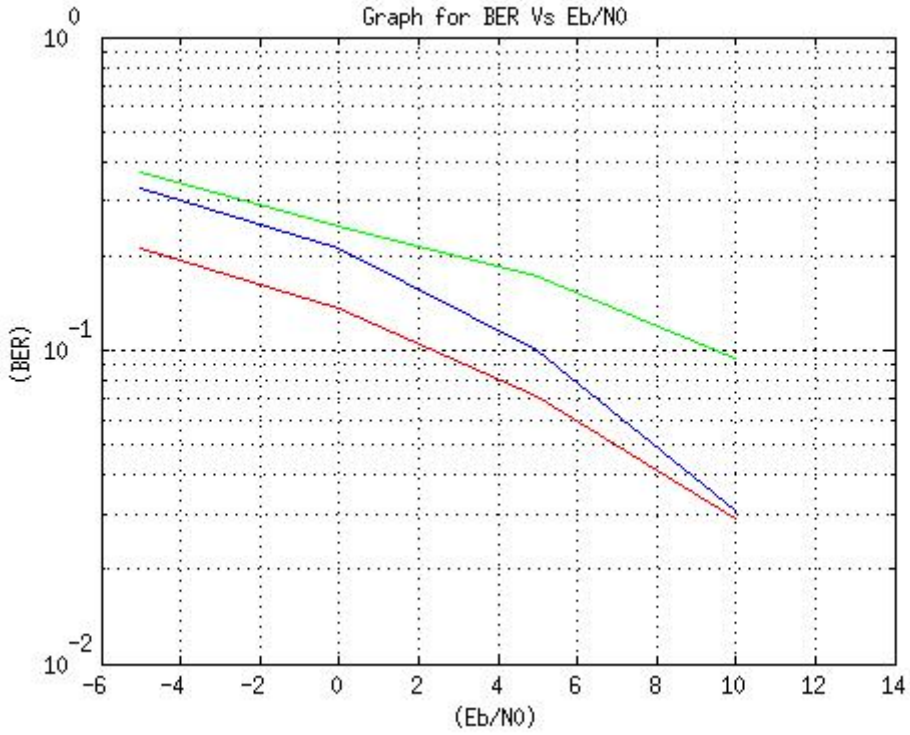
Digital Communications
Lab 5.1

Rajula Vineet Reddy
IMT2014045



Appendix:

Match Filter:

```
function outp = matchfilter(r)
Matrix1 = [1 1 1 1];
matrix2 = [-1 -1 -1 -1];
[rows,columns] = size(r);
output = [];
valu = 1;
col = columns/4;
for increment = 1:lenght(r)/4
   vec = r((incremet-1)*4)+1:((increment-1)*4)+4;
   valu = valu+4;
   u2 = conv(matrix2,vec);
   u1 = conv(Matrix1,vec);
   vec2 = max(u2);
   vec1 = max(u1);
   if vec1>vec2
      output(end+1) = 1;
   end
   if vec1<vec2
      output(end+1) = 0;
   end
end
outp = output;
end
```

Noise:

```
function p =noise(symbol,variance)
s = size(symbol);
r=sqrt(variance)*randn(s(1),s(2));
end
```

Input:

```
SNR_db = [-5 -2 0 2 5 8 12 15];
SNR = 10.^(SNR_db/10);
inverse = SNR.^(-1)
Eb = 0.01*(10.^(-3));
N0=Eb*inverse;
B = 400 * (10.^3);
N=b*(N0/2);
```

```
1 bit correcting:
Number = 100000;
data = 11;
TotalBits = 15;
Parity = TotalBits-Data;
numBlocks = ceil(Number/Data);
data = [randi([0 1], Number, 1); zeros(Data*numBlocks - Number,
1)];
Vec1 = [1; 1; 0; 0; 1];
Matrix1 = reshape(data, Data, numBlocks);
AJ = mod(conv2(Matrix1, Vec1), 2);
Parity = 0.01;
Temporary = reshape(AJ, TotalBits*numBlocks, 1);
KQ = mod(Temporary + (unifrnd(0, 1, size(Temporary)) < Parity),
2);
KQ = reshape(KQ, TotalBits, numBlocks);
clear Temporary;
% Parity check matrixVec2 = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1; 1 1 0 0; 0 1 1 0; 0 0 1
1;
1 1 0 1; 1 0 1 0; 0 1 0 1; 1 1 1 0; 0 1 1 1; 1 1 1 1; 1 0 1 1; 1 0 0
1];
% Retrive the syndrome polynomial
syndromePolynomial = mod(KQ'*Vec2, 2);
[bool, loc] = ismember(syndromePolynomial, Vec2, 'rows');
clear syndromePolynomial;
for k = 1:numBlocks
if(loc(k) > 0)
KQ(loc(k), k) = 1 - KQ(loc(k), k);
endif
endfor
clear bool, loc;
% To obtain the original out we need to convert each out of size
Number to a out of size Data using polynomial division
out = [];
for k = 1:numBlocks
out = [out, deconv(KQ(:,k), Vec1)];
end
out = mod(out, 2);
opt = reshape(out,N1,1);
Ber=abs(opt-INP);
Answer=numel(find(Ber==1))/N1;
```

```
2 bit correcting:
Number = 100000;
Data = 7;
Parity = 8;
TotalBits = Data + Parity;NumberOfBlocks = ceil(Number/Data);
data = [randi([0 1], Number, 1); zeros(Data*NumberOfBlocks -
Number, 1)];
M = reshape(data, Data, NumberOfBlocks);
G = [1; 0; 0; 0; 1; 0; 1; 1; 1]; % generator polynomial
U = mod(conv2(M, G), 2);
Temporary = reshape(U, TotalBits*NumberOfBlocks, 1);
Parity = 0.1;
R = mod(Temporary + (unifrnd(0, 1, size(Temporary)) < Parity),
2);
R = reshape(R, TotalBits, NumberOfBlocks);
Vec1 = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1; 1 1 0 0; 0 1 1 0; 0 0 1 1;
1 1 0 1; 1 0 1 0; 0 1 0 1; 1 1 1 0; 0 1 1 1; 1 1 1 1; 1 0 1 1; 1 0 0 1];
Vec3 = Vec1(mod(3*(0:14), 15)+1,:);
Sm1 = mod(R'*Vec1, 2);
Sm3 = mod(R'*Vec3, 2);
[bool, S1Pow] = ismember(Sm1,Vec1,'rows'); [bool, S3Pow] =
ismember(Sm3,Vec1,'rows');
SIG1 = Sm1; SIG2 = [];
for k = 1:NumberOfBlocks
if(S1Pow(k) != 0 && S3Pow(k) != 0)
x_var = S1Pow(k)-1; y = S3Pow(k)-1;
% Multiplicative inverse of a^x_var
z = 15-x_var;
x_var = mod(3*x_var, 15);
t1 = mod(z+x_var, 15); t2 = mod(z+y, 15);
val = mod(Vec1(t1+1,:) + Vec1(t2+1,:), 2);
% Append val to the vector of sigma2
SIG2 = [SIG2; val];
elseif(S1Pow(k) != 0)
x_var = S1Pow(k)-1;
x_var = mod(2*x_var, 15); % (3*x_var + 15-x_var) mod
15
val = Vec1(x_var+1,:);
% Append val to the vector of sigma2SIG2 = [SIG2; val];
else
% Two bit error does not exist, append 0 to sigma2
SIG2 = [SIG2; [0 0 0 0]];
end
end
[bool, SIG1Pow] = ismember(SIG1,Vec1,'rows'); [bool,
```

```
SIG2Pow] = ismember(SIG2,Vec1,'rows');
for k = 1:NumberOfBlocks
for j = 1:15
% Two roots exist
if(SIG1Pow(k) != 0 && SIG2Pow(k) != 0)
p1 = SIG1Pow(k)-1; q1 = SIG2Pow(k)-1;
t1 = mod(p1+j-1, 15); t2 = mod(q1+2*(j-1), 15);
val = mod(Vec1(1,:) + Vec1(t1+1,:) + Vec1(t2+1,:),
2);
% Has the polynomial evaluated to 0?
if(val == [0 0 0 0])
% If yes then it is a root, and the power of its
inverse is a bit error location`
pos = mod(15-(j-1), 15)+1;
R(pos, k) = 1 - R(pos, k);
end
% Only one root exists
elseif(SIG1Pow(k) != 0)
R(SIG1Pow(k), k) = 1 - R(SIG1Pow(k), k);
break;
% There is no erroneous bit in this block of R
else
break;end
end
end
out = [];
for k = 1:NumberOfBlocks
out = [out, deconv(R(:,k), G)];
end
out = mod(out, 2);
if(R == U && out == M)
printf("Success!! The out has succesfully been transfered
across the channel\TotalBits");
else
printf("All errors could not be corrected with total number of
errors = %d\TotalBits", nnz(out-M));
endif
```