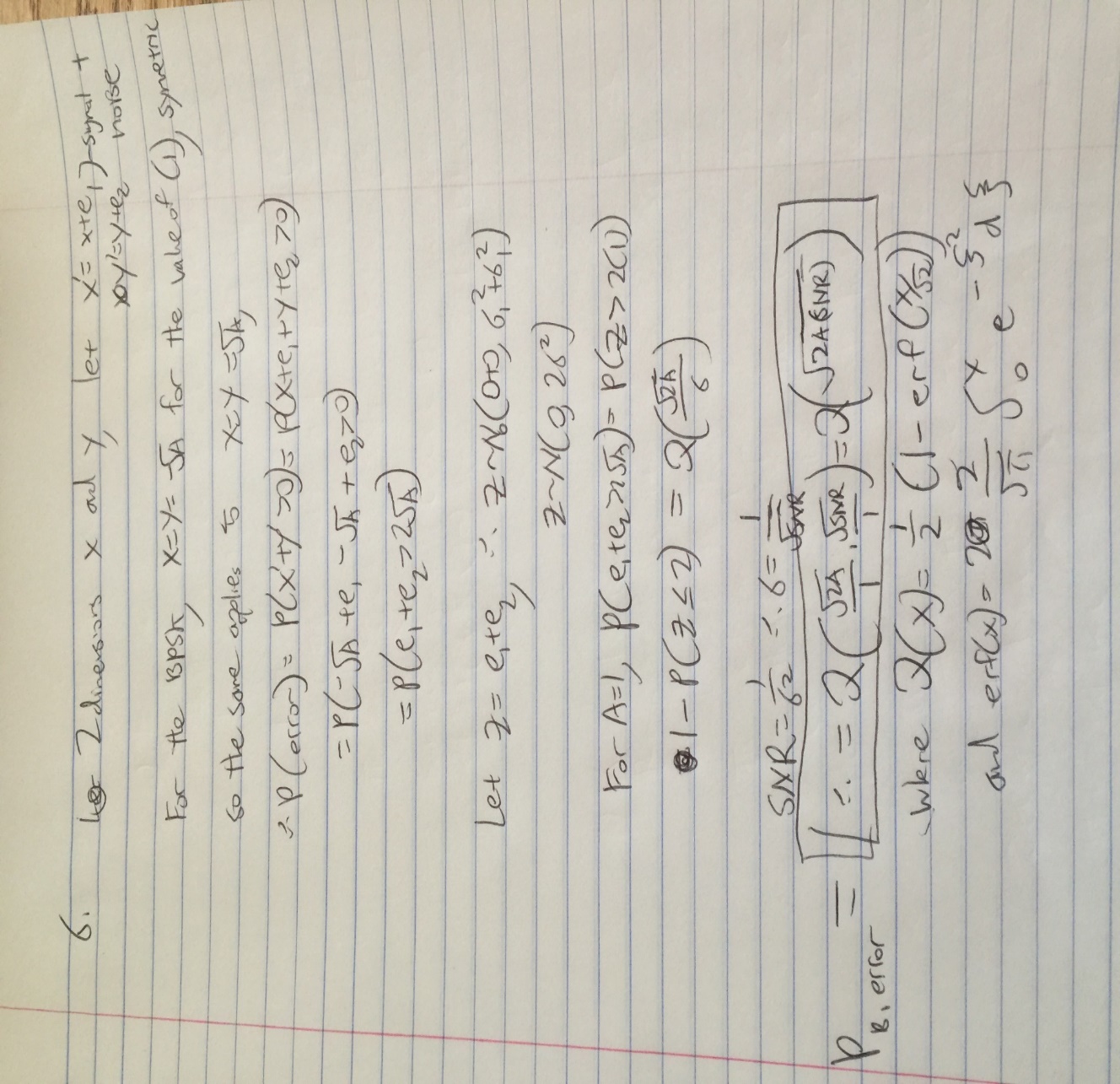
# MatLABProject 1

The code is copied and pasted from MATLAB into the appendix and has comments to explain how coded, uncoded and theoretical curves were created.

**Theoretical Transmission and extra credit**: we apply the following theory of error functions and Q functions to derive the formula for theoretical transmission. Then you plug in A=1 to the value of A:

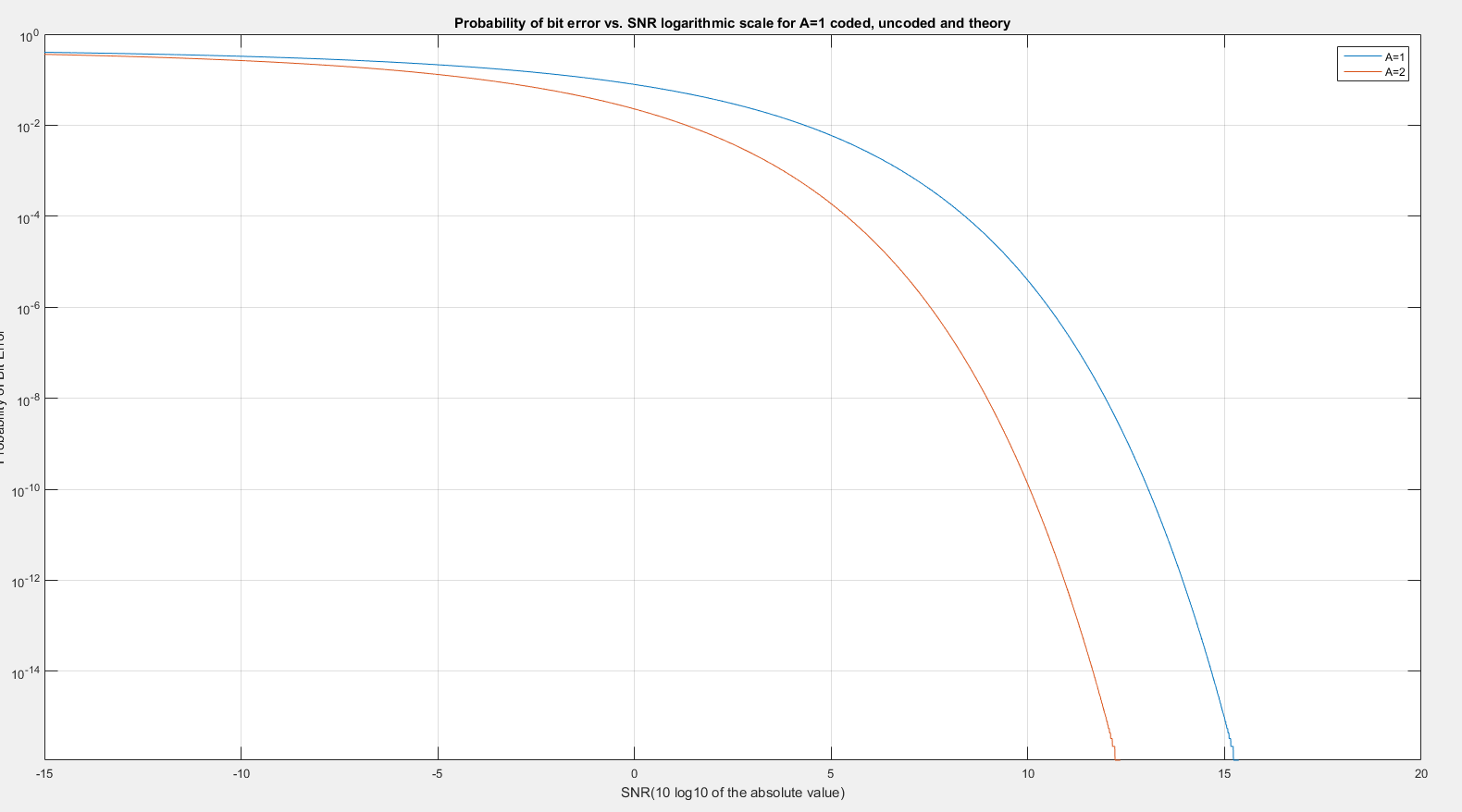


Thus, all you have to do is plot 1-the normal cumulative from negative infinity to 2 for

p=1-normcdf(2,0,var.^0.5). This is the theoretical plot for A=2, and the relationship for bit error is shown here as well. If A= 1 a succinct way to state the relationship is to say that P(error)=Q(root of 2\*SNR)).

**Final Question:**

There are a number of things we can observe that would let us plot for a = 2 without needing a simulation. The first is that everything is kept constant except the value of A. This means that the theoretical curve for A=2 is going to be of the same formula and general shape as the theoretical formula we derived above for the extra credit question. Already we know that the Probability of bit error values are going to be smaller for any constant value of SNR. This is because the mappings will be of higher energy since the magnitude of A in both directions of the 2 dimensions is greater. This means that the value of constant noise relative to an increased energy signal is going to be greater and thus that the decoder should pick up more accurate readings since noise is smaller relative to the whole system. The most accurate way to determine the relationship would be to use the equation above. As we can see, the probability of both errors being greater than 2 root A (Prob(e1+e2)>2root(a)) is equal to the normal cdf subtracted from 1 for the bounds negative infinity to 2. Instead, we merely have to change the formula to use 2\*root(2) instead of 2\*root(1). Thus we can derive the following formula:

If we plot this against the graph of A=1 theoretical curve, we can therefore obtain the relationship without having to run any simulation (blue is A=1 and red is A=2): 

Thus our derivation makes sense; there is a smaller probability of error at each point and it also increases as you go towards greater values of snr (smaller sigma, so the error is even less impactful).

var=2.\*(sigma.^2);

SNR=1./(sigma.^2);

%p is the theoretical curve A=1, pfinal is A=2

p=1-normcdf(2,0,var.^0.5);

pfinal=1-normcdf(2\*sqrt(2),0,var.^0.5);

figure(1);

semilogy(10\*log10(SNR),p,10\*log10(snr),pe, 10\*log10(SNR),pfinal);

ylabel('Probability of Bit Error');

xlabel('SNR(10 log10 of the absolute value)');

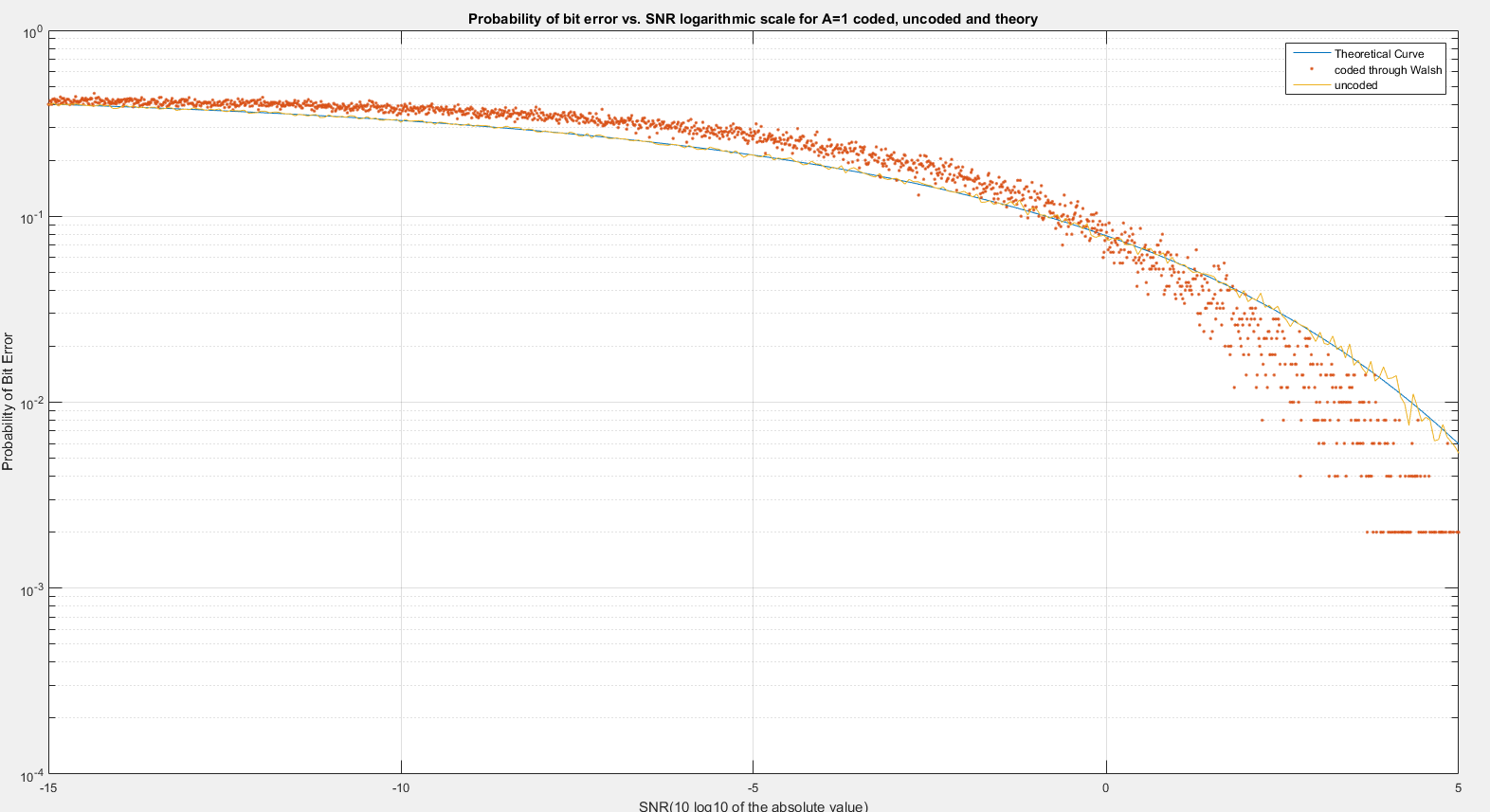
title('Probability of bit error vs. SNR logarithmic scale for A=1 and A=2);

grid on;

legend('Theoretical Curve', 'coded’, 'final');

axis([-15 5 0 1]) % probability is between 0 and 1 only

Below is also the final plot which includes all of the simulations for each simulation run over 10000 SNR values over 10000 trials for each value. The code took a while to compile:



As we can see, at sigma=1 and below the coding transmission bit error (red curve) start have less error probability than the uncoded version.

The curves are blue for theory, yellow for uncoded and red for coded. As we can see all three curves follow the same basic magnitudes and shapes which we want. The uncoded transmission follows the basic shape of the theory which means that our theoretical calculations are correct as their function accurately demonstrates the overall simulation pattern for uncoded The probability of error is higher in the Walsh functions when the value of SNR is relatively low (or the value of sigma is high compared to noise energies of 1). However as the SNR goes up (sigma goes between 0 and 1) the bit error starts to be better for coded transmission. We can thus conclude that the Walsh coding functions make beneficial bit error reductions when the SNR is greater than 1 (0 on the x axis for log values), so we can use the coding transmission when we have energies around 1 if the value of sigma is less than that.

Appendix:

%%

clear;

%from the hadamard transforms we get the following walsh function r

%vectors. Get the columns which are the transposes so we can run inner

%products.

%% **uncoded section**

%set A to

A=1;

% create the logspaced snr values and sigma values

snruncoded=logspace(-3,2,10000);

sigmauncoded=linspace(0,1,10000);

probuncoded=zeros(1,10000);

mu=0;

%iterate for the number of SNR terms

for i=1:length(snruncoded)

sigmauncoded=sqrt(1./snruncoded(i));

%create random bit input steam of 10000

b=randi([0,1],1,10000);

output=zeros(1,length(b));

%map from binary 0 and 1 to 2 dimension A vectors from the graph on

%project guidelines. a 0 maps to the right side which is postivie and a

%1 maps to the left side which is negative

for j=1:length(b)

if(b(j)==0)

vector=[sqrt(A),sqrt(A)];

end

if(b(j)==1)

vector=[-sqrt(A),-sqrt(A)];

end

%add noise of the specific mean and variance, mean 0 and variance

%sigma squared

y=normrnd(mu,sigmauncoded,[1 2]);

z(1)=vector(1)+y(1);

z(2)=vector(2)+y(2);

%according to the line on the graph, if the noise makes the mapping

%flip to the right side the output is 0 and left side the output is

%1. this can cause error because the noise values can make the

%output flip

if(z(2)>=-z(1))

output(j)=0;

end

if(z(2)<-z(1))

output(j)=1;

end

end

count=0;

%count the number of errors over all trials and add it up, this is 10000

%bits per trial for more accuracy.

for k=1:length(output)

if(output(k)~=b(k))

count=count+1;

end

end

%divide count of error over total count to get bit error probability

biterrProb=count./length(output);

%get all bit error probabilities per each SNR valu

probuncoded(i)=biterrProb;

end

%% **coded transmission** on walsh functions first create each of the Hadamurd function rows and their negatives from the rows.

row1=[1 1 1 1 1 1 1 1];

row1neg=-1.\*row1;

row2=[1 1 1 1 -1 -1 -1 -1];

row2neg=-1.\*row2;

row3=[1 -1 1 -1 1 -1 1 -1];

row3neg=-1.\*row3;

row4=[1 -1 1 -1 -1 1 -1 1];

row4neg=-1.\*row4;

row5=[1 1 -1 -1 1 1 -1 -1];

row5neg=-1.\*row5;

row6=[1 1 -1 -1 -1 -1 1 1];

row6neg=-1.\*row6;

row7=[1 -1 -1 1 1 -1 -1 1];

row7neg=-1.\*row7;

row8=[1 -1 -1 1 -1 1 1 -1];

row8neg=-1.\*row8;

%these values are obtained as the rows of the H matrix which can be found

%by doing the Hadmurd transformations.

z=[row1;row2;row3;row4;row5;row6;row7;row8];

zneg=[row1neg;row2neg;row3neg;row4neg;row5neg;row6neg;row7neg;row8neg];

%%

% create sigma which is the x variable of the plot and then

%variance and SNR from that. create bit error probability array.

snr=logspace(-3,2,10000);

sigma=zeros(1,10000);

pe=zeros(1,10000);

for i=1:length(sigma)

sigma(i)=sqrt(1./snr(i));

%% create input noisy vectors'

totalbits=0;

correctbits=0;

% create the generator matrix

P=[1 1 1 1 1 1 1 1; 0 0 0 0 1 1 1 1; 0 0 1 1 0 0 1 1; 0 1 0 1 0 1 0 1];

for trials=1:10000

%initialize total bits and incorrectbits;

% create the input data bits

databits=round(rand(1,4));

%multiplying the data bits by P and then mod 2 gives you the

%8-bit codeword, then map that to BPSK

%data bits become code bits 3, 5, 6 and 7

inputbinary=(databits\*P);

% do mod 2

for j=1:8

inputbinary(j)=mod(inputbinary(j),2);

end

% do BPSK of input

bpsk=zeros(1,8);

for k=1:8

if inputbinary(k)==0

bpsk(k)=1;

elseif inputbinary(k)==1

bpsk(k)=-1;

end

end

%add randomly generated noise of mean 0 and std=sigma.

% this is done by making a 1x8 matrix of randn values and multiplying

%it by sigma and then adding the mean which is 0 (to transform from

%standard normal.

x=randn(1,8);

noise=sigma(i)\*x;

noisybits=bpsk+noise;

walshvalues=zeros(1,16);

%% plug in walsh functions by doing inner products with each. Each walsh function is a row in the H matrix (z) or the negative H matrix (-z). You multiply by the noisy bits for each H row and negative H row, and the maximum value from these 16 inner products is the walsh function and BPSK that the coded transmission maps to.

for l=1:8

zwalshp=transpose(z((8\*(l)-7):((8\*(l)))));

walshvalues(l)=noisybits\*zwalshp;

end

for m=1:8

zwalshn=transpose(zneg((8\*(m)-7):((8\*(m)))));

walshvalues((m+8))=noisybits\*zwalshn;

end

%% we have all the walsh values, positive and negative.

%now compare maximum magnitude and sign for the proper mapping

%get index, the index is the same in both these H matrix and the

%negative H matrix

[walshmax,index]=max(abs(walshvalues));

walsh1bpsk=z(((8\*index)-7):(8\*(index)));

walsh2bpsk=zneg(((8\*index)-7):(8\*(index)));%now have both the postive and negative, check for sign

curr1=0;

curr2=0;

%this is how I checked for the sign, you see if the positive or

%negative version is right by which one has the closest number of bits.

%They are inverses of each other so one or the other has to have more.

for r=1:8

if walsh1bpsk(r)==bpsk(r)

curr1=curr1+1;

elseif walsh2bpsk(r)==bpsk(r)

curr2=curr2+1;

end

end

if curr1>=curr2;

%choise the correct walsh function, negative or positive

bpskout=walsh1bpsk;

else

bpskout=walsh2bpsk;

end

%BPSK output is proportional to output data bits (it is set indices in the BPSK) so compare input BPSK to output BPSK to get the same exact

%probabilities.

binaryout=zeros(1,8);

datacoded=zeros(1,4);

for q=1:8

if bpskout(q)==1

binaryout(q)=0;

elseif bpskout(q)==-1

binaryout(q)=1;

end

end

%data bits become code bits 3, 5, 6 and 7, you could also check data

%coded vs data input and you will get the same Pe if you run it.

%However I chose to do the other way because you get more total bits

%per trial.

% datacoded(1)=binaryout(3);

%datacoded(2)=binaryout(5);

%datacoded(3)=binaryout(6);

%datacoded(4)=binaryout(7);

for p=1:8

totalbits=totalbits+1;

if (bpsk(p)==bpskout(p))

correctbits=correctbits+1;

end

end

%$ this will give you a correct/total, subtract this from 1 to get the

%error bits/total which is probability of bit error

end

pe(i)=1-(correctbits/totalbits);

end

%% **theoretical curve** for A=1 Theory is explained in the page on the writeup

var=2.\*(sigma.^2);

SNR=1./(sigma.^2);

%p is the theoretical curve.

p=1-normcdf(2,0,var.^0.5);

%% cumulative graph plot all 3 on the same with a log y axis to get proper scaling

figure(1);

semilogy(10\*log10(SNR),p,10\*log10(snr),pe,'.',10\*log10(snruncoded),probuncoded,'-');

ylabel('Probability of Bit Error');

xlabel('SNR(10 log10 of the absolute value)');

title('Probability of bit error vs. SNR logarithmic scale for A=1 coded, uncoded and theory');

grid on;

legend('Theoretical Curve','coded through Walsh','uncoded');

axis([-15 5 0 1]) % probability is between 0 and 1 only