```
close all; clear all; clc;
% Bipolar, Baseband PAM transmitter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Pulse_duration_in_seconds = 0.2;
Time_per_symbol = Pulse_duration_in_seconds/2; % Assignment requirement
Symbols_sent_per_second = 10;
Numbers_sent_per_symbol = 50; % Upsample rate- adding in 0s for padding
Numbers_sent_per_second = Numbers_sent_per_symbol * Symbols_sent_per_second;
Tp = Pulse_duration_in_seconds / 2; % Half of the pulse duration
SPB = Numbers_sent_per_symbol;
dt = 1 / Numbers_sent_per_second; % Sampling period
Ts = 1 / Symbols_sent_per_second; % Symbol period

% Generate pulse SINC
pulse = sinc((-Numbers_sent_per_symbol:Numbers_sent_per_symbol)/
Numbers_sent_per_symbol);
pulse_energy = sum(pulse.*pulse);
pulse = pulse / pulse_energy^0.5; % Normalize the pulse


%Generate Pulse SRRC
%pulse = SRRC(0.5,Numbers_sent_per_symbol);
%pulse_energy = sum(pulse.*pulse);
%pulse = pulse / pulse_energy^0.5; % Normalize the pulse



% Encode DATA
data1 = text2bits('I Love MATLAB  ');
data2 = text2bits('Sike Python    ');
data3 = text2bits('is better      ');

f1 = 2 * pi * 20;
f2 = 2 * pi * 40;
f3 = 2 * pi * 60;

[y1,y1BUS] = encode_NHern(data1, pulse, f1, Numbers_sent_per_symbol, dt);
[y2,y2BUS] = encode_NHern(data2, pulse, f2, Numbers_sent_per_symbol, dt);
[y3,y3BUS] = encode_NHern(data3, pulse, f3, Numbers_sent_per_symbol, dt);
y = y1 + y2 + y3;

figure(86)
subplot(2, 1, 1)
hold on
title('Modulated y(t) with SINC Before Upsample')
xlabel('Time (s)')
ylabel('Amplitude')

xlim([0 2])
plot(dt*(1:length(y1BUS)), y1BUS)
plot(dt*(1:length(y2BUS)), y2BUS)
plot(dt*(1:length(y3BUS)), y3BUS)
```

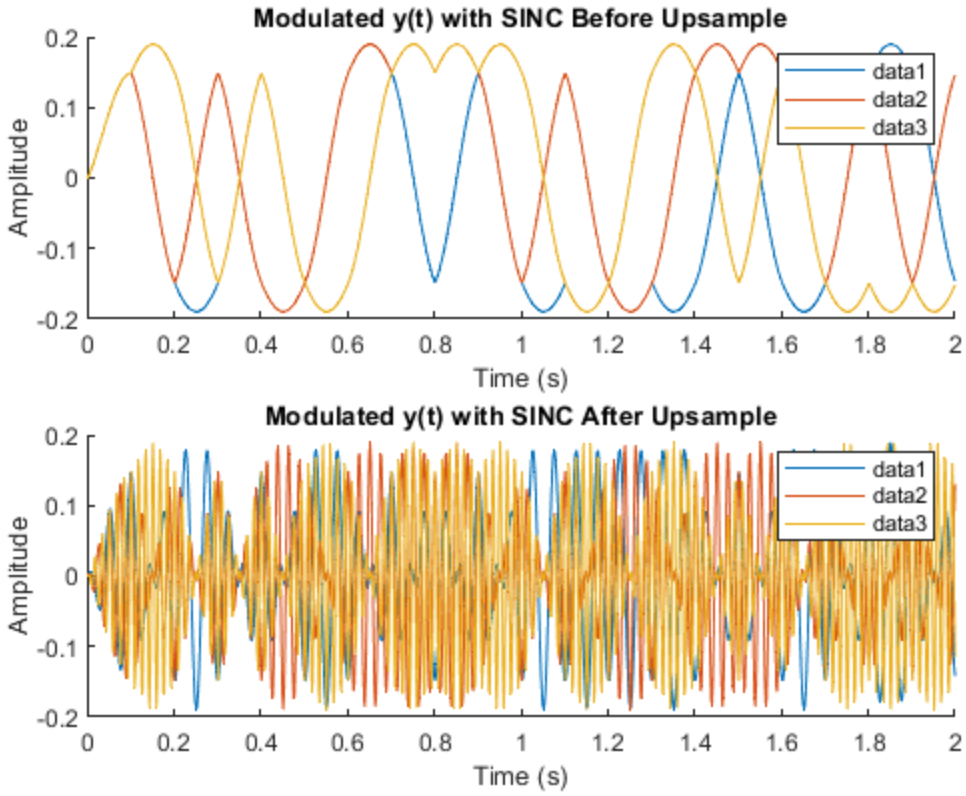
```
legend('data1','data2','data3')

subplot(2, 1, 2)
hold on
title('Modulated y(t) with SINC After Upsample')
xlabel('Time (s)')
ylabel('Amplitude')
xlim([0 2])

plot(dt*(1:length(y1)), y1)
plot(dt*(1:length(y2)), y2)
plot(dt*(1:length(y3)), y3)
legend('data1','data2','data3')


figure(88)
hold on
title('Fourier Analysis of SINC Encoded Message at Bandwith of 20Hz')
xlabel('Frequency (Hz)')
ylabel('Magnitude')
xlim([0, 80])
legend('20Hz,30Hz', '40Hz')
plot(abs(fft(y1, 500)))
plot(abs(fft(y2, 500)))
plot(abs(fft(y3, 500)))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Zero-noise case
r = y;  % y is sent, r is received, identical with 0 noise
data_out1 = decode(r, pulse, f1, Numbers_sent_per_symbol, dt, 1);
data_out2 = decode(r, pulse, f2, Numbers_sent_per_symbol, dt, 1);
data_out3 = decode(r, pulse, f3, Numbers_sent_per_symbol, dt, 1);
% Translate to ascii text
message_out1 = binvector2str(data_out1)
message_out2 = binvector2str(data_out2)
message_out3 = binvector2str(data_out3)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Measure noise vs accuracy, first using the sinc (as we had before)
noise = randn(1, length(y)); % For consistent testing, use the same noise each
 time

sigmaList = 0:0.1:2;
errorRate = zeros(1,length(sigmaList));
SNR = zeros(1,length(sigmaList));
for i = 1:length(sigmaList)
 sigma = sigmaList(i);

 % y is sent, r is received, so add noise here
 r = y + sigma * noise;
 data_out1 = decode(r, pulse, f1, Numbers_sent_per_symbol, dt, 0);
 data_out2 = decode(r, pulse, f2, Numbers_sent_per_symbol, dt, 0);
 data_out3 = decode(r, pulse, f3, Numbers_sent_per_symbol, dt, 0);
```

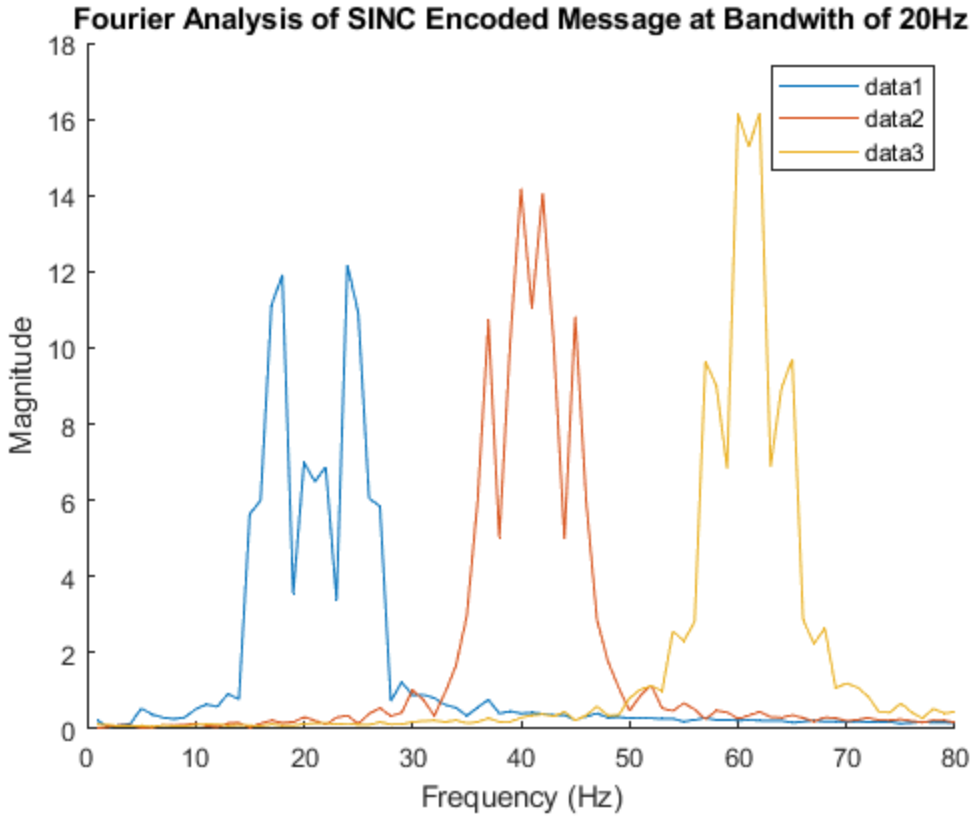
```
 % Calculate Error Rate
 total_bits_sent = length(data_out1) + length(data_out2) + length(data_out3);
 msg1_errors = sum(data_out1 ~= data1);
 msg2_errors = sum(data_out2 ~= data2);
 msg3_errors = sum(data_out3 ~= data3);
 total_errors = msg1_errors + msg2_errors + msg3_errors;
 errorRate(i) = total_errors / total_bits_sent;
 % fprintf('Error rate: %.2f%%\n', errorRate(i) * 100);
 signal_energy = sum(y.*y);
 noise_energy = sigma^2 * length(y);
 SNR(i) = 10*log10(signal_energy / noise_energy);
end

% Measure noise vs accuracy, now using the SRRC
errorRate_SRRC = zeros(1,length(sigmaList));
SNR_SRRC = zeros(1,length(sigmaList));

pulse = SRRC(0.5,Numbers_sent_per_symbol);
pulse_energy = sum(pulse.*pulse);
pulse = pulse / pulse_energy^0.5; % Normalize the pulse

y1 = encode_NHern(data1, pulse, f1, Numbers_sent_per_symbol, dt);
y2 = encode_NHern(data2, pulse, f2, Numbers_sent_per_symbol, dt);
y3 = encode_NHern(data3, pulse, f3, Numbers_sent_per_symbol, dt);
y = y1 + y2 + y3;

for i = 1:length(sigmaList)
 sigma = sigmaList(i);

 % y is sent, r is received, so add noise here
 r = y + sigma * noise;
 data_out1 = decode(r, pulse, f1, Numbers_sent_per_symbol, dt, 0);
 data_out2 = decode(r, pulse, f2, Numbers_sent_per_symbol, dt, 0);
 data_out3 = decode(r, pulse, f3, Numbers_sent_per_symbol, dt, 0);

 % Calculate Error Rate
 total_bits_sent = length(data_out1) + length(data_out2) + length(data_out3);
 msg1_errors = sum(data_out1 ~= data1);
 msg2_errors = sum(data_out2 ~= data2);
 msg3_errors = sum(data_out3 ~= data3);
 total_errors = msg1_errors + msg2_errors + msg3_errors;
 errorRate_SRRC(i) = total_errors / total_bits_sent;

 % fprintf('Error rate: %.2f%%\n', errorRate(i) * 100);
 signal_energy = sum(y.*y);
 noise_energy = sigma^2 * length(y);
 SNR_SRRC(i) = 10*log10(signal_energy / noise_energy);
end

% Plot the Error Rate vs. Sigma
figure(23);
hold on
plot(sigmaList, errorRate);
plot(sigmaList, errorRate_SRRC);
```

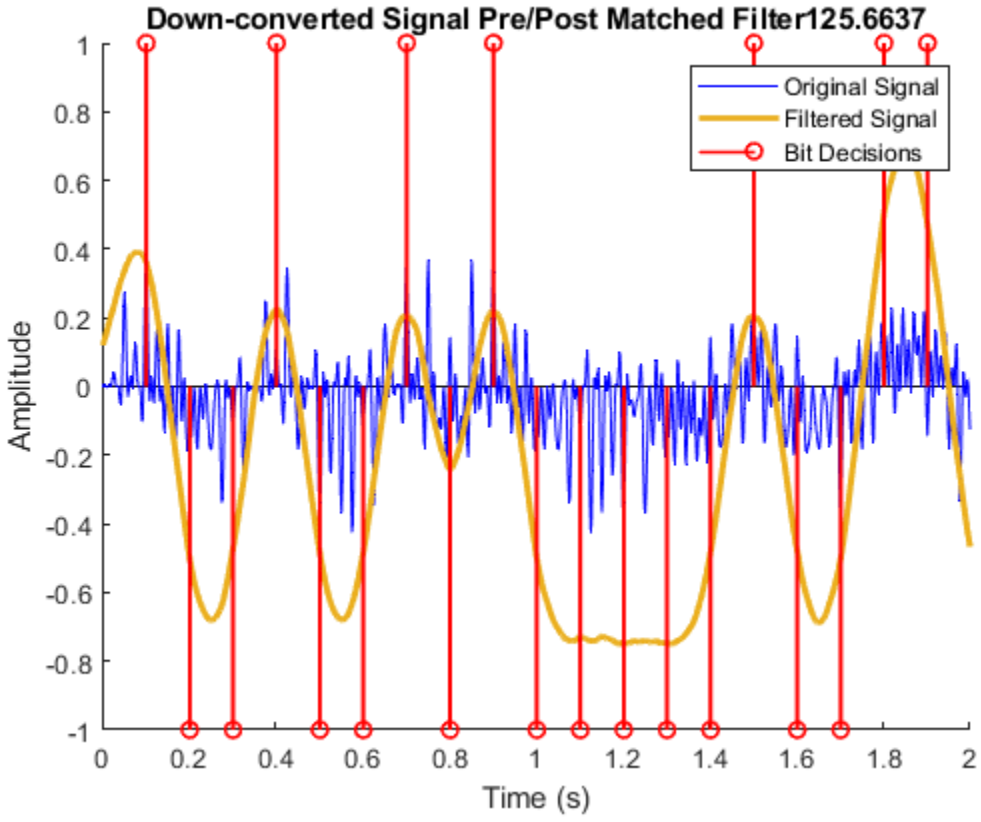
```
title('Error Rate vs. Noise Level');
xlabel('Sigma');
ylabel('Error Rate');
legend('SINC','SRRC')

% Plot the Error Rate vs. SNR
figure(24);
hold on
plot(SNR, errorRate);
plot(SNR_SRRC, errorRate_SRRC);
title('Error Rate vs. SNR');
xlabel('10 log( signal energy / noise energy )');
ylabel('Error Rate');
legend('SINC','SRRC')

function [y, y_before_upsample] = encode_NHern(data, pulse, modfreq,
 Numbers_sent_per_symbol, dt)
 % Modulation Scheme
 x = lut(data, [0   1], [-1  1]);
 Number_of_numbers_sent = Numbers_sent_per_symbol * length(data);
 t = 0:(Number_of_numbers_sent+length(pulse)-2); % room for convolution
 t = dt * t; % Convert from index of number sent to time (in seconds) sent
 %pulse=sinc(t/Ts);
 % Oversample and then Convolute
 UpSample_data = oversample(x, Numbers_sent_per_symbol);
 y_before_upsample = conv(UpSample_data, pulse);
 %  Upconvert
 z = cos(modfreq*t(1:length(y_before_upsample)));
 y = y_before_upsample .* z;
end

function data_out = decode(r, pulse, modfreq, Numbers_sent_per_symbol, dt,
 do_plot)
 % DownConvert each frequency carried
 t = dt*(1:length(r));
 z = cos(modfreq*t);

 %r = r+ .3 * randn(1, length(r)); % Add noise to sent signal; % Add noise to
 sent signal
 r = r.*z;

 % Use the pulse as a filter for what we received
 rFiltered = conv(r, pulse);

 % Find Bit Indices
 originial_length = length(r) - (length(pulse) - 1);
 symbols_sent = originial_length / Numbers_sent_per_symbol;
 sample_points = length(pulse) + Numbers_sent_per_symbol*(0:symbols_sent-1);

 % Bit Decisions
 data_out = (rFiltered(sample_points) > 0);

if do_plot
  figure(randi(10000))
```

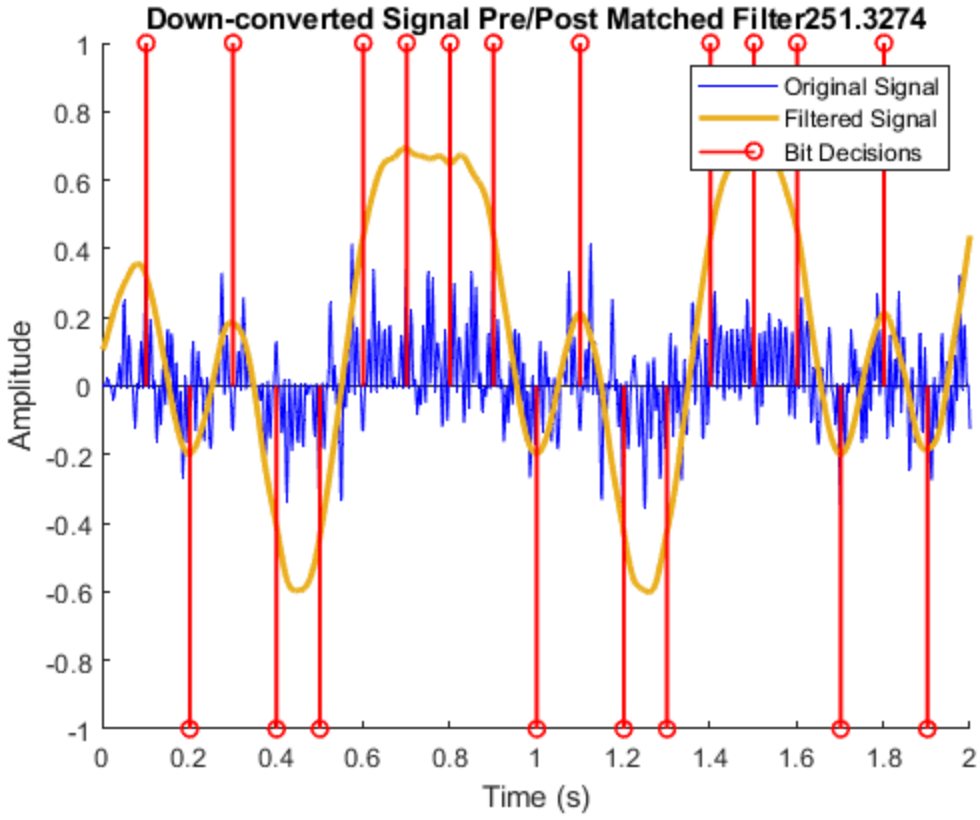
```
  title("Down-converted Signal Pre/Post Matched Filter"+ modfreq)
  xlabel('Time (s)')
  ylabel('Amplitude')
  xlim([0, 2])
  hold on;
  plot(t, r, 'b');
  rFiltIndex = 1:length(r);
  rFiltIndex = rFiltIndex + floor(length(pulse)/2);
  plot(t, rFiltered(rFiltIndex), 'Color', '#EDB120', 'LineWidth', 2);
  stem(t(sample_points - floor(length(pulse)/2)),
 (2*data_out)-1, 'r', 'LineWidth', 1.2);
  legend('Original Signal', 'Filtered Signal', 'Bit Decisions')
end

end

function h = SRRC(alpha, N)
% alpha = rolloff factor
% N = 1/2 of the quantity of numbers that will be sent per pulse
% Lp = number of pulses we'd like to leave room for
n = [-N : N] + 10^-9;
% Add epsilon to the n values to avoid numerical problems
% Plug into time domain formula for the SRRC pulse shape
h = (1 / sqrt(N)) * (sin(pi * (1 - alpha) * n / N) + ...
   (4 * alpha * n / N) .* cos(pi * (1 + alpha) * n / N)) ...
   ./ (pi * n / N .* (1 - (4 * alpha * n / N).^2));
end

function upsampled_data = oversample(data, upsample_rate)
 num_symbols = length(data);
 % Initialize an array with zeros for the upsampled data
 upsampled_data = zeros(1, num_symbols * upsample_rate);
  % Iterate through the symbols and insert them into the upsampled_data array
 for i = 0:(num_symbols - 1)
     % Copy the signal of -1 and +1 into a list of zeros with correct spacing
     upsampled_data(upsample_rate * i + 1) = data(i + 1);
 end
end
```

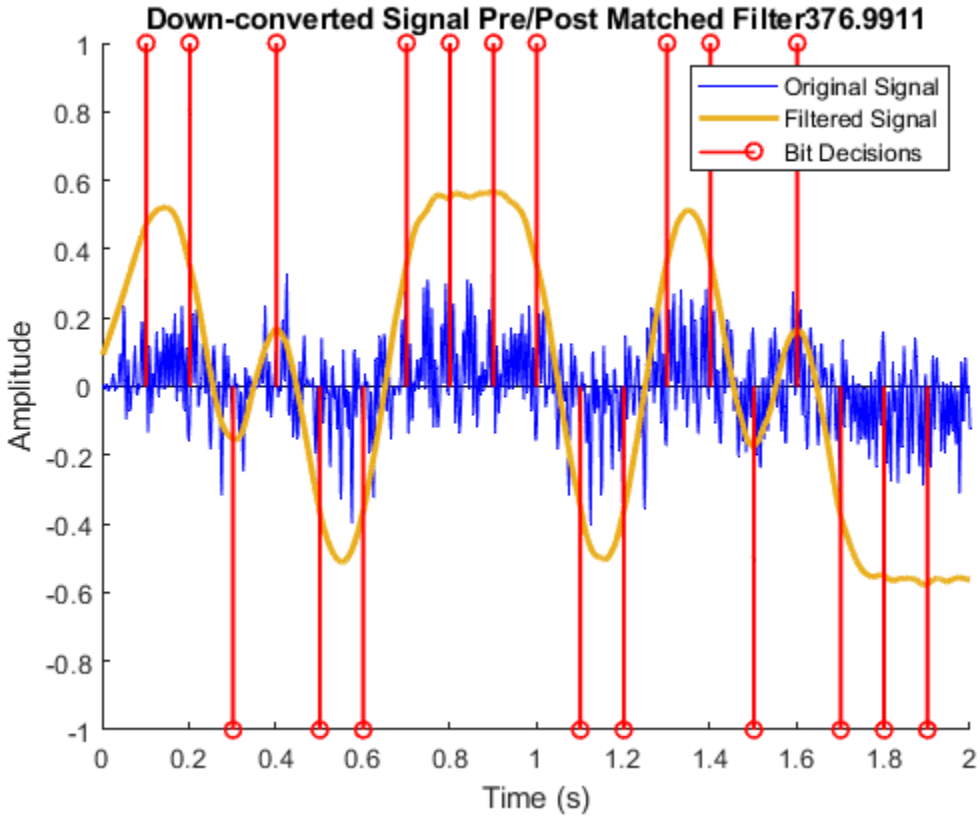
```
Warning: Ignoring extra legend entries.

message_out1 =

    'I Love MATLAB  '


message_out2 =

    'Sike Python    '


message_out3 =

    'is better      '
```

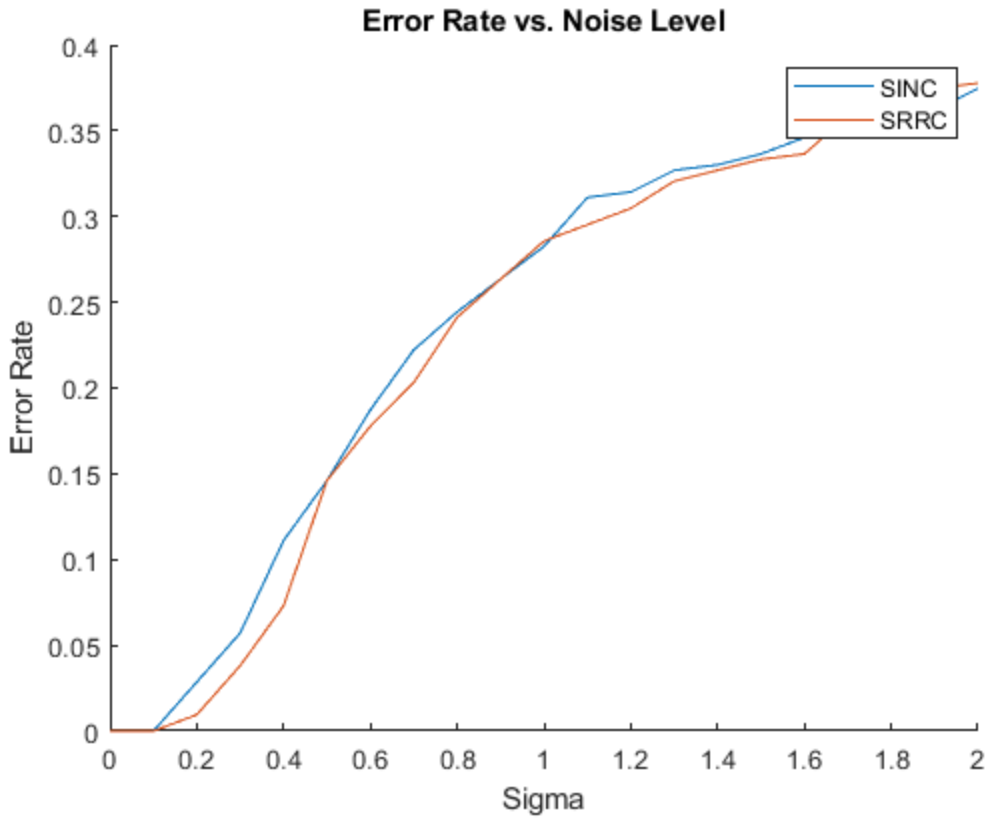

Modulated y(t) with SINC Before Upsample
Amplitude
Time (s)
data1
data2
data3
Modulated y(t) with SINC After Upsample
Amplitude
Time (s)
data1
data2
data3

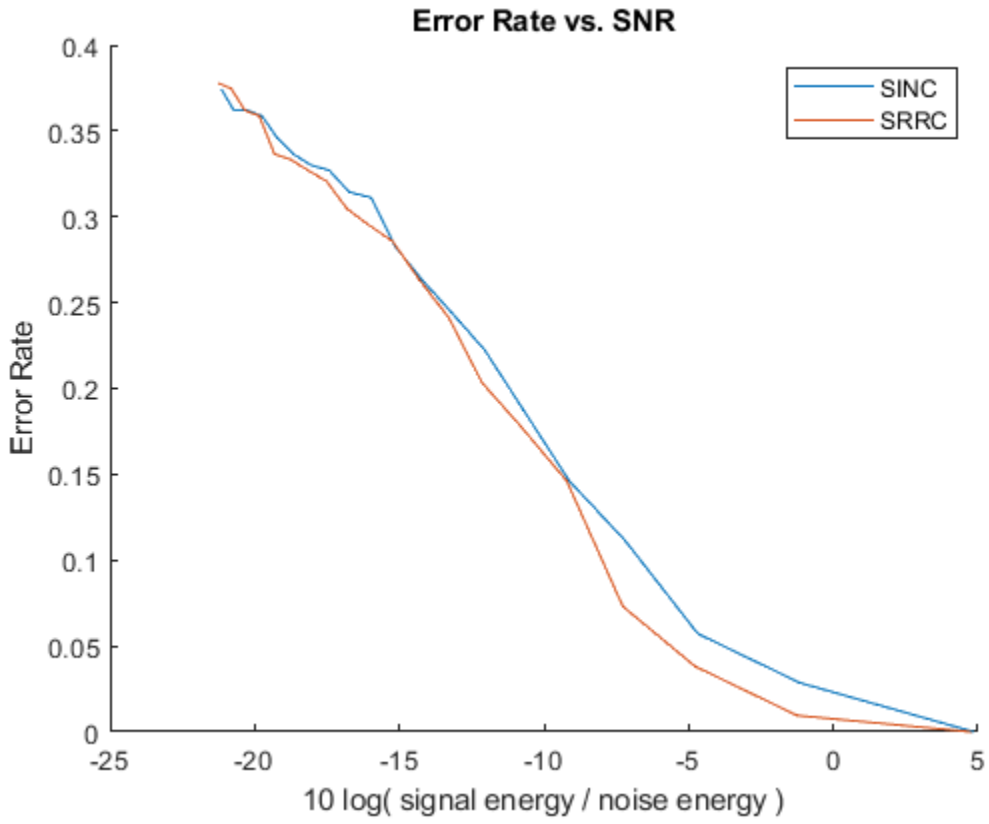

Fourier Analysis of SINC Encoded Message at Bandwith of 20Hz
data1
data2
data3
Magnitude
Frequency (Hz)
0
2
4
6
8
10
12
14
16
18
0
10
20
30
40
50
60
70
80


Down-converted Signal Pre/Post Matched Filter125.6637
Original Signal
Filtered Signal
Bit Decisions
Amplitude
Time (s)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Down-converted Signal Pre/Post Matched Filter251.3274
Original Signal
Filtered Signal
Bit Decisions
Amplitude
Time (s)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Down-converted Signal Pre/Post Matched Filter376.9911
Original Signal
Filtered Signal
Bit Decisions
Amplitude
Time (s)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Error Rate vs. Noise Level
SINC
SRRC
Error Rate
Sigma
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Error Rate vs. SNR
SINC
SRRC
Error Rate
10 log(signal energy / noise energy)
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
-25
-20
-15
-10
-5
0
5

Published with MATLAB® R2022b