Trabalho Final de Comunicações Digitais

Thiago Carneiro Pita
Departamento de Engenharia Elétrica
Engenharia de Redes de Comunicação
Universidade de Brasília
Matrícula: 15/0047037

João Tribouillet Marcial de Menezes
Departamento de Engenharia Elétrica
Engenharia de Redes de Comunicação
Universidade de Brasília
Matrícula: 14/0043209

Henrique Carneiro Pita
Departamento de Engenharia Elétrica
Engenharia Elétrica
Universidade de Brasília
Matrícula: 13/0050903

***Resumo*—Este documento é um relatório referente a um experimento de simulação de geração de um transmissor digital em um canal com ruído branco Gaussiano em *Matlab*.**

I. Descrição Experimental e Análise Experimental

A. Questão 1

Para a realização desta etapa experimental, criou-se um vetor com **Nb** bits aleatórios. Para isto, utilizou-se da função *rand* para gerar **Nb** números reais aleatórios entre 0 e 1 em um vetor e, em seguida, criou-se outro vetor, em que atribuía-se o valor de 1 para os números maiores que 0,5 e 0 para os valores menores que 0,5 constituintes do vetor gerado. Isso está demonstrado nó Código 1 a seguir. Onde *array_Nb* é o vetor binário.

```
Nb=12;

%%%%%%%%%%Questao 1%%%%%%%%%%%%%%
array_Nb=rand(1,Nb);
array_Nb=array_Nb>0.5;
```

Código 1. Código com a Criação de um Vetor de Bits Aleatórios

B. Questão 2

Esta etapa experimental consistiu na criação de mapas para as modlulações **BPSK**, **8-PSK**, **16-QAM** e **64-QAM**. Para isso, utilizou-se, primeiramente, os ***Modulators*** do *ToolBox* do *Matlab* citados a seguir

- **PSKModulator**
  Foi utilizado para as modulações de **BPSK** e **8PSK**. Para isso, indicou-se os diferentes ordens de cada modulação (no caso 2 e 8), fase 0, nomeação de '*SymbolMapping*' e uma especificação do valor para *Binary*.
- **RectangularQAMModulator**
  Foi utilizado para as modulações de **16QAM** e **64QAM**. Para isso, indicou-se a propriedade de como o mapa será criado como *ModulationOrder* seguido da ordem da modulação (no caso 16 e 64), nomeação de '*SymbolMapping*' e uma especificação do valor para *Binary*.

A partir disso feito, é possível obter-se a constelação dos possíveis símbolos de cada modulação especificada, tendo seu valor de amplitude e fase determinados. o Código 2 a seguir, mostra como foi forjado tais constelações e as Figuras 1, 2, 3 e 4 indicam as constelações obtidas para as modulações **BPSK**, **8-PSK**, **16-QAM** e **64-QAM**, respectivamente.

```
%%%%%%%%%%Questao 2%%%%%%%%%%%%%%
a='ModulationOrder';
b='SymbolMapping';
c='Binary';
d='BitInput';

%Define modulacoes
bpsk = comm.PSKModulator(2,0,b,c);
opsk = comm.PSKModulator(8,0,b,c);
hexqam = comm.RectangularQAMModulator(a,16,b,c);
ooqam = comm.RectangularQAMModulator(a,64,b,c);

%Plota
constellation(bpsk);
constellation(opsk);
constellation(hexqam);
constellation(ooqam);
```

Código 2. Código Referente às criações das Constelações das Diferentes Modulações em Questão

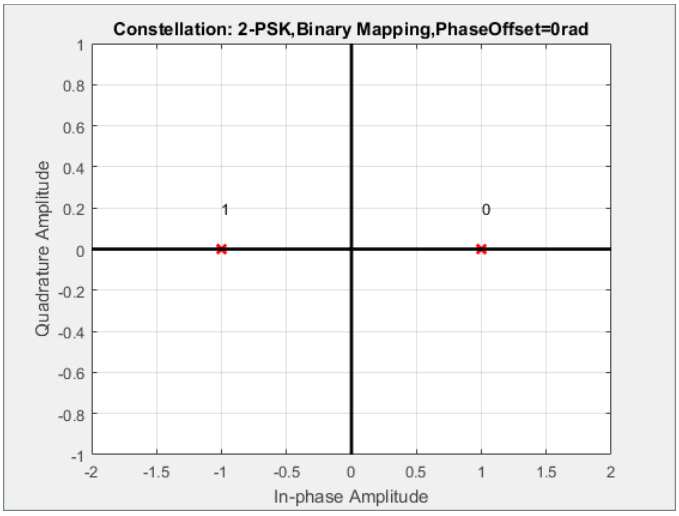

Figura 1. Constelação Obtida para o **BPSK**

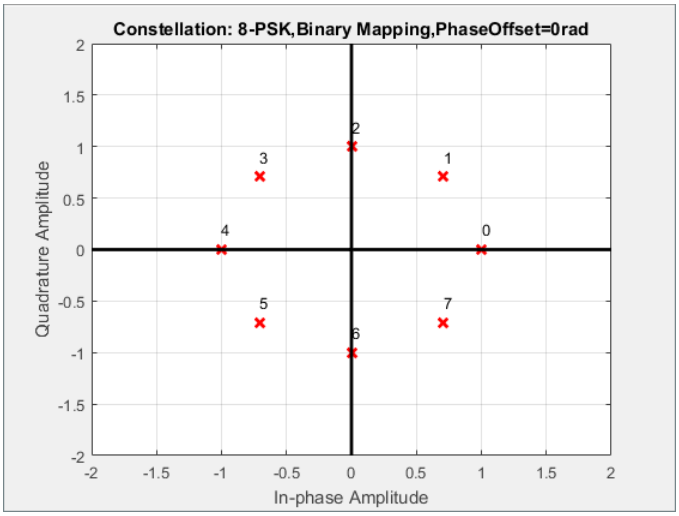

Figura 2. Constelação Obtida para o **8-PSK**

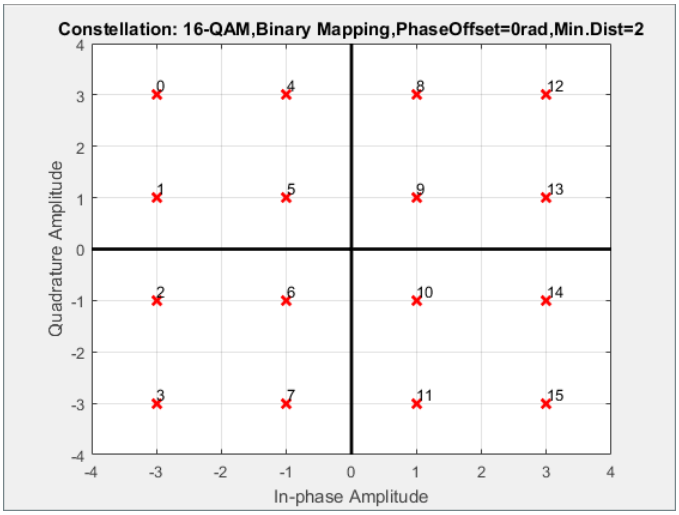

Figura 3. Constelação Obtida para o **16-QAM**

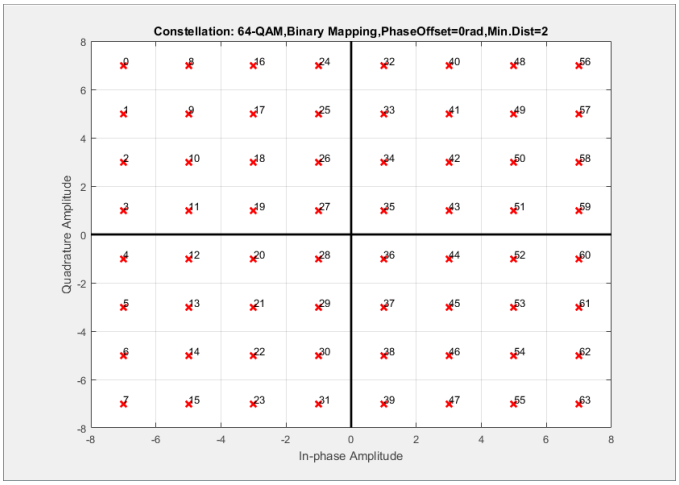

Figura 4. Constelação Obtida para o **64-QAM**

A partir destas constelações, é possível obter um mapeamento dos bits criados na Questão 1. Para isso, é necessário separar os bits que se encontram no vetor de bits **array_Nb** em grupos de bits que consistiam os símbolos das diferentes modulações. Para isso, criou-se uma função **separa** que, mediante envio do vetor e da ordem da modulação, obtém-se o valor em decimal do símbolo em questão. A função **separa** está demonstrada no Código 3 a seguir.

```
function [ v2, str, dec ] = separa( v1,M )
    v2=[];
    a=1;
    while(a<=length(v1))
        v2=[v2;v1(a:a+log2(M)-1)];
        a=a+log2(M);
    end
    str=num2str(v2);
    dec=bin2dec(str);
end
```

Código 3. Código Referente à Função de Separação de Símbolos

Essa função, como pode ser observado, retorna os símbolos de três formas diferentes: binário, *string* de binário e valor correspondente em decimal. Utilizou-se o valor correspondente em decimal de cada símbolo para que fosse feita a modulação do sinal para cada tipo específico de modulação. Isso está demonstrado no Código 4 a seguir.

```
%Mapeia
[~,~,bpsk]=separa(array_Nb,2);
[~,~,opsk]=separa(array_Nb,8);
[~,~,hexqam]=separa(array_Nb,16);
[~,~,ooqam]=separa(array_Nb,64);

bpsk=pskmod(bpsk,2);
opsk=pskmod(opsk,8);
hexqam=qammod(hexqam,16);
ooqam=qammod(ooqam,64);
```

Código 4. Código Referente ao Mapeamento dos Símbolos Presentes em **array_Nb**

Com essas etapas, tem-se o mapa de todos os símbolos existentes dentro de **array_Nb** para as modulações **BPSK**, **8-PSK**, **16-QAM** e **64-QAM**.

C. Questão 3

Esta etapa experimental consistiu em fitrar as sequências de símbolos de cada modulação com dois filtros, um de pulso de nyquist com fator de roll-off igual a 0.25 e um pulso retangular de largura T_s, na saída do filtro utilizou-se um fator de superamostragem de 16 amostras por símbolo. Para isso, criou-se uma função **PlotQ3** que realiza tais filtrações a uma taxa de bits de 1Mbps. Objetivamos, com isso observar, o espectro dos sinais. Essa função está representada no Código 5 a seguir.

```
function [ bpskrcosSig,qpskrcosSig,hexqamrcosSig,
    ooqamrcosSig, f1, rcos, array_Nb ] = PlotQ3( Nb
     )

array_Nb=rand(1,Nb);
array_Nb=array_Nb>0.5;
```

```
5
6  [~,~,bpsk]=separa(array_Nb,2);
7  [~,~,opsk]=separa(array_Nb,8);
8  [~,~,hexqam]=separa(array_Nb,16);
9  [~,~,ooqam]=separa(array_Nb,64);
10
11 bpsk=pskmod(bpsk,2);
12 opsk=pskmod(opsk,8);
13 hexqam=qammod(hexqam,16);
14 ooqam=qammod(ooqam,64);
15
16 sps = 16;
17
18 rcos = rcosdesign(0.25, 10, sps,'sqrt');
19 rect = [rectwin(sps)', zeros(1, 145)];
20
21 bpskrcosSig = upfirdn(bpsk, rcos, sps);
22 bpskrectSig = upfirdn(bpsk, rect, 16);
23
24 qpskrcosSig = upfirdn(opsk, rcos, sps);
25 qpskrectSig = upfirdn(opsk, rect, 16);
26
27 hexqamrcosSig = upfirdn(hexqam, rcos, sps);
28 hexqamrectSig = upfirdn(hexqam, rect, 16);
29
30 ooqamrcosSig = upfirdn(ooqam, rcos, sps);
31 ooqamrectSig = upfirdn(ooqam, rect, 16);
32
33 f1=PlotFigQ3(bpskrcosSig,' BPSK',bpskrectSig,Nb,2);
34 PlotFigQ3(qpskrcosSig,' 8-PSK',qpskrectSig,Nb,8);
35 PlotFigQ3(hexqamrcosSig,' 16-QAM',hexqamrectSig,Nb
     ,16);
36 PlotFigQ3(ooqamrcosSig,' 64-QAM',ooqamrectSig,Nb,64)
     ;
37
38 end
```

Código 5. Código Referente a Criação de Pulsos Filtrados por uma **rect** e um **raised cosine** a Partir de um **array_Nb** com **Nb** Bits Aleatórios

Nela, temos que entre as linhas 3 a 14 repetiu-se o que já foi feito e explicado em I-A e I-B. Após isso, indica-se o valor de amostragem que será utilizado na variável **sps**. Nas linhas 18 e 19 criam-se os pulsos referentes aos filtros que serão utilizados, sendo eles uma função **rect** e um cosseno levantado. Em seguida, criou-se, para cada tipo de modulação, uma variável para cada tipo de filtro que era aplicado sobre os valores complexos referentes ao vetor **array_Nb**, reconstruindo o sinal.

A partir dos sinais formados, chamou-se uma outra função que simplesmente gera uma imagem que contém os sinais formados pelos diferentes filtros para cada modulação. Tal função está representada no Código 6 a seguir.

```
1  function f=PlotFigQ3(Sinalrcos,Modulacao,Sinalrect,Nb,
       M)
2
3  Rb = 1e6;
4  sps = 16;
5  Rs = Rb/log2(M);
6  fsa = Rs*sps;
7
8  figure();
9  signalrcosft = fftshift(pwelch(Sinalrcos));
10 signalrectft = fftshift(pwelch(Sinalrect));
11 f = -1:2/(length(signalrcosft)-1):1;
12 f = f*(fsa/2);
13 plot(f,10*log10(signalrcosft));
14 title(strcat('Modula', Modulacao, ' Nb=',num2str(Nb)))
15     ;
16 hold on;
17 plot(f,10*log10(signalrectft));
18 grid
19 legend('Cosseno Levantado','Pulso Retangular')
20 xlabel('Frequia')
21 ylabel('Amplitude')
22
23 end
```

Código 6. Código Referente a Função de Geração de Gráficos com os Diferentes Filtros Aplicados Sobre o Sinal

Essa função recebe os dois sinais a serem plotados, uma *string* que indica o tipo de modulação e a quantidade de bits no sinal apenas para especificar cada imagem. Os sinais sofrem uma transformada de Fourier e seus espectros são mostrados em uma mesma imagem com suas especificações e plotados a partir de uma normalização da frequência por todo o espectro da função obtida.

Para gerarmos os espectros, foi chamada a função **PlotQ3** no *script* principal com os valores de **Nb** de 12, 120, 1200 e 12000, como mostrado no Código 7 a seguir.

```
41 %%%%%%%%%%Questo 3
42
43 PlotQ3(12);
44 PlotQ3(120);
45 PlotQ3(1200);
46 [bpskrcosSig,qpskrcosSig,hexqamrcosSig,ooqamrcosSig
     , f1, rcos, array_Nb]=PlotQ3(120000);
```

Código 7. Código Referente a Chamada da Função que Gerará o Espectro dos Símbolos Aleatórios

O resultado dessas chamadas foram 16 Figuras agrupadas de quarto em quarto para para cada tipo de modulação com as quatro variações de números de bits aleatórios já mensionados. Os resultados para essa etapa estão apresentados nas Figura 5, 6, 7 e 8 a seguir. as Variáveis que capturam o retorno da função com **Nb** = 12000 para uso futuro com os símbolos aleatórios iguais.

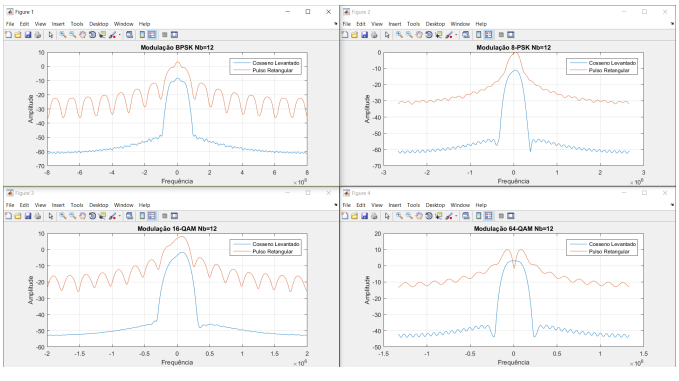

Figura 5. Espectros Gerados para Cada Modulação com **Nb** = 12.

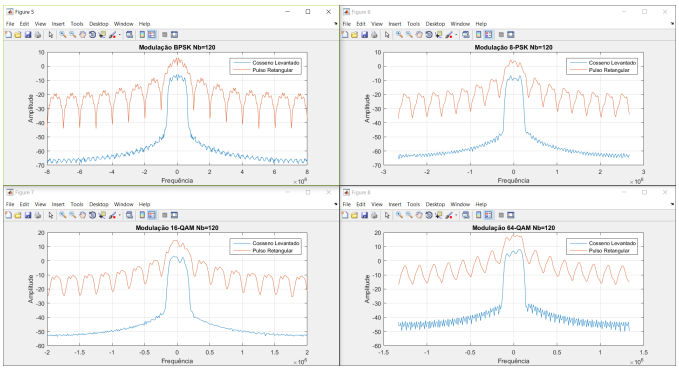

Figura 6. Espectros Gerados para Cada Modulação com **Nb** = 120.

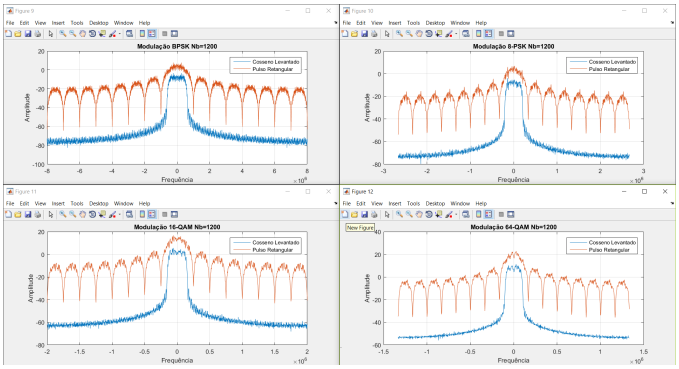

Figura 7. Espectros Gerados para Cada Modulação com **Nb** = 1200

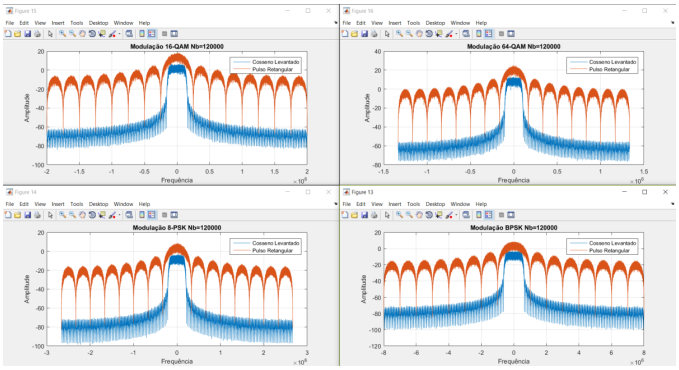

Figura 8. Espectros Gerados para Cada Modulação com **Nb** = 12000.

A partir dessas figuras, é possível verificar que quanto maior é o número de bits a serem considerados, os espectros tendem a se uniformizar devido a aleatoriedade dos sinais. Verifica-se também que tanto para as modulações **PSK** quanto para as modulações **QAM** houve uma menor variação dos valores dos espectros quanto maior era a sua ordem, por conta de um maior agrupamento de bits em um símbolo. Para todos os casos, o sinal que passou pelo filtro de pulso retangular possui uma banda muito maior do que os mesmos sinais que passaram pelo filtro de cosseno levantado, mostrando que o uso do cosseno levantado possui a vantagem sobre a banda, porém possui uma maior chance de ocorrer interferência intersimbólica caso o receptor e o transmissor não estejam corretamente sincronizados.

A banda do sinal retangular deveria ser a função **sinc** e a banda do cosseno levantado deveria ser algo próximo a um pulso retangular com um pequeno fator de correção. A banda do pulso retangular e do pulso de nyquist obtidos se mostram muito próximos dos esperados, pois a banda essencial do pulso retangular é aproximadamente a taxa de transmissão de símbolos e a banda do cosseno levantado é

$$B_t = \frac{(1+\rho)R_s}{2}$$

É válido comentar que como a taxa de transmissão de bits é constante quando transmitimos símbolos com um maior número de bits,ou seja, maior M, diminui-se o número de símbolos que devem ser enviados e, consequentemente, a banda. Como pode ser visto na equação anterior.

D. Questão 4

Para esta etapa experimental, gerou-se um ruído gaussiano complexo com variância N_0. Para isso, especificou-se duas razões de energia do simbolo por N_0, sendo elas:

$$\frac{E_s}{N_0} = 3dB$$

e

$$\frac{E_s}{N_0} = 10dB$$

A partir desses valores, fez-se os valores **SNR** a partir da razão E_s por N_0 subtraída do valor da taxa de amostragem em dB. O vetor resultante dessa operação foi passada para o método **awgn** cuja função é de adicionar um ruído branco gaussiano ao sinal. O sinal em que o ruído foi adicionado foi o **BPSK** com **Nb** igual á 12000 bits aleatórios após aplicação do filtro de cosseno levantado. A partir desse sinal, foi chamada uma Função **PlotFigQ4** que plotava o espectro do sinal em questão e especificava qual a SNR referente a cada imagem. Essa função está especificada no Código 8 e seus resultados estão representados na Figura 9 a seguir.

```
function PlotFigQ4( Plotar,db )

figure();
noisysig = fftshift(pwelch(Plotar));
f = -1:2/(length(noisysig)-1):1;
plot(f,10*log10(noisysig));
title(strcat('Modula BPSK, Es/N0=', db, ' db'));
grid
legend('Sinal ruidoso')
xlabel('Frequia Normalizada')
ylabel('Amplitude')

end
```

Código 8. Código Referente a Chamada da Função que Gerará o Espectro dos Símbolos Aleatórios

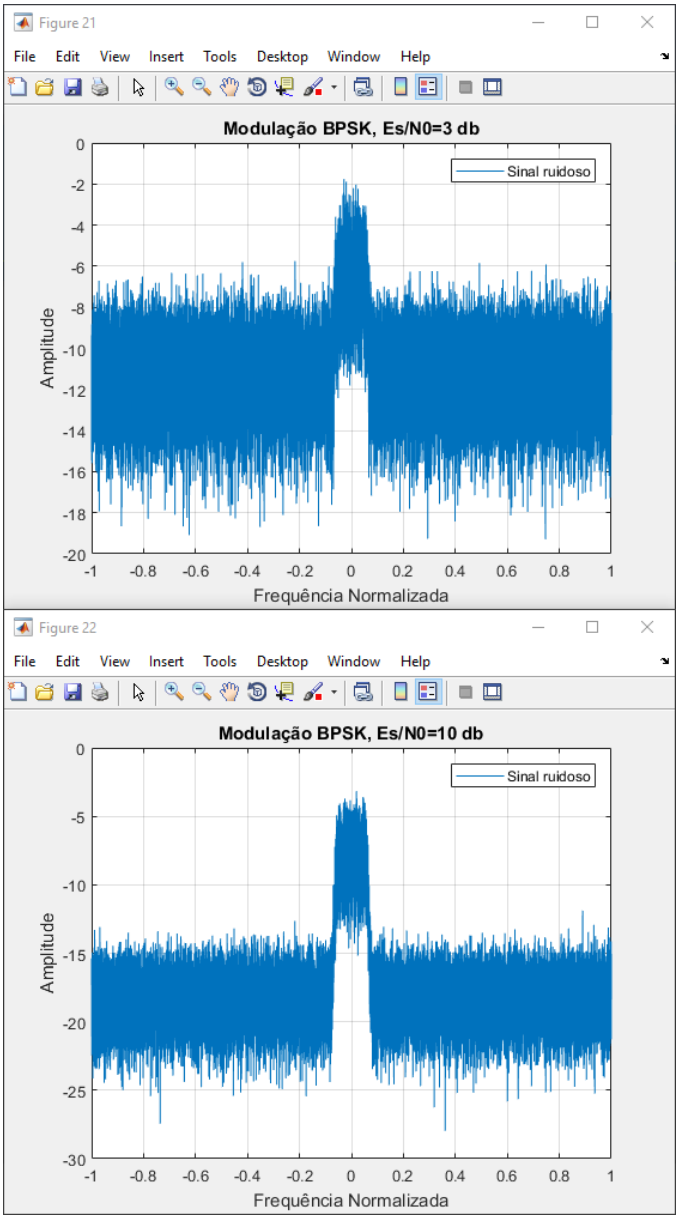

Figura 9. Espectros Gerados para **BPSK** com **Nb** = 12000 com Cosseno Levantado e com Ruído Branco de 3dB e 10dB.

Nessa Figura, é possível perceber que o ruido introduzido distorce o sinal consideravelmente quando comparado ao sinal sem ruído mostrado na Figura 8. Já quando comparados entre si, o sinal com a maior razão sinal ruído é mais característico e evidente, sendo mais próximo do sinal esperado quando comparado com a razão de 3dB. Ambos os sinais são destacáveis e visíveis, porém, para o caso de 3dB, a parte em que o sinal está presente e a parte em que o sinal só é constituído apenas de ruído se apresentam no mesmo nível em certas frequências, tornando mais difícil a diferenciação de ruído e do sinal nesses momentos, o que não acontece para a razão de 10dB.

E. Questão 5

Esta etapa experimental consistiu em filtrar o sinal ruidoso gerado em I-D a partir de um filtro casado, amostrou-se o sinal, detectou-se seu valor e seu símbolo mais provável para, assim, determinar os bits transmitidos. Isso foi feita a partir de uma Função chamada **CalculaBER**, demonstranda no Código 9 a seguir.

```
function [ BER ] = CalculaBER( array_Nb,Sig,rcos,M,
    mod )
BER=zeros(1,10);
EsN0 = 0; %dB
sps=16;
rcvFilter = fliplr(rcos);

while(EsN0<=30)
snr = EsN0 - 10*log10(sps);

noisySig = awgn(Sig, snr, 'measured');

rcvSig = upfirdn(noisySig, rcvFilter, 1, 16);
rcvSig = rcvSig(11:end-10);

scatterplot(rcvSig);
title(strcat(num2str(M),mod,' E_s/N_0=',num2str(EsN0)))

if(strcmp(mod,'psk'))
   rcv_data = pskdemod(rcvSig,M);
else
   rcv_data = qamdemod(rcvSig,M);
end

rcv_data1 = rcv_data';
rcv_data2=dec2bin(rcv_data1);
rcv_data3=reshape(rcv_data2',1,[]);
rcv_data4 = rcv_data3=='1';

rcv_data=rcv_data4;
Temp = rcv_data == array_Nb;
Temp = (length(Temp) - sum(Temp))/length(Temp);
BER(EsN0/3+1)=Temp;
EsN0=EsN0+3;
end
end
```

Código 9. Código Referente a Função que Calcula BER e Indica o Sinal Recebido

Nela, recebe-se o **array_Nb** que se refere ao *array* de 12000 bits aleatórios enviados, **Sig** o sinal sinal enviado já modulado, **rcos** o espectro do coseno levantado com taxa de amostragem de 16, **M** a ordem da modulação em questão que é identificada pela próxima variável - **mod**.

Foi feita simulações para E_b/N_0 de 0 até 30, caracterizado pelo *while*. Para cada ciclo, calculou-se o a razão sinal ruído e a utilizou para gerarmos o sinal recebido com ruído gaussiano, representado pela variável **noisySig**. A partir desse sinal, aplicou-se o filtro de coseno levantado e exclui-se as extremidades que são apenas resquícios do sinal na frequência no início e no final da que possuem detalhes do filtro que não representam o filtro. Tais sinais foram representados graficamente. As Figuras 10, 11, 12, 13, a seguir apresentando apenas os gráficos obtidos para um E_b/N_0 igual a 0, 9, 21 e 30 de até o valor máximo de 30 para cada modulação.

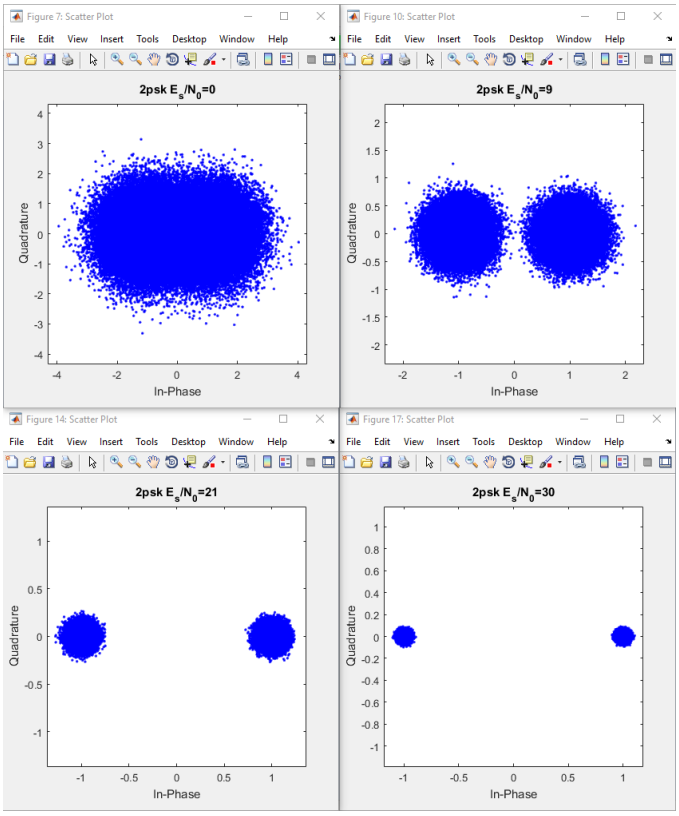

Figura 10. Sinais recebido da modulação **bpsk** com diferentes razões de sinal ruído.

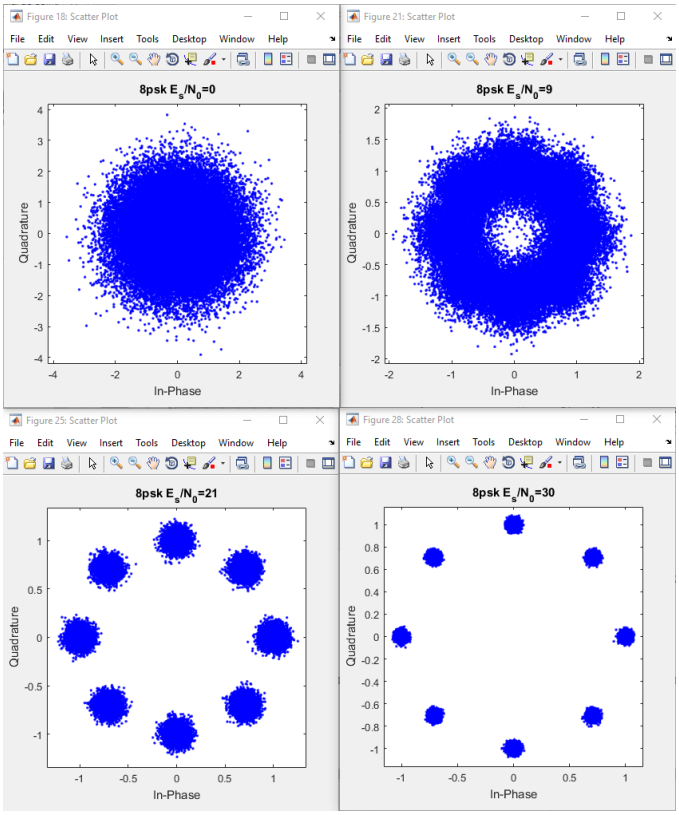

Figura 11. Sinais recebido da modulação **8-psk** com diferentes razões de sinal ruído.

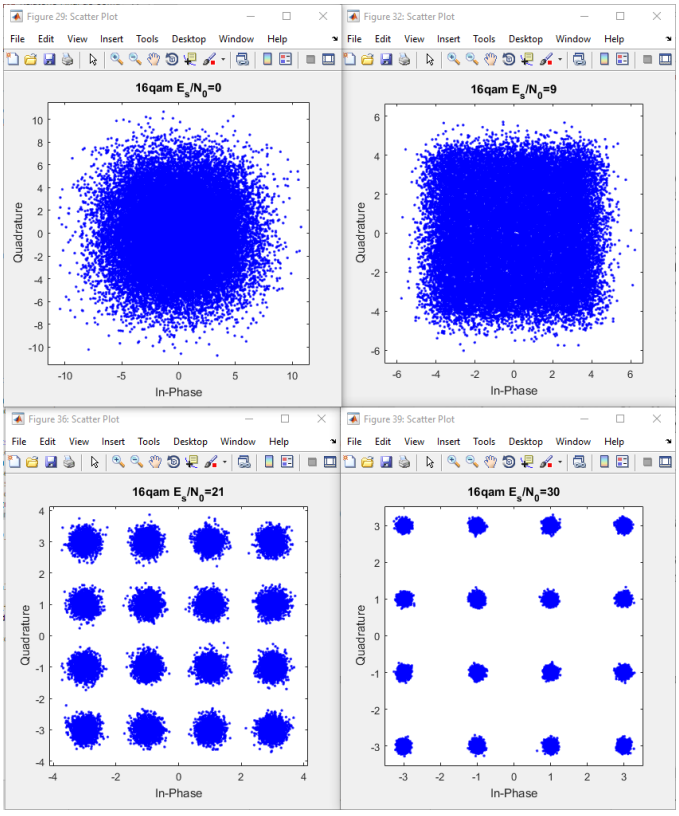

Figura 12. Sinais recebido da modulação **16-qam** com diferentes razões de sinal ruído.

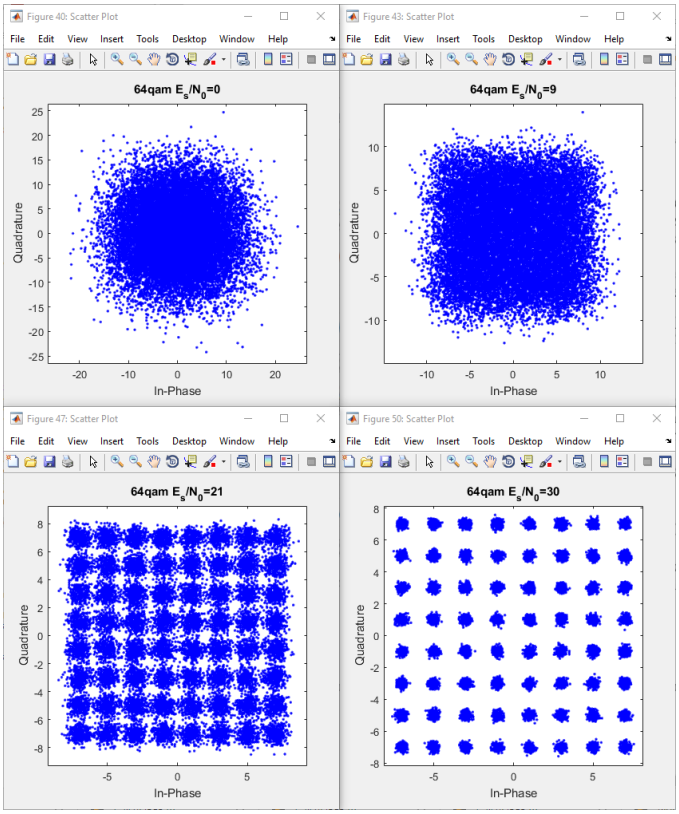

Figura 13. Sinais recebido da modulação **64-qam** com diferentes razões de sinal ruído.

Para todas as modulações, quando a razão E_b/N_0 é igual a 0, temos um sinal incompreensível, como de se esperar, pois não há destaque do sinal com relação ao ruído. Para E_b/N_0 é igual a 9, a única modulação que é capaz de discernir o sinal recebido é a modulação **bpsk**, com poucos possíveis erros. A partir da razão igual a 21 db é possível discernir todos os sinais com chances de erros próximos de 0 para todas as modulações.

A função **CalculaBER** também retorna a taxa de erro ocorridos pela demodulação. Isso é calculado a partir da verificação de quantos bits demodulados foram iguais aos bits presentes em **array_Nb** e calcula-se a porcentagem de erros pela razão entre a diferença entre o total de bits transmitidos e a quantidade de bits corretos pelo total de bits transmitidos e adiciona-se isso a um vetor que é retornado da função. Para cara retorno, fez-se um gráfico que mostrava a queda da taxa de erro pelo aumento da razão sinal ruído juntamente com os valores teóricos de tais valores. Isso foi feito a partir da função **PlotQ5**, representada a seguir.

```
1  function PlotQ5( BERBPSK,BER8PSK,BER16QAM,BER64QAM )
2
3  EsNo=0:3:30;
4
5  berbpsk = berawgn(EsNo-10*log10(log2(2)),'psk',2,'
       nondiff');
6  ber8psk = berawgn(EsNo-10*log10(log2(8)),'psk',8,'
       nondiff');
7  ber16qam = berawgn(EsNo-10*log10(log2(16)),'qam',16);
8  ber64qam = berawgn(EsNo-10*log10(log2(64)),'qam',64);
9
10 PlotFigQ5('BER tico e simulado da bpsk','Tea','
       Simulado',berbpsk,BERBPSK);
11 PlotFigQ5('BER tico e simulado da 8-psk','Tea','
       Simulado',ber8psk,BER8PSK);
12 PlotFigQ5('BER tico e simulado da 16-qam','Tea','
       Simulado',ber16qam,BER16QAM);
13 PlotFigQ5('BER tico e simulado da 64-qam','Tea','
       Simulado',ber64qam,BER64QAM);
14
15 figure();
16 semilogy(EsNo,berbpsk,'--');
17 hold on
18 semilogy(EsNo,BERBPSK);
19 semilogy(EsNo,ber8psk,'--');
20 semilogy(EsNo,BER8PSK);
21 semilogy(EsNo,ber16qam,'--');
22 semilogy(EsNo,BER16QAM);
23 semilogy(EsNo,ber64qam,'--');
24 semilogy(EsNo,BER64QAM);
25 title('Compara BERs para cada modula');
26 grid
27 legend('BPSK Teo','BPSK Simulado','8-PSK Teo','8-PSK
       Simulado','16-QAM Teo','16-QAM Simulado','64-
       QAM Teo','64-QAM Simulado');
28 xlabel('E_s/N_0 (dB)')
29 ylabel('Bit Error Rate')
30 ylim([10^-200,10^10])
31
32 end
```

Código 10. Código Referente a Função que Calcula os Valores Teórico da Taxa de Erro por Diferentes *SNRs* para Todas as Modulações Utilizadas

Nela, nas linhas 5 a 8 verifica-se os cálculos para os valores teóricos das taxas de erros por razão sinal ruído que serão plotados conjuntamente com os valores obtidos em 9. Para calcular esses valores utilizamos a função **berawgn** do próprio *matlab*. Os resultados obtidos dos gráficos comparativos estão representados nas Figuras 14, 15, 16, 17 a seguir, que melhor representam o comportamento de cada modulação com o aumento da razão sinal ruído.

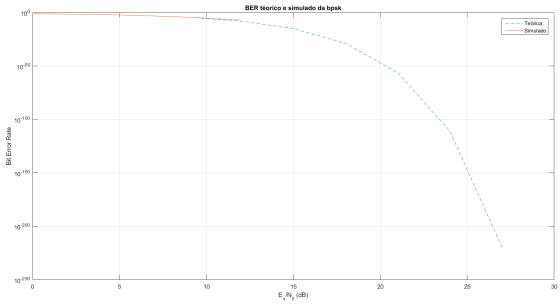

Figura 14. Sinais Teóricos e Práticos para Taxas de Erros por *SNRs* para Modulação **BPSK**.

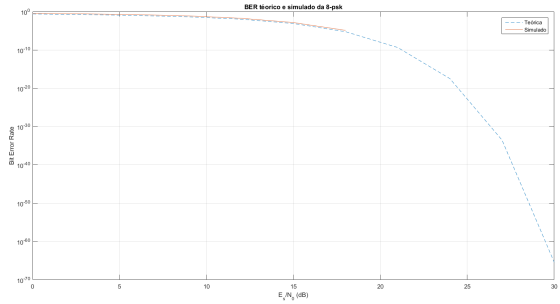

Figura 15. Sinais Teóricos e Práticos para Taxas de Erros por *SNRs* para Modulação **8-PSK**.

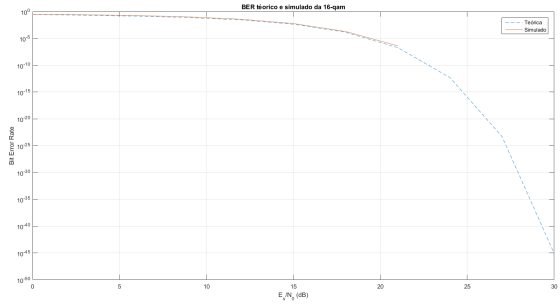

Figura 16. Sinais Teóricos e Práticos para Taxas de Erros por *SNRs* para Modulação **16-QAM**.

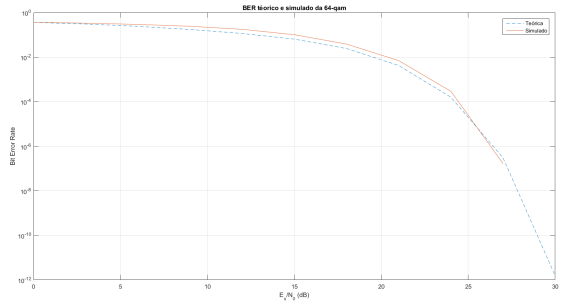

Figura 17. Sinais Teóricos e Práticos para Taxas de Erros por *SNRs* para Modulação **64-QAM**.

Fez-se, também, um gráfico comparativo entre todos os resultados obtidos, práticos e teóricos, para todas as modulações, apresentada na Figura 18 a seguir.

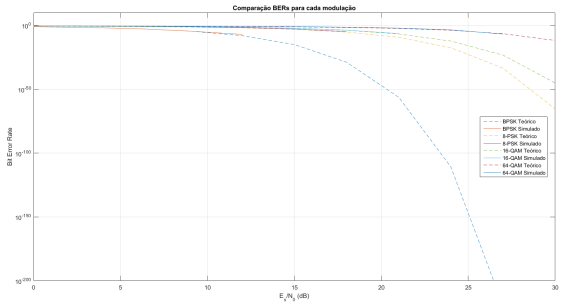

Figura 18. Sinais Teóricos e Práticos para Taxas de Erros por *SNRs* para Todas as Modulação Utilizadas.

Essa última figura nos mostra como as taxas de erros são relacionadas com as diferentes modulações, sendo as de maior banda possuem uma taxa de erro menor. Como podemos observar nos gráficos a curva teórica continua até o final da simulação enquanto a curva experimental acaba na metade dos elementos do eixo das abcissas. Isso ocorre porque como o número de bits da mensagem é limitado possuímos um pequeno espaço amostral, quando aplicamos uma taxa de erro pequena o processo está sujeito à sua aleatoriedade, sendo assim não ocorreu erro com nenhum dos elementos enviados. Como o gráfico contem o *log* do número de erros, nos casos em que estes são nulos, o próximo ponto não existe já que o *log* de 0 não está definido, sendo assim não é plotado no gráfico.

F. Questão 6

Neste item devemos aplicar a codificação de *Golay* para a **BPSK**, comparar os resultados teóricos com os simulados e comparar, também, o resultado teórico da **BPSK** com e sem a codificação de *Golay*.

Para aplicarmos a codificação utilizamos um código encontrado na internet no endereço da referência 2. A função *golay-codec* uma vez codificado o sinal repetimos os procedimentos explicados em I-D,ou seja, codificamos o sinal, aplicamos um ruído branco ao sinal, decodificamos o sinal, comparamos a mensagem recebida com a enviada e plotamos a curva da taxa de erros de bit pela razão sinal ruído.

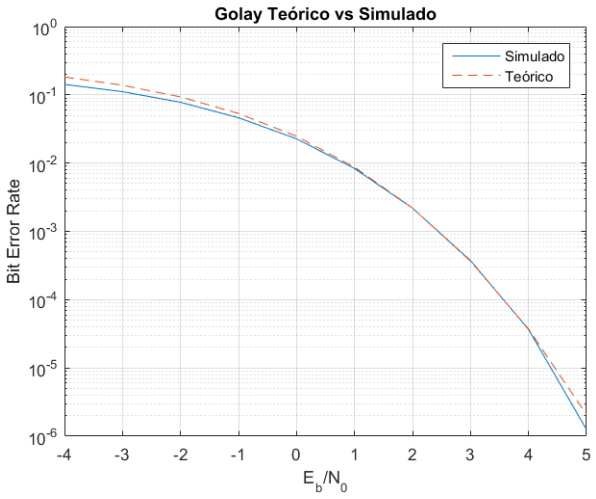

Figura 19. Comparação do resultado teórico e simulado com a aplicação da codificação de *Golay*.

Como podemos ver no na Figura 19 acima a curva do resultado simulado está sempre próxima ao resultado teórico e distancia-se apenas no começo e no fim da curva. Isso ocorre pelo mesmo motivo explicado no item anterior.

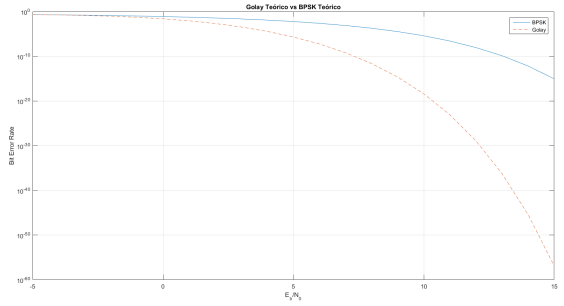

Figura 20. Comparação dos resultados teóricos com e sem codificação de *Golay*.

O gráfico apresentado na Figura 20 acima mostra que a codificação *Golay* reduz significativamente o número de erros na transmissão, pois este código é capaz de corrigir quaisquer combinações, de até 3 erros, em cada grupo de 12 bits da mensagem, a um custo de adição de redundância na mensagem.

II. Conclusão

Os resultados obtidos neste trabalho ilustraram que a banda da mensagem depende do pulso utilizado e da modulação podendo em um sistema real ambas as características serem ponderadas de forma que o sistema opere da melhor forma possível, para cada caso, na banda disponível.

Observamos também que a taxa de erros depende tanto da razão sinal ruído quanto da modulação escolhida. Portanto podemos ponderar esses fatores para que a uma dada potência

a transmissão seja confiável, em outras palavras, possua uma taxa de erros arbitrariamente pequena.

Utilizando a codificação adequada, neste caso a codificação *Golay*(24,12), podemos reduzir a taxa de erro ainda mais melhorando o sistema e permitindo melhorar outros aspectos que influenciam a taxa de erro a custo de um aumento nas mensagens a serem transmitidas.

REFERÊNCIAS

[1] Documentação de help do Matlab
https://www.mathworks.com/help

[2] Endereço onde foi obtida a função que aplica a codificação e decodificação de *Golay*
https://www.mathworks.com/matlabcentral/fileexchange/23341-golaycodec