```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rx.m simulates a digital radio receiver. Overcomes transmitter
% phase noise, transmitter-receiver frequency/pulse-shaping variances,
% transmission channel noise, spectral noise, and intersymbol
% interference on a 4-PAM multiuser TDMA transmission via various
% DSP techniques.
% Successfully decodes instructor-provided transmitter at highest
% level of difficulty with an error rate of 0%.
%
% 4/6/2018, Brian Willis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [decoded_text, y] = Rx(r, rolloff, desired_user)
    %Debug mode: Plots stack with higher number
    %0 = no plots
    %1 = Frequency and time domain plots
    %2 = Adaptive elements plots
    %3 = Filter plots
    DEBUG_MODE = 2;

    %Signal variables
    fs = 850e3;                         %Sampling frequency
    ts = 1/fs;                          %Sampling period
    i_f = 300e3;                        %Intermediate frequency
    s_p = 6.4e-6;                       %Symbol period
    srrc_width = 8;                     %Width of srrc pulse in symbol periods
    t = 0:ts:(length(r)-1)*ts;          %Time vector
    t_off = 0;                          %Timing offset
    M = fs/(1/s_p);                     %Oversampling factor

    %Decoding variables
    preamble = 'A0Oh well whatever Nevermind';
    preamble_s = letters2pam2(preamble);
    num_sym = 875;                      %Number of symbols in each user slot
    preamble_corr_amp = 1e3;            %Amplitude cutoffs for preamble correlation

    %Plot bounds
    fr_y_ub = 6.75e4;                   %Upper-bound for frequency response y-axes
    td_y_ub = 7.5;                      %Upper-bound for time-domain y-axes
    td_x_ub = 180000;                   %Upper-bound for time-domain x-axes

    %Create baseband LPF
    bb_lp_fo = 300;                     %Filter order
    bb_lp_fp = [0 0.24 0.26 1];         %Normalized frequency points
    bb_lp_fa = [1 1 0 0];               %Amplitudes
    bb_lp_h = firpm(bb_lp_fo, bb_lp_fp, bb_lp_fa);

    %Create matched filter
    m_f = fliplr(srrc(srrc_width/2, rolloff, M, t_off));

    theta_correction = 0;                       %If theta converges incorrectly, retry with new tau
    SIGNAL_INVERTED = 1;
```

```matlab
    while(SIGNAL_INVERTED == 1)
        %Carrier recovery variables
        bb_mu1 = 0.1;                       %Algorithm stepsizes
        bb_mu2 = 0.001;
        th1 = zeros(1, length(t));          %Allocate space for theta vectors
        th2 = zeros(1, length(t));
        th1(1) = -0.5;                      %Theta guesses
        th2(1) = -1 + theta_correction;
        f0 = i_f;                           %Carrier frequency guess

        %Preprocess received signal
        q=r.^2;                             %Square nonlinearity
        bp_fo = 320;                        %Filter order
        bp_fp=[0 .55 .57 .59 .61 1];        %Normalized frequency points
        bp_fa = [0 0 1 1 0 0];
        bp_h = firpm(bp_fo, bp_fp, bp_fa);  %BPF design via firpm
        rp = filter(bp_h, 1, q);            %BPF squared signal

        %Recover unknown frequency and phase of bandpassed signal using FFT
        fftrBPF=fft(rp);                                %Spectrum of preprocessed signal
        [~,imax]=max(abs(fftrBPF(1:floor(end/2))));     %Find frequency of max peak
        ssf=(0:length(rp))/(ts*length(rp));             %Frequency vector
        freqS=ssf(imax);                                %Freq at the peak
        [IR,f]=freqz(bp_h,1,length(rp),fs);             %Frequency response of filter
        [~,im]=min(abs(f-freqS));                       %At freq where peak occurs
        phaseBPF=angle(IR(im));                         %Angle of BPF at peak freq

        %Perform phase adjustments
        for k = 1:length(t) - 1
         %Top PLL
         th1(k+1) = th1(k) - bb_mu1*rp(k)*sin(4*pi*f0*t(k) + 2*th1(k) + phaseBPF);
         %Bottom PLL
         th2(k+1) = th2(k) - bb_mu2*rp(k)*sin(4*pi*f0*t(k) + 2*th1(k) + 2*th2(k) + phaseBPF);
        end

        %Begin manipulating signal
        bb = 2*cos(2*pi*i_f*t + th1 + th2).*r';         %Demodulate signal
        bb_lp = conv(bb, bb_lp_h);                      %LPF demodulated signal
        bb_lp = bb_lp(bb_lp_fo/2:end-bb_lp_fo/2);       %Trim leading zeros from convolution
        matched = conv(bb_lp, m_f);                     %Match filter LPF-ed signal

        %Interpolate-decimate match-filtered signal to downsample
        tnow = srrc_width*M + 1;                                                %Starting sample point
        tau = 0;                                                                %Adaptive element
        mu = 0.05;                                                              %Algorithm stepsize
        delta = 0.35;                                                           %Time for derivative
        dnsampled = zeros(1, ceil(length(matched)/M-M));
        tausave = zeros(1, ceil(length(matched)/M-M));
        tausave(1) = tau;
        i = 0;
        while tnow < length(matched)-srrc_width*M
          i = i + 1;
```

```
          dnsampled(i) = interpsinc(matched, tnow+tau, srrc_width);             %Interpolated value at sample point

          x_deltap = interpsinc(matched, tnow+tau+delta, srrc_width);           %Get value to the right
          x_deltam = interpsinc(matched, tnow+tau-delta, srrc_width);           %Get value to the left
          dx = x_deltap - x_deltam;                                             %Calculate numerical derivative

          tau = tau + mu*dx*dnsampled(i);                                       %Output Power algorithm update
          tnow = tnow + M;                                                      %Update sample point
          tausave(i) = tau;                                                     %Save for plotting
        end

        %Amplitude correct and quantize
        dnsampled = 0.72*dnsampled;
        quant = quantalph(dnsampled, [-3 -1 1 3]);

        %Find preamble locations via correlation
        p_c = xcorr(preamble_s, quant);
        p_c = fliplr(p_c);                          %Flip and shift result of correlation
        p_c = p_c(length(quant):end);
        if min(p_c) < -preamble_corr_amp            %Detect if signal is inverted
            SIGNAL_INVERTED = 1;                    %If so, retry with new demodulation tau
            theta_correction = pi;
        else
            SIGNAL_INVERTED = 0;
        end
    end
    p_i = find(p_c > preamble_corr_amp);            %Get indices

    %Create user variables composed of symbols
    user1 = zeros(1, length(p_i)*num_sym);
    user2 = zeros(1, length(p_i)*num_sym);
    user3 = zeros(1, length(p_i)*num_sym);

    %Populate users with slot data
    for i = 1:length(p_i)
        lower = (p_i(i)+length(preamble_s));        %Bounds of user slot in msg
        upper = (p_i(i)+length(preamble_s))+num_sym-1;
        user1(num_sym*(i-1)+1:num_sym*(i-1)+num_sym) = quant(lower:upper);

        lower = (p_i(i)+length(preamble_s)+num_sym);
        upper = (p_i(i)+length(preamble_s))+2*num_sym-1;
        user2(num_sym*(i-1)+1:num_sym*(i-1)+num_sym) = quant(lower:upper);

        lower = (p_i(i)+length(preamble_s)+2*num_sym);
        upper = (p_i(i)+length(preamble_s))+3*num_sym-1;
        user3(num_sym*(i-1)+1:num_sym*(i-1)+num_sym) = quant(lower:upper);
    end

    % Decode users
    user1 = pam2letters2(user1');
    user2 = pam2letters2(user2');
    user3 = pam2letters2(user3');
```

```
    if desired_user == 1
        decoded_text = user1;
    elseif desired_user == 2
        decoded_text = user2;
    elseif desired_user == 3
        decoded_text = user3;
    end

    y = dnsampled;

    %Plots
    if DEBUG_MODE ~= 0
        figure(1);

        %Original signal
        subplot(411);
        plotonlyspec(r, ts);
        ylim([0 fr_y_ub]);
        title('Original Signal Frequency Spectrum');

        %Signal pre-processed (squared, BPF-ed)
        subplot(412);
        plotonlyspec(rp, ts);
        ylim([0 fr_y_ub]);
        title('Preprocessed Signal for PLL');

        %Baseband signal demodulated
        subplot(413);
        plotonlyspec(bb, ts);
        ylim([0 fr_y_ub]);
        title('Demodulated Signal Frequency Spectrum');

        %Signal match-filtered
        subplot(414);
        plotonlyspec(matched, ts);
        ylim([0 fr_y_ub]);
        title('Match-Filtered Signal Frequency Spectrum');

        figure(2);

        %Signal match-filtered (Time domain)
        subplot(511);
        plot(bb_lp, '.');
        xlim([0 td_x_ub]);
        ylim([-td_y_ub td_y_ub]);
        grid on;
        xlabel('Samples');
        ylabel('Amplitude');
        title('Demodulated and LPF-ed Signal');

        %Signal match-filtered (Time domain)
```

```matlab
        subplot(512);
        plot(matched, '.');
        xlim([0 td_x_ub]);
        ylim([-td_y_ub td_y_ub]);
        grid on;
        xlabel('Samples');
        ylabel('Amplitude');
        title('Match-Filtered Signal');

        %Signal interpolator-decimated (Time domain)
        subplot(513);
        plot(dnsampled, '.');
        xlim([0 td_x_ub/M]);
        ylim([-td_y_ub td_y_ub]);
        grid on;
        xlabel('Samples');
        ylabel('Amplitude');
        title('Interpolator-Decimated Signal');

        %Signal quantized
        subplot(514);
        plot(quant, '.');
        xlim([0 td_x_ub/M]);
        ylim([-td_y_ub td_y_ub]);
        grid on;
        xlabel('Symbols');
        ylabel('Amplitude');
        title('Quantized Signal');

        %Locations of preambles
        subplot(515);
        plot(p_c);
        xlim([0 td_x_ub/M]);
        grid on;
        xlabel('Indices');
        ylabel('Amplitude');
        title('Locations of Preambles in Signal');

        if DEBUG_MODE == 2 || DEBUG_MODE == 3
            figure(3);

            %Plot phase offsets for demodulator
            subplot(311);
            plot(th1);
            xlabel('Iterations');
            ylabel('Angle (Radians)');
            title('Demodulator \theta_1');
            grid on;

            subplot(312);
            plot(th2);
            xlabel('Iterations');
```

```
            ylabel('Angle (Radians)');
            title('Demodulator \theta_2');
            grid on;

            %Plot interpolator-decimator convergence
            subplot(313);
            plot(tausave);
            xlim([0 td_x_ub/M]);
            xlabel('Iterations');
            ylabel('Amplitude');
            title('Interpolator-Decimator \tau');
            grid on;

            if DEBUG_MODE == 3
                figure(4);
                %Plot LPF characteristics
                freqz(bb_lp_h);
                title('Demodulation LPF');
                grid on;

                figure(5);
                %Plot BPF characteristics
                freqz(bp_h);
                title('Preprocessing BPF');
                grid on;

                figure(6);
                %Plot matched filter
                plot(m_f);
                title('Matched Filter');
                grid on;
            end
        end
    end
end

%Plotonlyspec(x, ts) plots just the spectrum of the signal x
%ts = time (in seconds) between adjacent samples in x
function plotonlyspec(x, ts)
    N=length(x);                                    %Length of the signal x
    ssf=(ceil(-N/2):ceil(N/2)-1)/(ts*N);            %Frequency vector
    fx=fft(x(1:N));                                 %Do DFT/FFT
    fxs=fftshift(fx);                               %Shift it for plotting
    plot(ssf,abs(fxs));                             %Plot magnitude spectrum
    xlabel('Frequency (Hz)');                       %Label the axes
    ylabel('Magnitude');
    grid on;
end
```