



**Área Departamental de Engenharia de Eletrónica e  
Telecomunicações e de Computadores**

**2º Módulo**

**Comunicações Digitais**

**Estudo e aplicação de conceitos fundamentais  
sobre SCD, cifra, técnicas de deteção de erros e  
codificação de canal.**

Autores:

48285 Aureliu Iurcu

48349 José Silva

**Licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2021/2022**

04/07/2022

# Introdução

O foco principal deste módulo foi o desenvolvimento de programas e aplicações nas linguagens *Python* e *C* a fim de aprofundarmos os estudos e aplicar os conceitos fundamentais sobre os *Sistemas de Comunicação Digital*, teoria de informação, técnicas de detecção de erros, técnicas de codificação de canal e aplicação de cifra.

## Desenvolvimento

### - Exercício 1

Neste exercício foram aprofundadas na prática a aplicação da cifra em ficheiros de texto e de imagem. O foco apareceu na implementação das cifras de *Caesar* e *Vernam*.

A cifra de *Caesar* usa a técnica de deslocamento de um certo número de caracteres positivamente ou negativamente para a codificação da informação e o seu deslocamento negado para a descodificação.

A cifra de *Vernam* aplica a técnica do *XOR* com uma certa chave aleatória para a codificação dos bits e aplica exatamente a mesma técnica para a descodificação da mesma, mas com o pormenor, de a chave ter de ser necessariamente a mesma que foi usada para a codificação da informação.

Para a verificação do funcionamento correto da aplicação de cada uma das cifras foram realizados testes onde os mesmos verificam a entropia de dois ficheiros e criam o seus histogramas.

#### 1 ° Teste ( ficheiros originais):

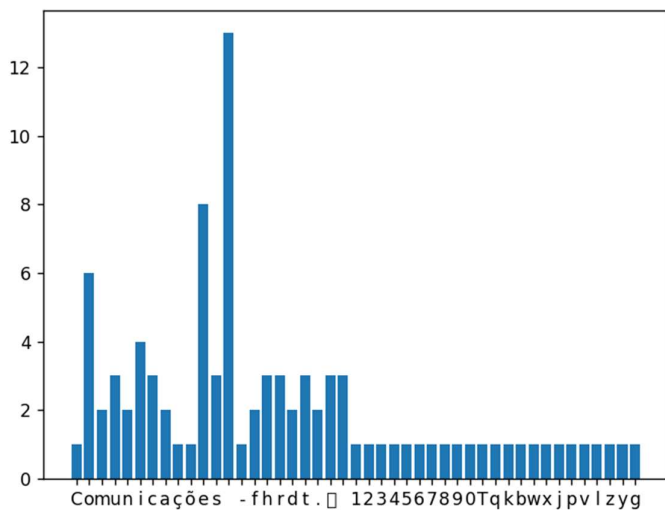


Figura 2 - Histograma "a.txt" original

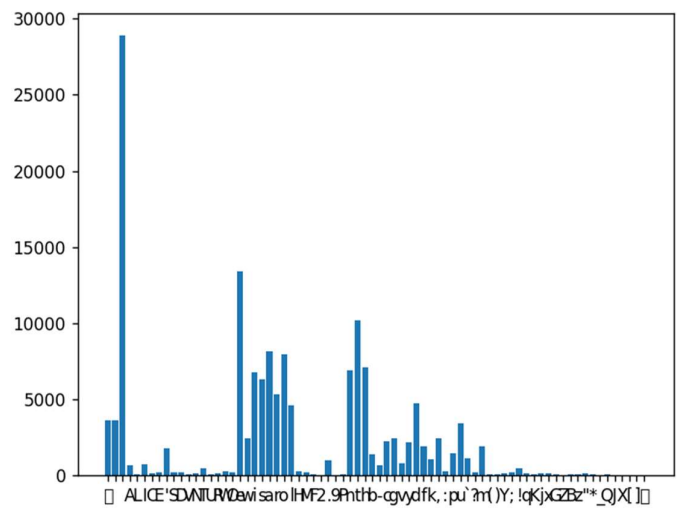


Figura 1 - Histograma "alice29.txt" original

Entropia do ficheiro "a.txt" -> 5.00507

Entropia do ficheiro "alice29.txt" -> 4.567666

## 2 ° Teste ( ficheiros cifrados com *Ceaser*):

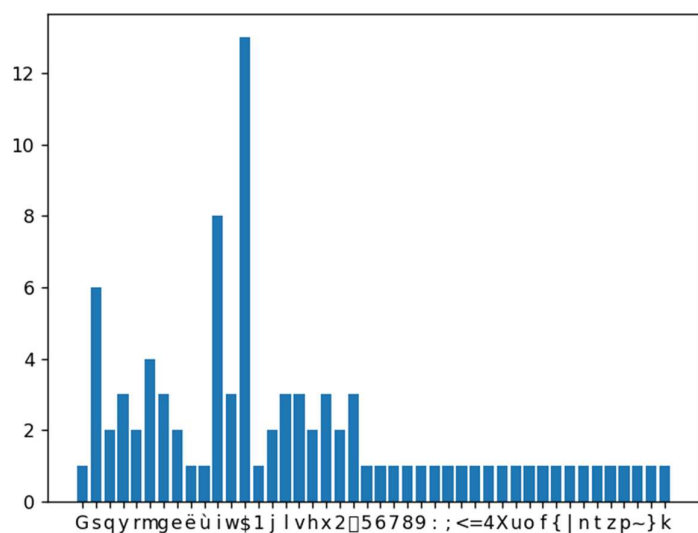


Figura 4 - Histograma "a.txt" cifrado com *Ceaser*

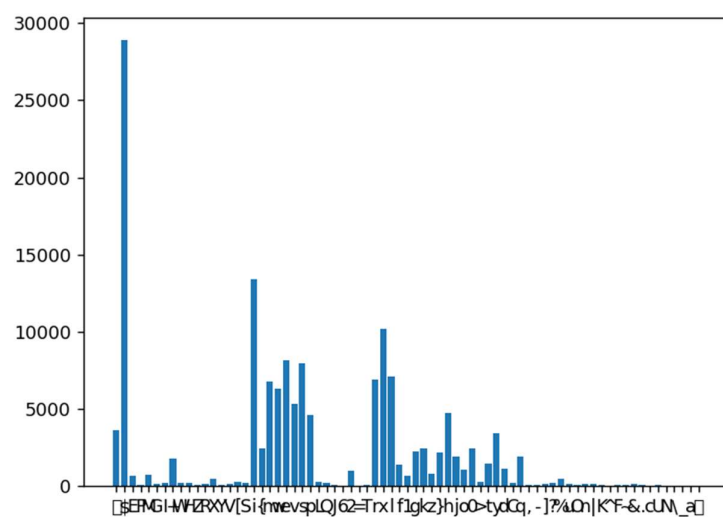


Figura 3 - Histograma "alice29.txt" cifrado com *Ceaser*

Entropia do ficheiro "a\_ciphared\_ceaser.txt" -> 4.959446

Entropia do ficheiro "alice29\_ciphared\_ceaser.txt" -> 4.512862

## 3 ° Teste ( ficheiros decifrados com *Ceaser*):

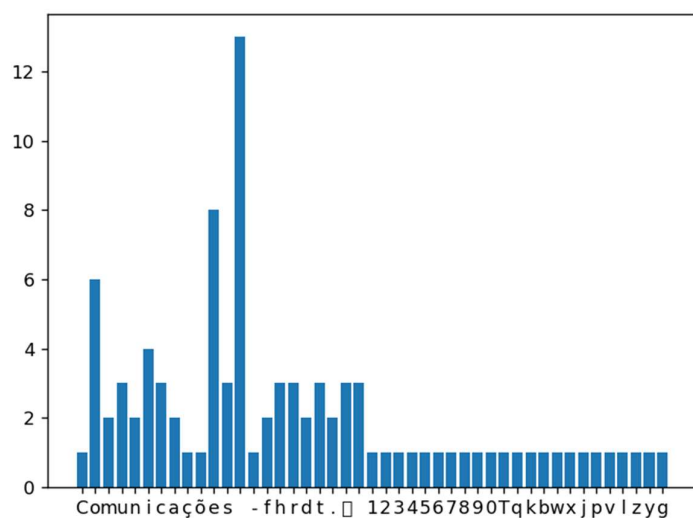


Figura 6 - Histograma "a.txt" decifrado com *Ceaser*

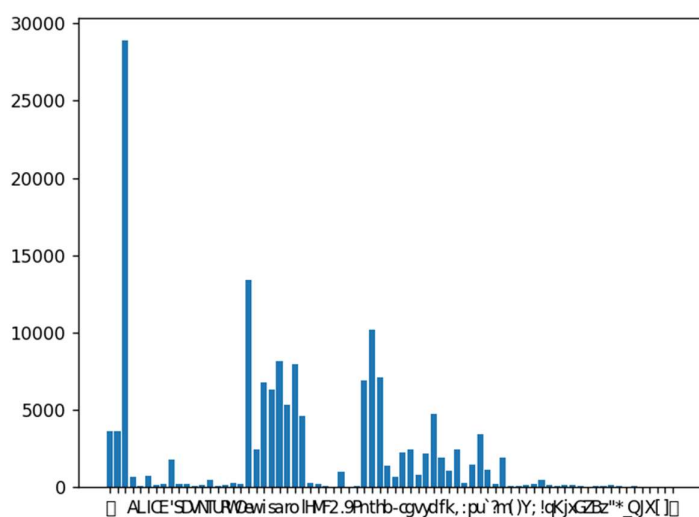


Figura 5 - Histograma "alice29.txt" decifrado com *Ceaser*

Entropia do ficheiro "a\_deciphared\_ceaser.txt" -> 5.00507

Entropia do ficheiro "alice29\_dedciphared\_ceaser.txt" -> 4.567666

## 2 ° Teste ( ficheiros cifrados com *Vernam*):

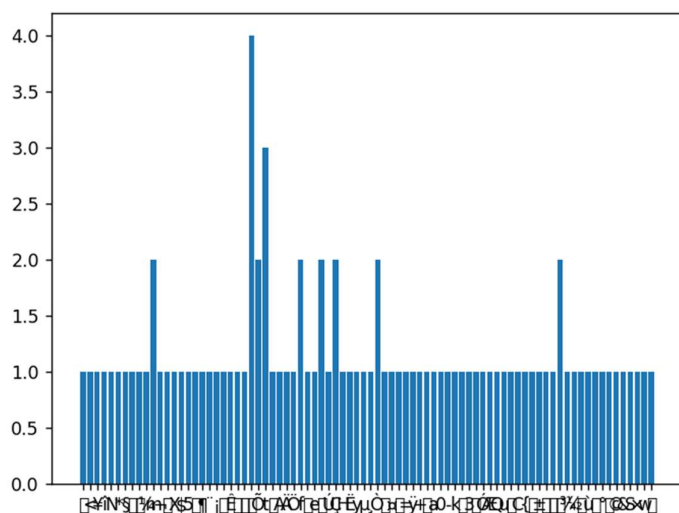


Figura 8 - Histograma "a.txt" cifrado com *Vernam*

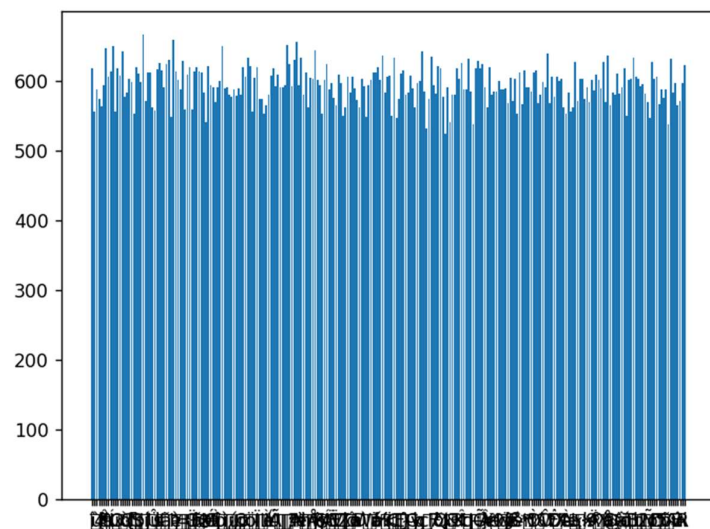


Figura 7 - Histograma "alice29.txt" cifrado com *Vernam*

Entropia do ficheiro “a\_ciphared\_vernam.txt” -> 6.269962

Entropia do ficheiro “alice29\_ciphared\_vernam.txt” -> 7.998646

## 3 ° Teste ( ficheiros decifrados com *Vernam*):

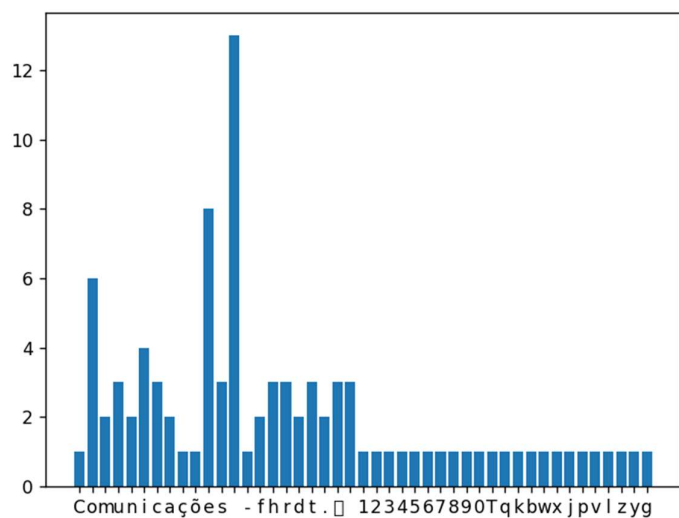


Figura 10 - Histograma "a.txt" decifrado com *Vernam*

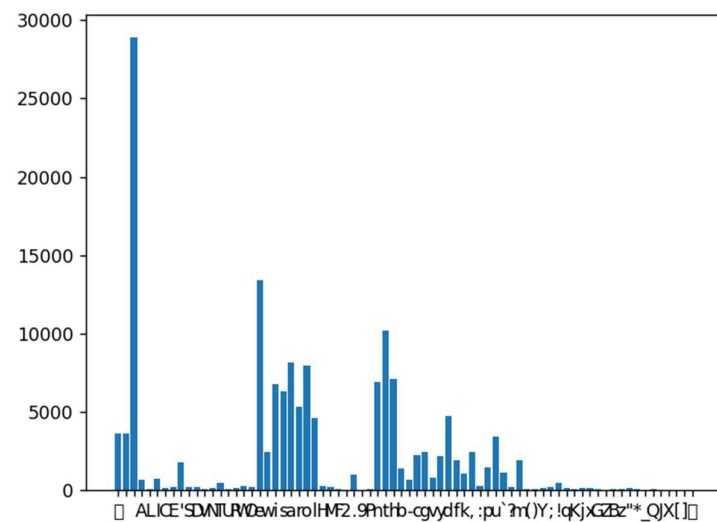


Figura 9 - Histograma "alice29.txt" decifrado com *Vernam*

Entropia do ficheiro “a\_ciphared\_vernam.txt” -> 5.00507

Entropia do ficheiro “alice29\_ciphared\_vernam.txt” -> 4.567666

Através das figuras anteriores e as entropias de cada ficheiro em causa podemos observar alguns aspetos importantes e específicos de cada técnica de cifragem:

1. Todos os ficheiros decifrados por qualquer uma das técnicas irão coincidir com o seu ficheiro original, tanto em termos de histograma como em termos de entropia o que leva a concluir que não existiu nenhum erro no processo
2. Aplicando a cifra de *Ceaser* na codificação de qualquer um dos ficheiros, o seu histograma vai se manter igual no seu eixo das ordenadas (ocorências) enquanto que no eixo das abcissas (carácter) irá mudar pois todos os caracteres sofreram um deslocamento.
3. Aplicando a cifra de *Vernam* na codificação o histograma do ficheiro irá mudar completamente criando um histograma muito mais equilibrado, sendo isso explicado pelo *XOR* feito dos bits de dados com a chave gerada aleatoriamente, o mesmo acontece com a entropia que é explicada pelo mesmo fator. Na aplicação dessa cifra garantimos que os caracteres que irão ser codificados irão ter aproximadamente a mesma ocorrência.

Na análise dos exemplos anteriores podemos concluir que é possível ter segurança perfeita num cenário em que é usado a cifra de *Vernami* e a chave gerada na codificação do mesmo seja enviada por um canal seguro.

## - Exercício 2

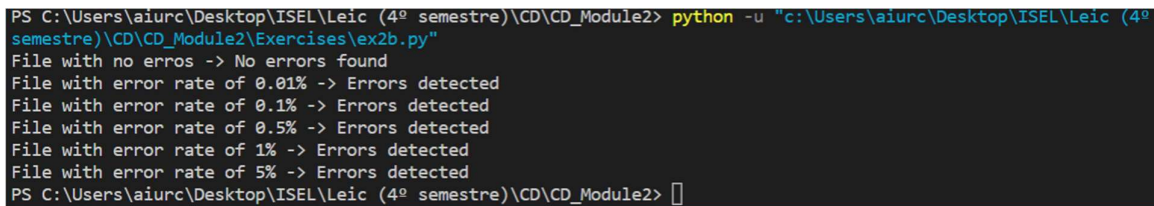
Neste exercício foram aprofundadas técnicas de detecção de erros, que na prática resumiu-se no estudo e aplicação do *CRC (Cyclic Redundancy Check)*.

Para o aprendizado do mesmo foram realizados os métodos “*crc\_file\_compute*” e “*crc\_file\_check*”, onde o “*crc\_file\_compute*” irá ser utilizado para a criação do cabeçalho do ficheiro com o seu respetivo *CRC* calculado, e a função “*crc\_file\_check*” que irá fazer a verificação de integridade do ficheiro usando para este efeito o *CRC* recebido no cabeçalho do ficheiro.

Para ser testado o correto funcionamento das funções foi realizada uma função auxiliar que injeta uma certa quantidade de erros no ficheiro que simulamente passou pelo emissor, canal e que no momento se encontra nas mãos do recetor.

Dessa forma foi realizada uma função *main* de teste que cria os ficheiros com a taxas de erros de (0%, 0.01%, 0.1%, 0.5%, 1% e 5%).

Usando um *CRC32* conseguimos observar os seguintes resultados



```
PS C:\Users\aiurc\Desktop\ISEL\Leic (4º semestre)\CD\CD_Module2> python -u "c:\Users\aiurc\Desktop\ISEL\Leic (4º semestre)\CD\CD_Module2\Exercises\ex2b.py"
File with no erros -> No errors found
File with error rate of 0.01% -> Errors detected
File with error rate of 0.1% -> Errors detected
File with error rate of 0.5% -> Errors detected
File with error rate of 1% -> Errors detected
File with error rate of 5% -> Errors detected
PS C:\Users\aiurc\Desktop\ISEL\Leic (4º semestre)\CD\CD_Module2> █
```

Figura 11 - *CRC* Test

Analisando a figura podemos concluir que no uso de um *CRC32* consegue detetar erros em qualquer que seja a percentagem de erros presentes no ficheiro.

### - Exercício 3

O objetivo do exercício em causa foi o entendimento e a correta implementação dos métodos de codificação de canal baseado nas técnicas de *NRZU* (*Non return to zero unipolar*), e *PSK* (*Phase Shift Keying*).

Para se conseguir alcançar o objetivo foram implementadas as funções “*NRZU\_Coder*”, “*PSK\_Modulator*”, “*NRZU\_Decoder*” e “*PSK\_Demodulator*”.

Todas as funções mencionadas anteriormente têm o foco da transformação de sinal digital em sinal analógico e vice-versa.

Para tal temos as funções “*NRZU\_Coder*” e “*NRZU\_Decoder*” que transformam o sinal digital em pulsos quadrados e as funções “*PSK\_Modulator*” e “*PSK\_Demodulator*” que passam o sinal digital para ondas sinusoidais. Sendo as ondas quadradas ou sinusoidais irão ser posteriormente da mesma forma enviados pelo canal.

Para o teste dos métodos mencionados anteriormente foram criadas duas funções auxiliares “*rect\_pulse\_presentation*” e “*sinusoidal\_pulse\_presentation*” que apresentam uma sequência de bits usando o formato *NRZU* e *PSK* respetivamente, as funções em causa também tem parâmetros que especificam a variância da amplitude e o ruído associado ao canal que ajudam a simular um certo canal de transmissão de dados.

No teste realizado foi usada a sequência binária b’10110001’. Os resultados simulando um canal perfeito, com com 75 amostras por bit com amplitude de 5V para o *NRZU* e amplitude máxima e frequência de 2V para a onda *PSK* estão apresentados nas figuras abaixo.

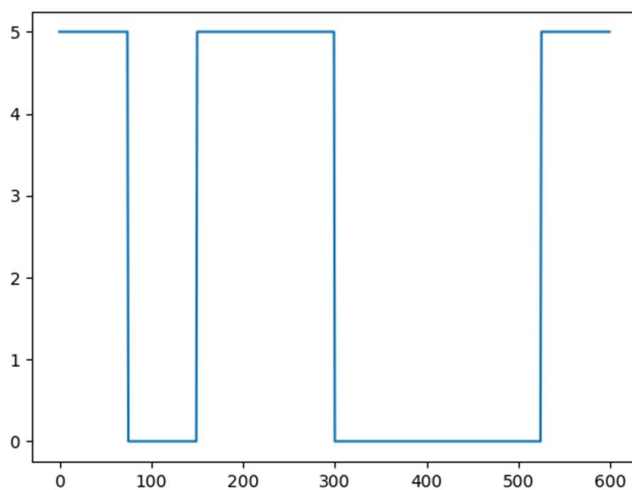


Figura 12 - *NRZU*

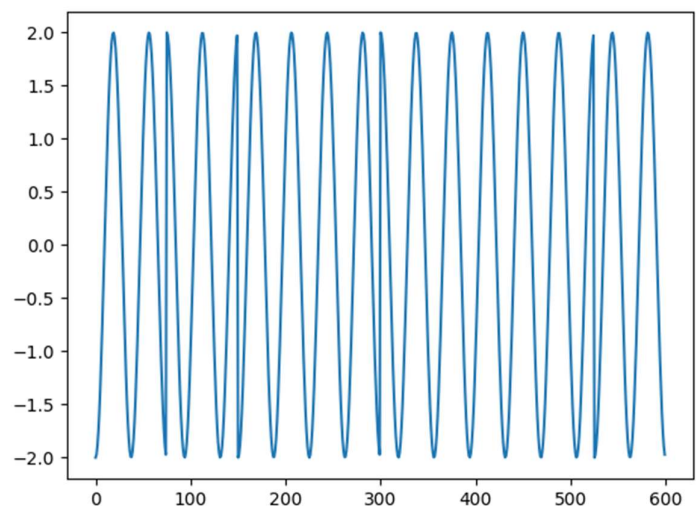


Figura 13 - *PSK*



Foram também realizados testes que simulam canal ruidoso e com variância na amplitude.

Os mesmos estão apresentados a seguir.

### Teste 1 ( $\alpha = 1$ & $n(t)$ in $-0.8..0.8$ ):

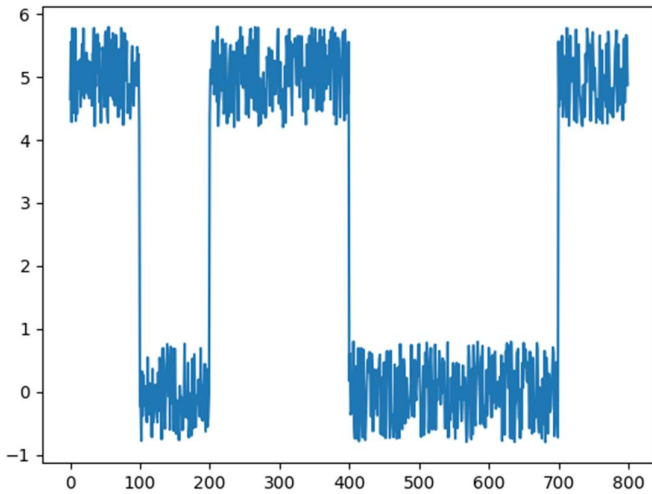


Figura 15 - NRZU with noise

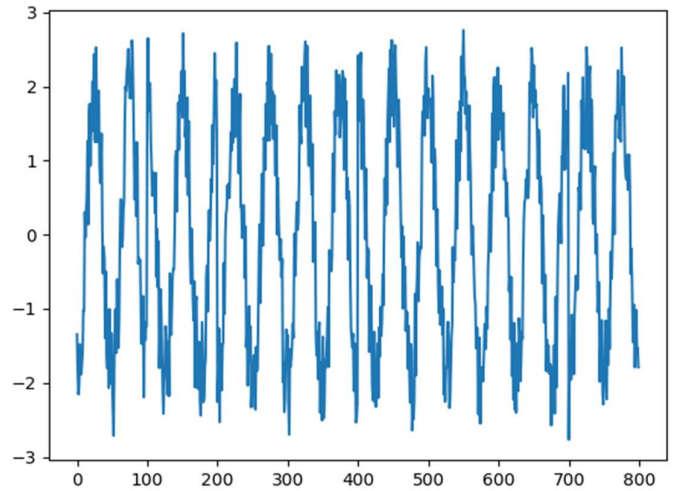


Figura 14 - PSK with noise

```
NRZU = [1, 0, 1, 1, 0, 0, 0, 1]
BER_NRZU with (alpha = 1) && (noise in -0.8..0.8) -> 0.0
PSK = [1, 0, 0, 0, 0, 1, 1, 1]
BER_PSK with (alpha = 1) && (noise in -0.8..0.8) -> 0.25
```

### Teste 2 ( $\alpha < 1$ & $n(t) = 0$ ):

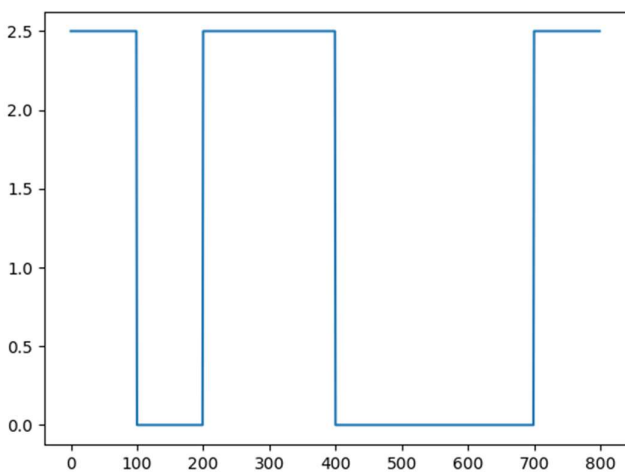


Figura 16 - NRZU with amplitude variance and no noise

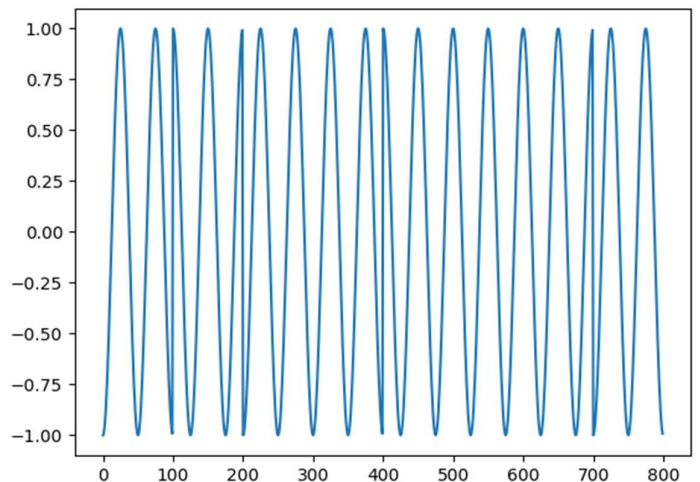


Figura 17 - PSK with amplitude variance and no noise

```
NRZU = [0, 0, 0, 0, 0, 0, 0, 0]
BER_NRZU with (alpha < 1) && (noise = 0) -> 0.5
PSK = [1, 0, 1, 1, 0, 0, 0, 1]
BER_PSK with (alpha < 1) && (noise = 0) -> 0.0
```

Através dos exemplos apresentados anteriormente podemos chegar as seguintes conclusões:

1. Caso a variância ultrapasse a metade da amplitude no sinal *NRZU* irá existir uma troca de 50% dos bits uma vez que apenas os bits iguais a 0 serão decodificados com sucesso enquanto que no sinal *PSK* a chance de errar um bit caso a variância aumenta não é tão grande desde a mesma não seja grande demais.
2. Em termos do ruído podemos concluir que quanto maior o ruído maior a chance de um bit ser mal decodificado, aplicando-se isto para qualquer que seja o canal de transmissão.

#### - Exercício 4

Neste exercício o principal objetivo foi aprender a criar um *SCD* na prática onde consigamos detetar o emissor, canal e o recetor.

O exercício consiste na criação de uma conexão entre dois dispositivos, em que no caso específico seria entre um *Arduino* e um *PC* em que o *Arduino* é o emissor que estaria conectado ao *PC*, que é o nosso emissor, através de um cabo *USB* que seria o nosso canal.

A conexão física está apresentada na figura abaixo



**Figura 18 - Conexão física entre o *Arduino* e o *PC***

Para conseguirmos enviar dados do *Arduino* para o *PC* foi usado o IDE *Arduino* para o controlo da informação presente no mesmo e a linguagem *Python* para o controlo da informação recebida no *PC*.

O processo foi dividido em 3 fases:

1. Criação de um programa em *Arduino* que envia uma sequência de 3 Strings (“Begin”, “Processing data...” e “Finished”) com um delay de 1s entre o envio de cada uma delas.

## 2. Configuração do canal de transmissão do *Arduino*

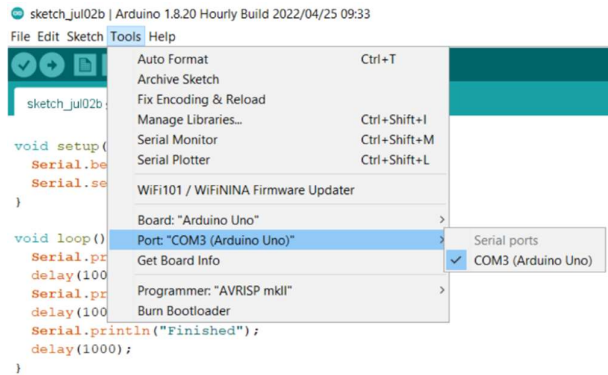


Figura 19 - Conexão ao canal de transmissão de dados *COM3*

3. Criação de um programa usando a linguagem *Python* e a biblioteca *serial*. O programa em *Python* tem como objetivo ler os dados transmitidos pelo *Arduino* e excrever os mesmos na consola. A biblioteca *serial* é usada para possibilitar a conexão ao *Arduino* através do canal acima mencionado

Nas figuras a seguir demonstrasse o correto funcionamento que comprova a conexão entre os dois dispositivos .



Figura 20 - Texto apresentado na consola logo após o inicio do programa



Figura 21 - Texto apresentado na consola 1s após o envio de "Begin"



Figura 22 - Texto apresentado na consola 1s após o envio de "Processing data..."

## **Conclusão**

Durante a realização desse módulo, fomos capazes de aplicar conceitos fundamentais sobre SCD, cifra, codificação de canal, implementação de códigos de controlo de erros, desempenho de sistemas de transmissão digital, ao nível físico com código de linha e modulação digital, sem esquecer a manipulação de dados e a interação com a plataforma Arduino. Acreditamos que após a realização desse módulo somos capazes de selecionar e aplicar códigos detetores e corretores de erros; reconhecer, identificar e aplicar as principais técnicas de codificação de dados com e sem perda, de acordo a matéria lecionada; e projetar e realizar sistemas que incluam comunicação digital entre dispositivos. Sem dúvidas que a resolução dos exercícios deste módulo serviu para a consolidação da segunda parte da matéria, que agora, temos muito maior domínio. Não obstante, não nos foi possível cumprir o desafio de realizar todos os exercícios propostos pelo enunciado, quer seja pela necessidade da utilização de conteúdos que não chegamos a implementar no módulo anterior, ou por não termos conseguido resolvê-los a tempo da data da entrega. Procuramos adotar as melhores soluções para as questões contidas no módulo em questão, apesar de nem todas serem as melhores do ponto de vista das complexidades (espaciais e temporais). Em suma, foi um módulo bastante desafiador, e fomos capazes de cumprir a maioria dos requisitos impostos, aplicando devidamente os conceitos aprendidos nas aulas