EC 415: Homework 5

Due by Friday 04/23/2021 6:00PM

Professor David Starobinski

Michael Kremer
kremerme@bu.edu

Exercise 8.1

The Matlab code in naivecode.m, which is on the website, implements the translation from binary to 4-PAM (and back again) suggested in (8.2). Examine the resiliency of this translation to noise by plotting the number of errors as a function of the noise variance v. What is the largest variance for which no errors occur? At what variance are the errors near 50%?

Solution

Using this code

Listing 1: MATLAB code for Exercise 8.1

```
mesLen=1000;                              %message length
bits=(sign(rand(1,mesLen)-.5)+1)/2;   %binary message to send
%index into constl = 1+ bits(i) + 2*bits(i+1)
constl=[-3 1 -1 3];
k=1;
pam4mes=zeros(1,length(bits)/2);
for i=1:2:length(bits)
  pam4mes(k)=constl(1+bits(i)+2*bits(i+1));   %switch to a PAM4 constellation
  k=k+1;
end

n=1000;
indx=1;
percErrs=zeros(1,n/.1);
num_errors=zeros(1,n/.1);
for v=0:.1:n

    %pass the signal through a noisy channel
    noisyPam=sqrt(v)*randn(1,length(pam4mes))+pam4mes;

    %quantize the received signal
    recSig=quantalph(noisyPam,[-3,-1,1,3]);

    k=1;
    recBits=zeros(1,2*length(recSig));
    %decode the signal using the naive code
    for i=1:length(recSig)
      if recSig(i)==3
        recBits(k)=1;
        recBits(k+1)=1;
      elseif recSig(i)==1
        recBits(k)=1;
        recBits(k+1)=0;
      elseif recSig(i)==-1
        recBits(k)=0;
        recBits(k+1)=1;
      elseif recSig(i)==-3
        recBits(k)=0;
```

```
            recBits(k+1)=0;
        end
        k=k+2;
    end

    %calculate the percentage error
    percErrs(indx)=sum((recBits~=bits))/length(recBits);
    num_errors(indx)=sum(recBits~=bits);
    %length(recBits)

    indx=indx+1;
end

% Plot
v=0:.1:n;
plot(v, num_errors)
title('number of errors as a function of variance for ' + string(length(recBits)) + ' bits
xlabel('variance')
ylabel('number of errors')
```

the following plot was generated.

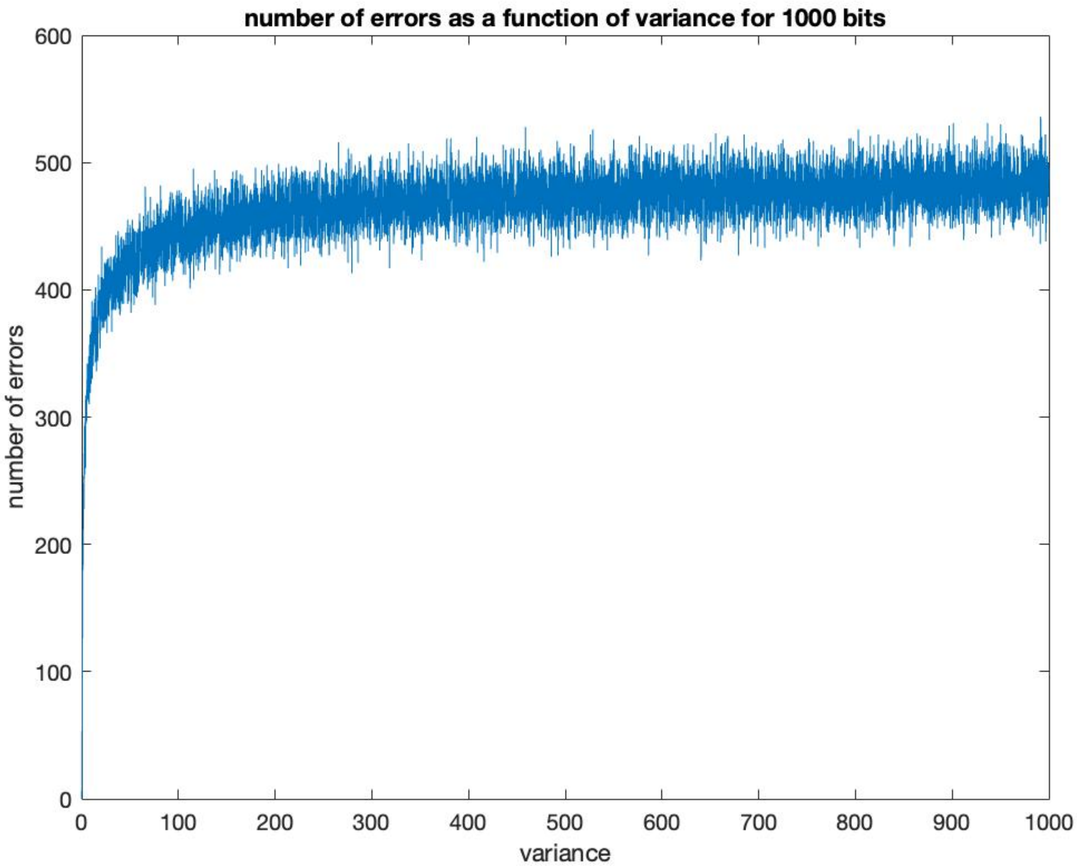


Figure 1: number of errors as a function of the noise variance v

From this we can see that the errors can be greater than 50% as soon as variance reaches about 300, but from the long term trend, the error count only approaches 50% on average.

Exercise 8.2

A Gray code has the property that the binary representation for each symbol differs from its neighbors by exactly one bit. A Gray code for the translation of binary into 4-PAM is
01 → +3
11 → +1
10 → -1
00 → -3
Mimic the code in naivecode.m to implement this alternative and plot the number of errors as a function of the noise variance v. Compare your answer with Exercise 8.1. Which code is better?

Solution

Using this code:

Listing 2: MATLAB code for Exercise 8.2

```
mesLen=1000;                              %message length
bits=(sign(rand(1,mesLen)-.5)+1)/2;   %binary message to send
%index into constl = 1+ bits(i) + 2*bits(i+1)
constl=[-3 1 -1 3];
k=1;
pam4mes=zeros(1,length(bits)/2);
for i=1:2:length(bits)
  pam4mes(k)=constl(1+bits(i)+2*bits(i+1));   %switch to a PAM4 constellation
  k=k+1;
end

n=1000;
indx=1;
percErrs=zeros(1,n/.1);
num_errors=zeros(1,n/.1);
for v=0:.1:n

    %pass the signal through a noisy channel
    noisyPam=sqrt(v)*randn(1,length(pam4mes))+pam4mes;

    %quantize the received signal
    recSig=quantalph(noisyPam,[-3,-1,1,3]);

    k=1;
    recBits=zeros(1,2*length(recSig));
    %decode the signal using the naive code
    for i=1:length(recSig)
      if recSig(i)==3
        recBits(k)=0;
        recBits(k+1)=1;
      elseif recSig(i)==1
        recBits(k)=1;
        recBits(k+1)=1;
      elseif recSig(i)==-1
```

```
            recBits(k)=1;
            recBits(k+1)=0;
        elseif recSig(i)==-3
            recBits(k)=0;
            recBits(k+1)=0;
        end
        k=k+2;
    end

    %calculate the percentage error
    percErrs(indx)=sum((recBits~=bits))/length(recBits);
    num_errors(indx)=sum(recBits~=bits);
    indx=indx+1;
end

% Plot
v=0:.1:n;
plot(v, num_errors)
title('number of errors as a function of variance for ' + string(length(recBits)) + ' bits
xlabel('variance')
ylabel('number of errors')
```

the following plot was generated.

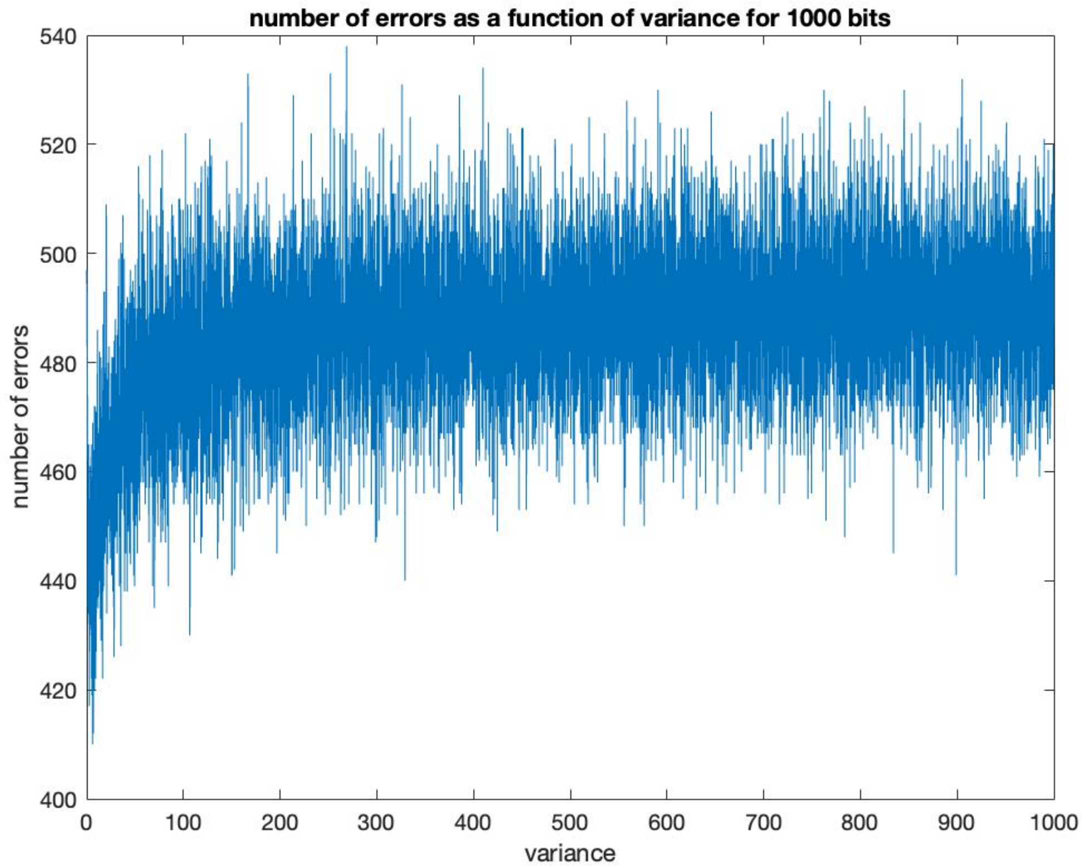


Figure 2: number of errors as a function of the noise variance v

By comparing this plot to the plot generated in Exercise 8.1, we can see that using a Gray code translation reduces the error count. This is to be expected since Gray code sequential transitions are more similar.

Exercise 8.5

Can you think of a pulse shape that will have a narrower bandwidth than either of the above but that will still be time limited by T ? Implement it by changing the definition of ps, and check to see whether you are correct.

Solution

Using this code:

Listing 3: MATLAB code for Exercise 8.5

```
str='Transmit this text string';            % message to be transmitted
m=letters2pam(str); N=length(m);            % 4-level signal of length N
M=10; mup=zeros(1,N*M); mup(1:M:N*M)=m;     % oversample by M
ps=sinc(linspace(-1,1,10));                 % blip pulse of width M
x=filter(ps,1,mup);                         % convolve pulse shape with data

t=1/M:1/M:length(x)/M;
subplot(2,1,1), plot(0:0.1:0.9,ps)
xlabel('The pulse shape')
subplot(2,1,2), plot(t,x)
xlabel('The waveform representing "Transmit this text"')
```

the following plot was generated.

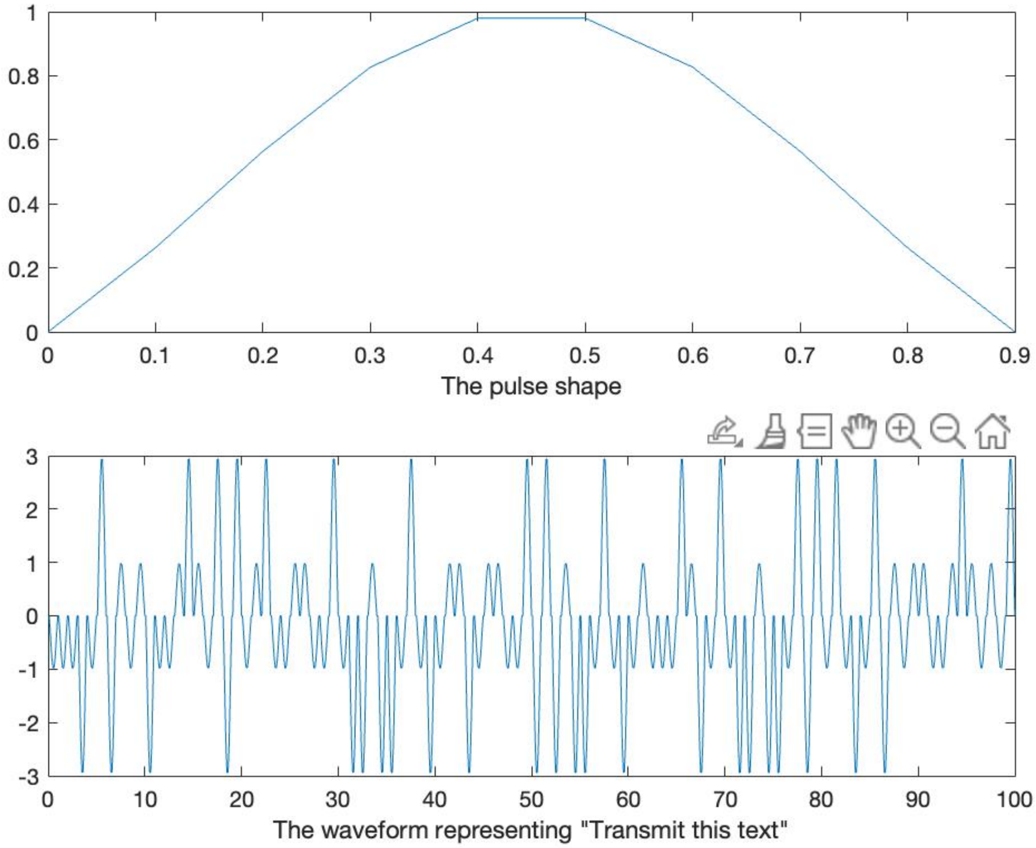


Figure 3: The waveform representing "Transmit this text"

In this solution a sinc pulse shape was used because it is the most efficient pulse shape.

Exercise 8.8

Rerun correx.m with different amounts of noise. Try sd=0, 0.1, 0.3, 0.5, 1, 2. How large can the noise be made if the correlation is still to find the true location of the header?

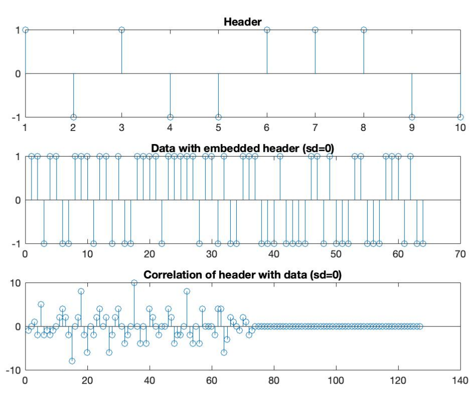
Solution

Using this code:

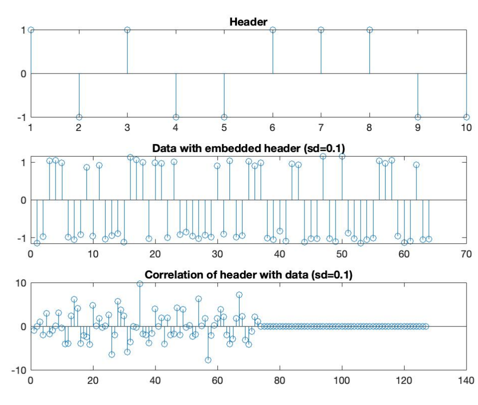
Listing 4: MATLAB code for Exercise 8.8

```
header=[1 -1 1 -1 -1 1 1 1 -1 -1];       % header is a predefined string
loc=30; r=25;                            % place header in position loc


sd=[0 0.1 0.3 0.5 1 2];

for i=1:1:length(sd)
    data=[sign(randn(1,loc-1)) header sign(randn(1,r))];  % generate signal
    data=data+sd(i)*randn(size(data));                  % add noise
    y=xcorr(header, data);                   % do cross correlation
    [m,ind]=max(y);                          % location of largest correlation
    headstart=length(data)-ind+1;            % place where header starts
    figure(i)
    subplot(3,1,1), stem(header)             % plot header
    title('Header')
    subplot(3,1,2), stem(data)               % plot data sequence
    title('Data with embedded header (sd='+string(sd(i))+')')
    subplot(3,1,3), stem(y)                  % plot correlation
    title('Correlation of header with data (sd='+string(sd(i))+')')
end
```

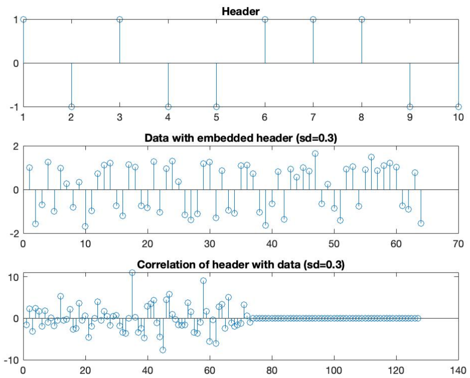
the following plots were generated.



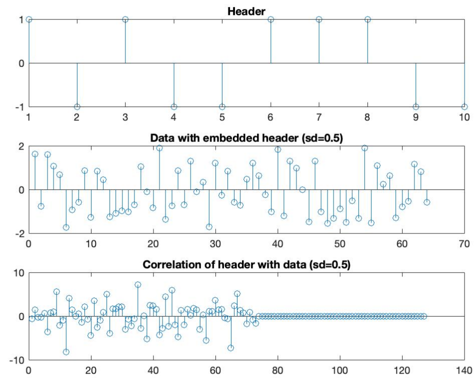
(a) sd=0



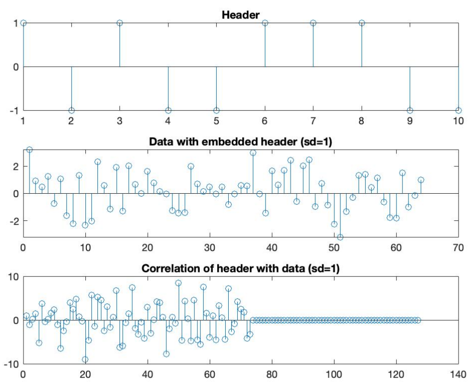
(b) sd=0.1



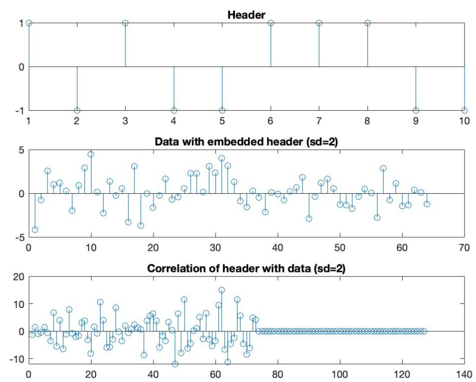
(a) sd=0.3



(b) sd=0.5



(a) sd=1



(b) sd=2

From the above plots it seems like the noise can be set to sd=0.1 before the true location of the header becomes hidden.

Extra Question

Solution

First, this code was used to convert a given ASCII string into QPSK symbols:

Listing 5: MATLAB code for Problem 5

```
% function complex_out = qpsk_enc(m)
%       M=4;
%       complex_out = exp(j*2*pi*(m-1)/M);
% end

function out = letters2QPSK(str) % call as Matlab function

% Convert ascii to binary string
binary=dec2bin(str);
[l, w] = size(binary);
bin="";
for i=1:l
    row="0"+binary(i:i,:);
    bin=bin+row;
end
bin=char(bin);

% Convert binary string to stuff
out = zeros(1,4*length(str));
M=4;
indx=1;
for i=1:2:length(bin)
    if bin(i) == '0'
        if bin(i+1) == '0'
            m=1;
        else
            m=2;
        end
    else
        if bin(i+1) == '0'
            m=4;
        else
            m=3;
        end
    end
    out(indx)= exp(j*2*pi*(m-1)/M);
    indx=indx+1;
end
```

The first thing the above code does is convert an ascii input into binary. Some manipulation has to be done to append a leading zero and write all values as on vector. Once the vector of binary values is created, I run a loop looking at two bit values to determine the value of m. After m is determined, the value is used to calculate the QPSK symbols and write them to an output vector.

Then, this code was used to convert a given ASCII string into a QPSK modulated passband signal:

Listing 6: MATLAB code for Problem 5

```
% String for testing
test_str='EC415';

% Get QPSK symbols;
QPSK_symbols=letters2QPSK(test_str);

time=8*length(test_str); Ts=1/1000; % sampling interval & time
t=Ts:Ts:time; lent=length(t);        % define a time vector
w=zeros(1,lent);
start=0;
for i=1:time/2
    w((2*start/Ts)+1:i*2/Ts)=QPSK_symbols(i);
end


fm=1; fc=10; c=exp(j*2*pi*fc*t);    % carrier at freq fc

%w=5/lent*(1:lent)+cos(2*pi*fm*t);    % create "message"
%w
v=c.*w;                              % modulate with carrier

% used to plot figure
subplot(2,1,1), plot(t,w)
axis([0,time, -1,3])
ylabel('amplitude'); xlabel('time'); title('(a) message signal');
subplot(2,1,2), plot(t,v)
axis([0,time, -2.5,2.5])
ylabel('amplitude'); xlabel('time'); title('(b) message after modulation');
```

I did not get the above code working for part two.