OOK SYSTEM

Rochester Institute of Technology

By: Rahul Singh Gulia
Instructor: Dr. Miguel Bazdresch

1. INTRODUCTION

2. OOK TRANSMITTER

2.1 Parameters defined

For the below mentioned parameters, we will design an On-Off Keying (OOK) transmitter (Tx) with Square Root Raised Cosine (SRRC) pulse.

i. Pulse interval, T
ii. Average symbol energy, E_s for the OOK constellation
iii. Tx carrier frequency, f_c
iv. Initial phase of the Tx carrier, Φ
v. SRRC pulse excess Bandwidth (BW), β

```
T=1;    % pulse interval or time period, and Symbol period = 1/T
Es = 1;   % Average energy/symbol in the OOK constellation
fc = 10;    % Carrier frequency in Hz
phi = 0.8;
beta = 0.5;   % SRRC pulse excess bandwidth
k = 1;
```

2.2 Sampling frequency

We must select a suitable sampling frequency to make sure that we avoid the aliasing in our model simulations.

```
fs = 100;    % Assuming sampling frequency
ts = 1/fs;  % Sampling period
```

2.3 SRRC pulse generation

Square Root Raised Cosine (SRRC) is often used to have minimum Intersymbol Interference (ISI). We generate it using the function srrcpulse.m.

```
[p, tt] = srrcpulse(beta, T, ts, 20);
plot(tt,p);
grid on;
% ylim([-0.08,0.4]);
title('SRRC pulse shape; T = 1'); xlabel('Time (s)');
```

2.4 OOK Constellation

On-Off keying (OOK) constellation C = {0,A} with a modulation index of 1 is the simplest digital modulation to implement.

```
N = randi(2,1,100)    % generating 100 random symbols
d = sqrt(2*Es);
C = [0 d];      % Generating the constellation
```

```
syms = C(N);   % Generating N symbols on the constellation
syms_up = upsample(syms, fs*T);


bb = conv(p, syms_up);  % baseband OOK signal
```

2.5 Carrier generation

```
t1 = 0:ts:ts*(length(bb)-1);
carrier = cos(2*pi*fc*t1 + phi);
```

2.6 Passband OOK signal generation

```
ook = bb.*carrier;
```

2.7 Passband signal spectrum Analysis

The spectrum of a signal *s(t)* is a frequency-shifted version of the baseband signal. The shape of the spectrum depends on the choice of pulse.

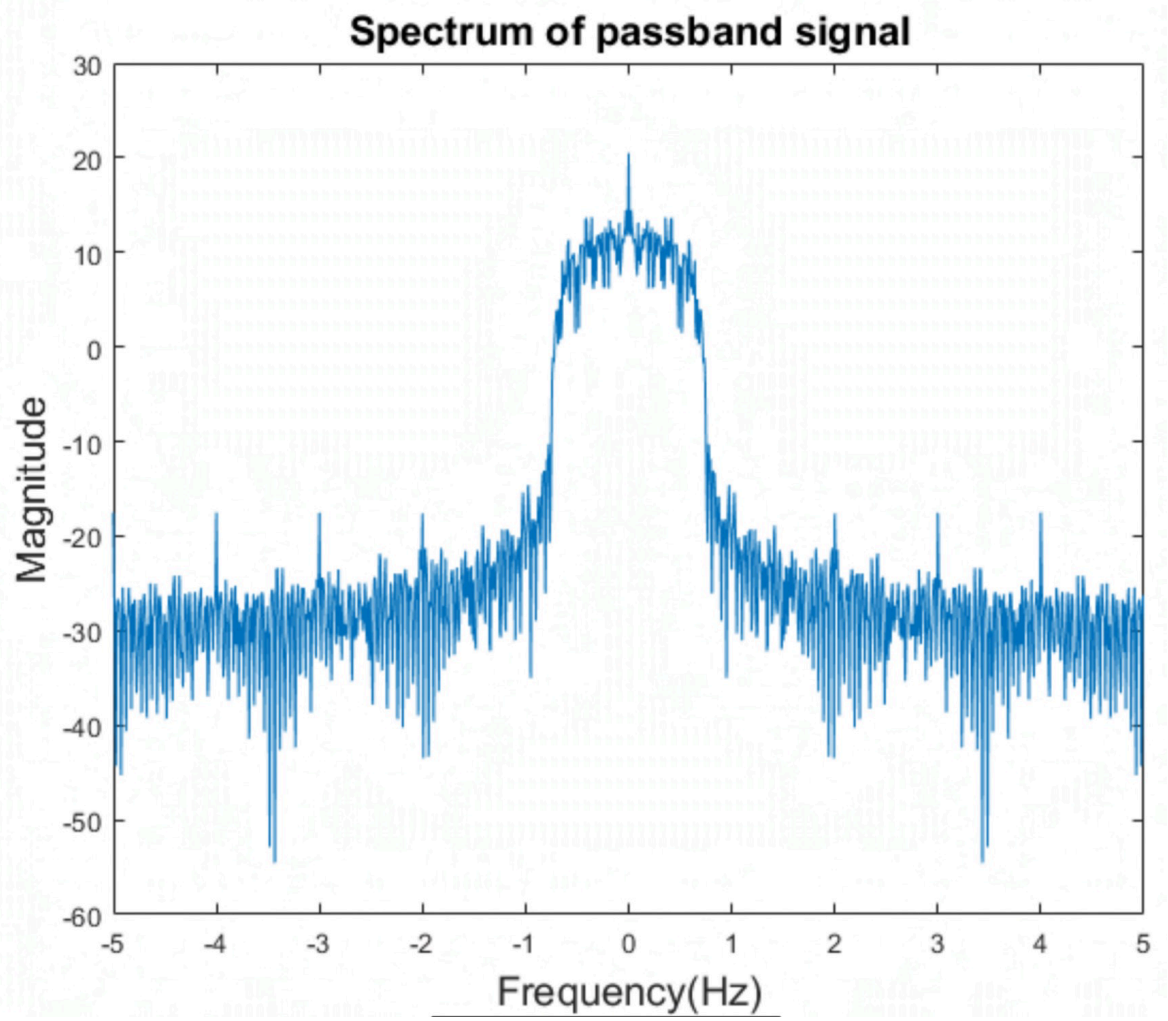
For a SRRC pulse, Bandwidth (BW) = $\frac{(1+\beta).R_S}{2}$ Hz

For β=0.5 and T=1, $R_s = \frac{1}{T} = 1$

$\therefore$ BW = $\frac{(1+0.5).1}{2} = 0.7$ Hz

2.8 Passband signal spectrum implementation

```
seglength = 100;
[pp, ff] = pwelch(bb, rectwin(seglength),0,256,fs,'psd');
plot(ff,10*log10(pp))

[m, f] = getspectrum(ook, ts,1);

figure
plot(f,10*log10(m))
xlabel("Frequency(Hz)", 'FontSize',15);
ylabel("Magnitude", 'FontSize',15);
title("Spectrum of passband signal", 'FontSize',15);
```

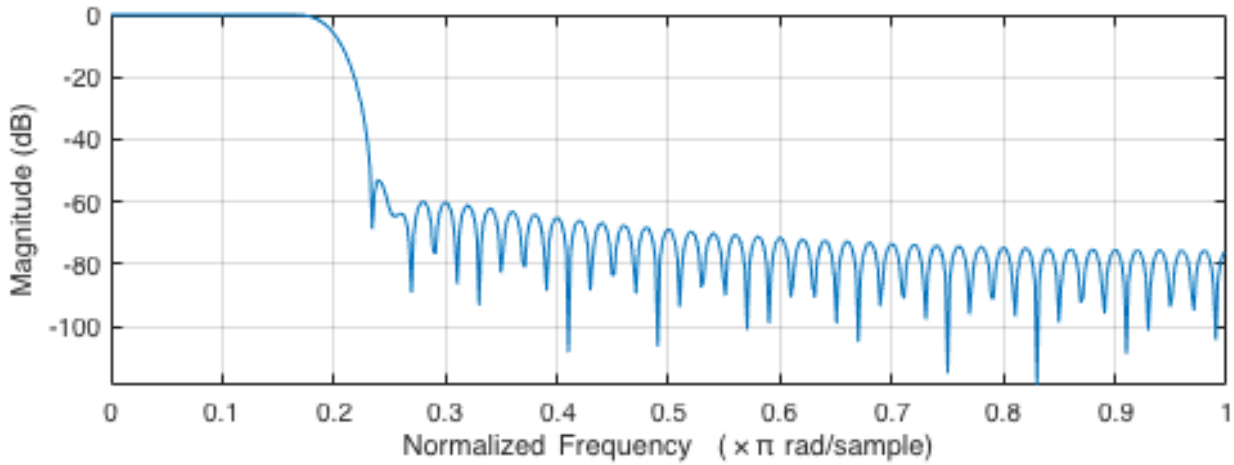


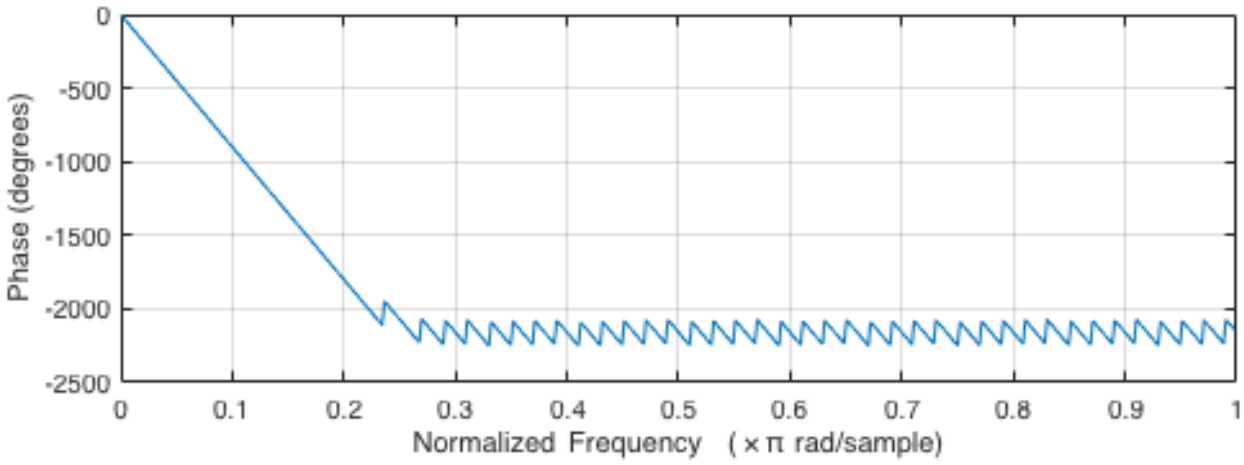
BW obtained = 0.7 Hz (approximately)

3. NON-COHERENT QUADRATURE OOK RECEIVER

3.1 Detecting In-phase and Quadrature phase parts of the transmitted data

```
% In-phase part
Spb_i = ook.*cos(2*pi*fc*t1);
fn = fs/2;  % Nyquist frequency
LPF = fir1(100, 10/fn);
freqz(LPF)
```






```
% S_i = filter(LPF,1,Spb_i);
S_i = conv(Spb_i,LPF);
```

```
r1 = S_i;
mf1 = conv(p, r1);
r_i = mf1;  %In-phase signals


% Quadrature-phase part
Spb_q = ook.*sin(2*pi*fc*t1);


% S_q = filter(LPF,1,Spb_q);
S_q = conv(Spb_q,LPF);


r2 = S_q;
mf2 = conv(p, r2);
r_q = mf2;  %In-phase signals
```

3.2 Sampling the Matched Filter output to recover the bits from the received signal

Now you will develop the code to recover the bits from the received signal. Since the eyes are open, we know that the bits are there. All that remains is actually recovering them.

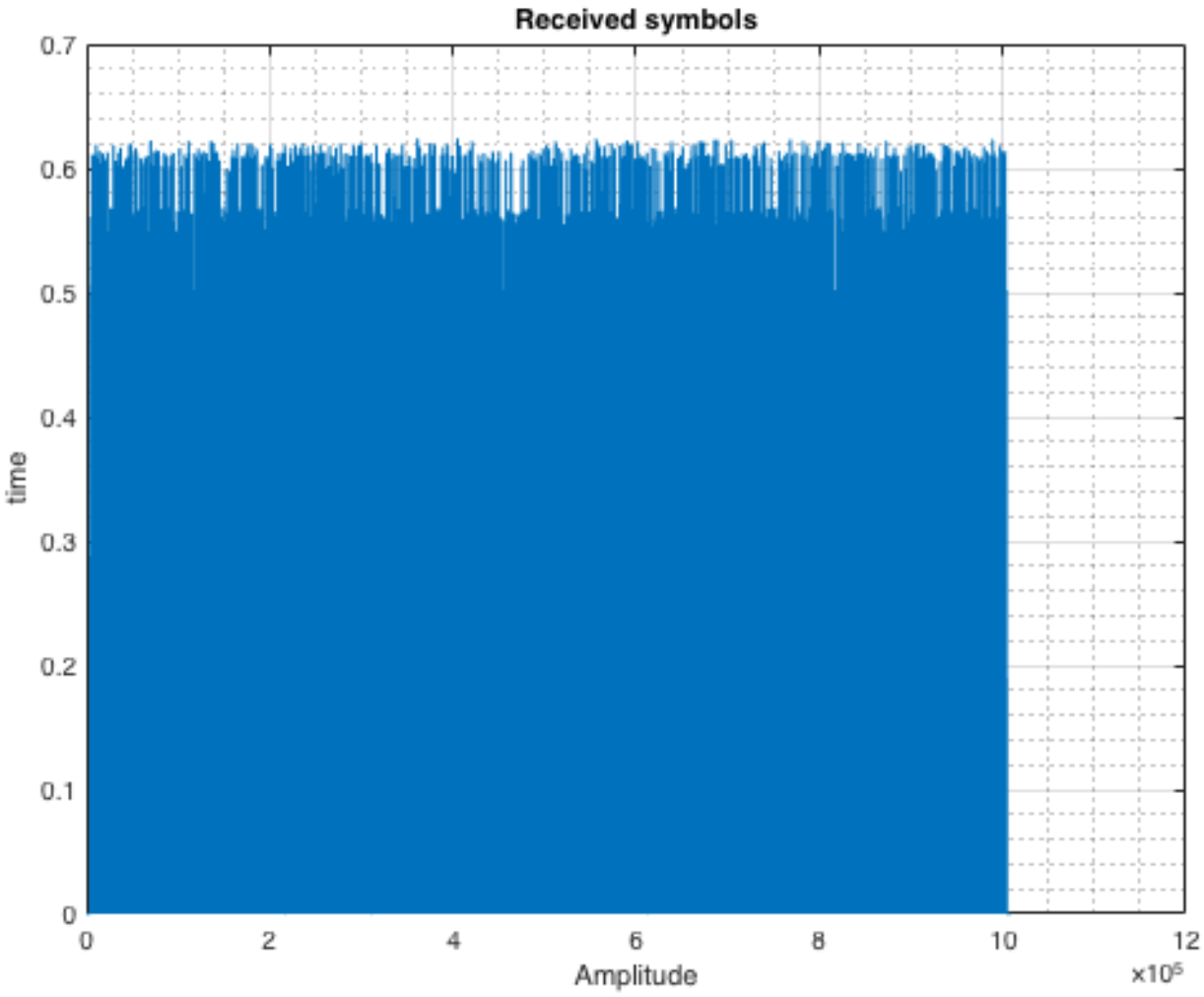
The first step is to sample the matched filter output at the correct times. You know that samples must be takes every *T* _seconds, which corresponds to T*fs samples. The difficulty is finding the initial sample time, because the received signal is delayed by the filtering operations in the receiver.

Let t_ini be the initial sample time. To find it, plot the output of the matched filter, and look for the first pulse. Find the times at which zeros and peaks are found; verify that zeros and peaks appear every T*fs samples. The figure below shows an example from my simulation, where T*fs is equal to 100.

I have found peaks and zeros every 100 samples as expected, starting from sample number 4070. In my simulation, I can recover the symbols by sampling from 2150 and every 100 samples afterward, until the end of the signal. (In an actual receiver this process, which is known as symbol synchronization, happens automatically)

```
Rxd_symbol = sqrt((r_i*cos(phi)).^2 + (r_q*sin(phi)).^2);


figure
plot(Rxd_symbol)
```

```
xlabel('Amplitude'), ylabel('time'), title('Received symbols')
grid on
grid minor
```




```
t_ini = 4070;
```

3.3 Detecting the Transmitted symbols

For a transmitted symbol a_k, the quadrature receiver obtains two samples, a_kI = a_k*cos(phi) in-phase and a_kQ = a_k*sin(phi) in quadrature. We will calculate all the transmitted symbols from a_kI and a_kQ.

```
Rx_matched = Rxd_symbol(t_ini:100:end);
```

```
Rx1 = [];
for idx1 = 1:length(Rx_matched)
    if Rx_matched(idx1)<0.4
        tmp1 = 0;
    elseif Rx_matched(idx1)>0.4
        tmp1 = d;
    end
    Rx1 = [Rx1 tmp1];
end
Rx1 = Rx1(1:N_tmp);
```

3.4 Checking the Received bits

Now that the receiver has recovered the transmitted symbols, we must make sure that they are correct. In MATLAB, given two vectors a and b of the same length, you may find how many of their elements are different by running sum(a ~= b).

```
fprintf('The error in the system is: ')
```

```
The error in the system is:
```

```
Error = sum(syms~=Rx1)
```

```
Error = 294
```

4. OOK SYSTEM NOISE ANALYSIS

In this section, we will add noise to the simulation of an OOK system.

Signal-to-noise ratio is defined as E_s/N_0. The average symbol energy, E_s, is controlled by a slider in your simulation. Since OOK is a binary system, $E_s=E_b$. The noise present in a communications system is defined by its variance $\sigma^2=N_0/2$. Then, the signal-to-noise ratio can be expressed as

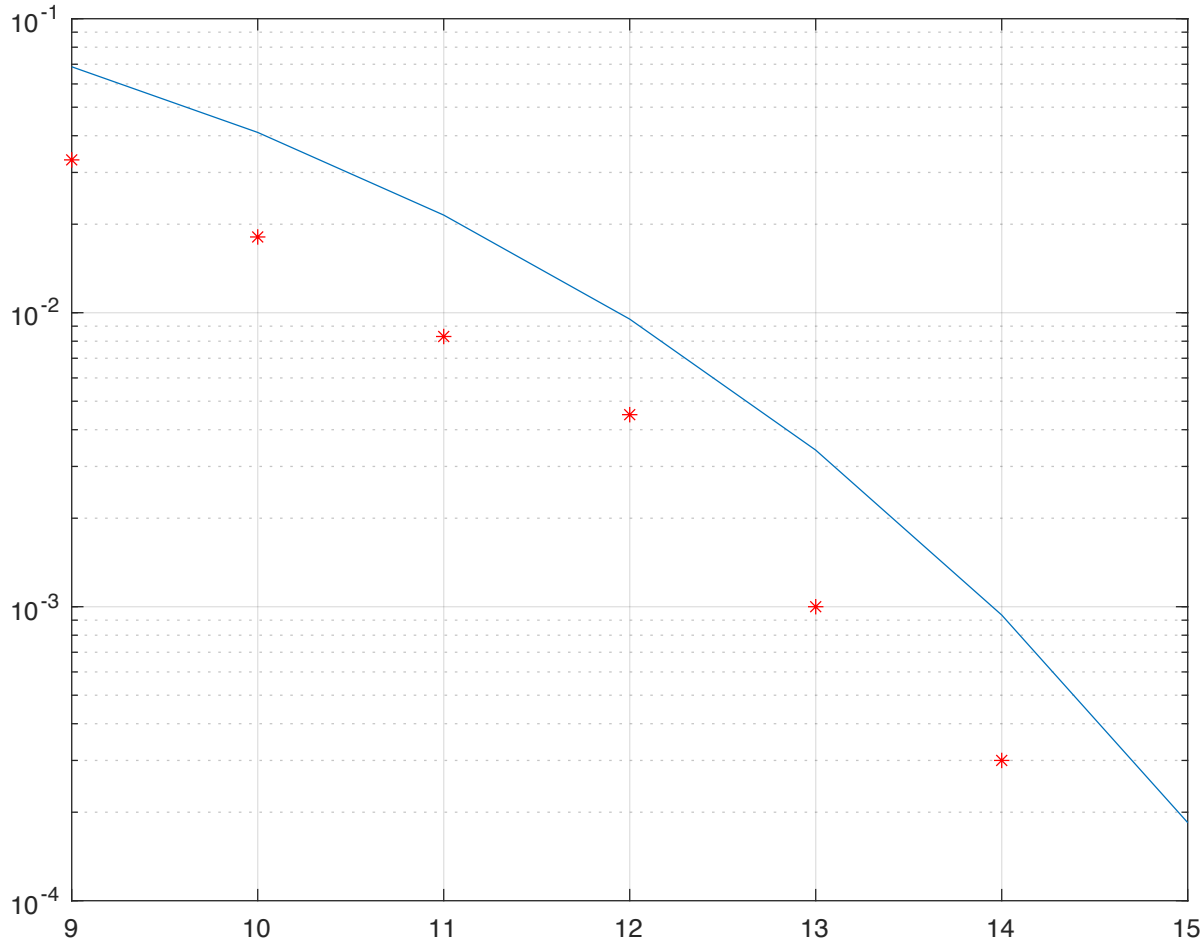
$$\text{SNR} = \frac{2.E_b}{N_o}$$

It is very important to note that a quadrature receiver has two separate sources of noise, one in each of its branches, and each with the same variance σ^2.

4.1 Theoretical Error rates

```
SNR_vec = 9:15;


for idx1 = 1:length(SNR_vec)
    SNR_tmp = SNR_vec(idx1);
    SNR_lin_tmp = 10^(SNR_tmp/10);


    No_var = (Eb)/SNR_lin_tmp;


    Err_expected_tmp(idx1) = 0.5*exp(-(SNR_lin_tmp/4));


    % Creating the noise with variance No/2 for the Received symbols
    n_i_tmp = randn(size(r_i))*(No_var);
    n_q_tmp = randn(size(r_q))*(No_var);

    % Adding the above calculated noise to the Received symbols
    r_i_noisy_tmp = r_i + n_i_tmp;
    r_q_noisy_tmp = r_q + n_q_tmp;


    Rxd_symbol_noisy_tmp = sqrt((r_i_noisy_tmp*cos(phi)).^2 +
(r_q_noisy_tmp*sin(phi)).^2);


    t_ini_noisy_tmp = 4076;
    Rx_matched_noisy_tmp = Rxd_symbol_noisy_tmp(t_ini_noisy_tmp:100:end);
```

```
    Rx1_noisy_tmp = [];
    for idx2 = 1:length(Rx_matched_noisy_tmp)
        if Rx_matched_noisy_tmp(idx2)<0.4
            tmp2 = 0;
        elseif Rx_matched_noisy_tmp(idx2)>0.4
            tmp2 = d;
        end
        Rx1_noisy_tmp = [Rx1_noisy_tmp tmp2];
    end
    Rx1_noisy_tmp = Rx1_noisy_tmp(1:N_tmp);


    Error_tmp = sum(syms~=Rx1_noisy_tmp);
    Error_rate_tmp(idx1) = Error_tmp/N_tmp;


end
```


10^{-1}
10^{-2}
10^{-3}
10^{-4}
9
10
11
12
13
14
15