

The following represents my final data for the uncoded and the lattice coded. The blue is the triangle error and the yellow is the coded error, which we can see is less than the uncoded error proving that the decoder does effectively reduce the proability of symbol error for the 16 QAM constellation. The differences are less significant, however, as we go down into the lower sigma (deviation) values because at this point the different between the noisy vector and the original zero vector is too small so that the uncoded will not be interpreted as an error. However, the overall decoder for higher values proves to be very effective in reducing error.

Code:

clear;

%% QAM Part 1 uncoded simulation.

%The method we will be using is to map the coordinates of the simulated

%input to the decision regions of a 16-region QAM constellation. If the

%decision region of the input with the gaussian noise becomes the wrong

%region, then we will say this is a symbol error. symbol errors/total erros

%is equal to the probability of error.

% create the logspaced snr values and sigma values

snruncoded=logspace(-3,5,3000);

sigmauncoded=sqrt(1./snruncoded);

probuncodedt=zeros(1,1000);

probuncodeds=zeros(1,1000);

x1t=0.5;

y1t=0.5;

%triangular quadrants

q1t=[x1t y1t];

q2t=[-1\*x1t y1t];

q3t=[-1\*x1t -1\*y1t];

q4t=[x1t -1\*y1t];

%square quadrants

x1s=1.5;

y1s=1.5;

q1s=[x1s y1s];

q2s=[-1\*x1s y1s];

q3s=[-1\*x1s -1\*y1s];

q4s=[x1s -1\*y1s];

%run simulation for each possible triangle or square symbol error,

%quadrants 1-4

for i=1:length(snruncoded)

%create counter for total square and triangle simulations error

totalt=0;

totals=0;

errort=0;

errors=0;

%gaussian noise, mean 0=sigma\*x+0

%there are 8 coordinates each with 2 dimensions of noise, so we create 16

%noise vectors of standard normal.

for j=1:3000

x=randn(1,16);

noise=x\*sigmauncoded(i);

%put the input through the noise, run channel for each independent

%dimension, which has its own random noise value.

s1=q1s+noise(1:2);

s2=q2s+noise(3:4);

s3=q3s+noise(5:6);

s4=q4s+noise(7:8);

t1=q1t+noise(9:10);

t2=q2t+noise(11:12);

t3=q3t+noise(13:14);

t4=q4t+noise(15:16);

totalt=totalt+4;

totals=totals+4;

%the decision regions are determined by the new coordinates of the vectors

%with noise. If these values are not within the decision regions then it is

%an error in uncoded transmission.

if (t1(1)<0 ||t1(1)>1 || t1(2)<0 ||t1(2)>1)

errort=errort+1;

end

if (t2(1)>0 ||t2(1)<-1 || t2(2)<0 ||t2(2)>1)

errort=errort+1;

end

if (t3(1)>0 ||t3(1)<-1 || t3(2)>0 ||t3(2)<-1)

errort=errort+1;

end

if (t4(1)<0 ||t4(1)>1 || t4(2)>0 ||t4(2)<-1)

errort=errort+1;

end

if (s1(1)<1 ||s1(2)<1)

errors=errors+1;

end

if (s2(1)>-1 ||s2(2)<1)

errors=errors+1;

end

if (s3(1)>-1 ||s3(2)>-1)

errors=errors+1;

end

if (s4(1)<1 ||s4(2)>-1)

errors=errors+1;

end

end

probuncodedt(i)=errort/totalt;

probuncodeds(i)=errors/totals;

end

input=[0 0 0 0 0 0 0 0];

snrcoded=logspace(-3,2,3000);

sigmacoded=sqrt(1./snrcoded);

probfinal=zeros(1,length(snrcoded));

for a=1:length(snrcoded)

errors=0;

total=0;

for b=1:3000 %run each sigma value through many trials, each trial has 4 QAM pairs so add 8 for number of symbols

%create random noise vectors

e=randn(1,8);

total=total+8;

%add gaussian noise

recieved = input+sigmacoded(a)\*e;

% need to check for this as well as -1/2 vectors

Half = recieved -.5;

%do the rounding

fr = round(recieved);

frhalf = round(Half);

%initialize gr but dont do the calculation for gr yet

grhalf = Half;

gr = recieved;

approximator = mod(recieved, 1);

roundUp = 1-approximator;

%find the index of the smallest norm so we can run calculations

diff = gr-round(gr);

[y,J] = max(abs(diff));

in = find(abs(diff) == y);

%use this to find the absolute smallest norm (with squared values)

if(length(in)>1)

[notneeded, findMe] = min(gr(in).^2);

in = in(findMe);

end

%do same stuff for gr

diff = grhalf-round(grhalf);

[y,J] = max(abs(diff));

halfindices = find(abs(diff) == y);

if(length(halfindices)>1)

[notneeded, findMe] = min(gr(halfindices).^2);

halfindices = halfindices(findMe);

end

gr = round(recieved);

grhalf = round(grhalf);

if(recieved(in)<gr(in))

gr(in) = gr(in)+1;

%do the same stuff to gr

else

gr(in) = gr(in)-1;

end

if(recieved(halfindices)>=grhalf(halfindices))

grhalf(halfindices) = grhalf(halfindices)-1;

else

grhalf(halfindices) = grhalf(halfindices)+1;

end

if(mod(sum(gr), 2) ~= 0)

%take the even of fr and gr

zone = fr;

else

zone = gr;

end

if(mod(sum(grhalf), 2) ~= 0)

ztwo = frhalf;

else

ztwo = grhalf;

end

if( sum(abs(recieved - zone)) <= sum(abs(recieved - ztwo)))

decoded = mod(zone,2);

else

decoded = mod(ztwo,2);

end

%check for QAM errors, so if the pairs don't equal the initial zero vector

if (mod(decoded(1),2)~=0)

errors=errors+1;

end

if (mod(decoded(2),2)~=0)

errors=errors+1;

end

if (mod(decoded(4),2)~=0)

errors=errors+1;

end

if (mod(decoded(6),2)~=0)

errors=errors+1;

end

if (mod(decoded(8),2)~=0)

errors=errors+1;

end

if (mod(decoded(3),2)~=0)

errors=errors+1;

end

if (mod(decoded(5),2)~=0)

errors=errors+1;

end

if (mod(decoded(7),2)~=0)

errors=errors+1;

end

end

%probability error = errors/total

probfinal(a)=errors/total;

end

%plot on the dBscale.

figure(1);

semilogy(10\*log10(snruncoded),probuncodedt,10\*log10(snruncoded),probuncodeds,10\*log10(snrcoded),probfinal);

ylabel('Probability of Bit Error');

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

title('Probability of bit error vs. SNR logarithmic scale for square and triangle in 16QAM uncoded and lattice coded');

grid on;

legend('triangle error','square error','error with gasset lattice');

axis([-30,20,10^-4,1]);