Sistema BPSK e Desmodulador de QPSK com Sincronização Simbólica

Laboratório Nº2

LEEC - Telecomunicações 23-24

Carlos Reis – 103166

Filipe Piçarra – 103592

Nuno Jorge – 102943

Índice

Introdução ao Trabalho

No dia-a-dia, confrontamo-nos frequentemente com a necessidade de transmitir informação digital através de canais passa-banda, como é o caso dos utilizados na comunicação sem fios e por satélite. Para isso, os terminais recorrem a técnicas específicas de modulação digital – por exemplo, BPSK (Binary Phase-Shift Keying) e QPSK (Quadriphase-Shift Keying).

Introdução Teórica

Tal como o nome indica, tanto BPSK como QPSK são métodos de modulação de fase, o que quer dizer que a fase da onda portadora é função do sinal modulante.

Como a sua amplitude é constante, estes sinais não podem ser desmodulados por detetores de envolvente. Em vez disso, compara-se o valor obtido na receção com os pontos da constelação que caracteriza o sinal recebido; recorre-se ao critério de máxima verosimilhança para escolher como ponto da constelação que determina o símbolo interpretado na receção aquele para o qual a distância ao valor lido é a menor.

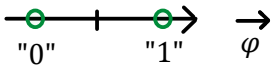
BPSK

Um sinal transmitido por BPSK é definido por

$$s(t) = \begin{cases} \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t)\,,\ para\ o\ símbolo\ "1" \\ -\sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t)\,,\ para\ o\ símbolo\ "0" \end{cases},$$

em que $0 \leq t \leq T_b$, T_b representa a duração do bit e E_b a energia, por bit, do sinal transmitido.

$\varphi = \sqrt{\frac{2}{T_b}}\cos(2\pi f_c t)$ define a base ortonormada que gera o espaço ao qual pertencem os sinais BPSK; assim, a constelação que caracteriza sinais deste tipo é a seguinte:



QPSK

Podemos reduzir a largura de banda do sinal transmitido para metade se, em vez de BPSK, recorremos à modulação por QPSK, na qual cada símbolo, constituído por dois bits - *dibit*, tem a duração $T = 2T_b$.

Um sinal transmitido por QPSK é definido por

$$s_i(t) = \begin{cases} \sqrt{\frac{2E}{T}}\cos\left[2\pi f_c t + (2i-1)\frac{\pi}{4}\right],\ 0 \leq t \leq T \\ 0,\ caso\ contrário \end{cases},$$

em que i = 1, 2, 3, 4 e E representa a energia, por símbolo, do sinal transmitido.

Ao contrário do que se verifica com BPSK, o espaço ao qual pertencem os sinais QPSK não é gerado por uma base de dimensão unitária, mas sim por uma base de dimensão 2, caracterizada por $\varphi_1 = \sqrt{\frac{2}{T}}\cos(2\pi f_c t)$ e $\varphi_2 = \sqrt{\frac{2}{T}}\sin(2\pi f_c t)$.

Assim sendo, um modulador pode ser o seguinte:

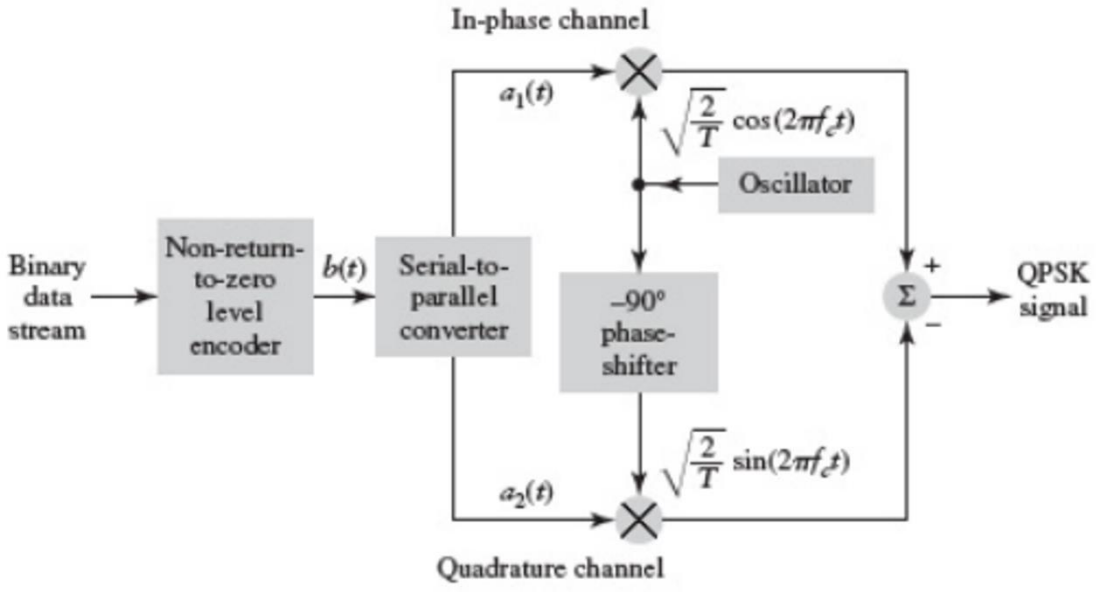


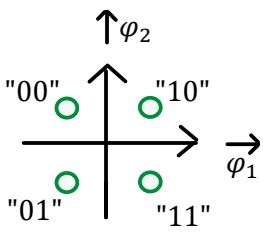
Figura 1 - Modulador QPSK

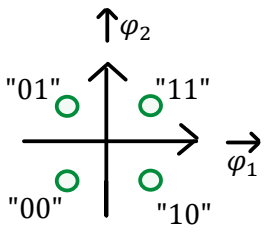
Repara que, com este modulador podemos receber um sinal digital NRZ Polar, onde a cada *dibit*, φ_1 varia com o bit mais significativo e φ_2 com o menos significativo. Assim o mapeamento de bits é o seguinte:

Index i	*Phase of QPSK signal (radians)*	*Amplitudes of constituent binary waves*		*Input dibit* $0 \le t \le T$
		Binary wave 1 $a_1(t)$	*Binary wave 2* $a_2(t)$	
1	$\pi/4$	$+\sqrt{E/2}$	$-\sqrt{E/2}$	10
2	$3\pi/4$	$-\sqrt{E/2}$	$-\sqrt{E/2}$	00
3	$5\pi/4$	$-\sqrt{E/2}$	$+\sqrt{E/2}$	01
4	$7\pi/4$	$+\sqrt{E/2}$	$+\sqrt{E/2}$	11

Figura 2 - Mapeamento de dibits.

Traduzindo para uma representação geométrica:





Constelação usada na prática

Esta será então a constelação escolhida para transmitir o sinal, pois se nos recordarmos que a Data Stream será código NRZ Polar, temos que o 0 vale -1 e 1 vale 1, como na figura abaixo. Deste modo, cada bit provocará uma variação direta do sinal de a_1 e a_2.

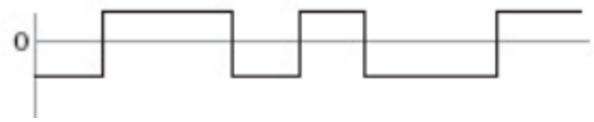


Figura 3 - Exemplo de um sinal NRZ Polar, com a sequência [0 1 1 0 1 0 0 1]

GNU Radio

O que segue é a explicação do projeto realizado no GNU Radio.

BPSK

Modulador

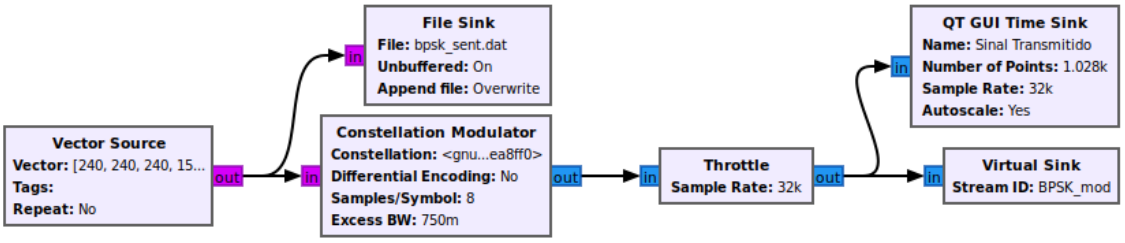


Figura 4 - Modulador BPSK

A figura acima mostra-nos o diagrama utilizado para modular o sinal digital gerado pelo **Vector Source** para a forma BPSK.

Começamos pelo **Vector Source**, que irá gerar a sequência de bytes correspondentes ao vetor: [240,240,240,15,15,15,240,240,240,]+ [10,29,43,10, 31, 66, 10, 35, 92,]+ [15,15,15,240,240,240,15,15,15,0,0,0,0,0,0,0,0].

Esta stream de bytes será gravada no ficheiro "*bpsk_sent.dat*" pelo bloco **File Sink** para futura analise, e modelada no bloco **Constellation Modulator**, que opera subordinado a um outro bloco – **Constellation Object** – definindo este o número de símbolos utilizados na comunicação e as coordenadas desses símbolos no espaço de sinal, que será explicado em mais detalhe abaixo. Este modulador irá gerar à saída sinais complexos que correspondem ao sinal modulado em banda-base (com espectro de *root-raised cosine*).

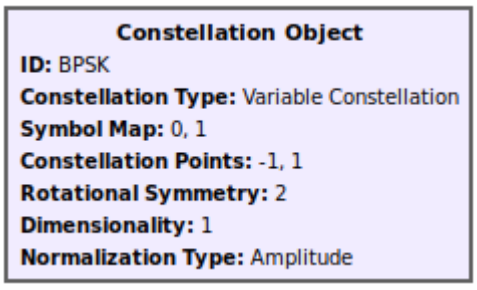


Figura 5 - Constellation Object

O bloco **Constellation Object** é responsável, como mencionado acima, por definir a constelação que irá reger a modulação. Para este caso iremos trabalhar com a constelação definida pelo simples espaço de sinais de: -1 e 1; mapeando assim os bits a zero para $s_0 = -1$ e os bits a 1 para $s_1 = 1$.

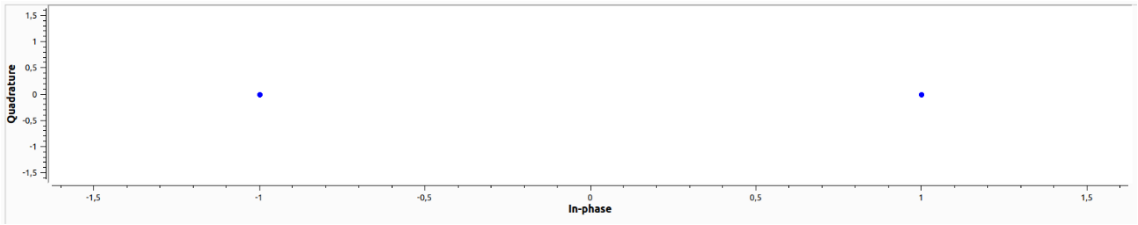


Figura 6 - Constelação Obtida na transmissão do código pretendido, sem ruído.

Observe-se o exemplo do vetor de bytes, correspondente a 255 0 0 (0b11111111 0b0 0b0), modulado:

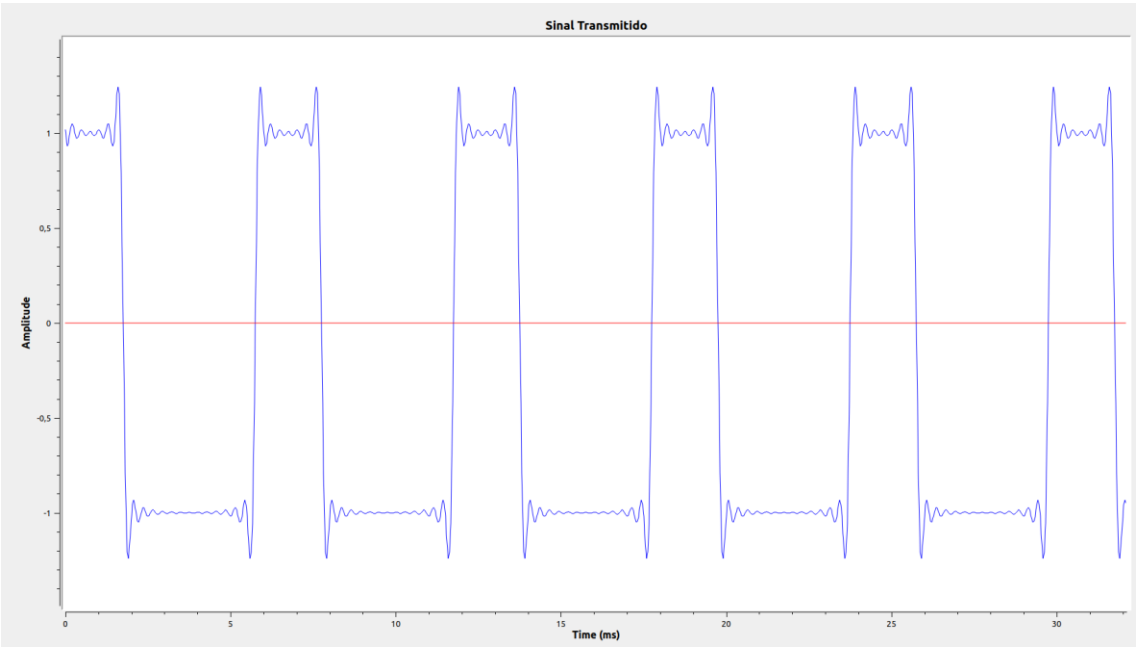


Figura 7 - Modulação BPSK em banda-base da sequência de valores: 255 0 0

Nesta imagem conseguimos diferenciar facilmente as zonas correspondentes à sequência de 1's (255) mapeadas pelo *plateau* a 1 e as zonas da sequência de 0's, por definição serão as contrárias.

Este sinal é depois despejado no bloco **Virtual Sink**, para ser posteriormente usado na desmodulação.

Desmodulador

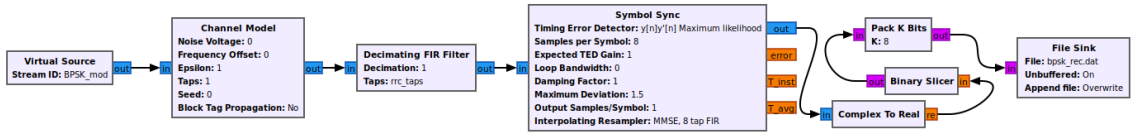


Figura 8 - Desmodulador BPSK

O sinal já modulado segue agora para o bloco **Channel Model**, que será utilizado para simular o ruído de canal, definido pelo parâmetro **noise voltage**, este módulo é inserido no início da cadeia recetora por se referir ao ruído de canal à entrada do recetor.

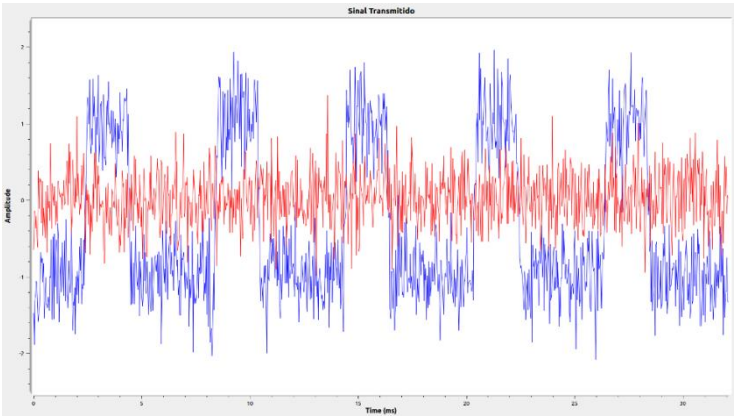


Figura 10 - Sinal digital (255 0 0) modulado, à saída do Channel Model, com o parâmetro de Noise Voltage a 0.5 V, entenda-se a vermelho a componente imaginária da variável, idealmente a zeros.

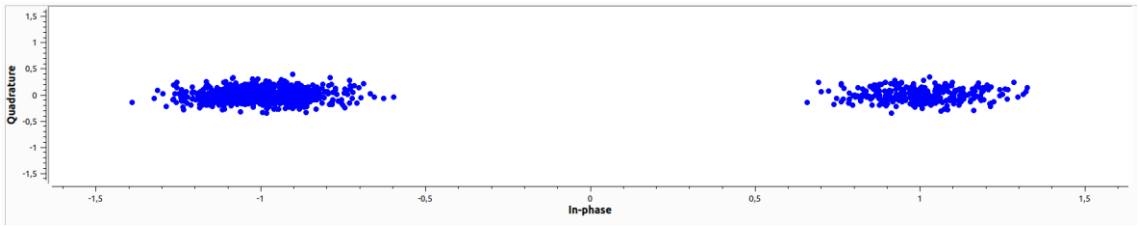


Figura 9 - Constelação obtida quando existe ruído de 0.5 V no canal

Aplica-se de seguida o bloco **Decimating FIR Filter**. Os parâmetros deste filtro são **Decimation=1** e **Taps=firdes.root raised cosine(1.0,samp rate, samp rate/sps,rolloff,11*sps)**. A função deste bloco é maximizar a razão entre potência instantânea dos impulsos e a potência instantânea do ruído facilitando a tarefa de decidir que símbolo foi enviado - diminuindo a probabilidade de erro.

O valor atribuído **Decimation=1** evita que a frequência de amostragem do sinal seja alterada. O valor atribuído a **Taps** procura recriar um filtro adaptado, este, denominado FIR (finite impulse response), aplica uma operação de convolução discreta do sinal com uma coleção de valores selecionados com Taps, assim sendo, selecionamos para Taps uma coleção de amostras do espetro de RRC(root-raised cossine) que é o espetro dos impulsos enviados.

O ganho do filtro é 1, "**samp rate,samp rate/sps,rolloff**" identifica as dimensões do espetro desejado, principalmente o rolloff que é responsável pela curvatura do espetro (rolloff=0 corresponde a um espetro retangular).

11*sps é o número de pontos usados na resposta impulsional do filtro. 11 vezes o ritmo de transmissão é um valor standard usado para estes filtros. Aumentar este valor aumenta a qualidade do filtro, mas aumenta também a complexidade computacional e assim o tempo de processamento.

À saída temos um sinal com muita mais potência na componente da mensagem do que na componente do ruído como podemos observar de seguida.

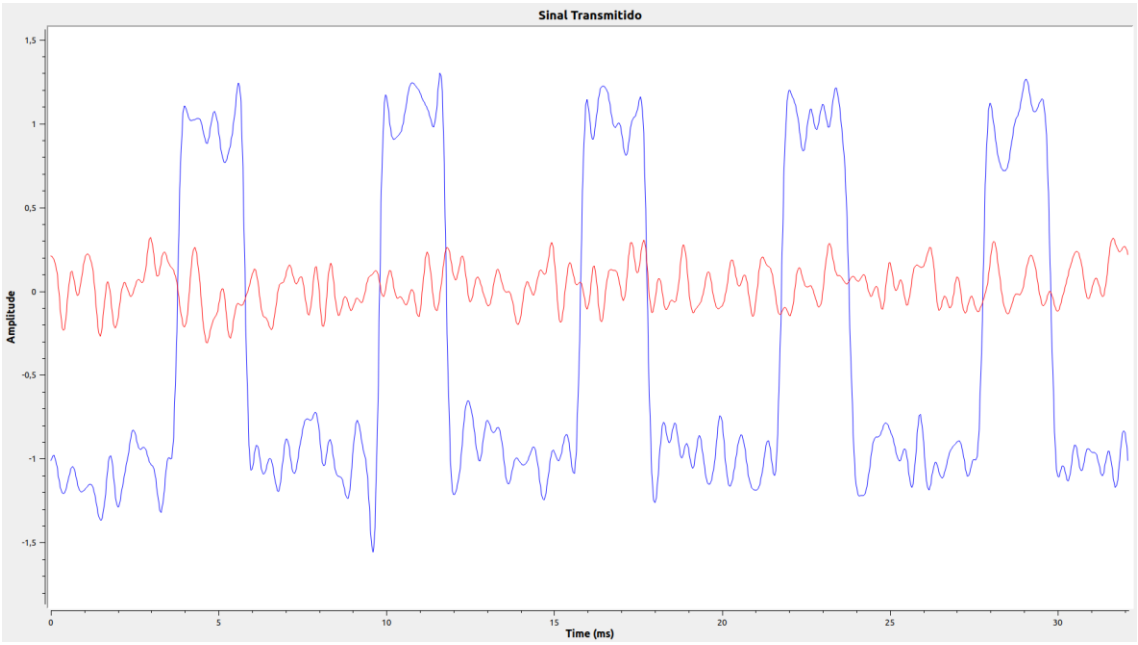


Figura 11 - Sinal Modulado após o Filtro, onde conseguimos claramente voltar a ver o sinal (a azul), mesmo depois de passar pelo canal com Noise Voltage a 0.5 V

De seguida aplicamos o bloco **Symbol Sinc.** A principal função deste bloco é corrigir desfasamentos entre o ritmo dos símbolos e o ritmo de amostragem. Este bloco usa uma malha de realimentação para se sincronizar com o sinal que está a ser recebido e amostrar o sinal nos momentos ideais para a decisão do símbolo a ser feita de seguida.

Os parâmetros escolhidos são:

- -**Timing Error Detector (TED)** - expressão a utilizar para a estimativa da distância entre o instante escolhido para amostrar e o instante ótimo;
- -**Samples per Symbol** – determina o ritmo de transmissão dos símbolos, a amostragem é feita perto desta frequência mais correções por causa do ruído;
- -**Expected TED Gain** - fundamentalmente é o ganho da malha de realimentação, deixamos a 1 e afinamos a largura de banda;
- **Loop Bandwidth (LB)** - usada para controlar a velocidade de resposta da realimentação, intuitivamente uma maior LB corresponde a uma resposta mais rápida e menor LB corresponde a uma resposta mais lenta. Quanto maior a LB mais rápidas são as variações a que a malha consegue reagir. Este valor deve ser ajustado de modo a que o TED reconheça as variações dos símbolos e ignore ao máximo o ruído;
- -**Damping Factor** - fator de amortecimento conhecido da teoria de realimentação, colocado a 1 para convergência mais rápida, amortecimento crítico;
- -**Maximum Deviation** -tempo normalizado máximo para o desvio dos períodos de amostragem originais;
- -**OutputSamples/Symbol** -colocamos a 1 porque queremos o mesmo número de bits na saída e na entrada;

Como a constelação mapeia apenas os bits 1 e 0 no eixo real, utilizamos o bloco **Complex to Real** para recuperar esta componente, onde no bloco **Pack K Bits**, serão agrupados conjuntos de 8 bites – bytes, para serem escritos no **File Sink**, com a saída denominada de "bpsk_rec.dat".

QPSK

Modulador

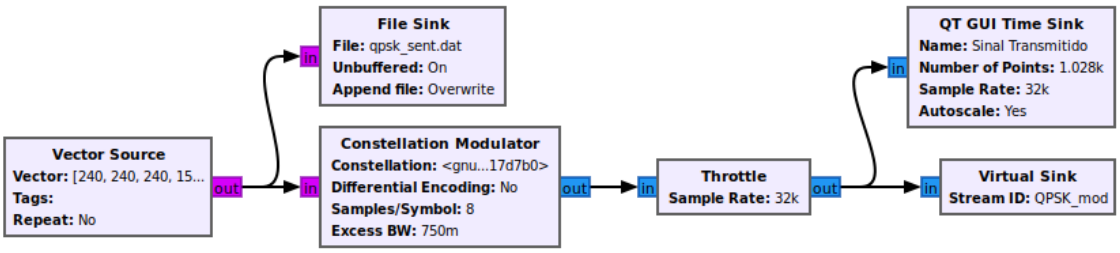


Figura 12 - Fluxograma do modulador QPSK

A arquitetura de blocos do modulador em QPSK é em tudo semelhante ao modulador de BPSK, com apenas a variação no **Constellation Object**, que por sua vez altera a natureza da modulação do sinal no **Constellation Modulator**. Os bytes gerados serão guardados em "qpsk_sent.dat".

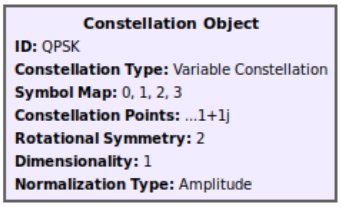


Figura 13 - Constellation Object QPSK

Nesta nova constelação vamos definir o mapeamento de *dibits* como já discutido na introdução teórica – **Figura 2** - Mapeamento de dibits. Assim sendo o GNURadio vai codificar cada dois bits numa componente real e imaginária, a real correspondendo ao bit mais significativo e a imaginária ao restante. Se um bit for 1, a sua respetiva componente assume o valor de 1, se estiver a 0, assume -1.

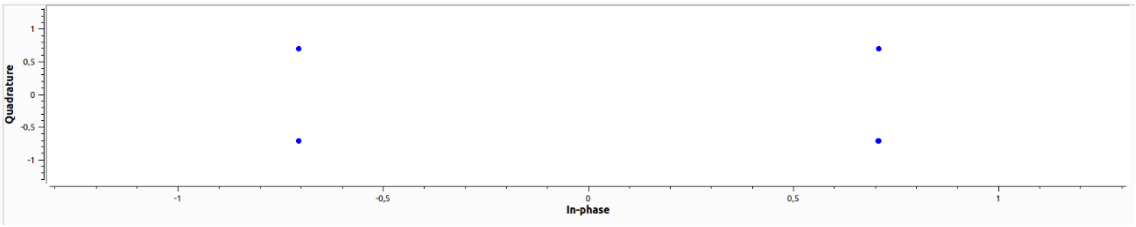


Figura 14 - Constelação Obtida na transmissão do código pretendido, sem ruído.

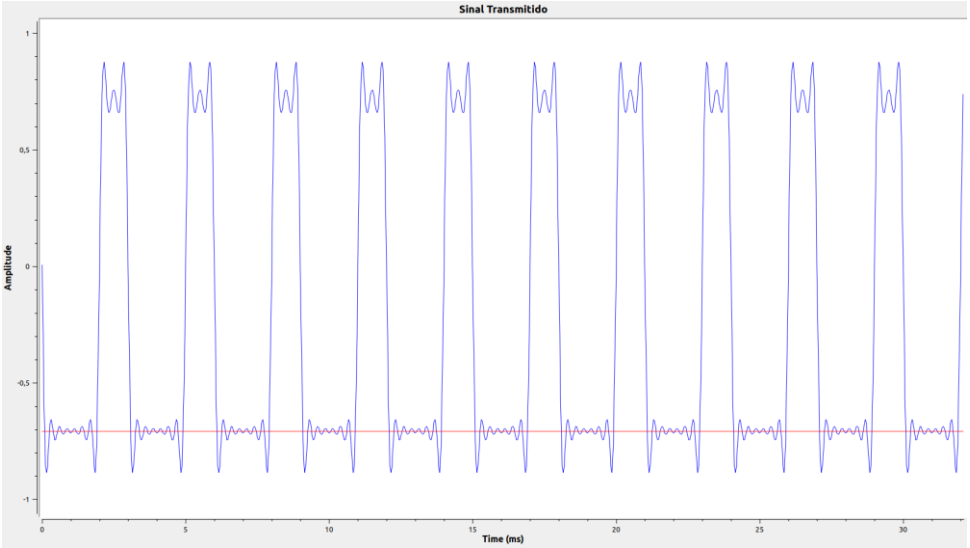


Figura 15 - Exemplo sinal transmitido 170 0 0 em modulação QPSK, a azul a componente real e a vermelho a imaginária.

Relembrando que o valor 170 corresponde a 10101010, podemos observar que a componente imaginária terá sempre o valor -1, já a componente real irá permanecer a 1 (nos intervalos que se transmite 170, e não nos de transmissão de 0).

Desmodulador

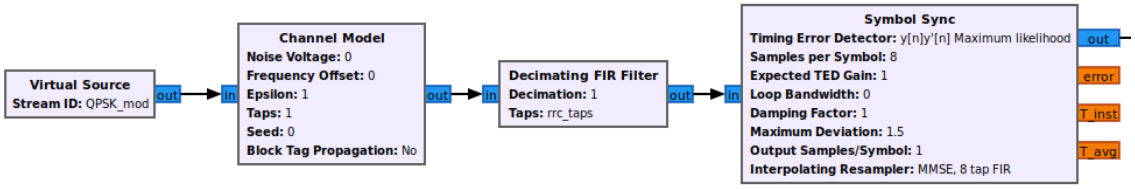


Figura 16 - Desmodulador QPSK (entrada de canal e sincronização de símbolo)

Do lado do desmodulador temos, outra vez, a mesma arquitetura usada no BPSK e explicada anteriormente.

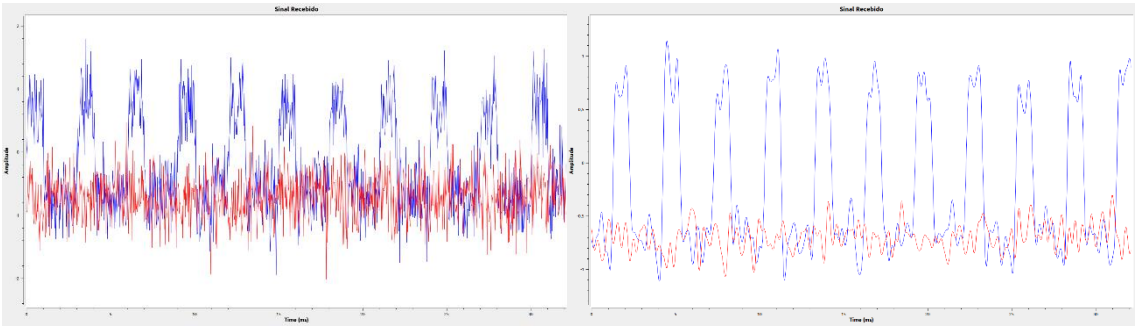


Figura 17 - Sinal recebido pelo desmodulador (170 0 0), antes e depois de passar pelo filtro, com a noise voltage da modelação de canal a 0.5 V.

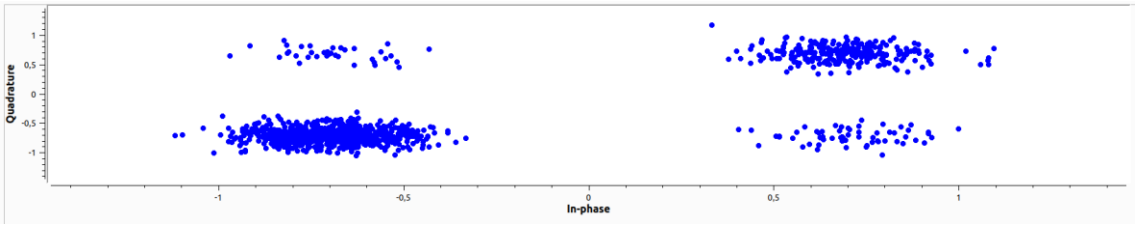


Figura 18 – Constelação obtida quando existe ruído de 0.5 V no canal

Como já havia sido explicado, ao utilizar o filtro à entrada do canal, estamos a aumentar significativamente a relação sinal ruído do sinal, voltando este a ser percetível.

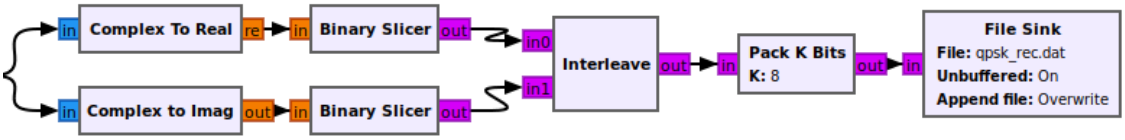


Figura 19 - Desmodulador QPSK (recuperação de bits e escrita de output).

Mais uma vez a lógica de recuperação de bits é semelhante à do BPSK, neste caso temos a exceção de cada símbolo ser um *dibit*, logo cada valor à saída do **Symbol Sync** tem dois bits codificados nele. Por isso extraímos a informação correspondente a estes bits com os blocos **Complex to Real** e **Complex to Imag**, aplicando o **Binary Slicer** para obter os bits correspondentes. De seguida, o **Interleave** intercala os bits por ordem de input. Esta byte stream é depois guardada no ficheiro "qpsk_rec.dat".

Erros de Bit na Transmissão

Para estudar este tópico, escrevemos umas alterações ao ficheiro de execução do nosso sistema. Deste modo conseguimos variar tanto o **noise voltage**, para simular vários ruídos de canal diferentes, e a **loop bandwidth**.

O **noise voltage** será variado de 0 a 4 em intervalos de 0.5 V. Intervalos maiores resultam em situações excessivas de erros de bits ao ponto de serem irrelevantes.

Variámos o valor de **Loop Bandwidth** entre 0 e 0.5 para procurar o valor ótimo. Este valor controla a velocidade de resposta da malha. Se a velocidade for muito elevada irá concentrar-se no ruído e a informação dos símbolos será pouco relevante para o erro calculado em TED (probabilidade de erro de símbolo elevada). Como os símbolos de facto variam, a **LB** não deve ser 0, mas deve ser reduzida para ignorar o ruído.

Este código irá então fazer um *run* do programa, cada vez com valores diferentes destes parâmetros, enviando a sequência de valores definida pelo enunciado e calculando os bits de diferentes no final de cada execução, estes valores são então escritos em ficheiros .csv para serem posteriormente processados.

Os ficheiros de código seguem em anexo com este relatório, mas será exemplificado o loop do BPSK.

```
def main(top_block_cls=bpsk, options=None):
    noise = np.arange(0, 4.1, 0.5)
    loopB = [round(num, 2) for num in np.arange(0, 0.501, 0.01)]

    results = [["Noise Voltage"]]
    results[0].extend(loopB)

    for n in noise:
        dif_list = [n]
        for lb in loopB:
            tb = top_block_cls(n, lb)

            def sig_handler(sig=None, frame=None):
                tb.stop()
                tb.wait()

                sys.exit(0)

            signal.signal(signal.SIGINT, sig_handler)
            signal.signal(signal.SIGTERM, sig_handler)

            tb.start()

            tb.wait()
            dif_list.append(Ver.runBPSK())
            print("Run over - Noise: {} Loop Bandwidth: {}".format(tb.noise, tb.loopB))
        results.append(dif_list)

    with open("results_bpsk.csv", "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerows(results)
```

Figura 20 - Código do programa, com a possibilidade de variar o noise voltage e a loop bandwidth

Neste programa iremos percorrer o vetor *noise*, com todos os valores definidos para o ruído no canal. Por cada um destes valores iremos iterar pelo vetor *loopB* com todos os valores da loop bandwidth. Então corremos o código de "execução normal" do GNU Radio com estes novos parâmetros, este irá gerar os dois ficheiros de saída que serão analisados pela função *Ver.runBPSK* para procurar bits diferentes entre o sinal recebido e enviado.

Deste modo, testamos todas as combinações possíveis para estas duas variáveis para encontrar o valor da LB que garanta o menor número de erros para cada nível de ruído.

BPSK

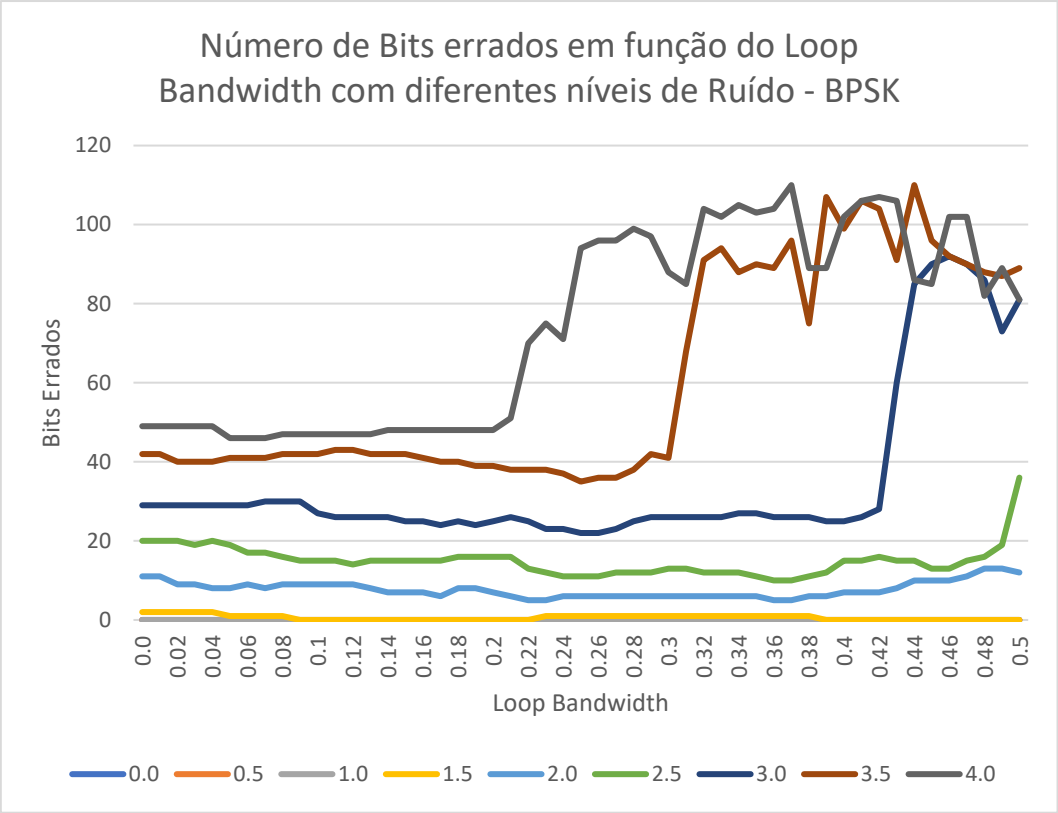


Figura 21- Gráfico de nº de bits diferentes BPSK

Com base nos dados extraídos desta análise podemos preencher a tabela com os valores da LB que minimizam o número de bits errados, relembrando que os mínimos selecionados são os de menor valor, pois idealmente a Loop Bandwidth deve ser a mais reduzida.

Noise [V]	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
L. B.	Todos	Todos	Todos	0.09	0.22	0.36	0.25	0.25	0.05
Err [%]	0	0	0	0	2.32	4.63	10.18	16.20	21.30

Podemos agora realizar uma análise mais em rigor, repetindo as medições para intervalos centrados nestes mínimos, variando agora a L.B. de 0.001 em 0.001.

Noise [V]	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
L. B.	Todos	Todos	Todos	0.086	0.221	0.360	0.244	0.251	0.043
Err [%]	0	0	0	0	1.85	4.63	10.18	14.35	19.00

QPSK

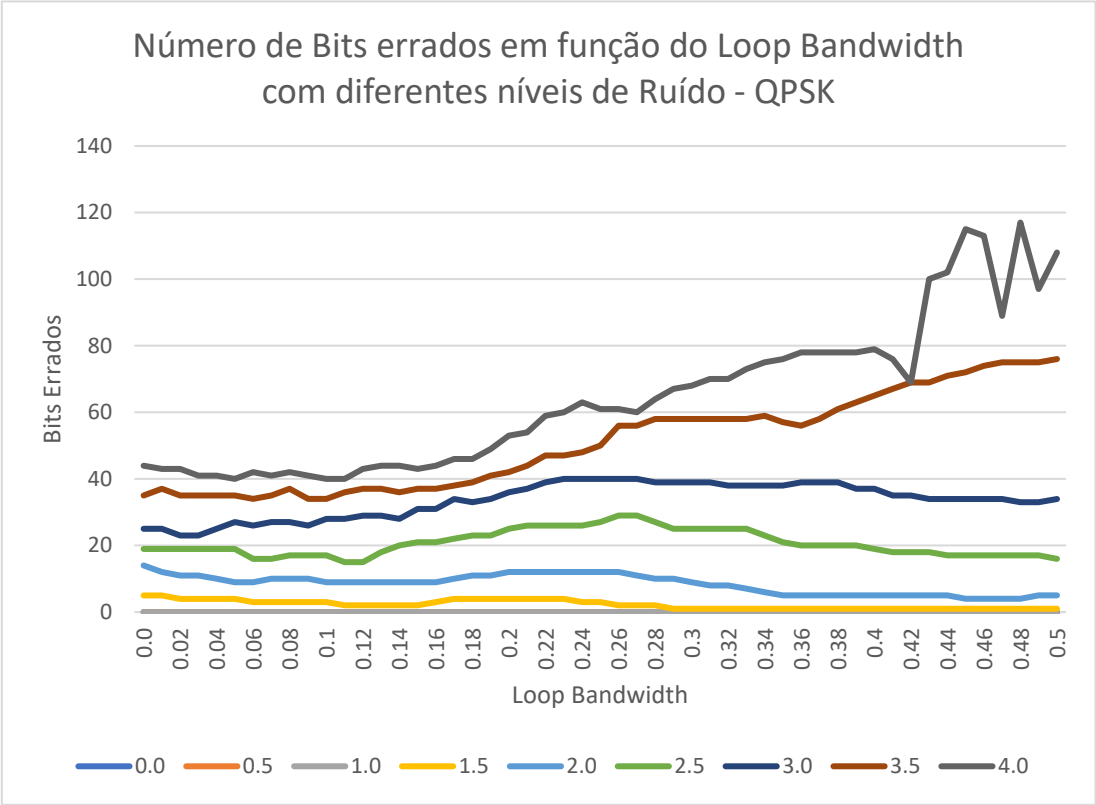


Figura 22 - Gráfico de nº de bits diferentes QPSK

A tabela de mínimos fica:

Noise [V]	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
L. B.	Todos	Todos	Todos	0.29	0.45	0.06	0.02	0.09	0.1
Err [%]	0	0	0	0.46	1.85	7.41	10.65	15.75	18.52

Com uma análise mais cuidada em torno dos mínimos:

Noise [V]	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
L. B.	Todos	Todos	Todos	0.29	0.448	0.061	0.018	0.083	0.108
Err [%]	0	0	0	0.46	1.85	6.94	10.56	15.28	18.05

Comentários Finais

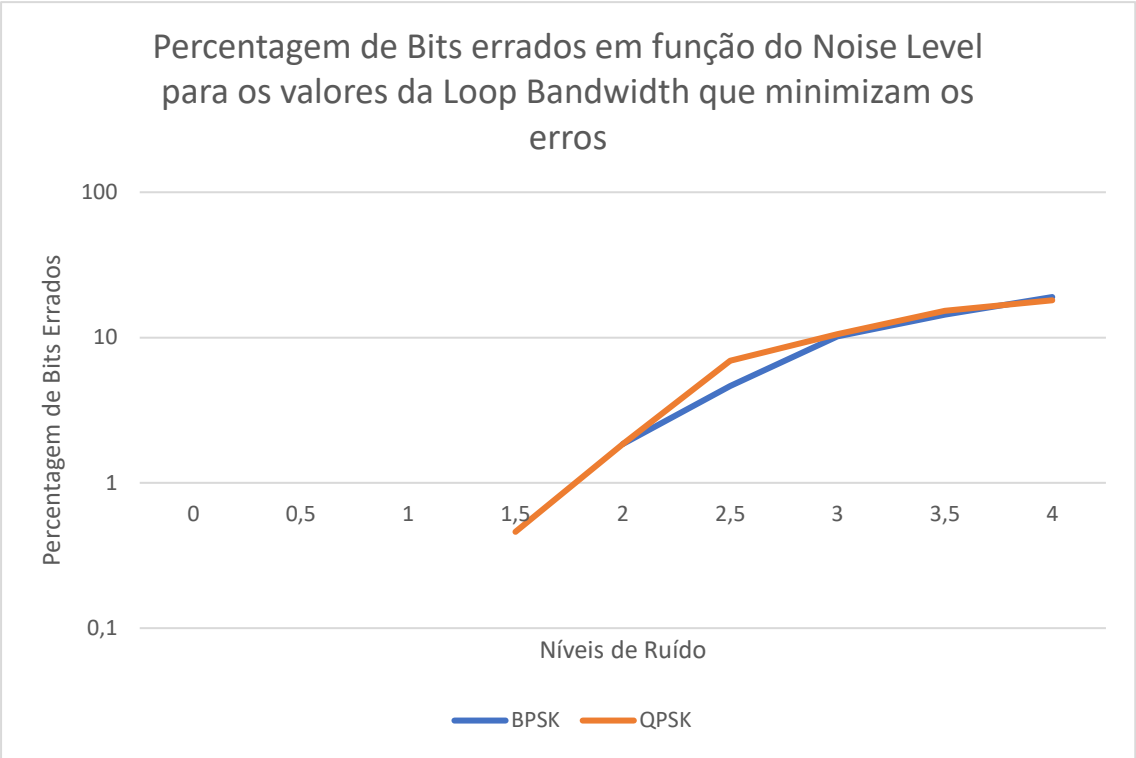


Figura 23 - Percentagem de Bits Errados em função do ruído do canal, para ambas as modulações.

Como podemos observar pelo gráfico acima, os dados confirma-nos que a probabilidade de erro de bits é semelhante em ambas as modulações, o mesmo não pode ser dito para a probabilidade de erro de símbolo que, em teoria, será superior na QPSK.

Algo interessante a observar é o quanto a probabilidade de erro de bit dispara caso o filtro adaptado seja removido:

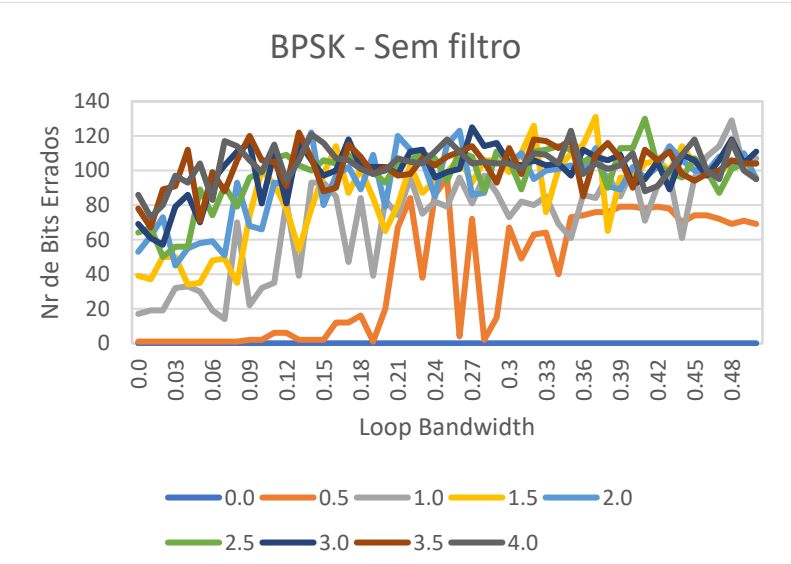


Figura 24 - Número de Bits errados, por nível de ruído, em função da Loop Bandwidth - Sem Filtro

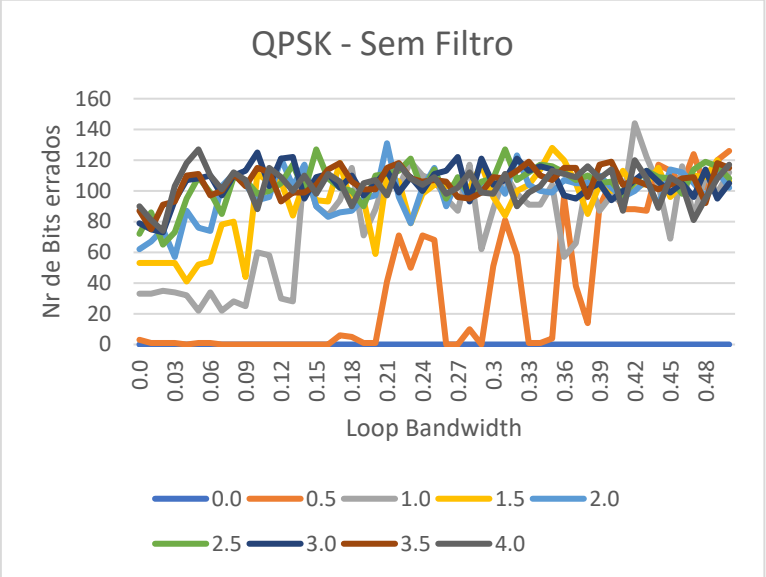


Figura 25 - Número de Bits errados, por nível de ruído, em função da Loop Bandwidth - Sem Filtro

Observa-se que para baixos valores de ruído e Loop Bandwidth, o número de bits errados pode ser desprezável. Isto deve-se ao facto de a relação sinal ruído já ser bastante alta, assim bastam pequenas iterações na malha de recuperação, no Symbol Sync, para se obter os bits corretos (em situações com ruído presente).

Observamos ainda que o número de bits errados se mantém constante em várias iterações do programa (para os mesmos valores de noise e loop bandwidth) nas mesmas posições do vetor. Acreditamos que isto se deve ao facto de o ruído ser gerado com uma sequência "aleatória" de valores que está, na realidade, predefinida por uma "seed" no programa.

É de notar que os pontos da constelação de QPSK não são $(1,1)$, $(-1,1)$, $(-1,-1)$ e $(1,-1)$, mas$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$, $\left(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$, $\left(-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right)$ e $\left(\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right)$; isto acontece para normalizar a energia dos sinais (igual ao quadrado da distância do respetivo ponto da constelação à origem).