

**Relatório do Segundo Trabalho Prático**

|  |  |
| --- | --- |
| Grupo | 13 |
| 38866 | Manuel Dias |

|  |  |
| --- | --- |
| Docente | Artur Ferreira |
|  |  |

Relatório final realizado no âmbito da Unidade Curricular de Comunicações,  
do curso de licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2019/2020

Junho de 2020

Índice

[Índice de Figuras 2](#_Toc43922066)

[1. Objetivos 3](#_Toc43922067)

[2. Grupo 1 4](#_Toc43922068)

[2.1 Alínea A 4](#_Toc43922069)

[2.1.1 Códigos de Linha 5](#_Toc43922070)

[2.1.1.1 Ausência de Código de Linha 5](#_Toc43922071)

[2.1.1.2 NRZ Bipolar Absoluto 6](#_Toc43922072)

[2.1.1.3 NRZ Unipolar Mark 6](#_Toc43922073)

[2.1.2 Modulação Digital 7](#_Toc43922074)

[2.1.2.1 Ausência de Modulação Digital 7](#_Toc43922075)

[2.1.2.2 Modulação OOK 8](#_Toc43922076)

# Índice de Figuras

[Figura 1 - Representação da GUI totalmente modificada 4](#_Toc43922060)

[Figura 2 - Representação do código de linha na GUI 5](#_Toc43922061)

[Figura 3 - Representação da modulação digital na GUI 7](#_Toc43922062)

# 1. Objetivos

Este trabalho tem como objetivos:

* Desenvolver programas e aplicações em MATLAB/OCTAVE;
* Estudo e aplicação de conceitos sobre codificação de fonte, cifra, correção de erros e desempenho de sistemas de transmissão digital, ao nível físico;

# 2. Grupo 1

No primeiro grupo do trabalho prático, é pretendido adicionar funcionalidades à GUI do trabalho anterior, adicionando 3 sub-blocos ao bloco CSD: emissor, canal e recetor.

## 2.1 Alínea A

As funcionalidades adicionadas foram separadas em 3 partes:

1. Suporte de códigos de linha;
2. Suporte de modulações digitais;
3. Suporte de simulação de um canal físico;

Em todas as funcionalidades, existem 2 parâmetros genéricos que podem ser escolhidos pelo utilizador, o tempo de bit em segundos e o número de amostras por cada bit. Este último parâmetro não convém ser um número muito elevado, pois, dependendo do ficheiro a analisar, este vai indicar a resolução do sinal e quanto maior for a resolução, mais memória vai consumir. Estes parâmetros podem ser verificados na Figura 1 no canto inferior direito.

Opcionalmente, foi adicionado um modo janela que mostra 10 bits no gráfico da GUI, quando houver uma simulação e o utilizador carregar num dos 2 botões do sub-bloco de código de linha ou de modulação digital. Esta janela pode ser movível no eixo do tempo de transmissão, deslocando a barra por debaixo do gráfico.

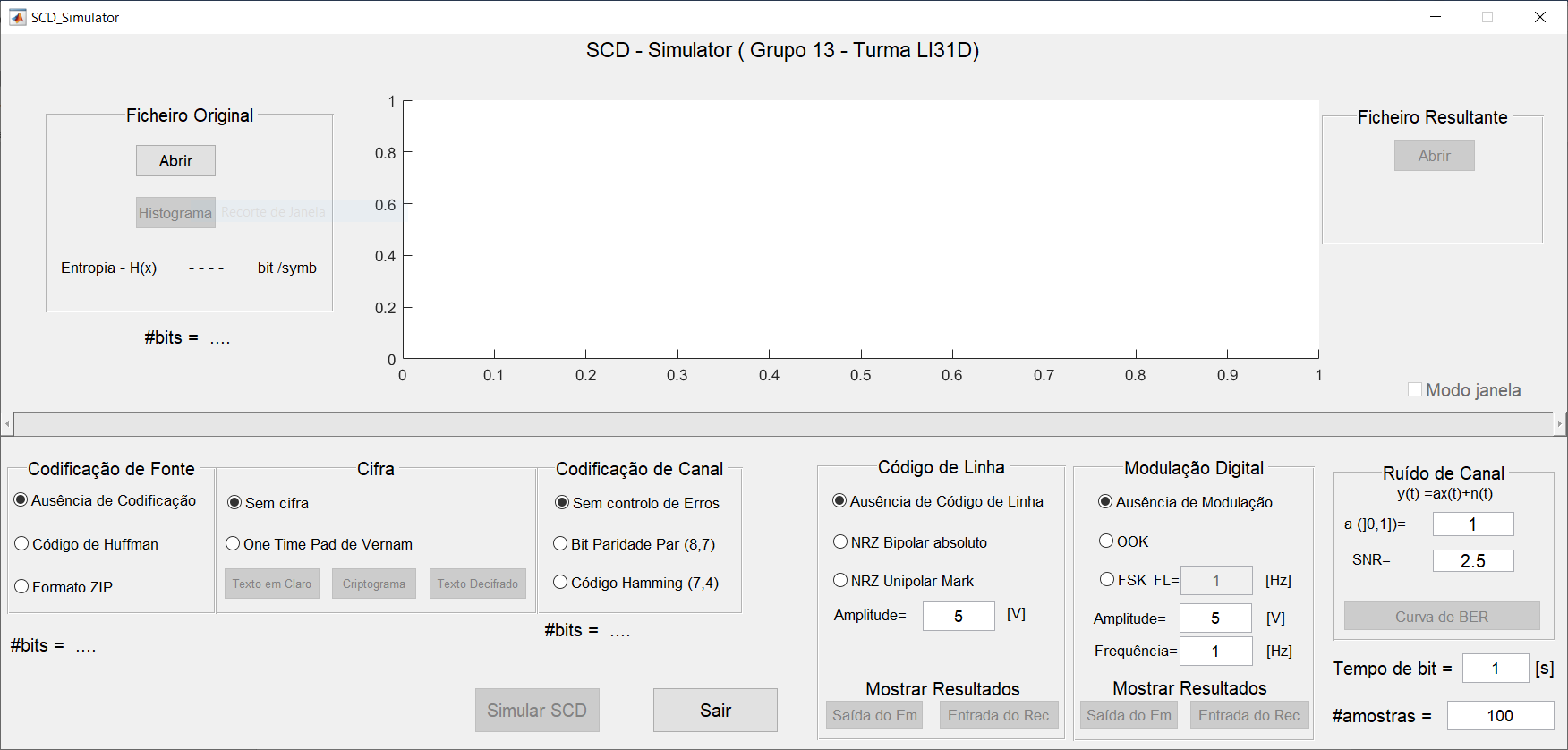


Figura - Representação da GUI totalmente modificada

### 2.1.1 Códigos de Linha

O código de linha está dividido em 3 partes:

1. Ausência de código de linha;
2. NRZ Bipolar Absoluto;
3. NRZ Unipolar Mark;

Esta implementação permite ao utilizador escolher que tipo de código de linha quer e, para além disto, o utilizador pode escolher a amplitude de cada pulso em Volts, como pode ser verificado na Figura 1 e na Figura 2.

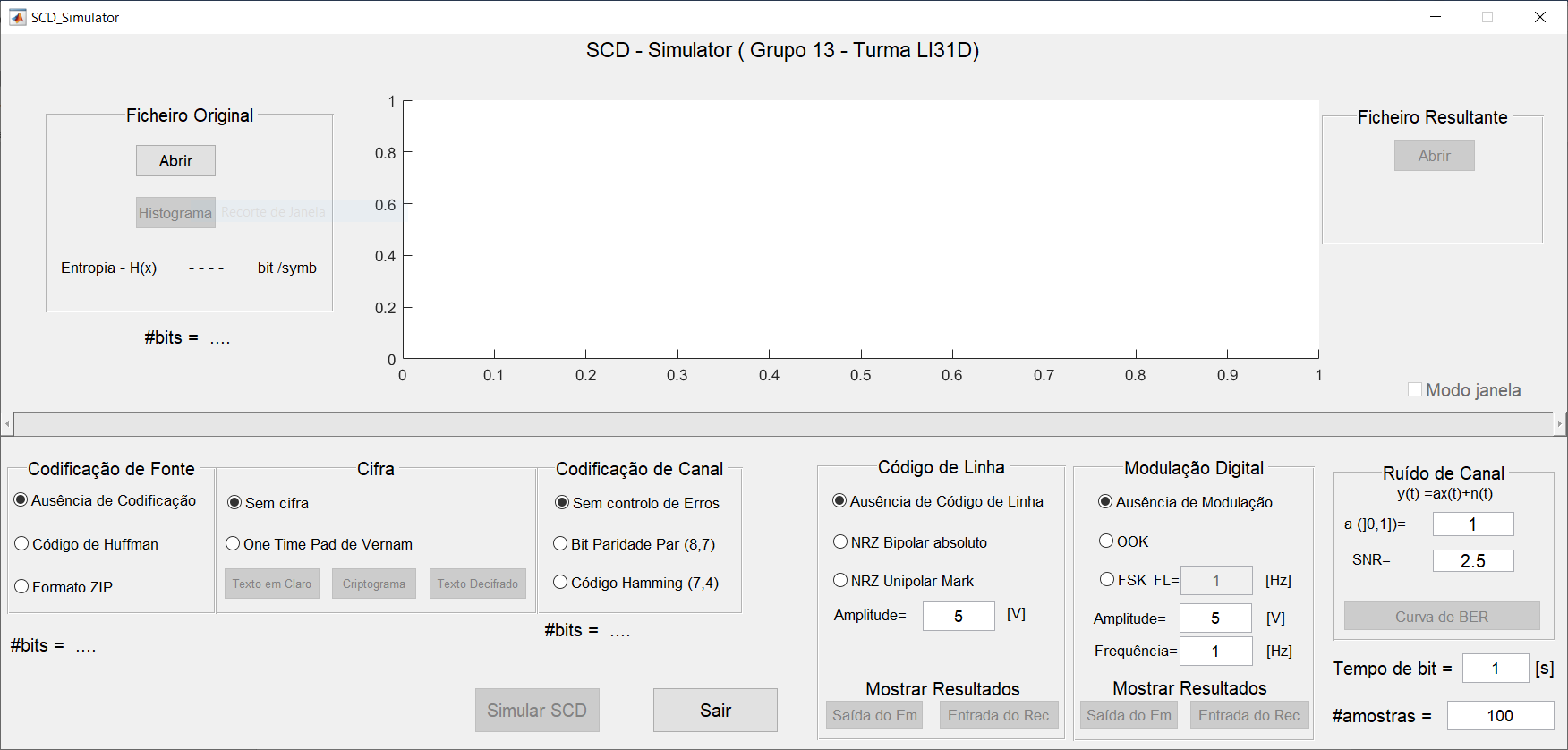


Figura - Representação do código de linha na GUI

#### 2.1.1.1 Ausência de Código de Linha

Na emissão de pulsos digitais não modulados, é necessário dos dados a transmitir a partir de um ficheiro, da amplitude de cada pulso, o número de amostras de cada pulso e o tempo de bit. A partir disto é possível transformar os dados do ficheiro num Stream de pulsos em que cada pulso pode ter entre 2 valores, ou 0, caso o bit seja 0, ou o valor de amplitude se o bit for 1.

Esta função, *Em\_Aus*, retorna 3 vetores:

1. Vetor de pulsos;
2. Vetor de tempo;
3. Vetor de energia de cada bit;

Estes vetores são essenciais para representar graficamente os pulsos, no caso do vetor de pulsos e do tempo, como também para simular e calcular o ruído de canal, no caso do vetor de energia de cada bit.

Na receção de pulsos digitais não modulados, é feito o procedimento inverso, escrevendo, no final, num ficheiro cujo nome é a concatenação de “CLA\_” com o nome do ficheiro passado na emissão.

#### 2.1.1.2 NRZ Bipolar Absoluto

Na emissão de pulsos do tipo NRZ Bipolar Absoluto, com a técnica de bit-suffing com blocos de 6 bits, é necessário o mesmo número de parâmetros que na Ausência de Código de Linha. No entanto, antes de transformar os bits em pulsos, os bits são postos em blocos de 6 ou 7 bits. Para tal, foi feita a função *Generate\_Bip\_Block* que gera blocos de N bits, neste caso 6, se houver transições de bits dentro desse bloco, caso contrário adiciona um bit inverso ao primeiro bit do bloco para o recetor detetar uma transição. A transição destes blocos para pulsos é feita quase da mesma maneira que na funcionalidade de ausência. A diferença está quando um bit no bloco for 0, o pulso não terá o valor 0, mas sim o valor negativo da amplitude.

Na receção de pulsos do tipo NRZ Bipolar Absoluto, com a técnica de bit-suffing com blocos de 6 bits, é feito o procedimento inverso, no entanto, a cada 6 pulsos, se não detetar uma transição, então o bit a seguir é ignorado, pois este é o bit de deteção de transição. No final é gravado os dados num ficheiro cujo nome é a concatenação de “NRZBA\_” com o nome do ficheiro recebido no emissor.

#### 2.1.1.3 NRZ Unipolar Mark

Na emissão de pulsos do tipo NRZ Unipolar Mark, com a técnica de bit-suffing com blocos de 6 bits, é necessário o mesmo número de parâmetros que na Ausência de Código de Linha. No entanto, antes de transformar os bits em pulsos, os bits são postos em blocos de 6 ou 7 bits. Para tal, foi feita a função *Generate\_Uni\_Block* que gera blocos de N bits, neste caso 6, se houver transições de bits dentro desse bloco, caso contrário adiciona um bit inverso ao primeiro bit do bloco para o recetor detetar uma transição. Para fazer a transição dos blocos para pulsos, é preciso definir primeiro, o valor inicial, mark, do sinal que foi 0. Este valor, inicialmente, tanto pode ser 0 como 1, mas tem de ser o mesmo entre o emissor e o recetor. Depois, durante o ciclo da transformação, cada vez que é detetado o bit 1, o valor de mark altera para , para fazer a transição de sinal. Este sinal está compreendido entre a amplitude escolhida pelo utilizador e 0.

Na receção de pulsos do tipo NRZ Unipolar Mark, com a técnica de bit-suffing com blocos de 6 bits, é feito o procedimento inverso, no entanto, a cada 6 pulsos, se não detetar uma transição, então o bit a seguir é ignorado, pois este é o bit de deteção de transição. No final é gravado os dados num ficheiro cujo nome é a concatenação de “NRZUM\_” com o nome do ficheiro recebido no emissor.

### 2.1.2 Modulação Digital

A modulação digital está dividida em 3 partes:

1. Ausência de modulação;
2. OOK;
3. FSK;

Esta implementação permite ao utilizador escolher que tipo de modulação quer e, para além disto, o utilizador pode escolher a amplitude de sinal em Volts como a frequência em Hertz, como pode ser verificado na Figura 1 e na Figura 3. Caso o utilizador escolha a opção FSK, a frequência à frente da opção indica a frequência quando um bit for 0.

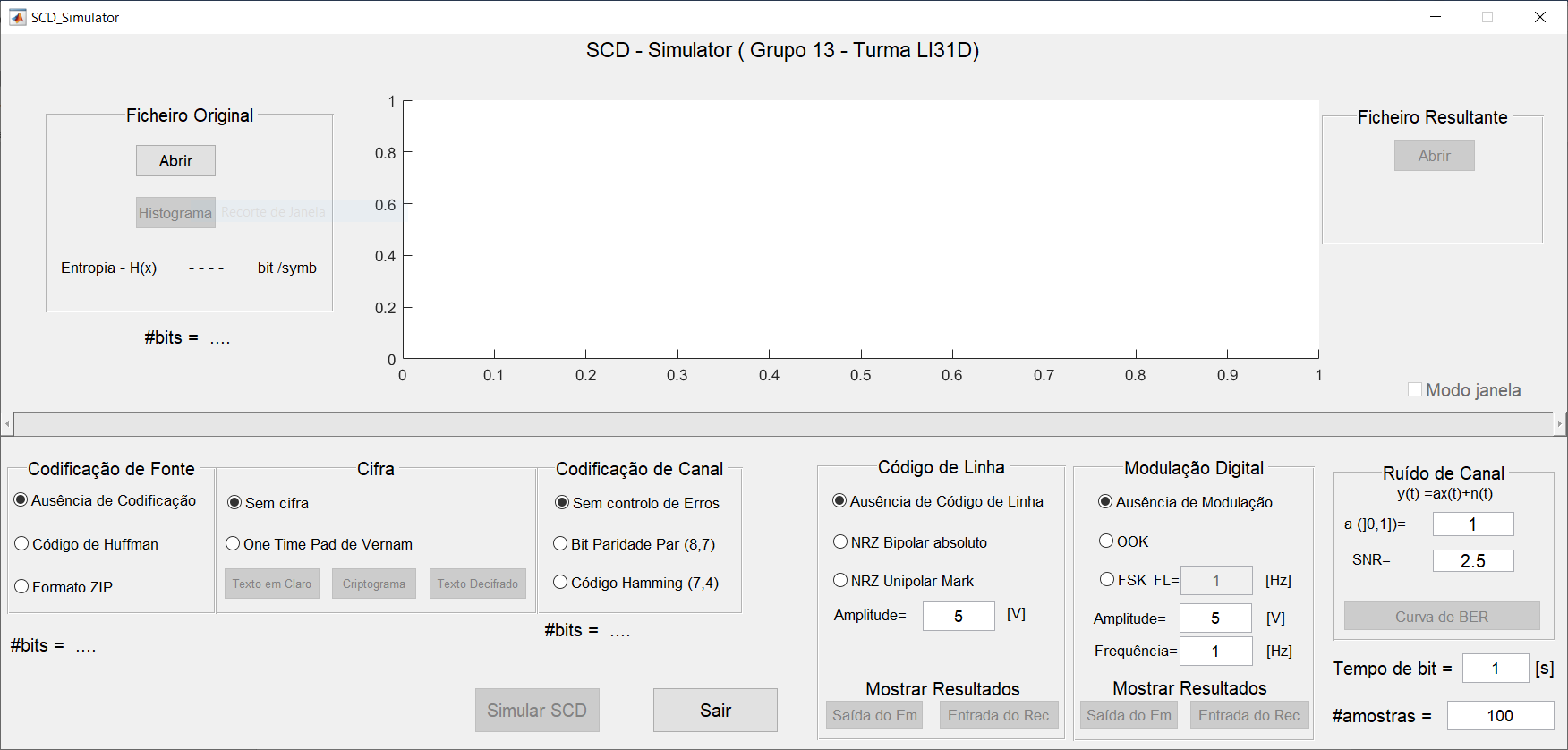


Figura - Representação da modulação digital na GUI

#### 2.1.2.1 Ausência de Modulação Digital

Neste modo, como não existe modulação, os dados à saída do emissor são iguais aos dados à saída do emissor de código de linha.

Na receção, é menos simples, pois o sinal pode vir com ruído. Para resolver isto, é calculado a energia de bit recebido em cada pulso. Se o código de linha escolhido for o NRZ Bipolar, então é verificado se esta energia é positiva ou negativa, devolvendo a amplitude de um pulso negativo caso a energia seja positiva. Caso contrário o pulso devolvido vem com amplitude negativa. Nos outros tipos de código de linha, é comparada esta energia com a energia de um pulso sem ruído e caso que a energia calculada com ruído for superior a metade da energia de um pulso sem ruído, então devolve um pulso com amplitude escolhida pelo utilizador, caso contrário o pulso terá uma amplitude de 0.

#### 2.1.2.2 Modulação OOK

Para fazer a emissão da modulação digital do tipo OOK, é necessário receber o vetor de dados a modular, a amplitude usada no código de linha, o tempo de bit, a amplitude de modulação escolhida pelo utilizador, a frequência e o número de amostras. Depois, num ciclo, é verificado pulso a pulso, se a amplitude do mesmo corresponde à amplitude do código de linha. Se for igual, então é gerado um sinal sinusoidal, caso contrário, o sinal é 0 no tempo de bit corrente.

Para a receção de sinais com modulação digital do tipo OOK é necessário calcular a energia absoluta do sinal, pois a partir disto é possível verificar se a amplitude de cada sinal de bit é igual ou superior à amplitude escolhida pelo utilizador. Se for superior ou igual, então o sinal representa um valor positivo, gerando um pulso com uma amplitude escolhida no código de linha. Caso contrário, o pulso terá ou uma amplitude com valor 0 ou com um valor negativo, dependendo do código de linha escolhido.

#### 2.1.2.3 Modulação FSK

Para fazer a emissão da modulação digital do tipo FSK, é necessário receber o vetor de dados a modular, a amplitude usada no código de linha, o tempo de bit, a amplitude de modulação escolhida pelo utilizador, a frequência que representa o bit 1, a frequência que representa o bit 0 e o número de amostras. Depois, num ciclo, é verificado pulso a pulso, se a amplitude do mesmo corresponde à amplitude do código de linha. Se for igual, então é gerado um sinal sinusoidal com a frequência com bit 1, caso contrário, é gerado com a frequência com bit 0.

Para a receção de sinais com modulação digital do tipo FSK é necessário calcular a frequência fundamental do sinal, pois a partir disto é possível verificar a frequência de cada sinal de bit. Se esta for superior ou igual à frequência do bit 1 ao quadrado, então o sinal representa um bit 1, gerando um pulso com uma amplitude escolhida no código de linha. Caso contrário, o pulso terá ou uma amplitude com valor 0 ou com um valor negativo, dependendo do código de linha escolhido.

### 2.1.3 Canal Físico

A simulação do canal físico é feita seguindo a expressão . Isto significa que é necessário que o utilizador insira um valor de atenuação do canal, α, e a relação sinal-ruído, SNR, para adicionar ruído ao canal. Isto pode ser verificado na Figura 1 e 4.

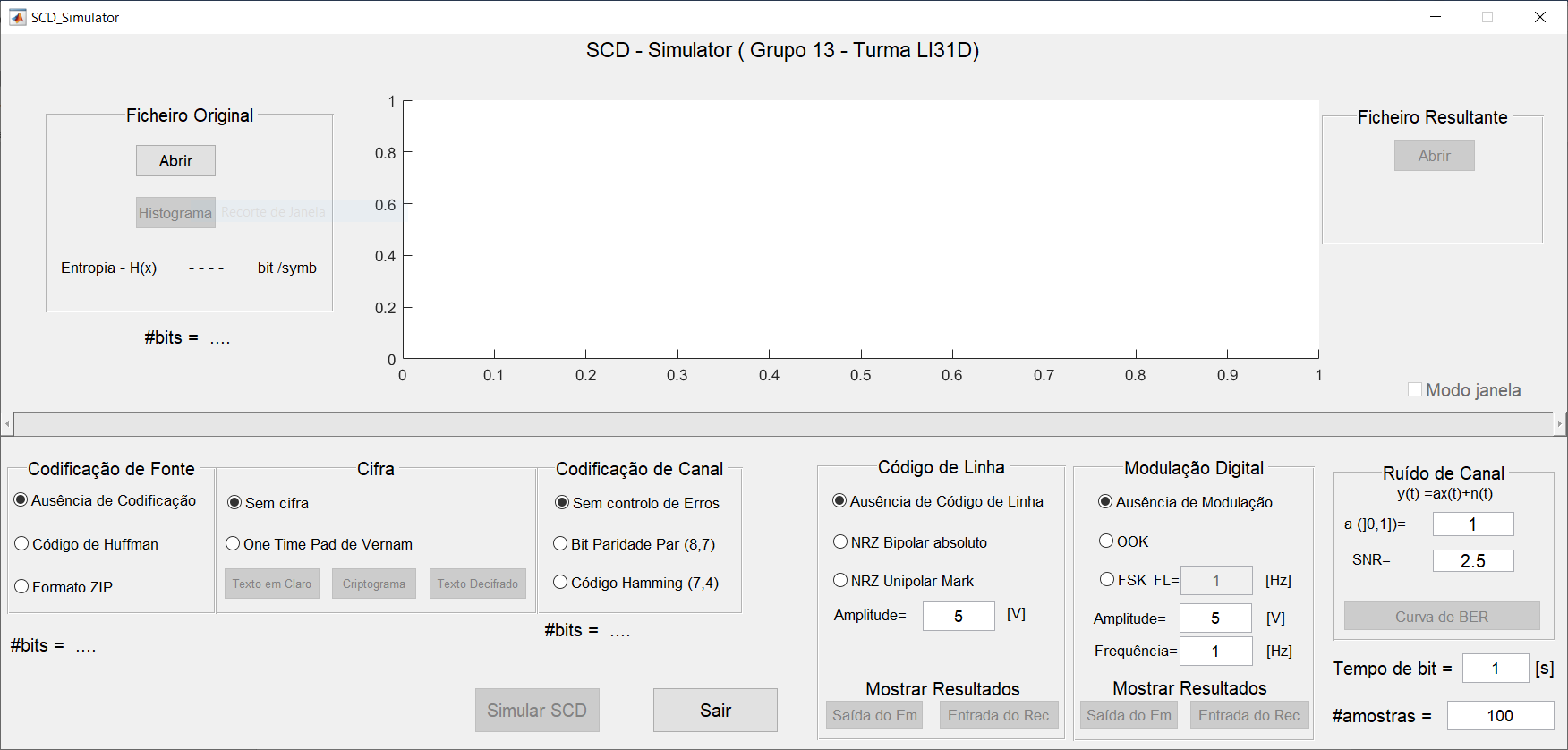


Figura - Representação do Ruído de Canal na GUI

Para calcular a adição de ruído, é preciso saber o SNR e a energia de sinal. A partir destes dois fatores, é possível calcular a energia do ruído e depois utiliza-se um gerador de números aleatórios normalmente distribuídos para simular o ruído. Por último é devolvido ao sinal de saída, os sinais de entrada atenuados com o ruído.

## 2.2 Alínea B

## 2.3 Alínea C

# 3. Grupo 2

# 4. Grupo 3

Neste grupo é pretendido fazer funções que contenham algoritmos de verificação de erros, contextualizados na validação do dígito de controlo do Bilhete de Identidade e do ISBN. Apesar de se poder usar qualquer linguagem de programação, foi utilizado à mesma o MATLAB.

## 4.1 Alínea (i)

Para fazer o algoritmo de cálculo do dígito de controlo, foi criada a função *BI\_generate\_control\_digit*. Esta recebe como parâmetro de entrada uma string com 8 caracteres, sendo estes todos números. Para obter o dígito de controlo, foi utilizada a fórmula , onde x é calculado a partir de um ciclo e o t foi calculado usando esta fórmula .

Para fazer o algoritmo de verificação, foi criada a função *BI\_check\_digit\_control*, que tem a capacidade de verificar se o dígito de controlo está correto e se todos os algarismos do BI estão bem e na ordem correta. Esta função recebe uma string com 9 caracteres, sendo estes todos números. Para fazer a verificação é usado o algoritmo do enunciado, recorrendo a uma instrução cíclica e no final é verificado se o resto da divisão do resultado com 11 é 0. Se for 0, então o número do BI é válido, caso contrário é inválido. A Figura 5 demonstra a funcionalidade das duas funções através de um script de teste.

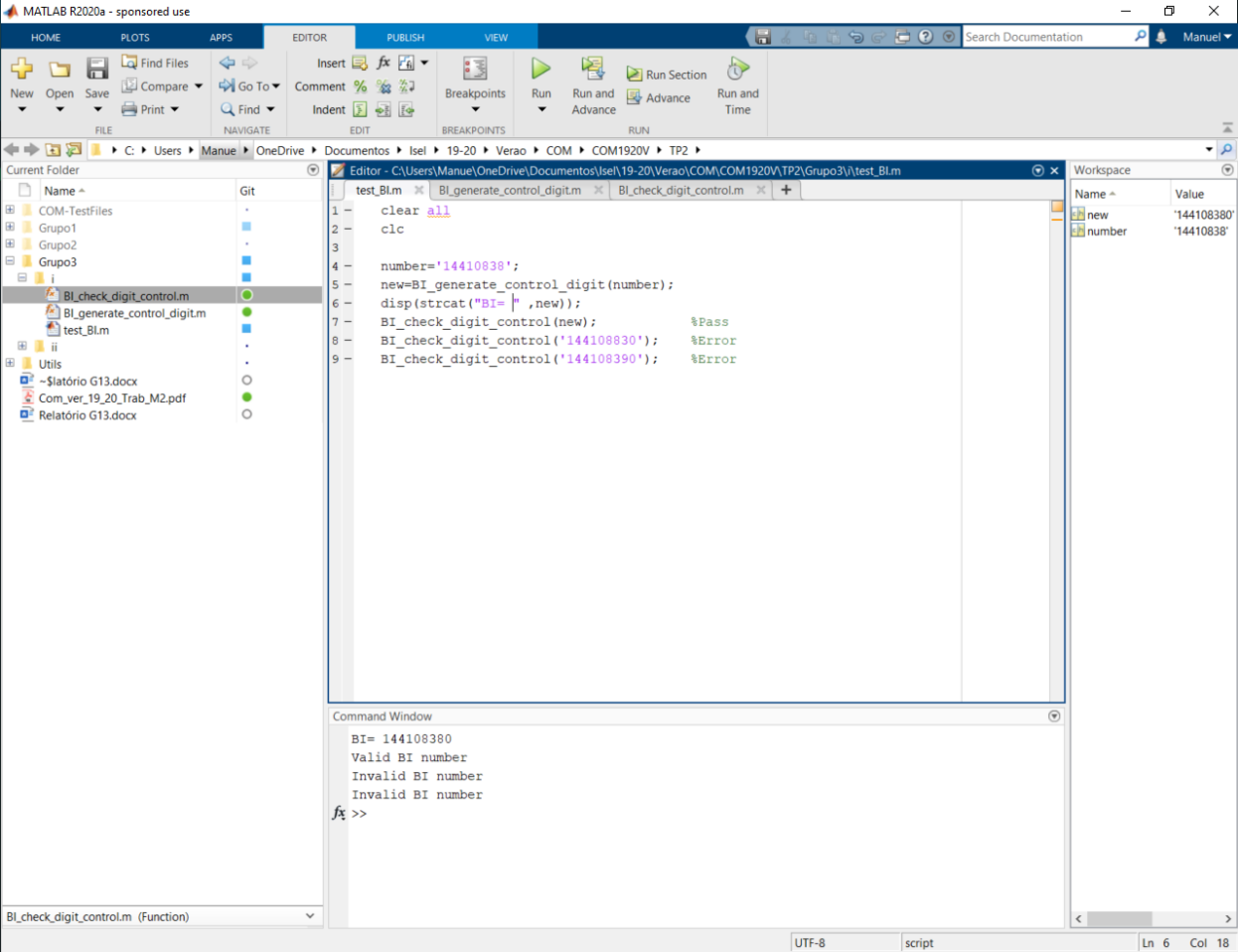


Figura - Resultados do script de testes das funções de BI

## 4.2 Alínea (ii)

Para fazer o algoritmo de cálculo do dígito de controlo, foi criada a função *ISBN\_generate\_control\_digit*. Esta recebe como parâmetro de entrada uma string com 12 caracteres, sendo estes todos números. Para obter o dígito de controlo, foi utilizada a fórmula , onde x é calculado a partir de um ciclo onde é verificado a paridade de cada algarismo e se o algarismo for par, soma-se o seu valor ao resultado, caso contrário é o triplo.

Para fazer o algoritmo de verificação, foi criada a função *ISBN\_check\_digit\_control*, que tem a capacidade de verificar se o dígito de controlo está correto e se todos os algarismos do ISBN estão bem e na ordem correta. Esta função recebe uma string com 13 caracteres, sendo estes todos números. Para fazer a verificação é usado o mesmo algoritmo que a função de geração do dígito de controlo, recorrendo à instrução cíclica. Depois é utilizada a fórmula , onde r é o resultado da instrução cíclica. Se o valor de d for diferente que 0, quer dizer que o valor de ISBN é inválido, caso contrário então é válido. A Figura 6 demonstra um script de teste onde é gerado o dígito de controlo e são feitas várias verificações a diferentes ISBN, que incluem um ISBN correto, uma troca de ordem entre 2 dígitos e um dígito errado.

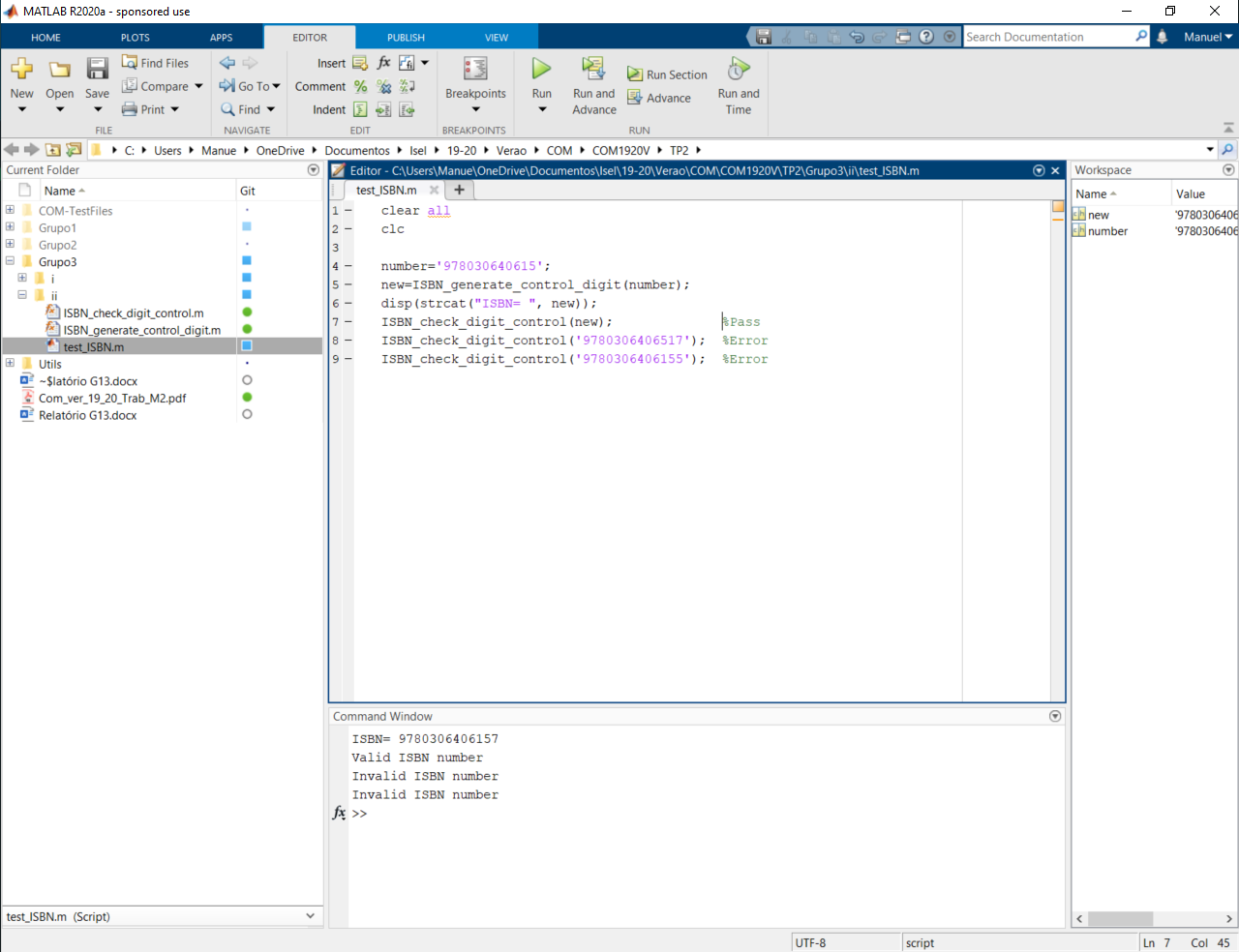


Figura - Resultados do script de testes das funções de ISBN