index_1C,y] = find(LTS21C-LTS11C == length(lts_t));

    [LTS11D, LTS21D] = meshgrid(lts_peaks_1D,lts_peaks_1D);
    [lts_last_peak_index_1D,y] = find(LTS21D-LTS11D == length(lts_t));

end


% Select best candidate correlation peak as LTS-payload boundary
% In this MIMO example, we actually have 3 LTS symbols sent in a row.
% The first two are sent by RFA on the TX node and the last one was sent
% by RFB. We will actually look for the separation between the first and the
% last for synchronizing our starting index.

[LTS1A, LTS2A] = meshgrid(lts_peaks_A,lts_peaks_A);
[lts_last_peak_index_A,y] = find(LTS2A-LTS1A == length(lts_t));

[LTS1B, LTS2B] = meshgrid(lts_peaks_B,lts_peaks_B);
[lts_last_peak_index_B,y] = find(LTS2B-LTS1B == length(lts_t));


% Stop if no valid correlation peak was found
if(isempty(lts_last_peak_index_A))
    disp('No LTS Correlation Peaks Found in Packet A!\n');
    return;
end

if(isempty(lts_last_peak_index_B))
    disp('No LTS Correlation Peaks Found in Packet B!\n');
    return;
end

if (MIMO_FLAG == 0)
    if(isempty(lts_last_peak_index_1A))
        disp('No LTS Correlation Peaks Found in Packet A!\n');
        return;
    end

    if(isempty(lts_last_peak_index_1B))
        disp('No LTS Correlation Peaks Found in Packet B!\n');
        return;
    end

    if(isempty(lts_last_peak_index_1C))
        disp('No LTS Correlation Peaks Found in Packet C!\n');
        return;
    end

    if(isempty(lts_last_peak_index_1D))
        disp('No LTS Correlation Peaks Found in Packet D!\n');
        return;
    end
end

% Set the sample indices of the payload symbols and preamble
% The "+32" here corresponds to the 32-sample cyclic prefix on the preamble LTS
% The "+192" corresponds to the length of the extra training symbols for MIMO channel estimation
mimo_training_ind_A = lts_peaks_A(max(lts_last_peak_index_A)) + 32;
mimo_training_ind_B = lts_peaks_B(max(lts_last_peak_index_B)) + 32;

if (MIMO_FLAG == 0)
    mimo_training_ind_1A = lts_peaks_1A(max(lts_last_peak_index_1A)) + 32;
```

```
    mimo_training_ind_1B = lts_peaks_1B(max(lts_last_peak_index_1B)) + 32;
    mimo_training_ind_1C = lts_peaks_1C(max(lts_last_peak_index_1C)) + 32;
    mimo_training_ind_1D = lts_peaks_1D(max(lts_last_peak_index_1D)) + 32;
end

if(MIMO_FLAG == 0)
    mimo_training_ind_tot = [mimo_training_ind_1A mimo_training_ind_1B mimo_training_ind_1C mimo_training_ind_1D];
    if(range(mimo_training_ind_tot) == 0)
        disp('Packet Starts Matches accross');
    end
end

if(mimo_training_ind_B == mimo_training_ind_A)
    disp('Packet Starts Matches accross');
end

%disp(snr_i);
```

```
Packet Starts Matches accross
Packet Starts Matches accross
```

CFO Correction

```
mimo_training_ind = mimo_training_ind_A;

payload_ind = mimo_training_ind + 192;

% Subtract of 2 full LTS sequences and one cyclic prefixes
% The "-160" corresponds to the length of the preamble LTS (2.5 copies of 64-sample LTS)
lts_ind = mimo_training_ind-160;

if(DO_APPLY_CFO_CORRECTION)
    %Extract LTS (not yet CFO corrected)
    rx_lts_A = raw_rx_dec_A(lts_ind : lts_ind+159); %Extract the first two LTS for CFO
    rx_lts1_A = rx_lts_A(-64 + [97:160]);
    rx_lts2_A = rx_lts_A([97:160]);

    rx_lts_B = raw_rx_dec_B(lts_ind : lts_ind+159); %Extract the first two LTS for CFO
    rx_lts1_B = rx_lts_B(-64 + [97:160]);
    rx_lts2_B = rx_lts_B([97:160]);

    if(MIMO_FLAG == 0)
        rx_lts_1A = raw_rx_dec_1A(lts_ind : lts_ind+159); %Extract the first two LTS for CFO
        rx_lts1_1A = rx_lts_1A(-64 + [97:160]);
        rx_lts2_1A = rx_lts_1A([97:160]);

        rx_lts_1B = raw_rx_dec_1B(lts_ind : lts_ind+159); %Extract the first two LTS for CFO
        rx_lts1_1B = rx_lts_1B(-64 + [97:160]);
        rx_lts2_1B = rx_lts_1B([97:160]);

        rx_lts_1C = raw_rx_dec_1C(lts_ind : lts_ind+159); %Extract the first two LTS for CFO
        rx_lts1_1C = rx_lts_1C(-64 + [97:160]);
        rx_lts2_1C = rx_lts_1C([97:160]);

        rx_lts_1D = raw_rx_dec_1D(lts_ind : lts_ind+159); %Extract the first two LTS for CFO
        rx_lts1_1D = rx_lts_1D(-64 + [97:160]);
        rx_lts2_1D = rx_lts_1D([97:160]);
    end

    %Calculate coarse CFO est
    rx_cfo_est_lts_A = mean(unwrap(angle(rx_lts2_A .* conj(rx_lts1_A))));
    rx_cfo_est_lts_A = rx_cfo_est_lts_A/(2*pi*64);

    rx_cfo_est_lts_B = rx_cfo_est_lts_A;
    rx_cfo_est_lts_1A = rx_cfo_est_lts_A;
    rx_cfo_est_lts_1B = rx_cfo_est_lts_A;
    rx_cfo_est_lts_1C = rx_cfo_est_lts_A;
    rx_cfo_est_lts_1D = rx_cfo_est_lts_A;

else
    rx_cfo_est_lts = 0;
    rx_cfo_est_lts_A = rx_cfo_est_lts;
    rx_cfo_est_lts_B = rx_cfo_est_lts;
```

```
    rx_cfo_est_lts_1A = rx_cfo_est_lts;
    rx_cfo_est_lts_1B = rx_cfo_est_lts;
    rx_cfo_est_lts_1C = rx_cfo_est_lts;
    rx_cfo_est_lts_1D = rx_cfo_est_lts;
end

% Apply CFO correction to raw Rx waveforms

rx_cfo_corr_t_A = exp(-1i*2*pi*rx_cfo_est_lts_A*[0:length(raw_rx_dec_A)-1]);
rx_cfo_corr_t_B = exp(-1i*2*pi*rx_cfo_est_lts_B*[0:length(raw_rx_dec_B)-1]);
if(MIMO_FLAG == 0)
    rx_cfo_corr_t_1A = exp(-1i*2*pi*rx_cfo_est_lts_1A*[0:length(raw_rx_dec_1A)-1]);
    rx_cfo_corr_t_1B = exp(-1i*2*pi*rx_cfo_est_lts_1B*[0:length(raw_rx_dec_1B)-1]);

    rx_cfo_corr_t_1C = exp(-1i*2*pi*rx_cfo_est_lts_1C*[0:length(raw_rx_dec_1C)-1]);
    rx_cfo_corr_t_1D = exp(-1i*2*pi*rx_cfo_est_lts_1D*[0:length(raw_rx_dec_1D)-1]);
end

rx_dec_cfo_corr_A = raw_rx_dec_A .* rx_cfo_corr_t_A;
rx_dec_cfo_corr_B = raw_rx_dec_B .* rx_cfo_corr_t_B;
if(MIMO_FLAG == 0)
    rx_dec_cfo_corr_1A = raw_rx_dec_1A .* rx_cfo_corr_t_1A;
    rx_dec_cfo_corr_1B = raw_rx_dec_1B .* rx_cfo_corr_t_1B;

    rx_dec_cfo_corr_1C = raw_rx_dec_1C .* rx_cfo_corr_t_1C;
    rx_dec_cfo_corr_1D = raw_rx_dec_1D .* rx_cfo_corr_t_1D;
end
```

Channel Estimation

MIMO Channel Estimatation

```
lts_ind_TXA_start = mimo_training_ind + 32 ;
lts_ind_TXA_end = lts_ind_TXA_start + 64 - 1;

lts_ind_TXB_start = mimo_training_ind + 32 + 64 + 32 ;
lts_ind_TXB_end = lts_ind_TXB_start + 64 - 1;

rx_lts_AA = rx_dec_cfo_corr_A( lts_ind_TXA_start:lts_ind_TXA_end );
rx_lts_BA = rx_dec_cfo_corr_A( lts_ind_TXB_start:lts_ind_TXB_end );

rx_lts_AB = rx_dec_cfo_corr_B( lts_ind_TXA_start:lts_ind_TXA_end );
rx_lts_BB = rx_dec_cfo_corr_B( lts_ind_TXB_start:lts_ind_TXB_end );

rx_lts_AA_f = fft(rx_lts_AA, N_SC);
rx_lts_BA_f = fft(rx_lts_BA, N_SC);

rx_lts_AB_f = fft(rx_lts_AB, N_SC);
rx_lts_BB_f = fft(rx_lts_BB, N_SC);

rx_lts_1A = rx_dec_cfo_corr_1A( lts_ind_TXA_start:lts_ind_TXA_end );
rx_lts_1B = rx_dec_cfo_corr_1B( lts_ind_TXA_start:lts_ind_TXA_end );
rx_lts_1C = rx_dec_cfo_corr_1C( lts_ind_TXA_start:lts_ind_TXA_end );
rx_lts_1D = rx_dec_cfo_corr_1D( lts_ind_TXA_start:lts_ind_TXA_end );

rx_lts_A_f = fft(rx_lts_1A, N_SC);
rx_lts_B_f = fft(rx_lts_1B, N_SC);
rx_lts_C_f = fft(rx_lts_1C, N_SC);
rx_lts_D_f = fft(rx_lts_1D, N_SC);

if(MIMO_FLAG == 0)
    H_A = rx_lts_A_f./lts_f;
    H_B = rx_lts_B_f./lts_f;
    H_C = rx_lts_C_f./lts_f;
    H_D = rx_lts_D_f./lts_f;
end

H_AA = rx_lts_AA_f./lts_f;
H_BA = rx_lts_BA_f./lts_f;
H_AB = rx_lts_AB_f./lts_f;
H_BB = rx_lts_BB_f./lts_f;
%
```

Rx payload processing, Perform combining for 1X4 and 2X2 separately

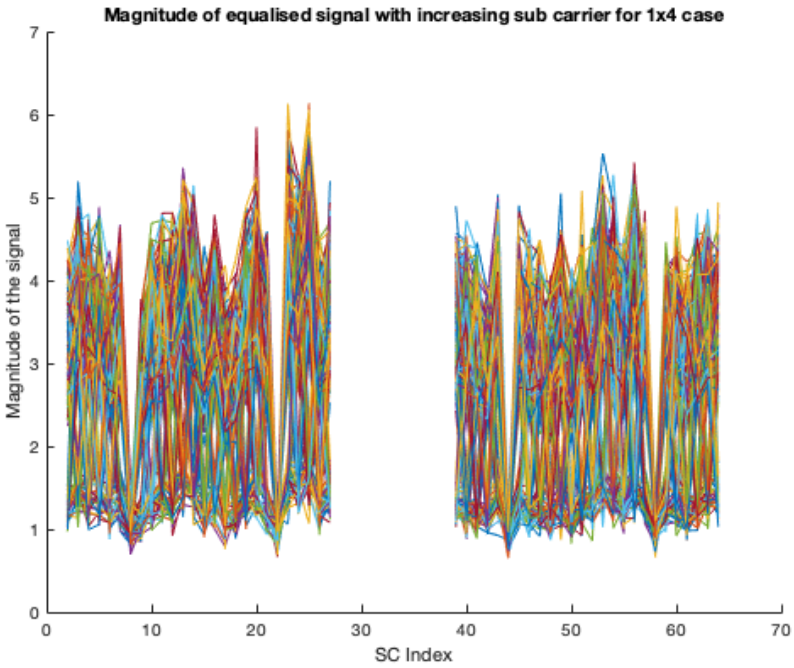
```
% Extract the payload samples (integral number of OFDM symbols following preamble)
val_A = (length(rx_dec_cfo_corr_A)-payload_ind)+1;
val_A = mod(val_A,(N_SC+CP_LEN));
padd_val_A = N_SC+CP_LEN - val_A;
zero_padd_A = zeros(1,padd_val_A);
rx_dec_cfo_corr_A = [rx_dec_cfo_corr_A zero_padd_A];
rx_dec_cfo_corr_A = rx_dec_cfo_corr_A(payload_ind:end);

val_B = (length(rx_dec_cfo_corr_B)-payload_ind)+1;
val_B = mod(val_B,(N_SC+CP_LEN));
padd_val_B = N_SC+CP_LEN - val_B;
zero_padd_B = zeros(1,padd_val_B);
rx_dec_cfo_corr_B = [rx_dec_cfo_corr_B zero_padd_B];
rx_dec_cfo_corr_B = rx_dec_cfo_corr_B(payload_ind:end);

if(MIMO_FLAG == 0)
    val_1A = (length(rx_dec_cfo_corr_1A)-payload_ind)+1;
    val_1A = mod(val_1A,(N_SC+CP_LEN));
    padd_val_1A = N_SC+CP_LEN - val_1A;
    zero_padd_1A = zeros(1,padd_val_1A);
    rx_dec_cfo_corr_1A = [rx_dec_cfo_corr_1A zero_padd_1A];
    rx_dec_cfo_corr_1A = rx_dec_cfo_corr_1A(payload_ind:end);

    val_1B = (length(rx_dec_cfo_corr_1B)-payload_ind)+1;
    val_1B = mod(val_1B,(N_SC+CP_LEN));
    padd_val_1B = N_SC+CP_LEN - val_1B;
    zero_padd_1B = zeros(1,padd_val_1B);
    rx_dec_cfo_corr_1B = [rx_dec_cfo_corr_1B zero_padd_1B];
    rx_dec_cfo_corr_1B = rx_dec_cfo_corr_1B(payload_ind:end);

    val_1C = (length(rx_dec_cfo_corr_1C)-payload_ind)+1;
    val_1C = mod(val_1C,(N_SC+CP_LEN));
    padd_val_1C = N_SC+CP_LEN - val_1C;
    zero_padd_1C = zeros(1,padd_val_1C);
    rx_dec_cfo_corr_1C = [rx_dec_cfo_corr_1C zero_padd_1C];
    rx_dec_cfo_corr_1C = rx_dec_cfo_corr_1C(payload_ind:end);

    val_1D = (length(rx_dec_cfo_corr_1D)-payload_ind)+1;
    val_1D = mod(val_1D,(N_SC+CP_LEN));
    padd_val_1D = N_SC+CP_LEN - val_1D;
    zero_padd_1D = zeros(1,padd_val_1D);
    rx_dec_cfo_corr_1D = [rx_dec_cfo_corr_1D zero_padd_1D];
    rx_dec_cfo_corr_1D = rx_dec_cfo_corr_1D(payload_ind:end);
end

payload_mat_A = reshape(rx_dec_cfo_corr_A,(N_SC+CP_LEN),[]);
payload_mat_B = reshape(rx_dec_cfo_corr_B,(N_SC+CP_LEN),[]);

% Remove the cyclic prefix
payload_mat_noCP_A = payload_mat_A(CP_LEN+[1:N_SC], :);
payload_mat_noCP_B = payload_mat_B(CP_LEN+[1:N_SC], :);

% Take the FFT
syms_f_mat_A = fft(payload_mat_noCP_A, N_SC, 1);
syms_f_mat_B = fft(payload_mat_noCP_B, N_SC, 1);

syms_f_mat_A = syms_f_mat_A(:,[1:N_OFDM_SYMS]);
syms_f_mat_B = syms_f_mat_B(:,[1:N_OFDM_SYMS]);

if(MIMO_FLAG == 0)
    payload_mat_1A = reshape(rx_dec_cfo_corr_1A,(N_SC+CP_LEN),[]);
    payload_mat_1B = reshape(rx_dec_cfo_corr_1B,(N_SC+CP_LEN),[]);

    % Remove the cyclic prefix
    payload_mat_noCP_1A = payload_mat_1A(CP_LEN+[1:N_SC], :);
    payload_mat_noCP_1B = payload_mat_1B(CP_LEN+[1:N_SC], :);

    % Take the FFT
    syms_f_mat_1A = fft(payload_mat_noCP_1A, N_SC, 1);
    syms_f_mat_1B = fft(payload_mat_noCP_1B, N_SC, 1);

    syms_f_mat_1A = syms_f_mat_1A(:,[1:N_OFDM_SYMS]);
    syms_f_mat_1B = syms_f_mat_1B(:,[1:N_OFDM_SYMS]);
```

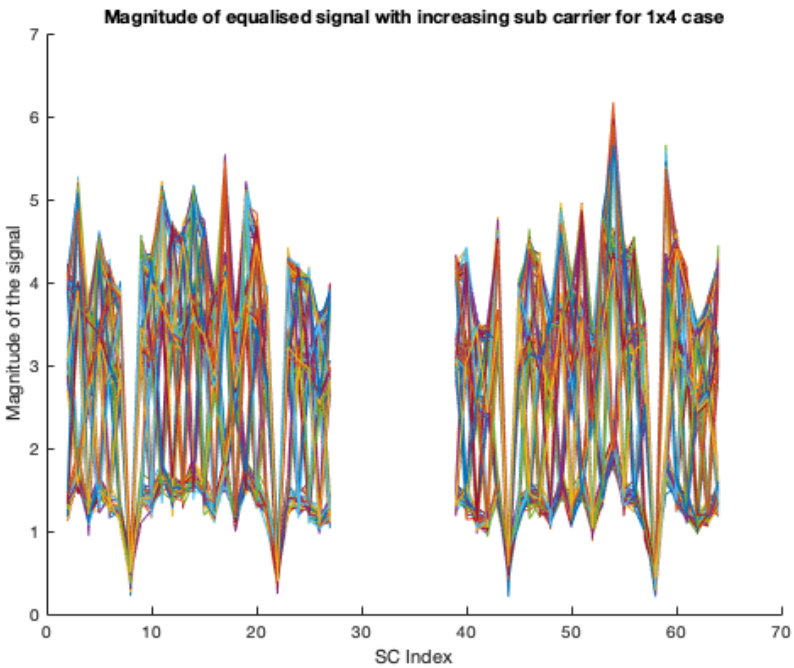
```
    payload_mat_1C = reshape(rx_dec_cfo_corr_1C,(N_SC+CP_LEN),[]);
    payload_mat_1D = reshape(rx_dec_cfo_corr_1D,(N_SC+CP_LEN),[]);

    % Remove the cyclic prefix
    payload_mat_noCP_1C = payload_mat_1C(CP_LEN+[1:N_SC], :);
    payload_mat_noCP_1D = payload_mat_1D(CP_LEN+[1:N_SC], :);

    % Take the FFT
    syms_f_mat_1C = fft(payload_mat_noCP_1C, N_SC, 1);
    syms_f_mat_1D = fft(payload_mat_noCP_1D, N_SC, 1);

    syms_f_mat_1C = syms_f_mat_1C(:,[1:N_OFDM_SYMS]);
    syms_f_mat_1D = syms_f_mat_1D(:,[1:N_OFDM_SYMS]);

end
```

MRC Equalization

```
pilots_mat_A = repmat(pilots_A,1,N_OFDM_SYMS);

if(MIMO_FLAG == 0)
    syms_f_mat_1A_eq = syms_f_mat_1A.*repmat(conj(H_A'),1, N_OFDM_SYMS);
    syms_f_mat_1B_eq = syms_f_mat_1B.*repmat(conj(H_B'),1, N_OFDM_SYMS);
    syms_f_mat_1C_eq = syms_f_mat_1C.*repmat(conj(H_C'),1, N_OFDM_SYMS);
    syms_f_mat_1D_eq = syms_f_mat_1D.*repmat(conj(H_D'),1, N_OFDM_SYMS);

    syms_f_mat_eq_1 = syms_f_mat_1A_eq + syms_f_mat_1B_eq + syms_f_mat_1C_eq + syms_f_mat_1D_eq;

    H_eq_1 = abs(H_A).^2 + abs(H_B).^2 + abs(H_C).^2 + abs(H_D).^2;
    H_eq_1_mat = repmat(H_eq_1',1, N_OFDM_SYMS);

    syms_f_mat_eq_1 = syms_f_mat_eq_1./H_eq_1_mat;
    syms_eq_mat_pilots_1 = syms_f_mat_eq_1;

    syms_eq_mat_pilots_mag_1 = syms_f_mat_eq_1(SC_IND_PILOTS,:).*pilots_mat_A;
    mag_corr_1 = mean(abs(syms_eq_mat_pilots_mag_1));

    syms_f_mat_eq_1 = syms_f_mat_eq_1./repmat(mag_corr_1,N_SC,1);

    figure(4)
    hold on
    for i = 1:size(syms_f_mat_eq_1,2)
        plot(abs(syms_f_mat_eq_1(:,i)))
    end
    hold off
    title('Magnitude of equalised signal with increasing sub carrier for 1x4 case');
    xlabel('SC Index'); ylabel('Magnitude of the signal');

end

syms_f_mat_A_eq = syms_f_mat_A.*repmat(conj((H_AA+H_BA)'),1, N_OFDM_SYMS);
syms_f_mat_B_eq = syms_f_mat_B.*repmat(conj((H_AB+H_BB)'),1, N_OFDM_SYMS);

syms_f_mat_A_eq_pilots = syms_f_mat_A.*repmat(conj((H_AA)'),1, N_OFDM_SYMS);
syms_f_mat_B_eq_pilots = syms_f_mat_B.*repmat(conj((H_AB)'),1, N_OFDM_SYMS);

syms_f_mat_eq_2 = syms_f_mat_A_eq + syms_f_mat_B_eq;
syms_f_mat_eq_2_pilots = syms_f_mat_A_eq_pilots + syms_f_mat_B_eq_pilots;

H_eq_2 = abs(H_AA + H_BA).^2 + abs(H_BB + H_AB).^2;
H_eq_mat_2 = repmat(H_eq_2',1, N_OFDM_SYMS);

H_eq_2_pilots = abs(H_AA).^2 + abs(H_AB).^2;
H_eq_mat_2_pilots = repmat(H_eq_2_pilots',1, N_OFDM_SYMS);

syms_eq_mat_pilots_2 = syms_f_mat_eq_2_pilots./H_eq_mat_2_pilots;
syms_f_mat_eq_2 = syms_f_mat_eq_2./H_eq_mat_2;

%syms_eq_mat_pilots_mag_2 = syms_eq_mat_pilots_2(SC_IND_PILOTS,:).*pilots_mat_A;
%mag_corr_2 = mean(abs(syms_eq_mat_pilots_mag_2));

%syms_f_mat_eq_2 = syms_f_mat_eq_2./repmat(mag_corr_2,N_SC,1);
```

```
figure(5)
hold on
for i = 1:size(syms_f_mat_eq_2,2)
    plot(abs(syms_f_mat_eq_2(:,i)))
end
hold off
title('Magnitude of equalised signal with increasing sub carrier for 1x4 case');
xlabel('SC Index'); ylabel('Magnitude of the signal');


%*This is optional -- SFO correction*

% Equalize pilots
% Because we only used Tx RFA to send pilots, we can do SISO equalization
% here. This is zero-forcing (just divide by chan estimates)
```





SFO and Phase Offset correction

```
if DO_APPLY_SFO_CORRECTION
    % SFO manifests as a frequency-dependent phase whose slope increases
    % over time as the Tx and Rx sample streams drift apart from one
    % another. To correct for this effect, we calculate this phase slope at
    % each OFDM symbol using the pilot tones and use this slope to
    % interpolate a phase correction for each data-bearing subcarrier.

        % Extract the pilot tones and "equalize" them by their nominal Tx values

        % Calculate the phases of every Rx pilot tone
    pilots_f_mat_2 = syms_eq_mat_pilots_2(SC_IND_PILOTS, :);
    pilot_phase_err_2 = (unwrap(angle(pilots_f_mat_2.*pilots_mat_A)));

    pilot_phase_1 = repmat(exp(-1i*pilot_phase_err_2(1,:)),16,1);
    pilot_phase_2 = repmat(exp(-1i*pilot_phase_err_2(2,:)),16,1);
    pilot_phase_3 = repmat(exp(-1i*pilot_phase_err_2(3,:)),16,1);
    pilot_phase_4 = repmat(exp(-1i*pilot_phase_err_2(4,:)),16,1);

    pilot_phase_corr_2 = [pilot_phase_1; pilot_phase_2; pilot_phase_3; pilot_phase_4];

        % Calculate the SFO correction phases for each OFDM symbol

    % Apply the pilot phase correction per symbol

else
        % Define an empty SFO correction matrix (used by plotting code below)
    pilot_phase_sfo_corr = zeros(N_SC, N_OFDM_SYMS);
end

%*This is optional*
% Extract the pilots and calculate per-symbol phase error
if DO_APPLY_PHASE_ERR_CORRECTION
    %pilots_f_mat_2 = syms_eq_mat_pilots_2(SC_IND_PILOTS, :);
    %pilot_phase_err_2 = mean(unwrap(angle(pilots_f_mat_2.*pilots_mat_A)));
    if(MIMO_FLAG == 0)
        pilots_f_mat_1 = syms_eq_mat_pilots_1(SC_IND_PILOTS, :);
        pilot_phase_err_1 = mean(unwrap(angle(pilots_f_mat_1.*pilots_mat_A)));
    end
else
        % Define an empty phase correction vector (used by plotting code below)
    pilot_phase_err_1 = zeros(1, N_OFDM_SYMS);
    pilot_phase_err_2 = zeros(1, N_OFDM_SYMS);
end

%pilot_phase_corr_2 = repmat(exp(-1i*pilot_phase_err_2), N_SC, 1);
if(MIMO_FLAG == 0)
    pilot_phase_corr_1 = repmat(exp(-1i*pilot_phase_err_1), N_SC, 1);
end

% Apply pilot phase correction to both received streams
%syms_f_mat_pc_2 = syms_f_mat_eq_2;
syms_f_mat_pc_2 = syms_f_mat_eq_2 .* pilot_phase_corr_2;
if(MIMO_FLAG == 0)
    syms_f_mat_pc_1 = syms_f_mat_eq_1 .* pilot_phase_corr_1;
end
%syms_f_mat_pc_B = syms_f_mat_eq .* pilot_phase_corr;

%syms_f_mat_pc_A = syms_f_mat_eq;

% Perform combining for MIMO 1X4 and 2X2
% you need to apply the MIMO equalization to each subcarrier separately and then perform combining

if(MIMO_FLAG == 0)
    payload_syms_mat_1 = syms_f_mat_pc_1(SC_IND_DATA, :);
end
payload_syms_mat_2 = syms_f_mat_pc_2(SC_IND_DATA, :);
```

Perform demodulation or demapping post combined symbols

```
rx_syms_case_2 = reshape(payload_syms_mat_2,1,[]);
if(MIMO_FLAG == 0)
    rx_syms_case_1 = reshape(payload_syms_mat_1,1,[]);
end
```

```
% plot the demodulated output rx_syms_case_1 and rx_syms_case_2

if(MIMO_FLAG == 0)
%    figure(6);
%    scatter(real(rx_syms_case_1), imag(rx_syms_case_1),'filled');
%    title(' Signal Space of received bits');
%    xlabel('I'); ylabel('Q');
end
%figure(7);
%scatter(real(rx_syms_case_2), imag(rx_syms_case_2),'filled');
%title(' Signal Space of received bits');
%xlabel('I'); ylabel('Q');


% FEC decoder for the rx_syms_case_1 and rx_syms_case_2


Demap_out_case_1 = demapper(rx_syms_case_1,MOD_ORDER,1);
Demap_out_case_2 = demapper(rx_syms_case_2,MOD_ORDER,1);

% viterbi decoder
trel = poly2trellis(7, [171 133]);
rx_data_final_1= vitdec(Demap_out_case_1,trel,7,'trunc','hard');
rx_data_final_2 = vitdec(Demap_out_case_2,trel,7,'trunc','hard');

% rx_data is the final output corresponding to tx_data, which can be used
% to calculate BER

[number1,ber1(1,1)] = biterr(tx_data_a,rx_data_final_1);
[number2,ber2(1,1)] = biterr(tx_data_b,rx_data_final_2);
%end

%end

%save("ber_1_16qam.mat","ber1");
%save("ber_2_16qam.mat","ber2");
```

Published with MATLAB® R2021b