

# On Off Keying with Additive White Gaussian Noise: Modulation and Demodulation

---

by Laurence G. Hassebrook  
2-27-2013

We simulate On Off Keying (OOK) modulation and demodulation. The modulated signal is synthesized by using an upsampled random bit stream, modulated by a carrier wave and then corrupted by Additive White Gaussian Noise (AWGN). Assuming no phase error, the modulated signal is demodulated using a mixer configuration. Both input and output signals are analyzed for noise distribution. The goal is to reproduce the figures and processing presented in this document using MATLAB. The signal length is  $N$ ; number of bits is  $N_{bit}$ ; Standard Deviation is  $STD$  and carrier frequency is  $k_c$ .

```
clear all;  
N=10000;  
Nbit=20;  
Nsample=floor(N/Nbit);  
Nbin=100;  
STD=.1;  
kc=4*Nbit;
```

Note that there are  $N_{sample}$  sample values for each bit. Hence the system is upsampled by  $N_{sample}$  to simulate continuous time or what we call pseudo-continuous time.

## 1. Binary Sequence Synthesis

Generate  $N_{bit}$  random bits using a the pseudo-random generater `rand()` such that

```
% Random Binary Signal  
wb=rand(1,Nbit);  
bits=binarize(wb);
```

The vector  $w_b$  is has only one sample per bit which is not a good model for continuous time so we upsample using a kronecker product such that.

```
ub=ones(1,Nsample);  
bk=kron(bits,ub);
```

where  $b_k$  is length  $N_b = N_{sample} \times N_{bit}$  which might be less than  $N$ . So to make sure we have a signal  $N$  long we first generate a zero vector  $N$  long and then we move  $b_k$  into it which effectively zero pads any mismatch in length.

```
Nb=Nsample*Nbit;
t=1:N;
b(1:Nb)=bk(1:Nb); %Force signal to be N samples long
```

The result along with the OOK modulation is shown in the next section.

## 2. OOK Modulation

OOK modulation is achieved by first generating a discrete cosine wave and then elementwise multiplying it by the upsampled binary signal sequence  $b$  such that

```
% modulate
sc=cos(2*pi*kc*t/N); % carrier signal
s=b.*sc;
```

To plot out both the binary signal and the modulated signal in the same plot, and store a jpeg image of this result, the matlab code is:

```
figure(1);
plot(t,b,t,s);
title('OOK Modulation');
xlabel('t');
ylabel('s(t)');
axis([1,N,-1.5,1.5]);
legend('binary message','OOK');
print -djpeg Fig1_BinarySignal;
```

The resulting figure is

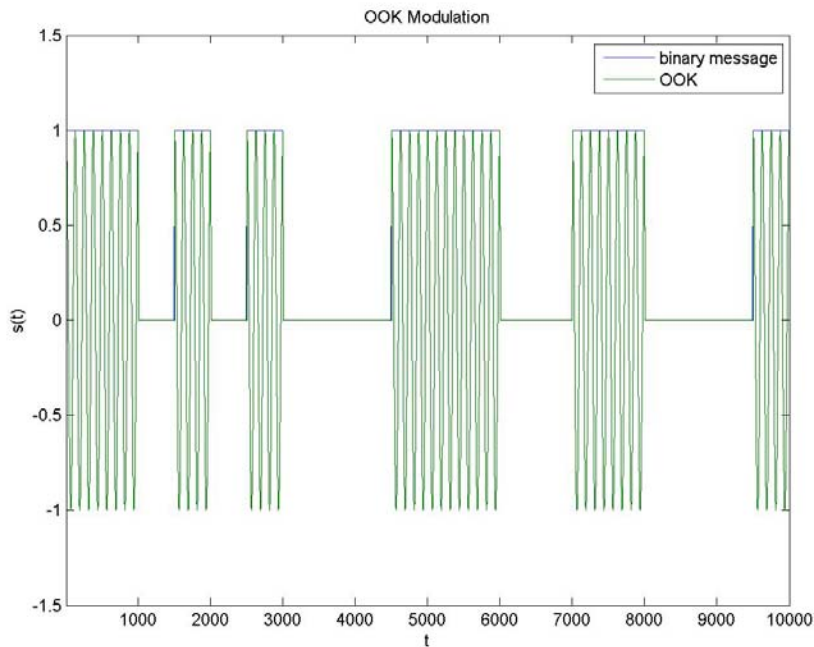


Figure 2.1: Composite plot of the binary signal and the modulated result.

The mathematical representation of the OOK modulated signal is

$$s(t) = b(t)\cos(2\pi k_c t/N) \quad (1)$$

where  $t$  is an integer index for sequence of length  $N$ .

### 3. AWGN and Signal Analysis

Generate a pseudo-random Gaussian sequence using `randn()` in MATLAB and add to the signal  $s(t)$  such that

$$r(t) = s(t) + \tilde{w}(t) \quad (2)$$

```
%% Gaussian distributed noise
w=STD*randn(1,N);
figure(2);
plot(w);
title('Gaussian Noise mean=0');
xlabel('t');
ylabel('w');
```

The noise signal is shown in Fig. 3.1 (left) along with its estimated pdf (right).

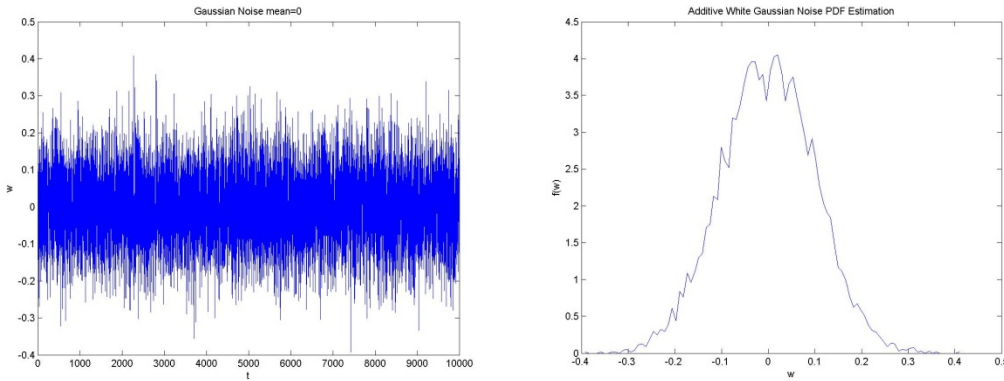


Figure 3.1: (left) Noise signal and (right) estimated pdf.

The code for the normalized pdf estimate in Fig. 3.1 (right) is

```
% Gaussian PDF
n=1:Nbin;
fg=hist(w,Nbin);
maxg=max(w);ming=min(w); % ming=ag * 1 + bg, maxg=ag * Nbin + bg,
% ag=(maxg-ming)/(Nbin-1); bg=ming-ag;
ag=(maxg-ming)/(Nbin-1);bg=ming-ag;
ng=ag * n + bg;
% normalize pdf to 1
dg=(maxg-ming)/Nbin;
fg=fg/(dg*sum(fg));
```

The resulting modulated signal with AWGN is shown in Fig. 3.2.

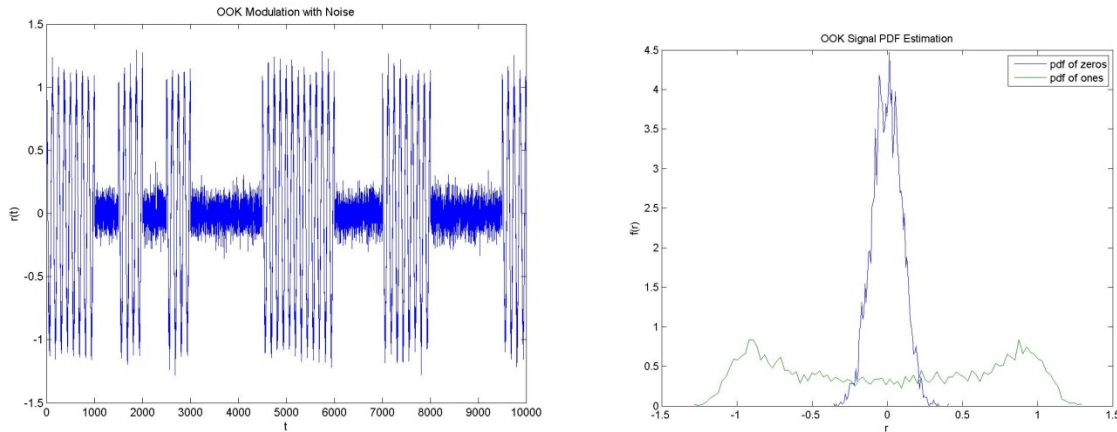


Figure 3.2 (left) OOK modulated signal with AWGN and (right) pdfs for 0 and 1 signals.

Where the noisy OOK signal in MATLAB is

```
r=s+w;
```

To separate out the distribution of the 0 and 1 bits, we use the original binary signal  $b$  such that

```
% separate out the two noise distributions
J0=find(b<0.5);
J1=find(b>=0.5);
f0=hist(r(J0),Nbin);
f1=hist(r(J1),Nbin);
max0=max(r(J0));min0=min(r(J0)); % min0=a0 * 1 + b0, max0=a0 * Nbin + b0,
max1=max(r(J1));min1=min(r(J1));
% a0=(max0-min0)/(Nbin-1); b0=min0-a0;
a0=(max0-min0)/(Nbin-1);b0=min0-a0;
n0=a0 * n + b0;
a1=(max1-min1)/(Nbin-1);b1=min1-a1;
n1=a1 * n + b1;
% normalize pdf to 1
d0=(max0-min0)/Nbin;
f0=f0/(d0*sum(f0));
d1=(max1-min1)/Nbin;
f1=f1/(d1*sum(f1));
```

The first J0 is where the bits are 0's and the J1 is where the bits are 1's. It can be seen in Fig. 3.2 that the "1" bit distribution is no longer Gaussian because the sine wave redistributes the noise.

## 4. Demodulation and Signal Analysis

We will use a mixer followed by a low pass filter to demodulate the OOK signal. A mixer multiplies the received signal with a replica of the carrier signal which must be in phase with the modulated carrier. The multiplication creates what is known as a baseband and two frequency

translated replicas of the baseband centered at  $\pm 2k_c$  frequencies. So to reconstruct the signal we simply low pass filter the multiplier output with a cutoff around  $k_c$ . Mathematically this is

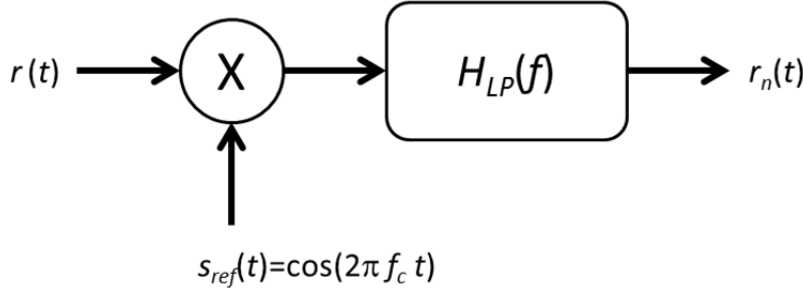
$$r_n(t) = r(t) \cos(2\pi k_c t / N) * h_{LP}(t) \quad (3)$$


Figure 4.1: Mixer based demodulator.

```

%% DEMODULATION USING A MIXER
sref=sc; % reference signal
% mix the reference with the input
r1=r.*sref;
% form reconstruction filter
fc=kc;
% filter with some recommended parameters
Norder=8;fmax=N/2;K=8; % filter gain
[f H]=lp_butterworth_oN_dft(kc,K,fmax,N,Norder);
% filter signal through channel via frequency domain
S=fft(r1);R=S.*H;
rn=real(ifft(R));

```

To detect “0” and “1” bits separately, we do not want to sample the bit boundaries else we would get errors from the transitions. So we define binary windows within the bit boundaries, one for the “1” detection and the other for the “0” detection.

```

% define sampling region for detector
ueye=zeros(1,Nsample);
ueye(1,floor(Nsample/4):floor(3*Nsample/4))=1;
bkeye=kron(bits,ueye);
bkeyenot=kron((1-bits),ueye);
beye=zeros(1,N);
beyenot=zeros(1,N);
beye(1:Nb)=bkeye(1:Nb); %Force signal to be N samples long
beyenot(1:Nb)=bkeyenot(1:Nb); %Force signal to be N samples long

```

where beye is within the one boundaries and beyenot is within the zero boundaries. All these signals are shown in Fig. 4.1

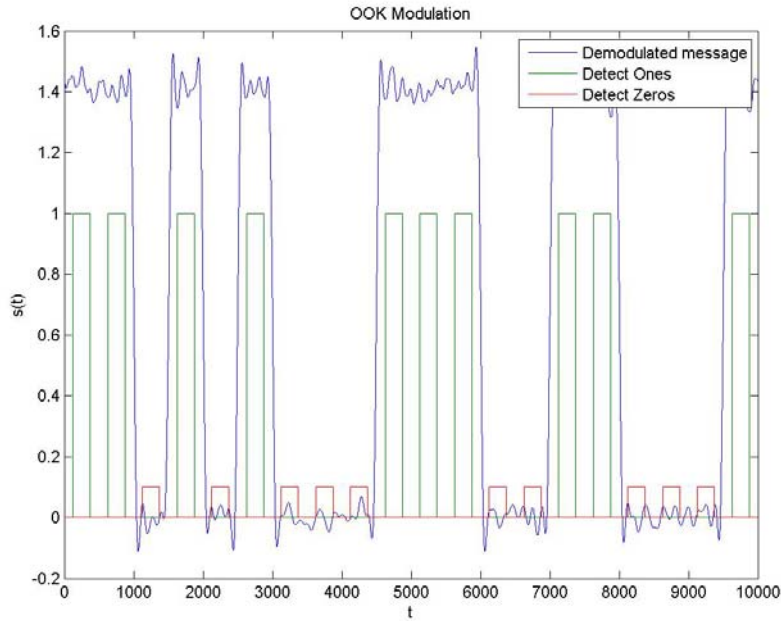


Figure 4.2: Demodulated signal with the detection windows superimposed.

Using the window sequences we estimate pdfs for the “0” and “1” signals separately as

```
% separate out the two noise distributions
J0=find(beyenot>0.5);
J1=find(beye>0.5);
f0=hist(rn(J0),Nbin);
f1=hist(rn(J1),Nbin);
max0=max(rn(J0));min0=min(rn(J0)); % min0=a0 * 1 + b0, max0=a0 * Nbin + b0,
max1=max(rn(J1));min1=min(rn(J1));
% a0=(max0-min0)/(Nbin-1); b0=min0-a0;
a0=(max0-min0)/(Nbin-1);b0=min0-a0;
n0=a0 * n + b0;
a1=(max1-min1)/(Nbin-1);b1=min1-a1;
n1=a1 * n + b1;
% normalize pdf to 1
d0=(max0-min0)/Nbin;
f0=f0/(d0*sum(f0));
d1=(max1-min1)/Nbin;
f1=f1/(d1*sum(f1));
```

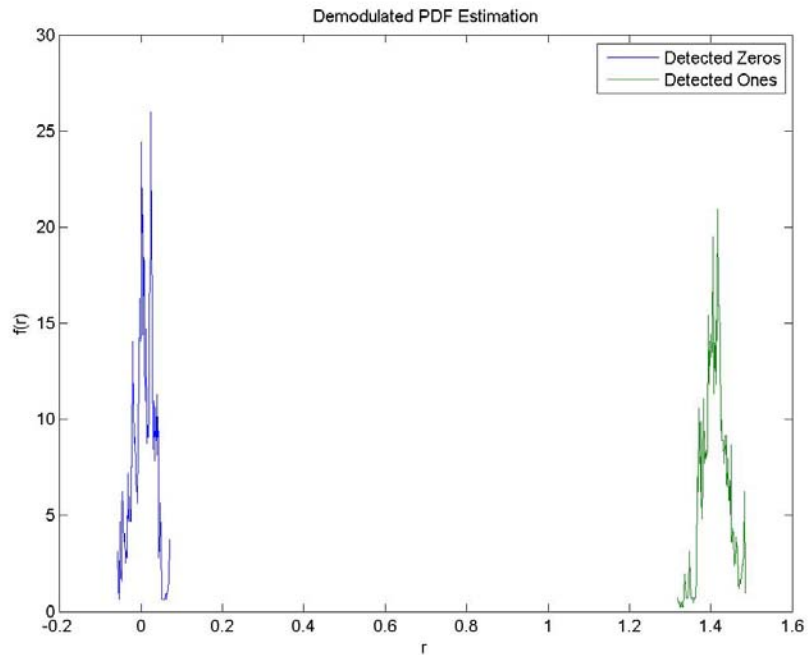


Figure 4.3: Estimated pdfs within the "0" and "1" detection windows with STD=0.1.

Rerun your simulation with STD=1 and plot the new detection pdfs in Fig. 4.3. Also look at the affect this increased noise has on Figs. 5 and 6.

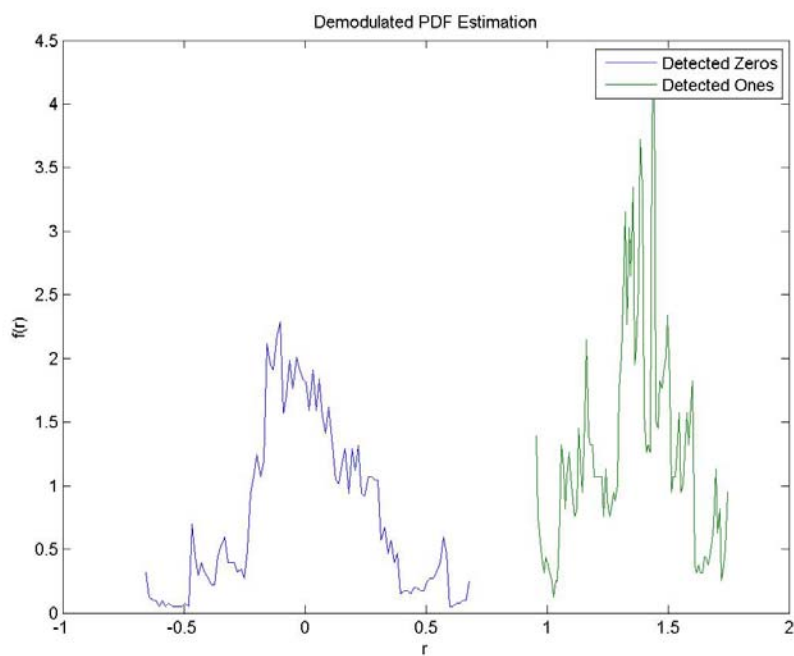


Figure 4.4: Estimated pdfs within the "0" and "1" detection windows with STD=1.

Fig. 4.3 shows the two pdfs separated by a gap between 0.8 and 0.9. This gap represents the value of a threshold used to test the data for a “0” or a “1” that would yield zero detection error.