Table of Contents


```
%RTL_CNOTCH
%
% Adaptive complex notch filter and RTL-SDR.
% First-order complex all-pass filter and gradient descent
%
% By R.W.

clear all, close all

% Script to set some essential parameters you need to figure yourself:
% - low-pass filter FLOW and decimator factor NDEC
% - Coefficients for the adaptive notch filter:
% - Filter parameter proportional to the bandwith: ALPHA (assumed by
 us?)
% - step size MU ( trial )

%rw_fmrx_init

%NDEC = FESR/40e+03;
NDEC = 6;
% Manually Inputting the decimation factor as we know that
% we should get down the sampling freq from 240k to 40~44k
fmax = 40e+03;
nyq = fmax/2;
```

Task & Explanation: Describe briefly the algorithm used in the filter design function The algorithm used is 48th order with frequency magnitude characterstics specified by how we set out cutoff frequency, in the second vector containing the desired magnitude response at each of the points specified in the vector where cutoff frequency is specifies. Frequency sampling-based FIR filter design is adopted where FLOW holds the values for filter coeffcients which consequently is used to filter

```
FLOW = fir2(48, [0 15e3/nyq 17e3/nyq 1], [1 1 0 0]);%filter esign
%FLOW = fir1(1, (FESR+ 15e3)/(2*FESR));
%freqz(FLOW,1);
h=fvtool(FLOW,1);



%adaptive filter parameters, These parameters were selected after
 couple of
%trials to get the ideal Frequency Offset
ALPHA = 0.8;
MU = 0.02 ;
```

Radio parameters

FM transmitter

```
%expFreq = 89.5e6;
% YLE 1
%expFreq = 87.9e6;
% YLE Puhe
expFreq = 103.7e6;
% YLE Radio Suomi
%expFreq = 94e6;

% Front-end sampling rate etc. Change the numbers at will
FESR = 240e3;
nSample = 4096*6;
nFrame = 500; %12e2/8;

hSDRrRx = comm.SDRRTLReceiver(...
    'RadioAddress', '0',...
    'CenterFrequency',     expFreq, ...
    'EnableTunerAGC',      true, ...
    'SampleRate',          FESR, ...
    'SamplesPerFrame',     nSample, ...
    'FrequencyCorrection', -40, ...
    'OutputDataType',      'double')
fprintf('\n')

hSpectrumAnalyzer = dsp.SpectrumAnalyzer(...
    'Name',              'Received signal',...
    'Title',             'Received signal', ...
    'SpectrumType',      'Power density',...
    'FrequencySpan',     'Full', ...
    'SampleRate',        FESR, ...
    'YLimits',           [-60,0],...
    'SpectralAverages', 10, ...
    'FrequencySpan',     'Start and stop frequencies', ...
    'StartFrequency',    -50e3, ...
    'StopFrequency',     50e3,...
    'Position',          figposition([50 30 30 40]));

% Check out low-pass filtered signal if necessary
hSA3 = dsp.SpectrumAnalyzer(...
'Name','Frequency corrected signal','Title','frequency signal',...
'SampleRate',FESR/NDEC,...
'FrequencySpan',    'Start and stop frequencies', ...
'StartFrequency',-15e3,'StopFrequency',15e3,...
'YLimits', [-60,0], 'SpectralAverages',10);

hAudio = audioDeviceWriter(FESR/NDEC,'BufferSize',nSample*2/NDEC);
getAudioDevices(hAudio);

% Time display object
hTime = dsp.TimeScope(...
```

```
 'SampleRate',  FESR, ...
 'NumInputPorts', 1,...
 'ShowGrid', 1, 'ShowLegend',1,...
 'TimeSpanSource', 'auto',...
 'TimeSpanOverrunAction', 'scroll',...
    'AxesScaling', 'Auto',...
 'Title', 'Frequency estimate (Rad)',...
 'PlotAsMagnitudePhase', 0);
% 'YLimits',[-1,1],...
```

```
hSDRrRx =

  comm.SDRRTLReceiver with properties:

           RadioAddress: '0'
        CenterFrequency: 103700000
         EnableTunerAGC: true
             SampleRate: 240000
         OutputDataType: 'double'
        SamplesPerFrame: 24576
    FrequencyCorrection: -40
        EnableBurstMode: false
```

Stream Processing

```
if ~isempty(sdrinfo(hSDRrRx.RadioAddress))

 fprintf('Receive time %f [s]   \n', nSample/FESR*nFrame)

    % Dimension of the vector after down-sampling
    ndim = nSample/NDEC;
    % Vectors to store the results. They must column vector
    evec = zeros(ndim,1);
    theta = zeros(ndim,1);
 sigvec = zeros(ndim,1);
 memo = zeros(1,length(FLOW)-1);
    xprev = 0; % notch filer's internal state
    thprev = 0; % Initial frequency
    for iFrame = 1 : nFrame
        rxSig = step(hSDRrRx);
        rxSig = rxSig - mean(rxSig);  % Remove DC component

        % Low-pass filter and down-sample to make loop to run at lower
        % sampling rate
        [rxfilt,memo] = filter(FLOW,1,rxSig,memo);
        rxdec = rxfilt(1:NDEC:end);
        hSpectrumAnalyzer(rxfilt);

        % Adaptive notch filter loop
        for kk=1:ndim
```

```
            [evec(kk),theta(kk),xprev,xprev_buffer(kk)]=...
                rw_adaptive_notch(ALPHA,MU,rxdec(kk),xprev,thprev);
            thprev = theta(kk);
        end

        rxdemod = rw_demodulate(theta(end),rxdec,ndim);
        filter_adaptive_signal = filter(theta,1,rxdec);
        %fvtool(filter_adaptive_signal);
        % Time evolution of the frequency estimate
        hTime(theta);
        % Demodulated/frequency corrected signal
        hSA3(rxdemod);

        %nUnderrun = hAudio(real(rxdemod));
        %if nUnderrun > 0
        %    fprintf('Audio player queue underrun by %d samples.
\n',nUnderrun);
        %end
    end
else
    warning(message('SDR:sysobjdemos:MainLoop'))
end
fprintf('The last estimate of the angular frequency is %f'
 ,theta(end));
%Notch filter’s magnitude response using fvtool()
```

Receive time 51.200000 [s]

Release all System objects

```
release(hSDRrRx);
clear hSDRrRx
release(hAudio)

f = linspace(-1,1,100);
z = exp(1i*f);
ALPHA = 0.001;% The parameter ? controls the width of the notch
%At SMT6,Otaniemi Location, at Alpha = 0.002 with Last Estimate being
 -12.94, the
%notch was evident at -0.375
%Implementing the paradigm of G(z) from Slide 21
%G1 = Numerator part of G(z)
%G1 = Denominator part of G(z)
G1 = exp(1i*theta(end))*z.^-1-ALPHA;
G2 = 1-ALPHA*exp(1i*theta(end))*z.^1;
G = 0.5*(1-G1./G2); % as G(z)
figure
plot(f,10*log(abs(G)))
xlabel('Normalized frequency')
ylabel('Magnitude response (dB)')
```

Functions for Adaptive Notch and exponential demodulation

```
function [error, theta, xprev,xprev_buffer] =
 rw_adaptive_notch(ALPHA,MU,rxdec,xprev,thprev)

xprev_buffer = xprev;
error = (-(0.5*(sqrt(1-ALPHA^2)*xprev))) + ((1+ALPHA)/2)*(rxdec);
theta  = thprev + (MU* imag(error * conj(xprev)));
xprev = (exp(1i*thprev)*ALPHA*xprev )+ (exp(1i*thprev))*(sqrt(1-
ALPHA^2))*rxdec;

end

function [demod] = rw_demodulate(theta,rxdec,ndim)


demod = rxdec.*exp((-1i)*theta*(0:ndim-1)');

end
```

```
The last estimate of the angular frequency is -1.359998
```

Published with MATLAB® R2018a