Moduladores e Demoduladores CPFSK, GMSK, MSK com canal

Emmanuel K. M. Johnson
180011464
Engenharia Eletrônica - UnB
johnson.emmanuel322@gmail.com

João Marcelo Almeida de Macedo
170106080
Engenharia Eletrônica - UnB
joaomacedodf@hotmail.com

1. Introdução

As modulações digitais são fundamentais direta ou indiretamente do cotidiano de toda a população em torno do globo, sendo muito usadas em todos os níveis de telecomunicações. Divididas em muitas formas diferentes, estas modulações podem se concentrar tanto em amplitude quanto em fase ou frequência. Os canais são os meios por onde o sinal é transmitido e podem influenciar na modulação ou demodulação do sinal, exigindo correções como, por exemplo, os equalizadores.

As modulações estudadas neste trabalho são a CPFSK, MSK E GMSK. Todas possuem a característica de uma fase contínua. Esta continuidade evita sinais com grandes lobos laterais e garante um melhor uso da banda do sinal. Mas esta vantagem aumenta a complexidade das arquiteturas de comunicação, principalmente o receptor.

Os canais estudados no trabalho serão o de Rayleigh, Rician e o canal com filtro FIR. Será comparado a taxa de erro de bits das modulações acima com sua respectiva BER considerando apenas o AWGN.

2. CPFSK

Na CPM (modulação de fase contínua), a fase da portadora é gradualmente alterada de um símbolo para outro símbolo. a fase instantânea depende da fase dos símbolos anteriores. Portanto, o CPM é classificado como modulação com memória. A modulação e demodulação do CPM são mais complicadas pelo fato de que a fase de um símbolo depende da fase dos símbolos anteriores.

O Continuous Phase Frequency Shift Keying (CPFSK) é uma variação da técnica de Frequency Shift Keying (FSK), que elimina as descontinuidades de fase. Na modulação FSK binária convencional, a frequência instantânea da portadora é comutada entre uma das duas frequências. Devido à comutação abrupta

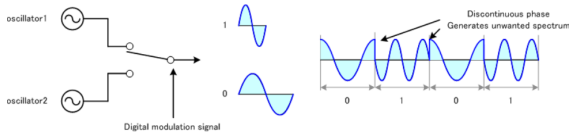

Figure 1: Imagem da modulação FSK.

entre as duas frequências, a fase não é contínua no FSK convencional. No CPFSK binário, a continuidade de fase suave é assegurada definindo o sinal do transmissor como pode ser visto na equação 1 onde E_b é a média da energia, T_b é o periodo de bit, f_c é a frequência da portadora base e $\theta(t)$ é a evolução da fase.

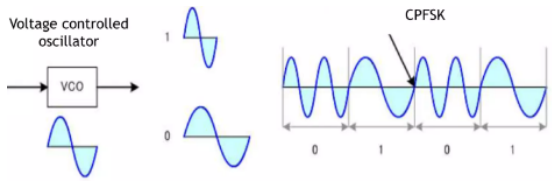

Figure 2: Imagem da modulação CPFSK.

$$s_i(t) = \sqrt{\frac{2E_b}{T_b}} cos(2\pi f_c t + \theta(t)) \quad , \quad 0 \leq t \leq T_b \tag{1}$$

Em qualquer instante de tempo, a evolução da fase pode ser dada como;

$$\theta(t) = \theta(0) + \frac{\pi h}{T_b}\int_0^t b(t)dt \qquad (2)$$

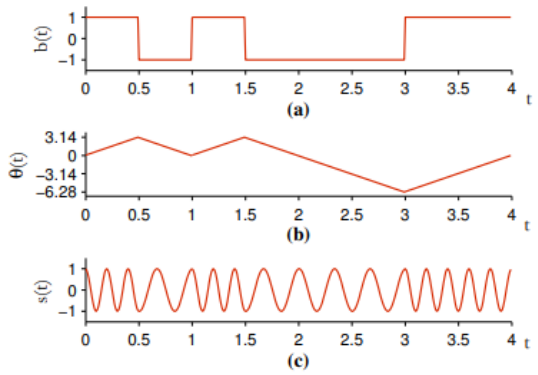

Figure 3: (a) Sequencia de Informação, (b) Evolução da fase de CPFSK do sinal (c)O sinal CPFSK modulado.

Os diagramas de fase também são chamados de árvores de fase. Observa-se que as árvores de fase para CPFSK são lineares por partes como consequência do pulso g(t) ser retangular. Trajetórias de fase e árvores de fase mais suaves são obtidas usando pulsos que não contêm descontinuidades, como a classe de cossenos elevados. A trajetória de fase para CPFSK pode ser visto na figura 4.

Como apenas a fase é modulada, o CPFSK possui um envelope constante. Isso permite o uso de um amplificador Classe C barato e com baixo consumo de energia. Não há distorção de intermodulação no amplificador e, portanto, não há respingos de espectro. Geralmente, porém, não podemos carregar tantos bits/s em uma determinada largura de banda com CPFSK quanto podemos com modulação linear (por exemplo, BPSK com raiz quadrada de cosseno elevado), porque usamos apenas uma coordenada (fase) e ignoramos a outra (raio). Devido às vantagens mencionadas acima, os esquemas CPFSK são os preferidos esquemas de modulação na maioria das aplicações de comunicação sem fio. O custo do amplificador é aproximadamente proporcional à sua potência, e os PAs dominam o custo das estações base. Por outro lado, detectores caros e que consomem muita energia também não são bem-vindos em aparelhos.

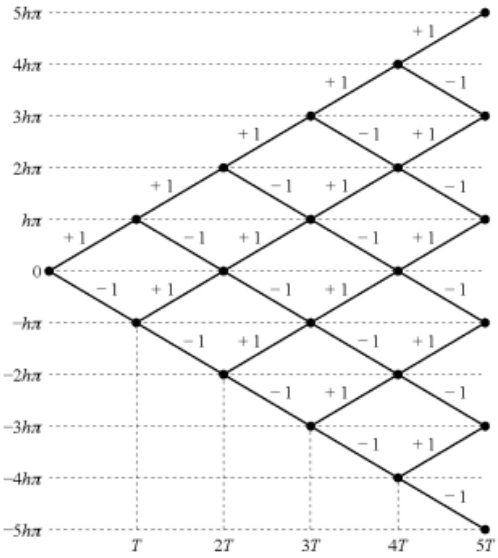

Figure 4: Imagem da Trajetória de fase para CPFSK..

3. *Minimum Shift-Keying (MSK)*

A modulação MSK carrega diversas características positivas: Ela mantém um envelope constante, um espectro compacto se comparado a modulações como QPSK e OQPSK, com boa performance em relação a sua taxa de erros, imutabilidade frente a limite em banda. Esta modulação é nada mais do que um caso especial da modulação CPFSK binária com índice de modulação igual a 0.5 e usando um pulso retangular de duração T. Assumindo então $\theta(0) = 0$ (zerando as contribuições de fase em t ¡ 0) e $h = 2f_dT = 0.5$, a componente em fase e quadratura da modulação MSK tem o formato senoidal e estão nas equações 3 e 4.

$$s_I(t) = \pm\sqrt{\frac{2E_b}{T_b}}cos(\frac{\pi t}{2T_b}) \quad , \quad -T_b < t < T_b \qquad (3)$$

$$s_Q(t) = \pm\sqrt{\frac{2E_b}{T_b}}sen(\frac{\pi t}{2T_b}) \quad , \quad 0 < t < 2T_b \qquad (4)$$

Onde E_b é a energia do bit e T_b é o tempo de duração de cada bit. O nome *Minimum Shift-Keying* vem do fato do índice de modulação ter a distância mínima de separação em frequência para garantir a ortogonalidade dos sinais. Uma representação gráfica para o sinal em fase e em quadratura está na figura 5.

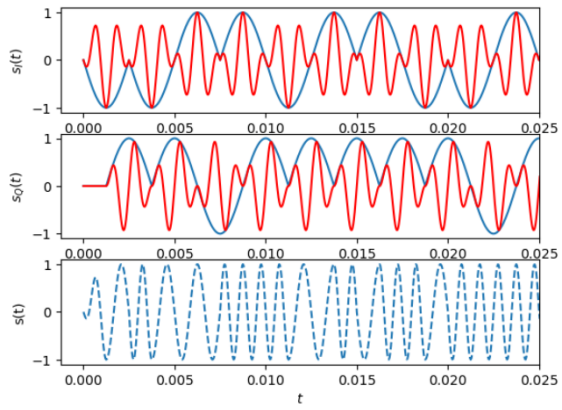

Figure 5: Componente em fase (primeiro) comparado com o componente em quadratura (segundo) do sinal MSK em banda-passante (terceiro). Os sinais em vermelho são as componentes com adição da portadora. Em azul, banda básica.

Uma forma compacta de escrever o sinal MSK em banda passante está na equação 5, para t entre 0 e $2T_b$. Nesta expressão, é importante ressaltar que a sequência de informação em fase e quadratura estão omitidas, mas consideradas partes de s_I e s_Q. A imagem 6 ilustra um modulador MSK, projetado de modo a enfatizar sua equivalência com a modulação OQPSK. Ele divide o os bits em componentes em fase e quadratura, realiza um *upscale* para aumentar o tempo de duração de cada bit e aplica um atraso na componente em quadratura antes de multiplicar as componentes por suas respectivas senóides, gerando assim o sinal modulado descrito na equação 5.

$$s(t) = s_I(t)cos(2\pi f_c t) - s_Q(t)sen(2\pi f_c t) \qquad (5)$$

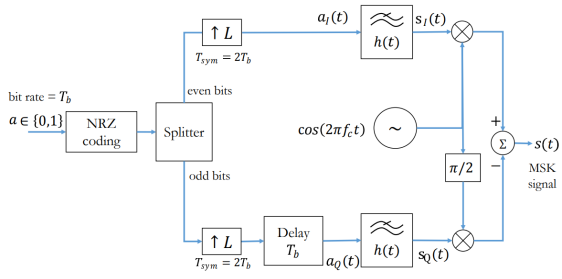

Figure 6: Modulador MSK com estrutura similar a da modulação OQPSK.

A equivalência com o OQPSK acontece porque ambas as modulações tem suas componentes em fase e em quadratura defasadas por T_b segundos (ou meio símbolo), mas o MSK tem suas curvas em formato senoidal em meio-ciclo enquanto o OQPSK utiliza-se de ondas retangulares.

O demodulador MSK está na figura 7, onde é possível ver que os sinais em fase e quadratura são multiplicados pelo módulo das funções de meio ciclo. Um integrador então recupera a informação no tempo e cada componente é amostrado para recompor o sinal que havia sido dividido em bits pares e ímpares. Um decisor no fim utiliza os dados da amostragem para estimar o bit correto. Por ser uma modulação com memória, é de se esperar que a arquitetura de demodulação seja mais complexa do que as sem memória.

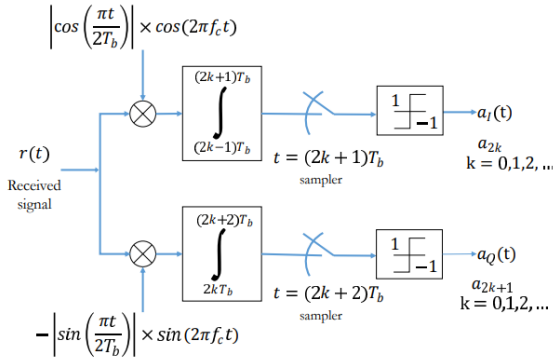

Figure 7: Demodulador MSK com equivalência enfatizada ao OQPSK.

Por ser um caso especial da modulação CPFSK binária, ou seja, por ser uma modulação cuja fase do sinal precisa ser estritamente contínua, a modulação MSK possui memória, ou não poderia ser capaz de garantir a continuidade do sinal em fase.

A taxa de erro de bits é dada pela equação 6, sendo essencialmente a mesma da modulação BPSK convencional. Sendo a função erro uma sigmoide, a energia de cada bit é inversamente proporcional a probabilidade de erro de bits, enquanto N_0 é diretamente proporcional a taxa de erro. A imagem 8 apresenta sua curva se comparada a modulação BPSK. É possível perceber similaridades entre as duas mesmo com as outras vantagens intrínsecas ao MSK.

$$P_b = \frac{1}{2} erfc(\sqrt{\frac{E_b}{N_0}}) \quad (6)$$

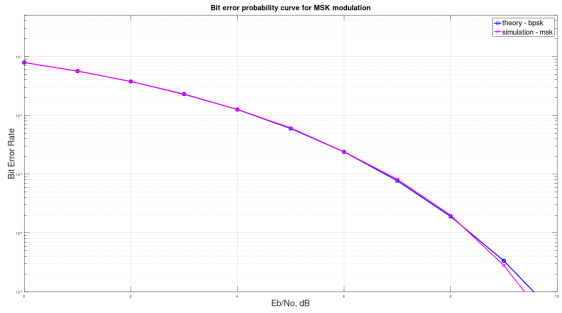

Figure 8: Taxa de erro de bits da modulação MSK.

A densidade espectral de potência para a modulação MSK se encontra na figura 9, onde é possível um decaimento acentuado dos lobos em frequências mais altas se comparado a modulações como BPSK. O primeiro lobo decai rapidamente com a frequência. Comparado ao QPSK, é possível perceber então que a interferência inter-canal é menor para o MSK. O lobo principal do MSK é mais largo, indicando mais energia na banda *null-to-null*. Entretanto, pode ser considerada uma desvantagem caso seja necessário uma banda muito estreita.

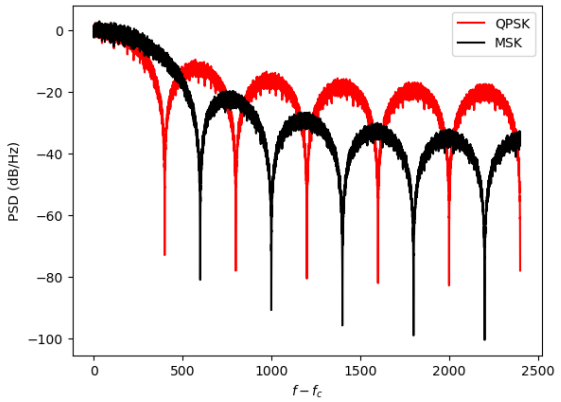

Figure 9: Densidade espectral de potência da modulação MSK.

4. *Gaussian Minimum Shift Keying* - GMSK

A modulação GMSK é semelhante a modulação MSK, pois é uma modificação da mesma, onde sequência de bits de entrada passa por um filtro passas baixas Gaussiano. O objetivo de utilizar a modulação GMSK é suavizar as transições de frequência, ela também reduz a complexidade dos sistemas tornando seu custo muito baixo em relação às demais modulações existentes. Possui memória.

Na figura 10 podemos observar que o modulador GMSK consiste em um modulador MSK com a adição do filtro passa baixa gaussiano.

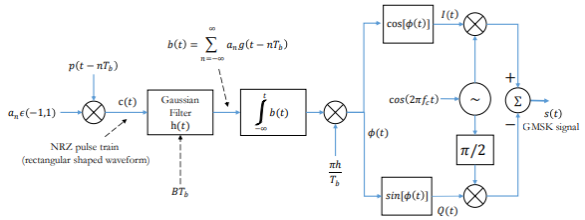

Figure 10: Arquitetura do Modulador GMSK

Já para a demodulação podemos observar na Figura 11, onde, através de um filtro de passagem de banda, o sinal de entrada é convertido para uma frequência intermediária. Com a portadora recuperada, o sinal é traduzido para a banda base. Os filtros LPF servem para remover a componente de frequência $2f_{if}$. Por fim os sinais de banda básica são utilizados pelo algoritmo de decisão para a detecção de dados.

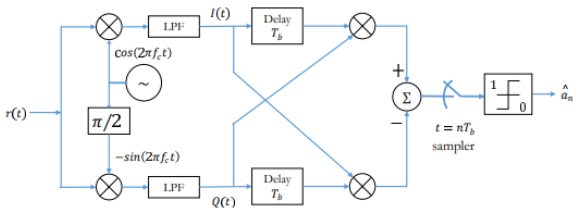

Figure 11: Arquitetura do Demodulador GMSK

Para a modulação GMSK, temos os seguintes gráficos na figura 12, onde é possível analisar um comportamento contínuo, outro ponto interessante é o formato de um círculo na sua constelação, isso se dá

principalmente pelo fato da modulação GMSK ter uma amplitude constante.

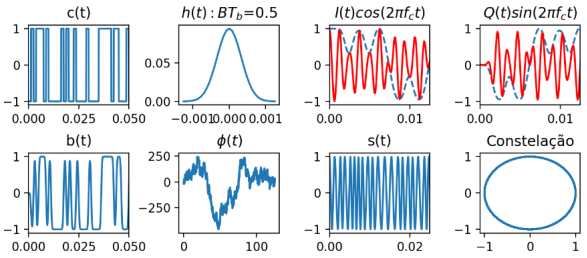

Figure 12: Gráficos da Modulação GMSK

A taxa de erro de bits (BER), para a modulação GMSK, é observada na figura 13. Temos então que a probabilidade de erros aumenta com o aumento de bits transmitidos e da razão $\frac{Eb}{N0}$.

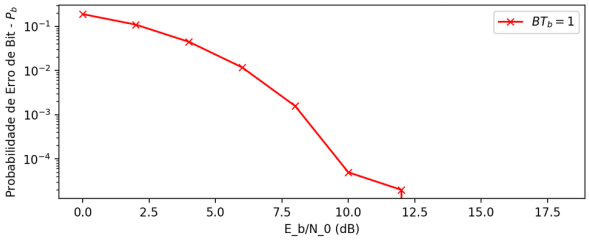

Figure 13: Probabilidade de Erro de Bits GMSK

A densidade espectral de potência da modulação GMSK é calculada usando o método de estimativa de espectro Welch em janela. Na Figura 14 temos o espectro para diferentes valores de BT_b.

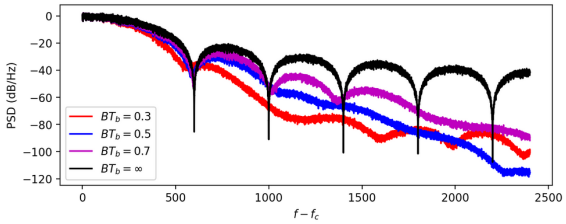

Figure 14: Densidade Espectral de Potência da Modulação GMSK

Como comparação de referência, a imagem 15 contém a densidade espectral de todas as modulações estudadas considerando BT_b = 0.7 para o GMSK. É possível observar o quão maior é o lobo principal do CPFSK. Isso pode aponar uma maior interferência inter-canal para o caso do CPFSK.

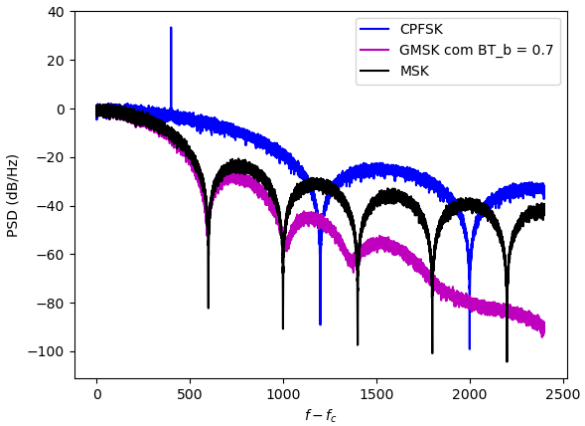

Figure 15: Densidade Espectral das modulações estudadas.

5. Canais

5.1.AWGN

Ao se tratar de canais, o modelo mais simples é chamado de *Additive white Gaussian Noise*. Como o nome indica, este canal adiciona um ruído aleatório de distribuição normal ao sinal, com potência uniforme em todas as frequências. Ela é mais utilizada como um modelo simples teórico onde nenhum fator relevante do ambiente é levado em conta. Por isso, não costuma ser usado em aplicações reais em ambientes urbanos.

Ainda assim, ele é tomado como um ”modelo ideal” em relação aos canais, visto que há bastante teoria por trás deste canal e como lidar com sua interferência. Por isso, ele será utilizado no decorrer desta seção como comparação entre os diversos tipos de canais.

Para o MSK, sua taxa de erros de bit teórica é descrita pela equação 6, citada acima, e sua taxa de erro de bits simulada em AWGN já foi mostrada na figura 8. De forma similar, a taxa de erro de bits da modulação GMSK em AWGN é mostrada na figura 13.

O modelo mostrado na figura 16 é o que será implementado neste relatório. Será tratado como uma

variável aleatória com distribuição normal já implementada por uma função extraída das referências utilizadas. Ele se somará ao sinal transmitido.

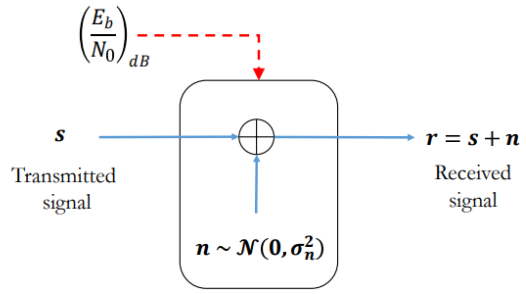

Figure 16: Modelo de aplicação do ruído AWGN.

5.2.Rayleigh Fading

O canal Rayleigh Fading é um canal que aplica um ruído ao sinal de acordo com uma distribuição de Rayleigh, a componente radial da soma de duas variáveis aleatórias gaussianas. Este modelo é útil em sinais que refletem entre o transmissor e o receptor. O valor associado ao canal é então multiplicado ao sinal, e ambos são somados ao ruído *AWGN*. O termo *Fading* refere-se a variações na amplitude causada por variações no tempo das fases que chegam ao receptor. Em um canal multipercurso, o sinal enviado chega no receptor com fases diferentes, o que pode causar uma soma (e o vetor resultante aumenta) ou uma subtração (onde o sinal recebido se torna muito pequeno).

Transmissões em canais limitados em banda são vulneráveis a interferência inter-simbólica. Uma forma de lidar com este problema é através do uso de equalizadores. Eles são uma espécie de filtro digital que diminui esta interferência inter-simbólica. Equalização perfeita é difícil de conseguir, mas há diversos métodos para mitigar grande parte dos efeitos.

O equalizador utilizado é o dos quadrados mínimos, que se baseia na minimização da equação 7 para r, onde H representa o canal e δ_k é a função Delta de Dirac.

$$\|hr - \delta_k\|_2^2 \tag{7}$$

A solução da minimização é $\frac{h^*}{\|h\|_2^2}$, multiplicada ao sinal recebido logo antes de ser demodulado.

Quanto a implementação, é possível usar o fator dos mínimos quadrados e distribuí-lo como na figura 17. Como assumimos que o canal é conhecido, este sistema realiza uma filtragem casada. A resposta ao impulso do filtro é casado com a resposta ao impulso do canal. o resultado desta operação, o vetor y, serve como uma aproximação suficiente para que se possa estimar o sinal resultante a partir do recebido. Isto será utilizado na implementação do canal *Rayleigh Fading* e *Rician Fading*.

Por ser um canal *flat*, a inserção deste canal no sinal $y = r + n$ pode ser dada como uma multiplicação do canal em r, resultando em $y = hr + n$. Este resultado também é utilizado no canal *Rician Fading*.

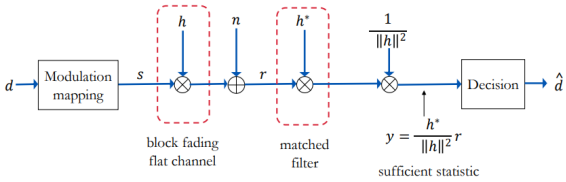

Figure 17: Modelo para simulação do canal utilizando o método dos quadrados mínimos.

O sinal GMSK passando por um canal modelado por *Rayleigh Fading* está mostrado na imagem 18. O valor de BT_b está sendo variado de acordo para mostrar como o canal se comporta dependendo de BT_b. É possível notar como o equalizador parece melhorar bastante o sinal, assemelhando-se ao modelo AWGN em formato.

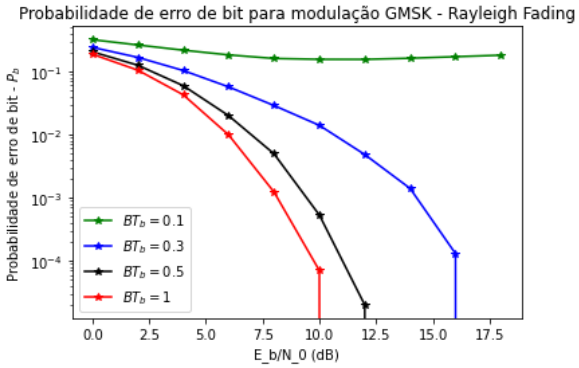

Figure 18: Modulação GMSK com Rayleigh Fading.

Para o sinal MSK, sua BER no canal *Rayleigh Fading* está na imagem 19. É possível ver a comparação

com o AWGN e o quão preciso ele se torna graças ao equalizador de quadrados mínimos.

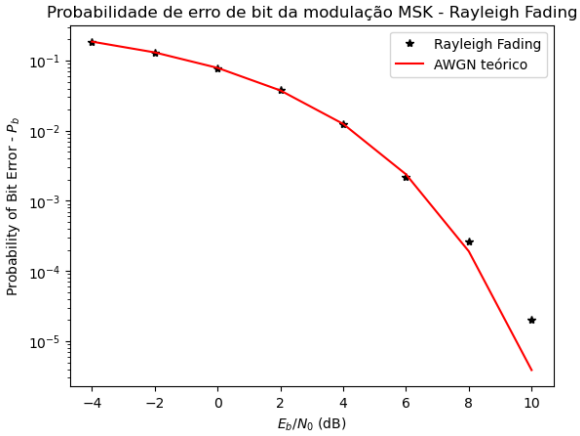

Figure 19: Modulação MSK com Rayleigh Fading.

5.3.Rician Fading

O modelo de *Rayleigh* é apropriado para a falta de um percurso dominante com linha de visada. Se houver um percurso dominante, a distribuição do envelope deixa de ser *Rayleigh* e se torna *Rician*. O processo de *fading* é então representado pela soma de uma exponencial complexa e um processo gaussiano complexo de banda estreita. Em efeitos de simulação, ele é representado de forma similar ao modelo de *Rayleigh*, mas com média não zero dado pela equação 8 e desvio padrão dado pela equação 9.

$$\mu = \sqrt{\frac{K}{2(K+1)}} \tag{8}$$

$$\sigma = \sqrt{\frac{1}{2(K+1)}} \tag{9}$$

Percebe-se que se k é 0, o modelo se reduz a um *Rayleigh fading*, que é então um caso especial do *Rician fading*. O fator K será variado nas simulações e o mesmo equalizador será usado em ambos os casos. O modelo para o MSK está na figura 20. É possível perceber que quanto maior o valor de K (dado em dB), melhor a taxa de erro de bits se torna quando a razão E_b por N_0 aumenta.

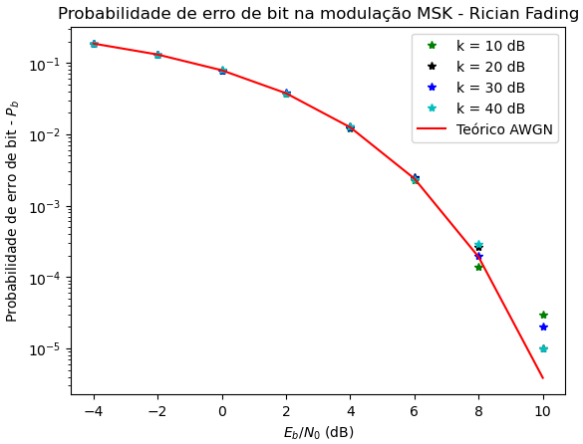

Figure 20: Modulação MSK com Rician Fading.

Para o GMSK, o modelo está na figura 21, onde se percebe algo similar ao MSK. O ganho na taxa de erros vem com um preço em potência, embora nesta modulação esse ganho represente uma porcentagem bem menor em relação a variação no K, sendo menos efetivo que na modulação MSK. Foi utilizado $BT_b = 0.3$ para esta simulação.

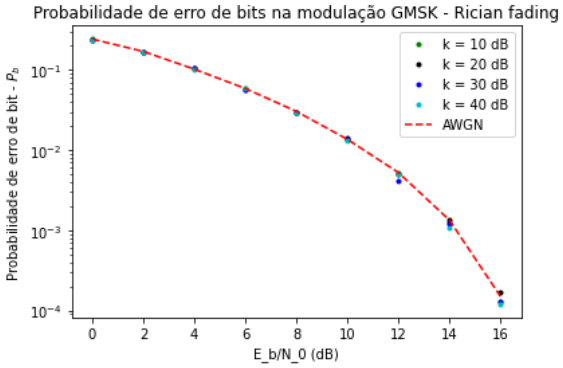

Figure 21: Modulação GMSK com Rician Fading.

5.4.canal FIR

O canal FIR é um modelo de canal que leva em conta uma característica importante dos canais na vida real: Ele não responde imediatamente a entrada. Efeitos como a interferência inter-simbólica podem afetar a saída do canal. Esses efeitos podem ser melhor estudados em sistemas LTI, onde a resposta do canal para

uma entrada depende apenas da resposta ao impulso do canal.

Um canal com resposta ao impulso de tamanho finito pode ser interpretado como um filtro FIR. Sua saída no tempo então é a convolução desta resposta ao impulso com o sinal no tempo (ou uma multiplicação na frequência). A imagem 22 exibe a arquitetura do modelo de canal considerando um filtro fir. O ruído AWGN é sempre-presente quando o canal é considerado na transmissão.

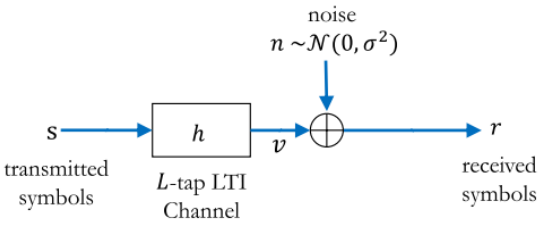

Figure 22: Modelo de canal considerando um filtro FIR.

Na implementação em python, o filtro escolhido veio da biblioteca *Scipy*. Ela possui o filtro *Wiener2* integrado, disponível com a função *scipy.signal.wiener()*. Aplicado ao sinal transmitido, o AWGN foi aplicado logo após, somando-se ao sinal. A imagem da taxa de erros na modulação MSK está na figura 23.

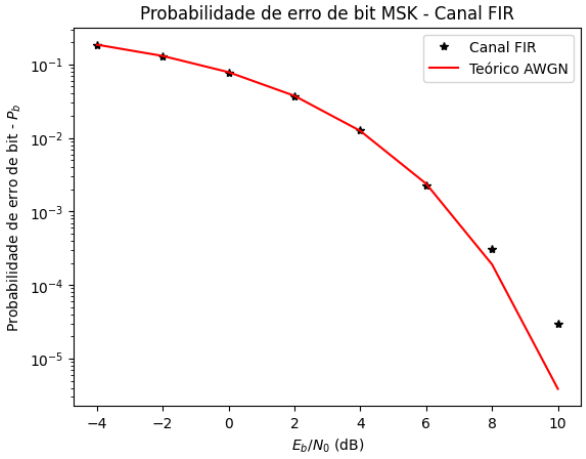

Figure 23: Taxa de erro de bits da modulação MSK com o canal FIR.

É possível perceber uma similaridade com os outros canais. Na figura 24, a taxa de erro de bits para o GMSK é mostrada. É notável que o baixo valor de BT_b acarretou em uma taxa de erros alta no experimento com o canal FIR.

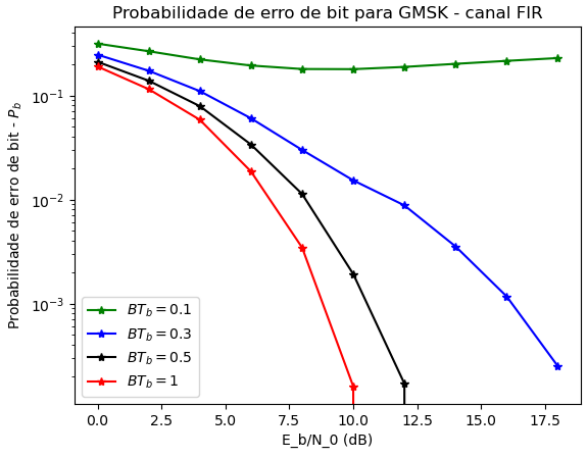

Figure 24: Taxa de erro de bits da modulação GMSK com o canal FIR.

6. Conclusão

As modulações analisadas possuem todas o seu propósito. A CPFSK é uma modulação geral, que pode abrangir uma vasta gama de requerimentos de projetos. A MSK é uma modulação bastante específica, mas que gera artifícios no design de receptores ao diferenciar entre uma transição de fase positiva e uma negativa sem ambiguidade, provando ser um atalho na concepção dos mesmos. Entretanto, é uma modulação que pode causar bastante interferência fora de sua banda para sistemas com canais adjacentes pouco espaçados.

A GMSK entra como uma manipulação do espectro da MSK para satisfazer o problema do espectro pouco compacto da mesma. Para isso, ela utiliza-se de um filtro passa-baixas gaussiano e soluciona o problema a custo de uma maior complexidade no design de receptores e transmissores. Todas possuem taxas de erro de bits similares tomando valores médios.

Cabe ao projetista comparar os diferentes tipos e escolher a melhor para sua aplicação, levando em conta os requerimentos do projeto e as dificuldades não apenas do sistema no qual está trabalhando, mas sistemas adjacentes que podem sofrer com demasiada interferência.

Em relação aos canais, o uso de um equalizador ajuda muito a estabilizar sua taxa de erros e controlar a transmissão. Os canais estudados se assemelharam ao AWGN, o que indica um bom sucesso do experimento e uma boa demodulação do sinal.

7. revisão bibliográfica

J. Proakis, Digital Communications, sér. Electrical engineering series. McGraw-Hill, 2001

M. Viswanathan, Digital modulations using Python, en. 2019

Appendix A.
Função da Modulação GMSK

```
# Função da Modulação

def gmsk_mod(a,fc,L,BT,enable_plot=False):
    """
    Function to modulate a binary stream using GMSK modulation
    Parameters:
        BT : BT product (bandwidth x bit period) for GMSK
        a : input binary data stream (0's and 1's) to modulate
        fc : RF carrier frequency in Hertz
        L : oversampling factor
        enable_plot: True = plot transmitter waveforms (default False)
    Returns:
        (s_t,s_complex) : tuple containing the following variables
            s_t : GMSK modulated signal with carrier s(t)
            s_complex : baseband GMSK signal (I+jQ)
    """
    fs = L*fc; Ts=1/fs;Tb = L*Ts; # derived waveform timing parameters
    c_t = upfirdn(h=[1]*L, x=2*a-1, up = L) #NRZ pulse train c(t)

    k=1 # truncation length for Gaussian LPF
    h_t = gaussianLPF(BT,Tb,L,k) # Gaussian LPF with BT=0.25
    b_t = np.convolve(h_t,c_t,'full') # convolve c(t) with Gaussian LPF to get b(t)
    bnorm_t = b_t/max(abs(b_t)) # normalize the output of Gaussian LPF to +/-1

    h = 0.5;
    # integrate to get phase information
    phi_t = lfilter(b = [1], a = [1,-1], x = bnorm_t*Ts) * h*np.pi/Tb

    I = np.cos(phi_t)
    Q = np.sin(phi_t) # cross-correlated baseband I/Q signals

    s_complex = I - 1j*Q # complex baseband representation
    t = Ts* np.arange(start = 0, stop = len(I)) # time base for RF carrier
    sI_t = I*np.cos(2*np.pi*fc*t); sQ_t = Q*np.sin(2*np.pi*fc*t)
    s_t = sI_t - sQ_t # s(t) - GMSK with RF carrier

    if enable_plot:
        fig, axs = plt.subplots(2, 4)
        axs[0,0].plot(np.arange(0,len(c_t))*Ts,c_t);
        axs[0,0].set_title("c(t)");axs[0,0].set_xlim(0,40*Tb)
        axs[0,1].plot(np.arange(-k*Tb,k*Tb+Ts,Ts),h_t);
        axs[0,1].set_title('$h(t): BT_b$='+str(BT))
        axs[0,2].plot(t,I,'--');axs[0,2].plot(t,sI_t,'r');
        axs[0,2].set_title('$I(t)cos(2 \pi f_c t)$');axs[0,2].set_xlim(0,10*Tb)
        axs[0,3].plot(t,Q,'--');axs[0,3].plot(t,sQ_t,'r');
        axs[0,3].set_title('$Q(t)sin(2 \pi f_c t)$');axs[0,3].set_xlim(0,10*Tb)
        axs[1,0].plot( np.arange(0,len(bnorm_t))*Ts,bnorm_t);
        axs[1,0].set_title("b(t)");axs[1,0].set_xlim(0,40*Tb)
        axs[1,1].plot(np.arange(0,len(phi_t))*Ts, phi_t);
        axs[1,1].set_title('$\phi(t)$')
        axs[1,2].plot(t,s_t);axs[1,2].set_title('s(t)');
        axs[1,2].set_xlim(0,20*Tb)
        axs[1,3].plot(I,Q);axs[1,3].set_title('Constelação')
        fig.show()
    return (s_t,s_complex)
```

Appendix B.
Modulação MSK em Python

```python
import numpy as np
def awgn(s,SNRdB,L=1):
    gamma = 10**(SNRdB/10) #SNR to linear scale

    if s.ndim==1:# if s is single dimensional vector
        P=L*sum(abs(s)**2)/len(s) #Actual power in the vector
    else: # multi-dimensional signals like MFSK
        P=L*sum(sum(abs(s)**2))/len(s) # if s is a matrix [MxN]

    N0=P/gamma # Find the noise spectral density
    if np.isrealobj(s):# check if input is real/complex object type
        n = np.sqrt(N0/2)*np.random.standard_normal(s.shape) # computed noise
    else:
        n = np.sqrt(N0/2)*(np.random.standard_normal(s.shape)+1j*np.random.standard_normal(s.shape))
    r = s + n # received signal
    return r
import matplotlib.pyplot as plt
def msk_mod(a, fc, OF, enable_plot = False):

    ak = 2*a-1 # 0->-1 1->+1
    ai = ak[0::2]; aq = ak[1::2] # dividir pares e impares
    L = 2*OF # duração do símbolo é o dobro da duração do bit

    #upsample por L
    from scipy.signal import upfirdn, lfilter
    ai = upfirdn(h=[1], x=ai, up = L)
    aq = upfirdn(h=[1], x=aq, up = L)

    aq = np.pad(aq, (L//2,0), 'constant') # delay em Tb (L/2)
    ai = np.pad(ai, (0,L//2), 'constant') # igualar o tamanho dos vetores

    #filtros passa-baixa
    Fs = OF*fc;Ts = 1/Fs;Tb = OF*Ts
    t = np.arange(0,2*Tb+Ts,Ts)
    h = np.sin(np.pi*t/(2*Tb))# filtro LPF
    sI_t = lfilter(b = h, a = [1], x = ai) # banda básica em fase
    sQ_t = lfilter(b = h, a = [1], x = aq) # banda básica em quadratura

    t=np.arange(0, Ts*len(sI_t), Ts) # portadora
    sIc_t = sI_t*np.cos(2*np.pi*fc*t) #com portadora
    sQc_t = sQ_t*np.sin(2*np.pi*fc*t) #com portadora
    s_t =  sIc_t - sQc_t# sinal em banda passante

    if enable_plot:
        fig, (ax1,ax2,ax3) = plt.subplots(3, 1)

        ax1.plot(t,sI_t);ax1.plot(t,sIc_t,'r')
        ax2.plot(t,sQ_t);ax2.plot(t,sQc_t,'r')
        ax3.plot(t,s_t,'--')
        ax1.set_ylabel('$s_I(t)$');ax2.set_ylabel('$s_Q(t)$')
        ax3.set_ylabel('s(t)')
        ax1.set_xlim([-Tb,20*Tb]);ax2.set_xlim([-Tb,20*Tb])
        ax3.set_xlim([-Tb,20*Tb])
        ax1.set_xlabel('$t$');ax2.set_xlabel('$t$')
        ax3.set_xlabel('$t$')
        fig.show()

    result = dict()
    result['s(t)']=s_t;result['sI(t)']=sI_t;result['sQ(t)']=sQ_t;result['t']=t
    return result
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erfc

N = 100000 # Números de símbolos a serem transmitidos
EbN0dB = np.arange(start=-4,stop = 11,step = 2) # range de Eb/N0 pra simulação
fc = 800
OF = 32 # fator de oversampling

BER = np.zeros(len(EbN0dB)) # valores de BER pra cada EB/N0

a = np.random.randint(2, size=N) # símbolos 0s e 1s uniformes
result = msk_mod(a,fc,OF,enable_plot=True)
s = result['s(t)']
```

Appendix C.
Taxa de erros de bits MSK em Octave

```
clear
N = 5*10^5; % número de bits / simbolos

fsHz = 1; % período de amostragem
T    = 4; % duração do símbolo

Eb_N0_dB = [0:10]; % multiplos valores Eb/N0
ct = cos(pi*[-T:N*T-1]/(2*T));
st = sin(pi*[-T:N*T-1]/(2*T));

for ii = 1:length(Eb_N0_dB)

    % transmissor MSK
    ipBit = rand(1,N)>0.5; % 0s e 1s com igual probabilidade
    ipMod =  2*ipBit - 1; % 0->-1 e 1->0

    ai = kron(ipMod(1:2:end),ones(1,2*T));  % bit par
    aq = kron(ipMod(2:2:end),ones(1,2*T));  % bit impar

    ai = [ai zeros(1,T)  ]; % igualar as dimensões das matrizes
    aq = [zeros(1,T) aq ];  % Delay na quadratura

    % forma de onda a ser transmitida
    xt = 1/sqrt(T)*[ai.*ct + j*aq.*st];

    % AWGN
    nt = 1/sqrt(2)*[randn(1,N*T+T) + j*randn(1,N*T+T)]; % variancia 0db

    % ruído
    yt = xt + 10^(-Eb_N0_dB(ii)/20)*nt; % AWGN

    %% receptor
    % multiplicar pelas senóides
    xE = conv(real(yt).*ct,ones(1,2*T));
    xO = conv(imag(yt).*st,ones(1,2*T));

    bHat = zeros(1,N);
    bHat(1:2:end) = xE(2*T+1:2*T:end-2*T) > 0 ; % bits pares
    bHat(2:2:end) = xO(3*T+1:2*T:end-T) > 0 ;  % bits ímpares

    % contagem de erros
    nErr(ii) = size(find([ipBit - bHat]),2);

end

simBer = nErr/N; % ber simulada
theoryBer = 0.5*erfc(sqrt(10.^(Eb_N0_dB/10))); % ber teorica

% plot
close all
figure
semilogy(Eb_N0_dB,theoryBer,'bs-','LineWidth',2);
hold on
semilogy(Eb_N0_dB,simBer,'mx-','LineWidth',2);
axis([0 10 10^-5 0.5])
grid on
legend('theory - bpsk', 'simulation - msk',"fontsize", 20);
xlabel('Eb/No, dB',"fontsize",24);
ylabel('Bit Error Rate',"fontsize", 24);
title('Bit error probability curve for MSK modulation',"fontsize", 20);
```

Appendix D.
Função da demodulação GMSK

```
# Função da Demodulação

def gmsk_demod(r_complex,L):
  """
  Function to demodulate a baseband GMSK signal
  Parameters:
  r_complex : received signal at receiver front end (complex form - I+jQ)
  L : oversampling factor
  Returns:
  a_hat : detected binary stream
  """
  I = np.real(r_complex); Q = -np.imag(r_complex); # I,Q streams
  z1 = Q * np.hstack((np.zeros(L), I[0:len(I)-L]))
  z2 = I * np.hstack((np.zeros(L), Q[0:len(I)-L]))
  z = z1 - z2
  a_hat = (z[2*L-1:-L:L] > 0).astype(int) # sampling and hard decision
  #sampling indices depend on the truncation length (k) of Gaussian LPF defined in the modulator
  return a_hat
```

Appendix E.
Função da probabilidade de erro de bits da GMSK

```
N=100000
EbN0dB = np.arange(start=0,stop = 19, step = 2) # Eb/N0 range in dB for simulation
BTs = [0.1, 0.3 ,0.5, 1] # Gaussian LPF's BT products
fc = 800 # Carrier frequency in Hz (must be < fs/2 and > fg)
L = 16 # oversampling factor

fig, axs = plt.subplots(nrows=1,ncols = 1)
lineColors = ['g','b','k','r']

for i,BT in enumerate(BTs):
  a = np.random.randint(2, size=N) # uniform random symbols from 0's and 1's
  (s_t,s_complex) = gmsk_mod(a,fc,L,BT) # GMSK modulation
  BER = np.zeros(len(EbN0dB)) # For BER values for each Eb/N0

  for j,EbN0 in enumerate(EbN0dB):
    r_complex = awgn(s_complex,EbN0) # refer Chapter section 4.1
    a_hat = gmsk_demod(r_complex,L) # Baseband GMSK demodulation
    BER[j] = np.sum(a!=a_hat)/N # Bit Error Rate Computation

axs.semilogy(EbN0dB,BER,lineColors[i]+'-x',label='$BT_b=$'+str(BT))
axs.set_title('[Probabilidade de Erro de Bit para a modulação GMSK]')
axs.set_xlabel('E_b/N_0 (dB)');axs.set_ylabel('Probabilidade de Erro de Bit - $P_b$')
axs.legend();fig.show()
```

Appendix F.
Função da densidade de espectro da GMSK

```
def plotWelchPSD(x,fs,fc,ax = None,color='b', label=None):

    from scipy.signal import hanning, welch
    from numpy import log10
    nx = max(x.shape)
    na = 16 # averaging factor to plot averaged welch spectrum
    w = hanning(nx//na) #// is for integer floor division
    # Welch PSD estimate with Hanning window and no overlap
    f, Pxx = welch(x,fs,window = w,noverlap=0)
    indices = (f>=fc) & (f<4*fc)   # To plot PSD from Fc to 4*Fc
    Pxx = Pxx[indices]/Pxx[indices][0] # normalized psd w.r.t Fc
    ax.plot(f[indices]-fc,10*log10(Pxx),color,label=label) #Plot in the given axes

N = 100000 # Number of symbols to transmit
fc = 800 # carrier frequency in Hertz
L = 16 # oversampling factor,use L= Fs/Fc, where Fs >> 2xFc
fs = L*fc
a = np.random.randint(2, size=N) # uniform random symbols from 0's and 1's
#'_':unused output variable
(s1 , _ ) = gmsk_mod(a,fc,L,BT=0.3, enable_plot=False) # BT_b=0.3
(s2 , _ ) = gmsk_mod(a,fc,L,BT=0.5) # BT_b=0.5
(s3 , _ ) = gmsk_mod(a,fc,L,BT=0.7) # BT_b=0.7
(s4 , _ ) = gmsk_mod(a,fc,L,BT=10000) # BT_b=very value value (MSK)
# Compute and plot PSDs for each of the modulated versions
fig, ax = plt.subplots(1, 1)
plotWelchPSD(s1,fs,fc, ax = ax , color = 'r', label = '$BT_b=0.3$')
plotWelchPSD(s2,fs,fc, ax = ax , color = 'b', label = '$BT_b=0.5$')
plotWelchPSD(s3,fs,fc, ax = ax , color = 'm', label = '$BT_b=0.7$')
plotWelchPSD(s4,fs,fc, ax = ax , color = 'k', label = '$BT_b=\infty$')
ax.set_xlabel('$f-f_c$'); ax.set_ylabel('PSD (dB/Hz)')
ax.legend(); fig.show()
```

Appendix G.
Função da modulação do CPFSK

```
def cpfsk_modulate(bit_str, bit_freq_map, baud, sample_rate):
    seconds_per_bit = 1 / baud
    samples_per_bit = int(sample_rate * seconds_per_bit)
    t = np.linspace(0, seconds_per_bit, samples_per_bit)

    # maps from bit sequence (like "10") to the modulated wave representing that "symbol"
    symbol_map = {bit_seq: np.sin(freq * 2 * np.pi * t) for bit_seq, freq in bit_freq_map.items()}

    signal = np.array([])
    bits_per_symbol = len(list(bit_freq_map.keys())[0])  # Assume all keys are the same length
    for symbol in [bit_str[i:i+bits_per_symbol] for i in range(0, len(bit_str), bits_per_symbol)]:
        symbol_wave = symbol_map[symbol]
        signal = np.append(signal, symbol_wave)
    return signal
```

Appendix H.
Função da demodulação do CPFSK

```
def cpfsk_demodulate(raw_signal, bit_freq_map, baud, sample_rate):
    seconds_per_bit = 1 / baud
    samples_per_bit = int(sample_rate * seconds_per_bit)
    t = np.linspace(0, seconds_per_bit, samples_per_bit)

    # maps from bit sequence (like "10") to the modulated wave representing that "symbol"
    wave_to_symbol_map = {bit_seq: np.sin(freq * 2 * np.pi * t) for bit_seq, freq in bit_freq_map.items()}

    bit_str = ""
    for index in range(0, len(raw_signal), samples_per_bit):
        best_symbol = ""
        highest_dot_abs = 0
        for symbol, symbol_wave in wave_to_symbol_map.items():
            raw_window = raw_signal[index:index+samples_per_bit]
            dot_abs = np.abs(np.dot(symbol_wave[0:len(raw_window)], raw_window))
            if dot_abs > highest_dot_abs:
                best_symbol = symbol
                highest_dot_abs = dot_abs
        bit_str += best_symbol
    return bit_str
```

Appendix I.
Canal Rayleigh e Rician Fading

```python
def rayleighFading(N):
    """
    Generate Rayleigh flat-fading channel samples
    Parameters:
        N : number of samples to generate
    Returns:
        abs_h : Rayleigh flat fading samples
    """
    # 1 tap complex gaussian filter
    h = 1/sqrt(2)*(standard_normal(N)+1j*standard_normal(N))
    return h
def ricianFading(K_dB,N):
    """
    Generate Rician flat-fading channel samples
    Parameters:
        K_dB: Rician K factor in dB scale
        N : number of samples to generate
    Returns:
        abs_h : Rician flat fading samples
    """
    K = 10**(K_dB/10) # K factor in linear scale
    mu = sqrt(K/(2*(K+1))) # mean
    sigma = sqrt(1/(2*(K+1))) # sigma
    h = (sigma*standard_normal(N)+mu)+1j*(sigma*standard_normal(N)+mu)
    return abs(h)
```

Appendix J.
Canal AWGN

```python
def awgn(s,SNRdB,L=1):
    """
    AWGN channel

    Add AWGN noise to input signal. The function adds AWGN noise vector to signal
    's' to generate a resulting signal vector 'r' of specified SNR in dB. It also
    returns the noise vector 'n' that is added to the signal 's' and the power
    spectral density N0 of noise added

    Parameters:
        s : input/transmitted signal vector
        SNRdB : desired signal to noise ratio (expressed in dB)
            for the received signal
        L : oversampling factor (applicable for waveform simulation)
            default L = 1.
    Returns:
        r : received signal vector (r=s+n)
    """
    gamma = 10**(SNRdB/10) #SNR to linear scale

    if s.ndim==1:# if s is single dimensional vector
        P=L*sum(abs(s)**2)/len(s) #Actual power in the vector
    else: # multi-dimensional signals like MFSK
        P=L*sum(sum(abs(s)**2))/len(s) # if s is a matrix [MxN]

    N0=P/gamma # Find the noise spectral density
    if isrealobj(s):# check if input is real/complex object type
        n = sqrt(N0/2)*standard_normal(s.shape) # computed noise
    else:
        n = sqrt(N0/2)*(standard_normal(s.shape)+1j*standard_normal(s.shape))
    r = s + n # received signal
    return r
```