TEORIA DA INFORMAÇÃO
E
TRANSMISSÃO DIGITAL

Material pedagógico para apoio às aulas da unidade curricular de Teoria da Informação e Transmissão Digital leccionada no Instituto Politécnico de Bragança - Escola Superior de Tecnologia e Gestão.

João Paulo Coelho

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Conteúdo

Capítulo 1

Introdução ao Matlab®

"Real men do it command-line..."

–Anónimo

O Matlab é uma das aplicações de cálculo numérico mais utilizadas em todo o mundo. O que começou por ser apenas uma interface ao LINPACK e EISPACK é hoje um programa informático maduro e repleto de potencialidades. Pelo papel que a computação numérica desempenha em tarefas de simulação, a unidade curricular de TITD começa por uma viagem ao conceito de programação em Matlab.

1.1 Introdução

Matlab é o acrónimo de **MAT**rix **LAB**oratory e nasceu, nos finais da década de 70 do milénio passado, pela mão de Cleve Moler. Inicialmente pretendeu ser apenas uma interface para as bibliotecas de cálculo numérico, *Linpack* e *Eispack*, que corriam sob *Fortran*. No entanto, devido à sua globalização e forte adesão por parte dos meios académicos foi criada, mais tarde por Jack Little entre outros, uma versão comercial.

Mais do que o núcleo em que assenta, o poder do *Matlab* advém da possibilidade da sua extensão, para um ou mais domínios concretos do conhecimento, por adição de rotinas específicas englobadas em pacotes designados por *Toolboxes*. Desde os meados da década de 1980, e até aos nossos dias, o número de *Toolboxes* tem vindo a aumentar. É possível encontrar uma miríade de livrarias que vão da estatística à inteligência artificial passando pela lógica difusa, bioinformática e muitas outras. Para além das *Toolboxes* comerciais é muito fácil encontrar livrarias na Internet desenvolvidas e partilhadas pela imensa rede de utilizadores do *Matlab*.

O *Octave* é, tal como o *Matlab*, um programa informático de cálculo numérico. No entanto é de livre acesso. Ou seja não requer a compra de licenças de utilização. O projecto *Octave* teve a sua origem nos finais da década de oitenta e a primeira versão foi disponibilizada nos inícios da década de noventa do século XX. É importante notar que o *Octave* interpreta comandos de *Matlab* mas possui também a sua própria semântica. Contudo essa particularidade é ignorada ao longo deste capítulo.

Dada a natureza comercial do *Matlab* é natural que esta ferramenta se encontre num nível muito acima do *Octave*. No entanto, em currículos introdutórios onde se necessite de uma ferramenta de computação numérica, o *Octave* pode ser utilizado, sem qualquer prejuízo, no lugar do *Matlab*. A unidade curricular de "Teoria da Informação e Transmissão Digital" enquadra-se nesse contexto. Por isso, e com excepção da última secção que requer obrigatoriamente o acesso ao *Matlab*, todos os exemplos documentados pode ser acompanhados com o *Octave*.

1.2 Operações elementares no Matlab

Nesta secção apresentam-se as operações básicas que podem ser executadas no *Matlab*. Começa-se por descrever a forma de criar variáveis no espaço de trabalho e como estas podem ser utilizadas como operandos em cálculos elementares.

1.2.1 Primeiras Operações

No fundo o *Matlab* foi desenvolvido para fazer operações algébricas. Neste sentido a primeira abordagem será a utilização desta ferramenta para a execução de cálculos simples. Por exemplo se se pretender determinar o resultado da operação de adição entre dois números, digamos 2 com 3, basta escrever na linha de comandos:

```
>> 2 + 3 [enter]
```

A expressão [enter] indica que a tecla *enter* do teclado deve ser pressionada e o símbolo >> refere-se ao *prompt* da linha de comandos no espaço de trabalho (*workspace*). O resultado desta operação é apresentado da seguinte forma:

```
ans = 5
```

Sem intenção explícita acabou-se de criar a primeira variável: a variável **ans**. Esta variável é concebida, de forma automática, e possui sempre o resultado da última operação executada (desde que não lhe seja atribuída outra variável). Por exemplo executando a seguinte instrução:

```
>>ans+3 [enter]
```

A resposta passa a ser:

```
ans = 8
```

Na operação anterior executou-se a adição do número 3 com a variável **ans** o que nos leva ao conhecimento da possibilidade de executar operações, não entre valores numéricos directo, mas entre valores numéricos encapsulados em variáveis.

Ao contrário da maior parte das linguagens de programação, onde é necessário definir uma variável antes de ser possível a sua utilização, no MATLAB esse passo é suprimido. A criação de variáveis é feita automaticamente sempre que um dado conteúdo, seja ele numérico ou não, é afectado a uma *string*[1]. Por exemplo se se desejar criar uma variável **A** com o valor 2 e uma variável **B** com o valor 3 basta executar a seguinte sequência de comandos:

```
>>A=2 [enter]
A = 2
```

```
>>B=3 [enter]
B = 3
```

A operação executada inicialmente pode agora ser conduzida da seguinte forma:

```
>>A+B [enter]
```

Note-se também que não foi definido explicitamente o tipo de dados que as variáveis **A** e **B** iriam receber. Por exemplo na linguagem C/C++ é necessário definir, *a priori*, se uma dada variável irá receber dados do tipo inteiro, caractere ou outros. De facto, esta é uma faculdade que permite simplificar bastante o uso do Matlab. Uma variável adapta-se sempre ao tipo de dado que vai receber. Assim, se na próxima instrução a variável A fosse armazenar um caractere, nenhuma alteração ao nível da variável deveria ser tomada. Esta iria adaptar-se automaticamente para absorver o novo tipo de dado. O que acabou de ser dito pode ser testado executando as seguintes linhas de código:

```
>>A=2.45 [enter]
A = 2.4500
```

```
>>A='a' [enter]
A = a
```

Observe o uso de pelica (') para armazenar o caractere **a** na variável **A**. O tipo de dado afecto a uma dada variável pode ser identificado executando o comando **whos**. Na linha que se segue mostra-se o resultado da execução da função **whos** sobre a variável **A**.

O tipo de dados pode ser identificado no campo **class**. Para além dessa informação apresenta-se também o espaço, em *bytes*, ocupado pela variável assim como a sua dimensão. Mais sobre este assunto adiante.

[1] A única restrição é a de que o primeiro caracter não pode ser um dos símbolos de 0 até 9.

```
>>whos A [enter]
*** local user variables:
Prot Name Size Bytes Class
==== ==== ==== ===== =====
rwd A 1x1 1 char

Total is 1 element using 1 byte
```

Para já, e antes de prosseguir, chama-se a atenção que o Matlab, tal como o C/C++, é sensível à utilização de maiúsculas e minúsculas. Assim, a variável **A** é distinta da variável **a** como pode ser testado da seguinte forma:

```
>>a=2.45 [enter]
a = 2.4500
```

```
>>whos A a [enter]
*** local user variables:
Prot Name Size Bytes Class
==== ==== ==== ===== =====
rwd A 1x1 1 char
rwd a 1x1 8 double

Total is 2 elements using 9 bytes
```

1.2.2 Eliminar variáveis do *workspace*

Uma variável criada no espaço de trabalho pode ser destruída recorrendo à função **clear**. Por exemplo, admitindo a existência de uma variável **A** o seguinte comando elimina-a:

```
>> clear A [enter]
```

Para apagar mais do que uma variável basta colocar o seus nomes, separados por espaço, a seguir à instrução clear. Por exemplo o seguinte comando elimina as variáveis **a** e **B**.

```
>> clear a B [enter]
```

Se se desejar limpar o espaço de trabalho de **todas** as variáveis basta escrever:

```
>> clear all
```

Este comando admite ainda o uso de *wildcards*. Por exemplo o comando,

elimina todas as variáveis no espaço de trabalho que começam pelo caractere **A** maíusculo.

```
>> clear A*
```

1.2.3 As instruções *Help* e *Lookfor*

Quando se trabalha com uma ferramenta como o Matlab temos na nossa mão um manancial de rotinas que executam as mais diversas operações sejam elas matemáticas, gráficas ou de outra natureza. Se bem que essas funções estejam, normalmente, compartimentadas em *toolboxes*, não deixa de ser uma tarefa complexa tentar encontrar uma função específica para uma determinada operação.

Para facilitar essa pesquisa o Matlab possui a palavra reservada **lookfor**. À frente dessa instrução coloca-se uma palavra-chave que seja descritiva daquilo que se pretende pesquisar. Por exemplo se se pretender encontrar quais as funções relacionadas com a palavra-chave "variable" basta escrever na linha de comandos.

```
>>lookfor variable [enter]
```

Por outro lado, quando se conhece a função mas se ignora a sua sintaxe é possível obter uma descrição dessa rotina recorrendo à instrução **help**. Esta instrução, executada na linha de comandos, leva como argumento o nome da função sobre a qual se quer conhecer o funcionamento. Por exemplo a seguinte linha de código,

```
>>help clear [enter]
```

mostra a formatação associada à utilização da função **clear**.

Para além de ajuda no modo texto o Matlab fornece também uma descrição mais detalhada no modo gráfico. Para tal basta activar o *hiperlink* no final do texto apresentado pela função **help**. Por exemplo na situação anterior o Matlab apresenta, no final, o link [doc clear].

1.2.4 Vectores

Admita-se agora que a variável **C** não é um escalar mas sim um vector (a verdade é que um escalar é um vector com dimensão 1x1). Mais concretamente desejamos introduzir o seguinte vector linha:

$$\begin{bmatrix} 3 & 5 & 8 & 2 \end{bmatrix}$$

Para isso basta proceder conforme se apresenta subsequentemente.

```
>>C=[3 5 8 2] [enter]
C =
3 5 8 2
```

A separação dos elementos é feita com espaços ou, alternativamente, com vírgulas. Note-se que a alocação do espaço é feita automaticamente, i.e. não é necessário definir antecipadamente a dimensão do vector. Executando agora,

```
>>whos C [enter]
*** local user variables:
Prot Name Size Bytes Class
==== ==== ==== ===== =====
rwd C 1x4 32 double

Total is 4 elements using 32 bytes
```

Observa-se que a dimensão da variável é agora 1x4. Em qualquer instante é possível aceder a um (ou mais) elementos do vector indicando, à frente da variável entre parênteses curvos, o índice do(s) elemento(s) a que se pretende aceder. Por exemplo se desejarmos aceder ao terceiro elemento do vector **C** basta escrever:

```
>>C(3) [enter]
ans = 8
```

Para aceder a mais do que um elemento simultaneamente, o argumento passa de um escalar a um vector, vector esse com os índices dos elementos a aceder. Por exemplo se pretendermos aceder ao primeiro e último valor do vector anterior basta executar:

```
>>C([1 4]) [enter]
ans =
3 2
```

Note-se que a indexação dos vectores em Matlab é feita a partir de 1 e não a partir de 0 como acontece em C/C++. Uma forma alternativa recorre à palavra reservada **end** para aceder ao último elemento do vector. Por exemplo,

```
>>C([1 end]) [enter]
ans =
3 2
```

Suponhamos agora que se pretende alterar o valor do segundo elemento do vector **C** de 5 para, por exemplo, 25. Para executar essa alteração basta aceder ao segundo elemento do vector e afectá-lo com o valor 25, i.e.

```
>>C(2)=25 [enter]
C =
3 25 8 2
```

O que acontece se se realizar a afectação C(5)=10? Como o vector inicial possui apenas quatro elementos, ao ser atribuído um valor ao quinto elemento o vector inicial será expandidos de modo a abarcar agora 5 elementos.

Se a afectação do valor 10 fosse feita, por exemplo, ao elemento 9, o Matlab encarrega-se de colocar, em todas as posições não inicializadas, o valor zero. Por exemplo:

```
>>C(5)=10 [enter]
C =
3 25 8 2 10
```

```
>>C(9)=10 [enter]
C =
3 25 8 2 10 0 0 0 10
```

Como se deve supor, é possível a alteração de mais do que um elemento simultaneamente. Nesse caso o vector e acedido, não por um escalar, mas por um vector. O exemplo que se segue ilustra como é possível alterar o valor dos índices 6, 7 e 8 do vector **C** para -1, 3 e 5.

```
>>C([6 7 8])=[-1 3 5] [enter]
C =
3 25 8 2 10 -1 3 5 10
```

Como eliminar um elemento de um vector? A resposta é muito simples: recorrendo à matriz vazio, denotada em Matlab, por [ ]. Assim, se se pretender eliminar o primeiro elemento do vector, basta executar...

```
>>C(1)=[] [enter]
C =
25 8 2 10 -1 3 5 10
```

O comprimento do vector passou de 9 elementos para 8. O primeiro elemento foi eliminado e todos os outros foram deslocados para a esquerda uma posição. A nova dimensão do vector pode ser confirmada recorrendo à função **length()**. Esta função leva como argumento um vector e retorna o número de elementos desse vector[2].

```
>>length(C) [enter]
ans = 8
```

Como inserir vectores coluna? De diversas formas. Uma delas consiste na introdução de uma matriz linha aplicando-lhe, de seguida, a operação de transposição. Por exemplo o vector:

$$\begin{bmatrix} 4 \\ 6 \\ -9 \\ 2 \end{bmatrix}$$

pode ser introduzido como:

```
>>D=[4 6 -9 2]' [enter]
```

[2] Também pode ser aplicado a matrizes. Nesse caso retorna o máximo da dimensão da matriz.

A *"plica"*, a seguir ao vector, denota a operação de transposição. Outra estratégia passa por separar os elementos do vector por ponto-e-vírgula. O exemplo que se segue ilustra isso mesmo.

```
>>D=[4;6;-9;2] [enter]
D =
4
6
-9
2
```

Ainda, e finalmente, os elementos podem ser inseridos individualmente segundo a forma ilustrada subsequentemente:

```
>>D=[4 [enter]
6 [enter]
-9 [enter]
2 ] [enter]
D =
4
6
-9
2
```

Nos vectores colunas aplica-se tudo o que foi dito para os vectores linha nomeadamente no que se refere à alteração, introdução e supressão de elementos.

1.2.5 Matrizes

Os vectores linha e coluna são casos particulares de matrizes em que uma das dimensões é exactamente 1. No caso genérico de uma matriz $m \times n$ esta pode ser introduzida segundo as regras enunciadas para os vectores. Por exemplo a matriz:

$$\begin{bmatrix} 16 & 2 & 3 \\ 5 & 11 & 10 \\ 9 & 7 & 6 \end{bmatrix}$$

Pode ser inserida como:

```
>>E=[16 2 3;5 11 10;9 7 6] [enter]
E =
16 2 3
5 11 10
9 7 6
```

ou
ou ainda

Em matrizes a identificação de um determinado elemento pode ser feito de duas formas distintas: fornecendo o índice da linha e da coluna ou, alternativamente, o número do elemento. Este último é definido, ao longo das colunas,

```
>>E=[[16;5;9] [2;11;7] [3;10;6]] [enter]
E =
16 2 3
5 11 10
9 7 6
```

```
>>E=[16 2 3 [enter]
5 11 10 [enter]
9 7 6] [enter]
E =
16 2 3
5 11 10
9 7 6
```

começando pelo primeiro elemento do topo da primeira coluna. Assim, se se pretender aceder ao elemento da terceira linha, segunda coluna pode-se escrever:

```
>>E(3,2) [enter]
ans = 7
```

onde o valor 3 representa o índice da linha e o 2 o índice da coluna ou, em alternativa,

```
>>E(6) [enter]
ans = 7
```

A dimensão de uma matriz pode ser questionada através da função **size()**. Esta função leva, como argumento, uma matriz e retorna o número de linhas e colunas dessa variável. Por exemplo:

```
>>size(E) [enter]
ans =
3 3
```

Em matrizes não faz sentido eliminar um elemento. No mínimo é necessário eliminar uma linha inteira ou uma coluna inteira. Assim, se fosse necessário eliminar a primeira coluna da matriz **E** a seguinte linha de código deveria ser escrita.

```
>>E([1 2 3],1)=[] [enter]
E =
2 3
11 10
7 6
```

Uma forma alternativa seria escrever:

```
>>E(:,1)=[] [enter]
```

Neste caso o '**:**' pode ser lido como "*todas as linhas*". Obviamente se o '**:**' ocupar o lugar de 1 deve ler-se "*todas as colunas*". Assim, para eliminar todos os elementos de uma matriz pode escrever-se:

```
>>E(:,:)=[] [enter]
E = []
```

ou, de forma mais compacta,

```
>>E=[] [enter]
E = []
```

Note-se que, apagar todos os elementos de uma matriz, não é sinónimo de apagar a variável. Como se pode deduzir do exemplo anterior, a matriz **E** continua a existir só que é vazia. Para uma variável ser eliminada do espaço de trabalho, independentemente do tipo, deve recorrer-se à função **clear()**.

1.3 Operações algébricas entre matrizes

No Matlab estão definidas todas as operações algébricas elementares entre matrizes. Para além da transposição, referida anteriormente, há ainda a destacar a adição, produto interno, inversa entre muitas outras. Nesta secção apresentam-se algumas das funções mais recorrentes.

Seja **A** uma matriz com dimensão $m \times n$ e **B** uma matriz com dimensão $n \times p$. O produto interno entre ambas as matrizes é uma terceira matriz, designemo-la por **C**, de dimensão $m \times p$ cujos elementos são obtidos pela soma do produto dos elementos de **A** (tomados coluna a coluna) pelos elementos de **B** (tomados linha a linha). No Matlab essa operação é executada por intermédio do operador '*****' conforme se mostra de seguida.

```
>>A=[1 2 3;4 5 6] [enter]
A =
1 2 3
4 5 6
```

```
>>B=[7 8 9]' [enter]
B =
7
8
9
```

É possível definir outro tipo de produto, que se designa normalmente por produto de Hadamard (ou Shur) e é efectuado elemento-a-elemento. Para essa

```
>>C=A*B [enter]
C =
50
122
```

operação ser executada é necessário que as matrizes (ou vectores) tenham a mesma dimensão. Admitindo as seguintes matrizes:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

O produto elemento-a-elemento é definido como:

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{23}b_{23} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{bmatrix}$$

No Matlab esta operação é conseguida colocando um ponto (.) a preceder o sinal de multiplicação. Mais genericamente, um ponto antes de qualquer operação atribui-lhe a competência de ser executada "*termo-a-termo*".

```
>>A=[1 2 3;4 5 6;7 8 9]; [enter]
>>B=[7 8 9;4 5 6;1 2 3]; [enter]
>>C=A.*B [enter]
C =
7 16 27
16 25 36
7 16 27
```

O leitor mais atento deve ter notado o aparecimento, à frente das duas primeiras linhas, do sinal de ponto-e-vírgula (;). Se comparar o resultado dessas duas operações com as do quadro da página anterior conclui que, utilizando essa pontuação elimina o eco, para a tela, do resultado da execução da operação ou função. Esta característica é muito útil em muitas circunstâncias tal como no caso de matrizes e vectores muito extensos.

Na tabela que se segue apresentam-se algumas das operações mais comuns que podem envolver matrizes:

1.4 Métodos para a criação de matrizes

A introdução dos elementos, um a um, pela linha de comandos apenas é razoável no caso de um número relativamente baixo de elementos. No entanto, no caso do número de elementos ser elevado e, simultaneamente, existir um padrão sistemático ou uma relação funcional entre eles, existem formas mais expeditas de introduzir esses vectores ou matrizes no espaço de trabalho.

Tabela 1.1: Operações com matrizes.

Operação		**Exemplo**

Inversão	**inv()**	`>>A=[1 2;3 4];`
		`>>inv(A)`
		`ans=`
		`-2.0000 1.0000`
		`1.5000 -0.5000`

Soma das colunas de uma matriz	**sum()**	`>>A=[1 2 3;4 5 6];`
		`>>sum(A)`
		`ans=`
		`5 7 9`

Produto das colunas de uma matriz	**prod()**	`>>A=[1 2 3;4 5 6];`
		`>>prod(A)`
		`ans=`
		`4 10 18`

Determinante de uma matriz	**det()**	`>>A=[1 2;3 4];`
		`>>det(A)`
		`ans = -2`

Valores próprios de uma matriz	**eig()**	`>>A=[1 2;3 4];`
		`>>eig(A)`
		`ans =`
		`-0.3723`
		`5.3723`

Independência linear (*rank*)	**rank()**	`>>A=[1 2;3 4];`
		`>>rank(A)`
		`ans = 2`

Matriz diagonal e diagonal de matrizes	**diag()**	`>>A=[1 2;3 4];`
		`>>diag(A)`
		`ans =`
		`1`
		`4`

Soma cumulativa	**cumsum()**	`>>A=[1 2 3;4 5 6];`
		`>>cumsum(A)`
		`ans=`
		`1 2 3`
		`5 7 9`

Traço de uma matriz	**trace()**	`>>A=[1 2;3 4];`
		`>>trace(A)`
		`ans = 5`

Por exemplo, suponha-se que se deseja criar um vector com os valores inteiros de 0 a 1000. Este pode ser formado facilmente seguindo a forma sintáctica subsequente:

```
>>A=0:1000; [enter]
```

A linha que acabou de ser escrita pode ser lida como "*cria um vector A com valores de 0* **até** *1000*". Como o espaçamento entre valores não está explicito, o interpretador admite um espaçamento unitário.

No entanto é possível definir, explicitamente, um espaçamento distinto, seja ele racional ou não. Os quadros que se seguem mostram, respectivamente, a criação de um vector com valores de 0 a 1000 com espaçamento de 0.1 e o mesmo vector mas agora formado com os elementos em ordem decrescente. Note-se que o passo está pontuado entre sinais (:).

Alternativamente, em vez de se especificar o passo é possível criar um vector conhecendo os limites superior e inferior e o número de elementos. Se o

```
>>A=0:0.1:1000; %ascendente [enter]
```

```
>>A=1000:-0.1:0; %descendente [enter]
```

espaçamento entre os elementos for linear recorre-se à função **linspace()**. No caso de a relação ser logarítmica existe a função **logspace()** para o mesmo efeito. O exemplo que se segue mostra a criação de um vector com 5 elementos com valores compreendidos entre 0 e 5 e, por um lado, com espaçamento uniforme e, por outro, com espaçamento logarítmico.

```
>>A=linspace(0,5,5) [enter]
A =
0.00000 1.25000 2.50000 3.75000 5.00000
```

```
>>A=logspace(0,5,5) [enter]
A =
1 1.7783e+001 3.1623e+002 5.6234e+003 1.0000e+005
```

Existem ainda outras funções capazes de gerar matrizes de uso comuns como é o caso da matriz identidade. A tabela que se segue resume algumas dessas funções.

Para finalizar apresenta-se na tabela **??** um quadro resumo, ilustrado com exemplos, de algumas das funções transcendentes em Matlab.

1.5 Gráficos

O Matlab possui um vasto conjunto de funções associadas ao esboço gráfico de funções. Neste documento, dado o seu objectivo, apenas se apresentam as mais básicas.

A função mais comum para o desenho de gráficos 2D é a função **plot()**. Esta função possui, no mínimo, um argumento: o vector (ou matriz) que se pretende desenhar. Neste caso o Matlab considera, como variável independente, os índices do vector. Por exemplo imagine que se pretende representar graficamente a função $y = x^2$ para valores de x entre $[0, 1]$. Para isso é necessário começar por criar o vector da variável independente.

```
>>x=linspace(0,1,100); [enter]
```

O número de elementos do vector está intimamente ligado à resolução com que se quer apresentar o gráfico da função. Contudo, um valor muito elevado, para além de não trazer nenhum benefício ao nível visual, obriga a um aumento excessivo da carga computacional tornando a função **plot()** muito lenta a ser executada. Por outro lado, um valor muito baixo pode levar à distorção da função. Em seguida cria-se o vector da variável dependente:

```
>>y=x.^2; [enter]
```

Tabela 1.2: Funções associadas à criação de algumas matrizes importantes.

Operação		**Exemplo**

Cria uma matriz de zeros com dimensão estabelecida no seu argumento.	**zeros()**	`>>A=zeros(2,4)`
		`A =`
		`0 0 0 0`
		`0 0 0 0`

Cria uma matriz de unidades com dimensão estabelecida no seu argumento.	**ones()**	`>>A=ones(1,3)`
		`A =`
		`1 1 1`

Cria a matriz identidade.	**eye()**	`>>A=eye(3,4)`
		`A =`
		`1 0 0 0`
		`0 1 0 0`
		`0 0 1 0`

Note a existência do ponto (.) antes da potenciação (^). Neste momento, tanto a variável dependente como a independente, existem no espaço de trabalho. O passo que se segue reside na apresentação do gráfico usando a seguinte formulação:

```
>>plot(x,y); [enter]
```

O resultado desta operação é apresentado, à frente, na figura 1.1. Como se pode reparar o gráfico aparece com alguma rudeza. Contudo é possível alterar muitas das suas características. É possível adicionar-lhe um título, etiquetas nos eixos, linhas de grade, marcadores, etc. De modo a ilustrar o que se acabou de dizer execute, sem fechar a janela do gráfico, os seguintes comandos:

```
>>title('y=x^2'); [enter]
```

Cria etiqueta no eixo das abcissas

```
>>xlabel('x'); [enter]
```

Cria etiqueta no eixo das ordenadas

```
>>ylabel('y'); [enter]
```

Observe que no Matlab uma *string* deve ser sempre limitada por plicas. Note-se ainda que a formatação do texto segue a norma sintáxica do LaTeX. Mais sobre este assunto adiante.

Voltando à sequência de comandos anterior, o resultado encontra-se ilustrado na figura 1.2.

Chama-se à atenção que no Matlab todas as características do gráfico podem ser alteradas directamente a partir da própria janela do gráfico. No entanto todas as propriedades de um objecto podem ser alteradas a partir de comandos específicos. Neste caso, essas alterações requerem os apontadores (*handles*) dos objectos (gráfico, etiquetas, etc) utilizando, posteriormente, as funções **set()set()** e **get()get()** para alterar as propriedades desses mesmos

Tabela 1.3: Código de 3 bit para uma mensagem com alfabeto de dimensão 8.

Operação		**Exemplo**

Funções trigonométricas	**sin()**
	cos()
	tan()
	cot()
		`>>sin([0 pi/2 pi])`
		`ans =`
		`0.00000 1.00000 0.00000`

Funções trigonométricas inversas	**asin()**
	acos()
	atan()
	acot()
		`>>atan(Inf)`
		`ans =`
		`1.5708`

Funções hiperbólicas	**sinh()**
	cosh()
	tanh()
	coth()
		`>>A=coth(eye)`
		`A =`
		`1.3130`

Funções hiperbólicas inversas	**asinh()**
	acosh()
	atanh()
	acoth()
		`>>acosh(0)`
		`ans =`
		`0 + 1.5708i`

Logaritmo natural	**log()**	`>>log(1)`
		`ans =`
		`0`

Logaritmo de base 10	**log10()**	`>>log10(10)`
		`ans =`
		`1`

Exponencial	**exp()**	`>>exp(1)`
		`ans =`
		`2.7183`

Raiz quadrada	**sqrt()**	`>>sqrt(2)`
		`ans =`
		`1.4142`

Factorial	**factorial()**	`>>factorial(10)`
		`ans =`
		`3628800`

Módulo	**abs()**	`>>abs(-1)`
		`ans =`
		`1`

Potenciação	^	`>>2^3`
		`ans =`
		`8`

```
>>hnd=plot(x,y); [enter]
>>set(hnd,'LineWidth',8); [enter]
```

objectos. Por exemplo se desejarmos aumentar a espessura da curva basta executar a seguinte sequência de comandos:
o resultado encontra-se ilustrado na figura 1.3.

O conjunto das propriedades, de um dado objecto, que admitem alteração podem ser acedidas através da função **get()**. Por exemplo:

1.5.1 Gráficos com múltiplos traçados

Por vezes existe a necessidade de se contrastarem dois gráficos simultaneamente. Em Matlab isso pode ser conseguido de, pelo menos, três formas: A primeira acontece naturalmente se a variável a representar for uma matriz em vez de um vector. A outra consiste em colocar, dentro da função **plot()**, os pares das

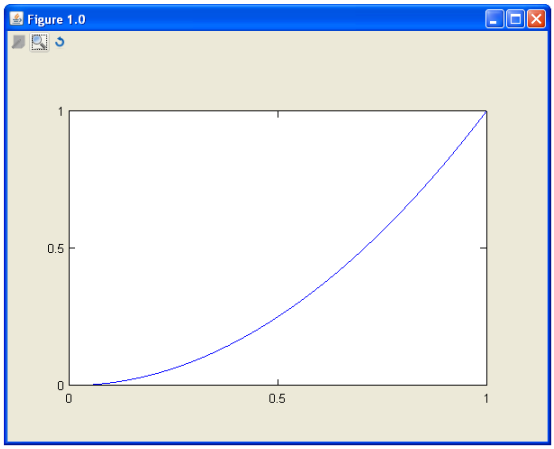


Figura 1.1: Representação gráfica da função $y = x^2$

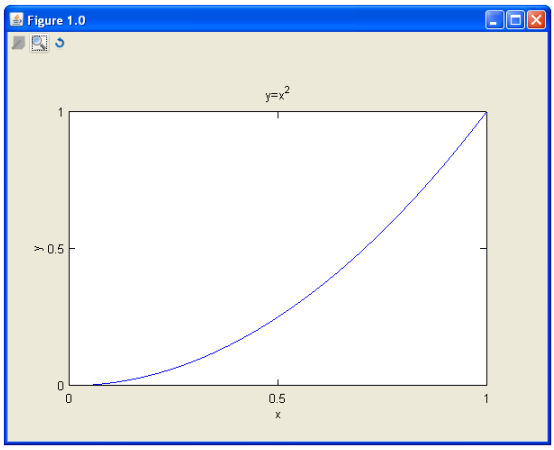


Figura 1.2: Gráfico de $y = x^2$ agora com etiquetas e título.

variáveis independente e dependente separadas por vírgulas. A terceira recorre ao uso da função **hold**. Comecemos por analisar esta última. Para isso imagine que se pretende traçar, no mesmo gráfico, as curvas associadas a duas funções distintas digamos,

$$y = x^2 \text{ e } z = x^3 \text{ com } x \in [0, 1] \tag{1.1}$$

Começa-se pela criação dos vectores associados às variáveis em questão:

Seguidamente ordena-se para traçar o gráfico associado à primeira função com:

Para desenhar o gráfico da segunda função na mesma figura é necessário "congelar" a janela de visualização, caso contrário o gráfico inicial desaparece para dar lugar ao novo. Para efectuar a retenção do primeiro gráfico executa-se a função:

Posteriormente já é possível executar o traçado da segunda função com:

Observe agora que a função **plot** possui três argumentos de entrada. Nesta formatação o último parâmetro refere-se a um atributo aplicado à curva traçada

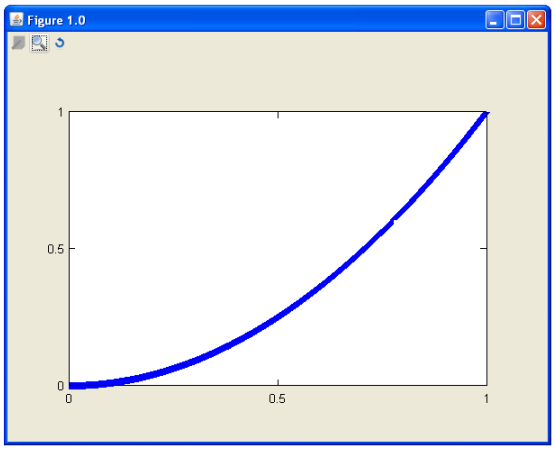


Figura 1.3: Gráfico de y=x2 com traço mais espesso.

```
>>get(hnd) [enter]
BeingDeleted = off
ButtonDownFcn =
Children = [ ]
Clipping = on
Color = [ 0,0000 0,0000 1,0000 ]
CreateFcn =
DeleteFcn =
HandleVisibility = on
KeyLabel =
LineStyle = -
LineWidth = 0,5000
Marker = none
MarkerEdgeColor = auto
MarkerFaceColor = none
MarkerSize = 6,0000
Parent = [ -41,3772 ]
Tag =
Type = line
UserData =
Visible = on
XData = [ (100 by 1) array of double ]
YData = [ (100 by 1) array of double ]
ZData = [ ]
```

```
>>x=linspace(0,1,100); [enter]
>>y=x.^2; [enter]
>>z=x.^3; [enter]
```

```
>>plot(x,y); [enter]
```

```
>>hold on; [enter]
```

```
>>plot(x,z,'r'); [enter]
```

(neste caso desenha o gráfico com a cor vermelho). Os tipos de atributos possíveis de aplicar à linha encontram-se documentados na tabela que se segue.

Tabela 1.4: Códigos para atributos de linhas em gráficos.

cores		**Marcadores**	
Magenta	**m**	Ponto	**.**
Ciano	**c**	Círculo	**O**
Vermelho	**r**	Marca x	**X**
Verde	**g**	Cruz	**+**
Azul	**b**	Estrela	*****
Amarelo	**y**	Quadrado	**S**
Preto	**k**	Losango	**D**
Tipo de Linha		Triângulo (baixo)	**V**
Sólida	**-**	Triângulo (cima)	**^**
Ponteada	**:**	Triângulo (esquerda)	**<**
Ponto/Traço	**-.**	Triângulo (direita)	**>**
Tracejado	**- -**	Pentagrama/Hexagrama	**p/h**

O resultado final possui o seguinte aspecto:

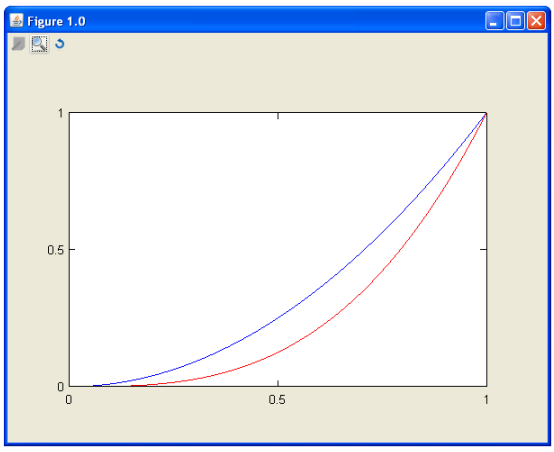


Figura 1.4: Gráfico de y=x2 com traço mais espesso.

Qualquer execução subsequente de funções **plot** irá acrescentar à janela já aberta uma nova curva. Por exemplo executando,

```
>>plot(x,y.*z,'k:'); [enter]
```

sem fechar a janela anterior tem como resultado, o aparecimento de um segundo traçado como se pode ver na figura 1.5. Para "descongelar" a janela basta escrever:

```
>>hold off; [enter]
```

A partir deste momento, qualquer novo traçado irá eliminar os existentes. Experimente executar o último comando de **plot**!

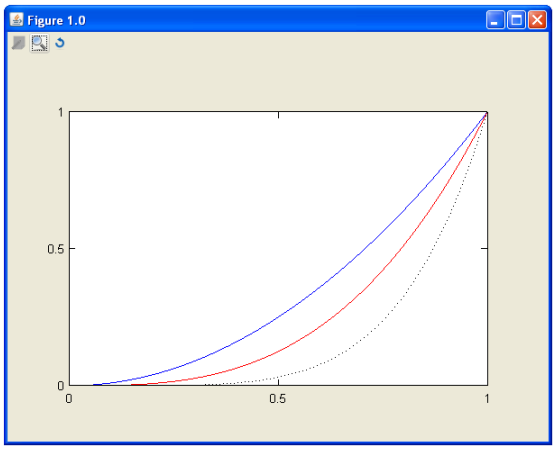


Figura 1.5: Gráfico de y=x2 com traço mais espesso.

Outra alternativa para o desenho de vários traçados sobre o mesmo gráfico passa pela utilização da função **plot** com tantos pares de argumentos quantos o número de gráficos a desenhar. Por exemplo a figura anterior também pode ser obtida por:

```
>>x=linspace(0,1,100); [enter]
>>y=x.^2; [enter]
>>z=x.^3; [enter]
>>w=y*z; [enter]
>>plot(x,y,x,z,x,w); [enter]
```

Note-se que agora a própria função **plot** distingue, automaticamente, cada um dos traçados com cores diferentes. O resultado desta execução encontra-se impressa na figura 1.6.

No entanto, se assim se desejar, é possível aplicar uma formatação particular para uma ou mais das curvas esboçadas. Por exemplo:

```
>>plot(x,y,'-',x,z,'pk',x,w); [enter]
```

cujo resultado se encontra visível na figura 1.7.

Para melhorar a legibilidade deste tipo de gráficos normalmente utiliza-se uma legenda onde se refere, através de um código de cores ou marcadores, qual o traçado correspondente a cada uma das funções.

No Matlab a função **legend()** executa essa operação. Assim, e sem fechar a janela do gráfico, escreva e execute a seguinte linha de código.

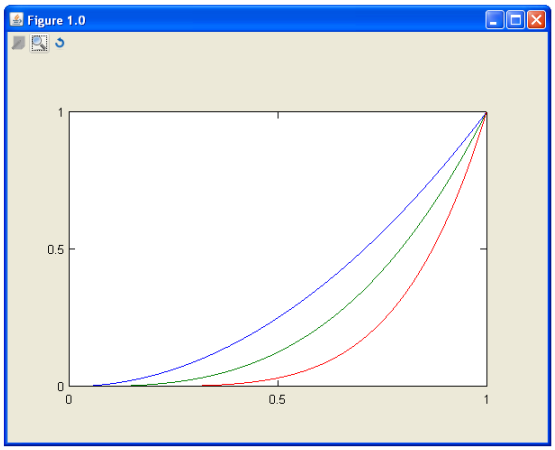


Figura 1.6: Gráfico de y=x2 com traço mais espesso.

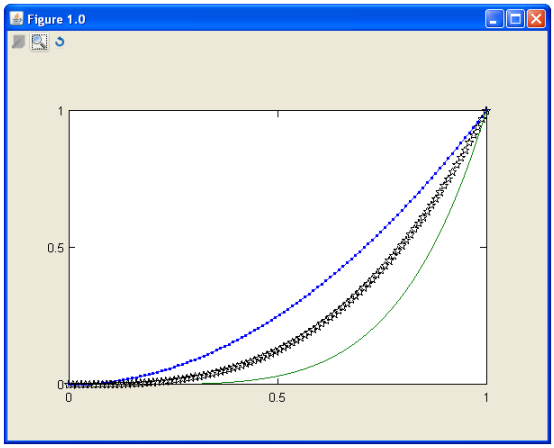


Figura 1.7: Gráfico de y=x2 com traço mais espesso.

```
>>legend('y=x^2','z=x^3','w=y*z'); [enter]
```

O gráfico anterior passa a ter, agora, o aspecto da figura 1.8.

A terceira, e última estratégia, para a apresentação de vários traçados simultaneamente segue do facto da função **plot** admitir matrizes como argumento. Neste caso, cada coluna da matriz é interpretada como uma função. Assim, o gráfico anterior poderia ser obtido pela execução do seguinte excerto,

```
>>plot(x,[y' z' (y.*z)']) [enter]
>>legend('y=x^2','z=x^3','w=y*z'); [enter]
```

As operações de transposição sobre os vectores da variável dependente devem-se ao facto desses vectores serem vectores linha e desejar-se que passem a vectores coluna. Observe ainda como podem ser concatenados vários vectores de modo a

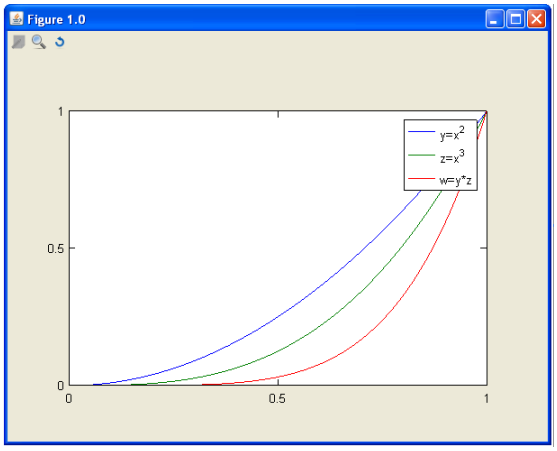


Figura 1.8: Gráfico com legenda.

formar uma matriz! (note que todos os vectores devem possuir o mesmo número de linhas)

1.5.2 Janela com múltiplos gráficos

Já se viu na secção anterior que existe a possibilidade de, no mesmo gráfico, coexistirem mais do que um traçado simultaneamente. Para além disso é possível definir em Matlab uma janela onde podem coexistir vários gráficos (que por sua vez podem admitir diversos traçados simultâneos). A função que permite executar este fraccionamento da figura em vários gráficos é designada por **subplot()**.

Para se perceber o modo de operação desta função admita que uma dada janela de visualização possa ser compreendida como uma matriz vazia. Se desejarmos desenhar dois gráficos a janela deve dividir-se em uma de duas formas conforme se ilustra na figura 1.9.

Neste contexto a divisão da esquerda pode ser interpretada como a estruturação da matriz vazia referida anteriormente, numa matriz com duas linhas e uma coluna. Por seu lado, a figura da direita pode ser entendida como uma matriz de gráficos com uma linha e duas colunas.

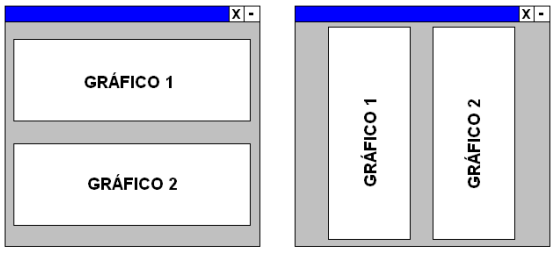


Figura 1.9: Estratégia de divisão da janela de visualização (I).

Este conceito pode generalizar-se para um número arbitrário de linhas e colunas conforme se pretende ilustrar com a figura subsequente.

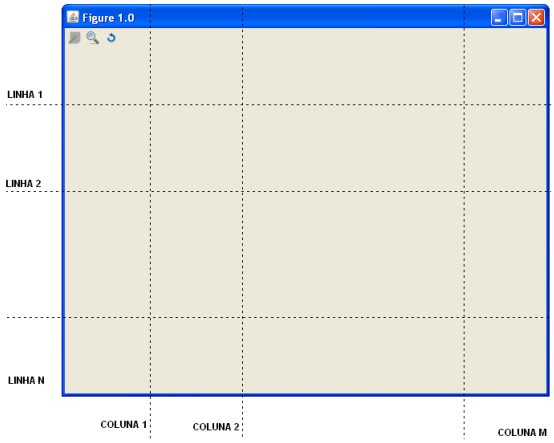


Figura 1.10: Estratégia de divisão da janela de visualização (II).

A função **subplot** deve ser executada com três argumentos de entrada. Os dois primeiros dizem respeito ao número de "linhas" e "colunas" com que se quer dividir a janela, i.e. definem o número de gráficos a desenhar e a sua organização. O terceiro representa um apontador que indica qual dos gráficos deve ficar activo. Qualquer instrução de **plot** executada subsequentemente irá traçar a curva no gráfico que estiver activo.

De modo a ilustrar o funcionamento desta função admita que se pretende traçar as funções:

$$\begin{aligned} y &= x * \sin(2\pi x), x \in [0,4] \\ z &= x * \cos(2\pi x), x \in [0,4] \end{aligned} \tag{1.2}$$

em dois gráficos diferentes dentro da mesma janela. Para isso basta executar a seguinte sequência de instruções:

```
>>x=linspace(0,4,100);y=x.*sin(2*pi*x);z=x.*cos(2*pi*x); [enter]
>>subplot(2,1,1);plot(x,y);title('func1'); [enter]
>>subplot(2,1,2);plot(x,z);title('func2'); [enter]
```

Observe que neste excerto existem vários comandos colocados na mesma linha. De facto é possível introduzir, na mesma linha, um número arbitrário de comandos devendo estes estar separados por vírgula ou, alternativamente, por ponto-e-vírgula.

A segunda linha começa por criar uma janela de visualização dividida, horizontalmente, em duas e activa o gráfico da linha superior. Seguidamente o traçado é esboçado recorrendo à função **plot**. A última linha activa o gráfico da linha inferior e, através do comando **plot**, desenha o gráfico de z contra x. O aspecto final deste conjunto de comandos encontra-se ilustrado na figura que se segue.

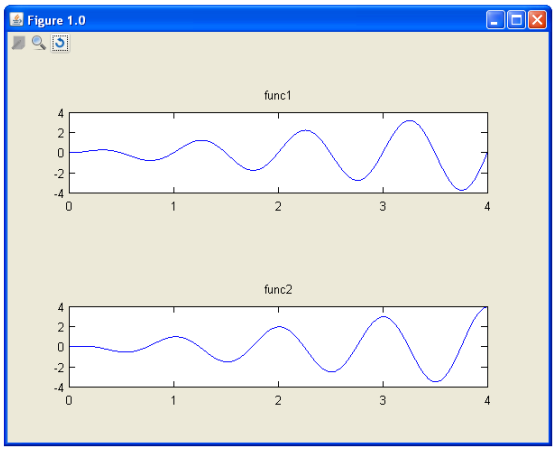


Figura 1.11: Dois gráficos sobre a mesma janela de visualização).

1.5.3 Gráficos 3D

No Matlab também é possível desenhar curvas em $\mathbb{R}^3$. O desenho de gráficos neste contexto não é muito diferente do que se viu anteriormente. A principal diferença reside no facto de agora existir uma terceira componente. Para absorver essa alteração o comando **plot** passa a ser **plot3**.

Por exemplo o desenho da seguinte função em 3D:

$$\begin{aligned} y &= x * \sin(2\pi x), x \in [0,4] \\ z &= x * \cos(2\pi x), x \in [0,4] \end{aligned} \tag{1.3}$$

Passa a ser codificada em Matlab por:

```
>> x=linspace(0,8,800); [enter]
>> y=x.*sin(2*pi*x); [enter]
>> z=x.*cos(2*pi*x); [enter]
>> plot3(x,y,z) [enter]
>> grid on [enter]
```

O aspecto final do gráfico encontra-se na figura que se segue. Observe ainda a função **grid()** na última linha de comandos. Esta função, quando activa, desenha linhas de grelha sobre a figura. Para inactivar esta característica basta escrever **grid off**.

Para terminar este tema, deixa-se aqui o reparo de que também é possível desenhar superfícies tridimensionais. Os exemplos que se seguem ilustram algumas, de entre muitas outras, possibilidades oferecidas.

```
>> [theta,psi]=meshgrid(-pi:pi/10:pi); [enter]
>> z=sin(theta); [enter]
>> surf(cos(theta).*cos(psi),cos(theta).*sin(psi),z); [enter]
>> colormap bone; [enter]
```

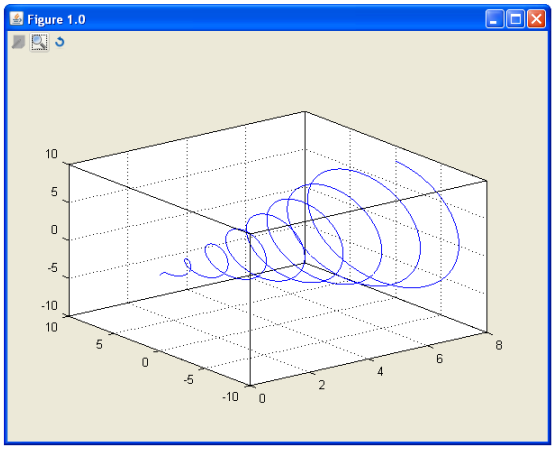


Figura 1.12: Gráfico com 3 componentes.

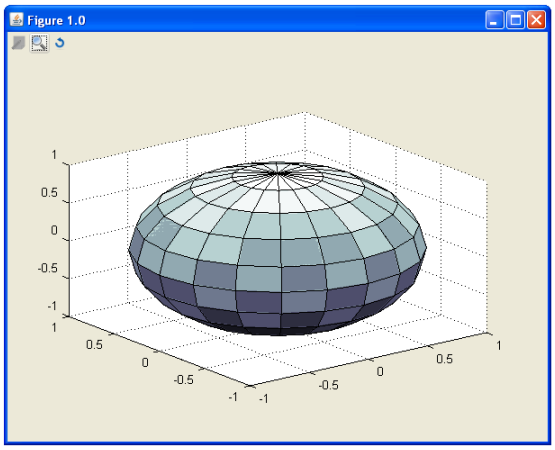


Figura 1.13: Desenho de uma superfície esférica.

```
>> [alfa,beta] = meshgrid(-10:0.25:10,-20:0.25:0); [enter]
>> gama = sinc(sqrt((alfa/pi).^2+(beta/pi).^2)); [enter]
>> mesh(alfa,beta,gama); [enter]
>> axis([-11 11 -21 1 -0.3 1.1]); [enter]
>> xlabel('\fontsize{18} \color{red}\bf \alpha'); [enter]
>> ylabel('\fontsize{18} \color[rgb]{.5 .5 .5}\bf\beta'); [enter]
>> zlabel('\fontsize{18}\bf\gama'); [enter]
>> hidden off [enter]
```

Ao encerrar esta secção tecem-se os últimos comentários aos excertos de código anteriores. A função **meshgrid** retorna todas as combinações possíveis entre as variáveis independentes dentro do domínio especificado no seu argumento. Note que a função pode devolver mais do que uma matriz. Sobre isso falaremos na secção que se segue.

A função **surf** serve para desenhar uma superfície e a função **mesh** serve

apenas para desenhar a malha da superfície. Cada uma delas leva, no mínimo, três argumentos (todos eles do tipo matricial) referentes aos três eixos ordenados.

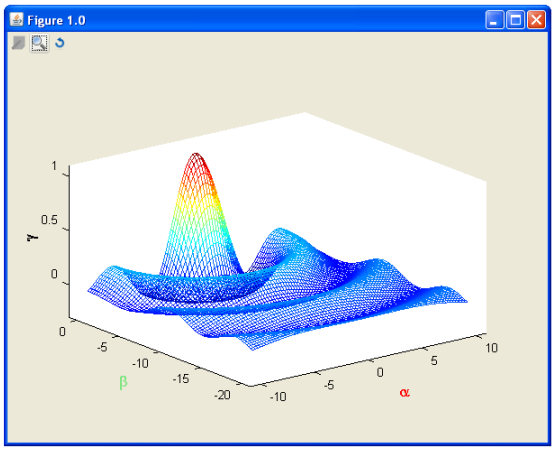


Figura 1.14: Desenho da malha de uma superfície suave..

A função **axis()** que aparece no código associado ao seno cardinal (**sinc()**) permite alterar os limites de cada um dos eixos. A sua sintaxe deve ser especificada da seguinte forma:

```
axis([xmin xmax ymin ymax zmin zmax])
```

Note que esta função também pode ser aplicada em gráficos 2D. Como se deve suspeitar, nesse caso, a componente z não é definida, i.e.

```
axis([xmin xmax ymin ymax])
```

Em parágrafos anteriores falou-se na eventualidade de formatar o texto recorrendo, para isso, a comandos do LaTeX. É exactamente isso que se fez quando se colocou as etiquetas sobre os eixos na segunda função 3D. Os comandos em LaTeXsão iniciados pelo caractere '\'. As tabelas que se seguem apresentam alguns comandos e símbolos que podem ser inseridos e executados nas etiquetas.

A função **colormap()** no primeiro exemplo atribui, à figura, uma determinada paleta de cores. As paletas admissíveis são, de entre muitas outras, **bone**, **copper**, **hot**, **cold** e **pink.**

Finalmente a função **hidden** possui dois argumentos possíveis, **on** ou **off**, e permite tornar, ou não, a superfície semi-transparente.

1.6 M-Files: *scripts* e funções

Se bem que esta secção trate simultaneamente de *scripts* e *funções*, ambos os conceitos, como veremos à frente, tem muito pouco a ver. No entanto encontram-se sobre a mesma secção dado que ambas estão associadas à execução de rotinas aninhadas em ficheiros.

Tabela 1.5: Propriedades associadas às *string*.

	Comando		**Comando**
Negrito	\bf	**Cor**	\color{nome_da_cor†}
Itálico	\it	Cor RGB	\color[rgb]{R G B‡}
Normal	\rm	Nome da fonte	\fontname{nome_da_fonte}
Oblíqua	\sl	Tamanho da fonte	\fontsize{tamanho_da_fonte}

† O nome da cor pode ser um de entre os seguintes: *red*, *green*, *yellow*, *magenta*, *blue*, *black* ou *white*.

‡ O valor associado a cada um dos componentes RGB é um número real entre **0** e **1**

Tabela 1.6: Letras e símbolos formato LaTeX.

Letras Gregas e Símbolos Matemáticos											
\alpha	α	\upsilon	υ	\sim	~	\rho	ρ	\forall	$\forall$	\partial	∂
\beta	β	\phi	φ	\leq	$=$	\sigma	σ	\exists	$\exists$	\bullet	$\bullet$
\gamma	γ	\chi	χ	\infty	8	\varsigma	ς	\ni	$\ni$	\div	$\div$
\delta	δ	\psi	ψ	\clubsuit	$\clubsuit$	\tau	τ	\cong	$\cong$	\neq	?
\epsilon	ε	\omega	ω	\diamondsuit	$\diamondsuit$	\equiv	$\equiv$	\approx	$\approx$	\aleph	à
\zeta	ζ	\Gamma	Γ	\heartsuit	$\heartsuit$	\Im	I	\Re	R	\wp	P
\eta	η	\Delta	Δ	\spadesuit	$\spadesuit$	\otimes	$\otimes$	\oplus	$\oplus$	\oslash	$\oslash$
\theta	θ	\Theta	Θ	\leftrightarrow	$\leftrightarrow$	\cap	$\cap$	\cup	$\cup$	\supseteq	$\supseteq$
\langle	$\langle$	\Lambda	Λ	\leftarrow	$\leftarrow$	\supset	$\supset$	\subseteq	$\subseteq$	\subset	$\subset$
\iota	ι	\Xi	Ξ	\uparrow	$\uparrow$	\int	?	\in	$\in$	\o	o
\kappa	κ	\Pi	Π	\rightarrow	$\rightarrow$	\rfloor	$\rfloor$	\lceil	$\lceil$	\nabla	∇
\lambda	λ	\Sigma	Σ	\downarrow	$\downarrow$	\lfloor		\cdot	$\cdot$	\ldots	$\ldots$
\mu	μ	\Upsilon	Υ	\circ	$\circ$	\perp	$\perp$	\neg	$\neg$	\prime	$\prime$
\nu	ν	\Phi	Φ	\pm	$\pm$	\wedge	$\wedge$	\times	x	\0	Ø
\xi	ξ	\Psi	Ψ	\geq	$=$	\rceil		\surd	v	\mid	$\mid$
\pi	π	\Omega	Ω	\propto	$\propto$	\vee	$\vee$	\varpi	ϖ	\copyright	©

1.6.1 Scripts

O MATLAB possui um histórico dos comandos que vão sendo executados. Esse histórico pode ser revisitado bastando, para isso, pressionar as teclas de *Up* e *Down* do teclado. De cada vez que essas teclas são pressionadas um determinado comando transacto toma o lugar no *prompt* da linha de comandos. Este conceito mostra-se muito útil no caso de se pretender alterar alguma das características de um comando já introduzido ou, tão simplesmente, se se desejar re-executar um dado comando.

No entanto, quando um dado comando depende do resultado de outros este método torna-se, na melhor das hipóteses, fastidioso. Para contornar este problema introduz-se agora o conceito de *script*.

Um *script* não é mais do que um conjunto de comandos armazenados num ficheiro ASCII. Para clarificar imagine o conjunto de comandos do exemplo anterior escritos, não na linha de comandos, mas num ficheiro aberto pelo programa *Notepad*§do *Windows*§como se ilustra na figura que se segue.

Seguidamente guarde esse ficheiro, com extensão **.m**, num directório (pasta) à sua escolha por exemplo guardar o ficheiro com nome *meu_script1.m* no directório *C:\matlab_files*.

Para executar o ficheiro recém-criado basta escrever o seu nome na linha de comandos do Matlab. No entanto é necessário ter o cuidado de verificar se o directório de trabalho [*Working Directory*] inclui o directório onde se encontra

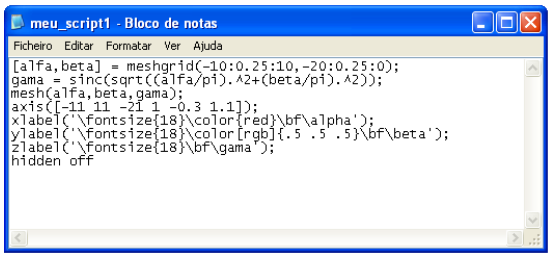


Figura 1.15: Comandos do MATLAB escritos na aplicação "bloco de notas" do Windows.

o ficheiro a executar. Caso contrário basta alterar o directório de trabalho na *combo-box* existente para o efeito.

Depois do directório de trabalho alterado escreva, na linha de comandos,

```
>> meu_script1 [enter]
```

Se todos os passos referidos anteriormente foram devidamente seguidos os comandos escritos no ficheiro serão executados sem erro.

Uma nota adicional sobre o editor de texto. Se bem que neste exemplo tenha sido utilizado o *Notepad*§não invalida que possa ser usado outro editor de texto não-formatado qualquer. Com efeito o próprio Matlab possui um editor de texto embebido. Para o executar basta escrever na linha de comandos:

```
>> edit [enter]
```

Frequentemente existe a necessidade de identificar o conjunto de instruções incluídas num ficheiro M [designado por *M-File*], i.e. qual o objectivo da sequência de comandos e/ou o resultado de cada instrução. Este facto apela para o uso de comentários no código escrito. Para o interpretador de comandos, o sinal % indica que, qualquer caracter ou conjunto de caracteres escritos posteriormente a esse sinal se destinam a comentários e serão ignorados. Um exemplo do que acabou de ser dito pode ser observado na figura subsequente.

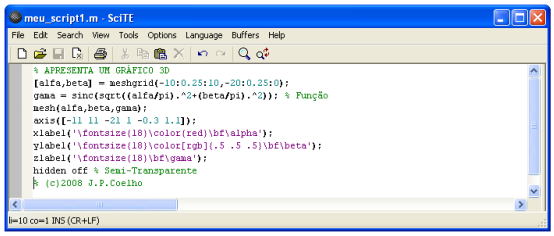


Figura 1.16: Comandos do MATLAB escritos na aplicação "bloco de notas" do Windows.

1.6.2 Funções

O número de funções embebidas no próprio núcleo do Matlab é bastante reduzido quando comparado com o número total de funções disponíveis. Isto porque é dada a possibilidade ao utilizador de criar as suas próprias funções utilizando, dentro destas, outras funções já existentes.

A criação de funções em Matlab é bastante simples bastando para isso respeitar uma sintaxe de base semelhante para todas as funções. O primeiro passo para a construção de uma função consiste em abrir um ficheiro de texto não-formatado (como aconteceu com o *script*). Para isso basta abrir um dos editores de texto do Matlab através, por exemplo, da instrução **edit** escrito na linha de comandos.

O que distingue um *script* de uma função é a palavra reservada **function** que deve ser escrita no inicio do arquivo. A seguir a **function** deve escrever-se o nome da função. Esse nome não pode conter espaços em branco nem caracteres especiais tais como o ponto-final, cedilhas etc.

Uma função pode ou não possuir argumentos de entrada e pode, ou não, possuir argumentos de saída. Se existirem argumentos de entrada estes devem ser colocados, separados por vírgulas, entre parêntesis curvos logo à frente do nome da função. Por outro lado, se existirem argumentos de saída, estes devem ser incluído, também separados por vírgulas, entre parêntesis rectos atrás do nome da função. Nesse caso o sinal de igual deve preceder o nome da função. As linhas que se seguem ilustram o cabeçalho para funções sem argumentos, só com argumentos de entrada, só com argumentos de saída e com argumentos, tanto de entrada como de saída.

```
function nome_da_funcao
function nome_da_funcao(in1,in2,...,inn)
function [out1,out2,...,outk]=nome_da_funcao
function [out1,out2,...,outk]=nome_da_funcao(in1,in2,...,inn)
```

Admita que se pretende criar uma função, a que chamaremos *soma_quadrado*, capaz de realizar a soma do quadrado das componentes de um vector, i.e.

$$Y = \sum_n X_n^2$$

Neste caso a função deverá possuir um parâmetro de entrada, o vector X, e um parâmetro de saída, o escalar Y. Assim sendo o cabeçalho da função terá o seguinte aspecto[3]:

```
function [Y]=soma_quadrado(X)
```

A rotina que a função deve executar será definida nas linhas que se seguem ao cabeçalho. Neste caso poderia ser, por exemplo:

Depois de escrever estas linhas no ficheiro guarde-o com o mesmo nome da função tendo o cuidado de lhe acrescentar a extensão ".m". A execução da função faz-se agora através da linha de comandos como se mostra a seguir.

[3]Efectivamente, quando existe apenas uma variável de saída os parênteses rectos podem ser eliminados.

```
X=X(:); % obriga a que X seja um vector coluna
Y=X'*X;
```

```
>> soma_quadrado([1 2 3 4 5])edit [enter]
ans = 55
```

Neste caso pretende determinar a soma do quadrado do vector linha [1 2 3 4 5]. O resultado desta operação não foi afectado a nenhuma variável por isso é destinada à variável **ans**. No entanto o resultado poderia ser canalizada para uma variável de ambiente como se mostra de seguida.

```
>> resultado=soma_quadrado([1 2 3 4 5])edit [enter]
resultado = 55
```

Edite novamente esta função e acrescente os seguintes comentários:

```
function [Y]=soma_quadrado(X)
% Calcula, e retorna, a soma quadrática de um vector.
% Utilização:
% Y=soma_quadrado(X)
% onde X é um vector e Y é um escalar

% (c)2008 J.P.Coelho

X=X(:); % obriga a que X seja um vector coluna
Y=X'*X;
```

Salve novamente o arquivo e execute na linha de comandos:

```
>> help soma_quadrado [enter]

Calcula, e retorna, a soma quadrática de um vector.
Utilização:
Y=soma_quadrado(X)
onde X é um vector e Y é um escalar
```

O que pode concluir?

O interpretador considera que qualquer comentário entre **function** e a primeira instrução, ou linha em branco, se refere a texto de apoio à rotina (*help*). Os restantes comentários são considerados documentação da rotina.

Uma última chamada de atenção antes de passar ao tema seguinte. Todas as variáveis criadas dentro de uma função são locais a essa função. Isto significa que duas funções distintas podem usar o mesmo nome para designar uma dada variável. Qualquer variável dentro de uma função é inacessível pelo espaço de trabalho. O mesmo não acontece com os *scripts* onde as variáveis utilizadas e criadas pertencem ao espaço de trabalho.

Um aparte também sobre a forma como uma função termina. Qualquer função termina após a execução da última instrução dentro do ficheiro. Contudo,

por vezes, é necessário terminá-la prematuramente (nos casos, por exemplo, em que um determinado erro surgiu). Nesses casos é possível utilizar a função **return**. Um exemplo sobre a sua utilização apresenta-se na secção que se segue.

A programação avançada em Matlab pode ainda ser feita de outras formas. Por exemplo é também possível escrever programas em C e FORTRAN capazes de interagir com o Matlab. No entanto o desenvolvimento destes tópicos está muito para além do escopo deste texto.

1.7 Instruções para controlo de fluxo

Como deve saber o seno cardinal de um argumento x, expresso como $\mathrm{sinc}(x)$, é por definição:

$$\mathrm{sinc}(x) = \begin{cases} \frac{\sin(x)}{x}, & x \neq 0 \\ 1, & x = 0 \end{cases}$$

Observe-se que a função está definida por dois ramos onde a decisão de qual utilizar depende da variável x. Como explicar isto ao Matlab? Viu-se até agora que o Matlab executa as instruções de forma sequencial. Contudo este caso exige que, em dado momento, a função se comporte de forma diferente.

Esta secção vem responder a este problema. Irá tratar-se de algumas das estratégias que existem para controlo da direcção do fluxo de um programa em Matlab. Essa reorientação pode ser radial, no sentido de que existem, em paralelo, vários caminhos que o programa pode seguir, ou então pode ser cíclica. Esta última no caso de se pretender executar o mesmo conjunto de operações um numero, pré-determinado ou não, de vezes.

1.7.1 Expressões Condicionais

O problema posto no início desta secção poderia ser resolvido por uma função com a seguinte arquitectura:

```
function Y=senocardinal(X)

if X~=0,
   Y=sin(X)./X;
else
   Y=1;
end
```

A escolha de um dos dois ramos é conseguida à custa da expressão **if**. A sintaxe associada a esta função encontra-se descrita subsequentemente:

```
if condição
...
elseif condição
...
else
...
end
```

A condição refere-se a uma expressão lógica que pode ser verdade ou falsa. Associada à definição da condição encontram-se as seguintes operações relacionais:

Tabela 1.7: Lista das condições lógicas em Matlab.

= =	Igual	<=	Menor ou igual
~=	Diferente	&&	Conjunção
>	Maior	||	Disjunção
<	Menor	~	Negação
>=	Maior ou igual		

Outra alternativa, em muitas situações, à expressão **if** é a expressão **switch**. Este comando alterna entre diversos casos dependendo da avaliação de uma dada expressão condicional. A forma sintáctica é a que se segue.

```
switch expressão
case condição1,
...
case {condição2,condição3, ...},
...
otherwise
...
end
```

Por exemplo a situação anterior poderia igualmente ser contornada da seguinte forma:

```
function Y=senocardinal(X)

switch X,
case 0, Y=1;
otherwise Y=sin(X)./X;
end
```

1.7.2 Ciclos

Não é uma tarefa singular, dentro de uma rotina de computação numérica, existirem partes do código que devem ser repetidos um número arbitrário de vezes. O MATLAB contempla essas situações com dois comandos distintos: o ciclo **for** e o ciclo **while**.

O primeiro é utilizado no caso de se conhecer *a priori* o número de iterações que o ciclo deve executar. Caso contrário recorre-se à expressão **while**.

Imagine-se, por exemplo, que se pretende uma função capaz de determinar a sequência de *Fibonacci* de ordem n. Esta sequência é calculada a partir da seguinte fórmula de recursão:

$$F_k = F_{k-1} + F_{k-2} \tag{1.4}$$

onde, inicialmente, $F_1 = F_2 = 1$ e $k \in \mathbb{N}$

Uma forma de dar resposta a esta questão consiste em executar a operação definida a partir de um ciclo conduzido n-2 vezes. A função que se segue representa uma sugestão para este problema:

```
function serie=fibonacci(ordem)

% Retorna a sequência de Fibonacci

if ordem<=0,
   disp('ERRO: A ordem deve ser > 0');
   return;
elseif ordem==1
   serie=1;
elseif ordem==2,
   serie=[1 1];
else
   serie(1:2)=1;
   for k=3:ordem,
      serie(k)=serie(k-1)+serie(k-2);
   end
end
```

O código acima descrito pode ser visto como formado por duas componentes em níveis hierárquicos distintos: uma de detecção de falhas e outra de execução da série propriamente dito.

Relativamente ao primeiro, é importante, e é também boa prática de programação, dotar qualquer código com mecanismos, nem que sejam básicos, para a detecção de erros. Neste caso apenas se prevê erros associados ao argumento de entrada. No entanto, em muitas situações, é necessário analisar possíveis violações ao longo do programa (por exemplo matrizes mal condicionadas, divisões por zero, etc.).

Voltando ao código, a cadeia de **if** testa se o argumento de entrada "ordem" é, ou não, inferior ou igual a zero. No caso afirmativo a sequência não pode ser calculada. O resultado é o aparecimento de uma mensagem de erro no espaço de trabalho seguido do fim da função. A mensagem de erro é gerada recorrendo à função **disp**. Esta função leva, como argumento, uma cadeia de caracteres (*string*) que apresenta no espaço de trabalho, i.e.

```
disp('string_a_colocar_no_espaço_de_trabalho')
```

o término da função neste caso é conseguido por meio da instrução **return**.

O núcleo da função criada encontra-se encabeçada pela instrução de ciclo **for**. Em termos sintácticos esta função segue o seguinte formato:

```
for var=v1:passo:v2
...
end
```

Inicialmente a variável `var` toma o valor de `v1`. Ao fim do primeiro ciclo é incrementada (decrementada) de uma quantidade igual a ***passo*** e passa ao segundo ciclo. O ciclo **for** é executado enquanto `var` for menor (ou maior) que `v2`.

Nos casos em que o número de iterações a ser executadas é desconhecido o comando **for** deve dar lugar ao comando **while** na implementação de ciclos. A instrução **while** deve ser usada de acordo com a seguinte estrutura:

```
while condição
...
end
```

Onde "*condição*" se refere a uma expressão lógica que é avaliada recorrentemente. Em cada iteração a condição lógica é testada. Enquanto o seu valor lógico for verdadeiro o ciclo é executado. Quando a condição passa a falso o ciclo é quebrado e a função executa a instrução seguinte. Note que é possível sair antecipadamente de um ciclo (tanto **while** como **for**) recorrendo à instrução **break**. Por exemplo observe o seguinte excerto de código.

```
X=0;
while X<100
   Y=X-80;
   if Y==0, break; end
   X=X+1;
end
```

Utilizando a instrução **while**, a função de *Fibonacci* pode tomar a seguinte forma:

```
function serie=fibonacci(ordem)
% Retorna a sequência de Fibonacci

if ordem<=0,
   disp('ERRO: A ordem deve ser > 0');
   return;
elseif ordem==1
   serie=1;
elseif ordem==2,
   serie=[1 1];
else
   serie(1:2)=1;
   k=3;
   while k<=ordem,
      serie(k)=serie(k-1)+...
      serie(k-2);
      k=k+1;
   end
end
```

Para além da utilização do comando **while**, esta versão da função de *Fibo-*

nacci possui uma pequena alteração na forma como a equação da série foi introduzida. Note a existência de reticências ($\cdots$) que permitem, dentro de uma função, *script* ou linha de comandos, partir uma função, operação ou equação em diversas linhas.

1.8 Leitura e escrita de dados em ficheiros

Em muitas situações existe a necessidade de importar dados de ficheiros para o Matlab para poderem ser manipulados. Adicionalmente pode também ser necessário guardar, num ficheiro, variáveis e resultados de operações executadas no Matlab. Devido a estes e outros motivos o Matlab possui funções capazes de realizar operações de manipulação de ficheiros. Algumas destas rotinas são exploradas no decorrer desta secção.

1.8.1 O par *fscanf* e *fprintf*

Para quem já tem alguma experiência de programação em C reconhece imediatamente as funções **fscanf** e **fprintf**. No entanto existem algumas pequenas diferenças dado que sobre o Matlab estas são vectorizadas.

A função **fscanf** permite importar, para uma variável, o contudo de um ficheiro sob uma determinada formatação. A sua forma mais simples é a seguinte:

```
conteúdo_do_ficheiro=fscanf(identificador,'formatação')
```

O identificador é um ponteiro para o ficheiro. Esse ponteiro é gerado a partir da função **fopen()**. A função **fopen** leva como argumentos o nome do ficheiro e o tipo de permissão relativamente ao acesso. A permissão pode ser, entre outras, de leitura (**r**) ou escrita (**w**). Em termos sintácticos a função **fopen** deve seguir a seguinte formulação:

```
identificador=fopen('nome_do_ficheiro','permissão')
```

As especificações de conversão, identificadas no campo de formatação, indicam o tipo de dados que devem ser lidos para a variável de destino. Estas especificações tomam a forma de uma *string* encabeçada pelo caractere '%'. Caracteres de conversão válidos são, entre outros, as letras **c** (ler sequencia de caracteres), **s** (ler sequencia de caracteres ignorando os espaços), **d** (ler valores decimais inteiros) e **f** (ler valores de virgula flutuante).

No fim do processo de leitura ou escrita sobre o ficheiro este deve ser fechado. Para isso existe a função antagónica ao **fopen()**: o **fclose()**. Esta instrução apenas requer o identificador do ficheiro devendo ser executada da seguinte forma:

```
fclose(identificador)
```

Em jeito de experiência passemos a importar o ficheiro readme.txt, que se encontra na raiz do directório do Matlab. Para isso executemos a seguinte sequência de comandos:

```
>> fp=fopen('README.txt','r'); [enter]
>> A=fscanf(fp,'%c'); [enter]
>> fclose(fp); [enter]
>> whos A [enter]
>> A [enter]
```

Sugestão: *compare o texto que aparece no ambiente de trabalho com o conteúdo do ficheiro abrindo-o com o* ***Notepad****.*

O processo de escrita num ficheiro é semelhante ao processo de leitura. Neste caso a função **fscanf()** é substituída pela função **fprintf()**. Esta função recebe, tal como a função **fscanf()**, o identificador do ficheiro e uma *string* de formatação. Para além disso necessita também do(s) vector(es) que se pretende(m) guardar no ficheiro. Segue, em baixo, a sintaxe para esta função:

```
fprintf(identificador,'formatação',variáveis)
```

Observe um exemplo da utilização dessa função no excerto que se segue:

```
>>x=0:0.1:10; y=x.^2; [enter]
>>fp=fopen('o_meu_ficheiro.txt','w'); [enter]
>>fprintf(fp,'%1.1f %1.2f\n',[x;y]); [enter]
>>fclose(fp); [enter]
>>edit o_meu_ficheiro.txt [enter]
```

Os valores entre '%' e 'f' indicam a forma como os valores serão guardados. No primeiro serão guardados com uma resolução de apenas uma casa decimal e no segundo com duas casas decimais. Observe ainda a existência do formatador '\n'. Este marcador indica que, após cada par de valores escritos, deve ser efectuada uma mudança de linha. Experimente correr novamente o exemplo anterior mas, agora, sem o '\n' no **fprintf**.

1.8.2 O par *fread* e *fwrite*

Um ficheiro binário possui a informação representada da forma mais compacta. Caracteres num ficheiro binário são registados consecutivamente sem espaços de separação.

Normalmente este formato é destinado à leitura por máquinas e não por indivíduos. Por isso mesmo o programa de computador que vai ler esse ficheiro deve saber o formato com que os dados foram registados no ficheiro original.

O par **fscanf()** e **fprintf()** apresentado na secção anterior é especialmente destinado à leitura/escrita no modo ASCII. O Matlab consegue lidar com ficheiros binários através das funções **fread()** e **fwrite()**.

Em termos de forma de operação estas funções assemelham-se às anteriores: É necessário criar um ponteiro para o ficheiro a ler ou escrever, executar a operação de leitura/escrita e finalmente fechar o ficheiro.

Segue um exemplo para comparar as funções **fread()** e **fwrite()** com as funções **fscanf()** e **fprintf()**. Começa-se por gerar um vector com 100 elementos. Por exemplo executando:

```
>> A=randn(100,1); [enter]
```

Armazenam-se agora os valores do vector `A` em dois ficheiros distintos: Um ficheiro ASCII e um binário denominados `exemplo.txt` e `exemplo.bin` respectivamente. A sequência de comandos apresentada em baixo ilustra o procedimento utilizado.

```
>> fp=fopen('exemplo.txt','w'); [enter]
>> fprintf(fp,'%f',A); [enter]
>> fclose(fp); [enter]
>> fp=fopen('exemplo.bin','w'); [enter]
>> fwrite(fp,A,'double'); [enter]
>> fclose(fp); [enter]
```

O resultado foi a criação dos dois ficheiros. O primeiro utiliza 855 bytes contra os 800 bytes[4] do ficheiro `exemplo.bin`.

O procedimento para a leitura é semelhante e encontra-se documentada nos comandos que se seguem:

```
>> fp=fopen('exemplo.txt','r'); [enter]
>> B=fscanf(fp,'%f'); [enter]
>> fclose(fp); [enter]
>> fp=fopen('exemplo.bin','r'); [enter]
>> C=fread(fp,'double'); [enter]
>> fclose(fp); [enter]
```

Existe um manancial de formatos para dados binários que o Matlab pode ler e escrever. Entre eles destacam-se os seguintes: A escolha do formato correcto para a leitura de um determinado ficheiro é muito importante. É necessário o conhecimento *a priori* do formato utilizado para salvar os dados.

Se os dados guardados em ficheiro são para ser utilizados apenas pelo Matlab a utilização do par **load()** e **save()** simplificam grandemente a tarefa. Na secção que se segue apresenta-se o modo de operação dessas funções.

1.8.3 O par *load* e *save*

Quando se termina a sessão do Matlab (escrevendo, por exemplo, a instrução **exit** na linha de comandos), todas as variáveis criadas no espaço de trabalho perdem-se para sempre. Uma forma de as salvaguardar consiste em envia-las para um ficheiro em disco[5]. Esta operação pode ser feita facilmente recorrendo à função **save()**. Por exemplo, para guardar as variáveis `x`, `y` e `z` existentes no espaço de trabalho basta escrever:

```
 >> save nome_do_ficheiro x y z
```

[4] 800 bytes = 100 elementos × 64 bit por elemento / 8 bit

[5] O comando **diary()** pode ser utilizado para guardar a sequência de comandos de uma sessão.

Tabela 1.8: Lista do formatos aceites pelo par fwrite/fread.

'uchar'	caracter sem sinal	8 bits
'schar'	caracter com sinal	8 bits
'int8'	inteiro	8 bits
'int16'	inteiro	16 bits
'int32'	inteiro	32 bits
'uint8'	inteiro sem sinal	8 bits
'uint16'	inteiro sem sinal	16 bits
'single'	vírgula flutuante	32 bits
'double'	vírgula flutuante	64 bits

Por defeito o Matlab utiliza um formato próprio para o ficheiro guardado (MAT file). Note que a função **save()** admite ainda outros argumentos (*switches*) que permitem forçar o ficheiro a obedecer a certas especificações. O leitor mais curioso pode escrever **help save** na linha de comandos e investigar todas as potencialidades desta função.

As variáveis guardadas desta forma podem ser carregadas para o espaço de trabalho recorrendo à função **load()**. Na sua forma mais simples a função **load()** requer apenas o nome do ficheiro onde se encontram as variáveis. Assim, para carregar todas as variáveis para o espaço de trabalho, basta escrever:

```
>> load nome_do_ficheiro
```

Tome sempre atenção se o ficheiro a carregar está, ou não, no directório de trabalho. Caso contrário mude o directório de trabalho ou inclua o caminho (*path*) antes do nome do ficheiro.

1.9 Tipos avançados de dados

Até ao momento apenas se apresentaram tipos de dados matriciais. Nestes todos os elementos devem pertencer à mesma classe. Por exemplo todos os elementos de uma matriz devem ser *double* ou *char*. No entanto o Matlab permite a utilização de estratégias mais avançadas para a manipulação e organização dos dados. Fala-se, de entre outras, das estruturas de dados e dos vectores de células (*cell arrays*).

1.9.1 Estuturas

Uma estrutura é um tipo de dado onde são definidos campos (fields). Cada campo pode conter um tipo de dado diferente. Um exemplo comum dado é o da ficha de um indivíduo numa determinada base de dados. Por exemplo a base de dados de uma biblioteca possui um conjunto de fichas em que cada uma das fichas é atribuída a um utilizador. Toda a informação relevante do utilizador deve constar nessa ficha. Por exemplo o nome, morada, telefone, endereço electrónico, número de sócio assim como as referências e datas de requisição dos livros actualmente em poder do utilizador. Cada um destes elementos é um campo da ficha e cada um desses campos possui diferentes tipos de dados. Por exemplo o campo nome é do tipo *char* e o número de telefone pode ser numérico.

No Matlab a ficha de um indivíduo pode ser construída como um estrutura. Por exemplo para criar uma variável chamada *ficha* com cinco campos, nome, morada, telefone, numero de sócio e livros requisitados, o seguinte sintaxe pode ser utilizado:

```
>> ficha.nome='João Paulo Coelho'; [enter]
>> ficha.morada='IPB'; [enter]
>> ficha.telefone=259000111; [enter]
>> ficha.socio=9567; [enter]
>> ficha.livros=['GR45';'PP10';'WE61'] [enter];
```

Para se observar o registo basta escrever o nome da variável (neste caso *ficha*). Procedendo desta forma obtém-se o seguinte resultado: Para aceder aos

```
>> ficha [enter]
ficha =
nome:  'João Paulo Coelho'
morada:  'IPB'
telefone:  259000111
socio:  9567
livros:  [3x4 char]
```

valores concretos apenas de um campo basta acrescentar, à frente do nome da estrutura, o campo que se pretende observar. Por exemplo:

```
>> ficha.nome [enter]
ans =
João Paulo Coelho
```

Para eliminar o campo de uma estrutura recorre-se à função **rmfiled()**. Esta função leva como argumento o nome da estrutura e os campos a eliminar. Por exemplo para eliminar os campos *morada* e *livros* da estrutura anterior basta escrever:

```
>> ficha=rmfield(ficha,['morada';'livros']) [enter]
ficha =
nome:  'João Paulo Coelho'
telefone:  259000111
socio:  9567
```

Os campos de uma estrutura podem ser organizados alfabeticamente utilizando a função **orderfields()**. Aplicando esta função à estrutura obtém-se:

```
>> ficha=orderfields(ficha) [enter]
ficha =
nome:  'João Paulo Coelho'
socio:  9567
telefone:  259000111
```

Uma forma alternativa para criar uma estrutura envolve a utilização da função **struct()**. O exemplo que se segue apresenta a forma como esta função poderia ser utilizada para criar uma segunda ficha com apenas três campos:

```
ficha2 = struct('nome','João','morada','IPB','telefone',123456789)
```

Note-se que este tipo de dados se comporta como outro qualquer. Ou seja é possível definir vectores ou matrizes ou mesmo estruturas de estruturas. Por exemplo:

```
>> ficha(1) = struct('nome','João','morada','IPB','tel.',1234);
>> ficha(2) = struct('nome','Maria','morada','ESTiG','tel.',1111);
>> ficha(3) = struct('nome','Pedro','morada','ESEB','tel.',9999);
```

```
ficha =
1x3 struct array with fields:
nome
morada
tel.
```

```
%Agora o exemplo de uma estrutura de estruturas:
ficha1=struct('nome', 'João', 'morada', 'IPB',...
'telefone',123456789); [enter]
ficha2 = struct('nome', 'Maria', 'morada', 'ESTiG',...
'telefone',111111111); [enter]
ficha.cliente(1)=ficha1;[enter]
ficha.cliente(2)=ficha2;[enter]
```

1.9.2 Vectores de Células

Quando se pretende criar uma matriz de strings é necessário garantir que cada linha tenha o mesmo número de caracteres. Por exemplo a seguinte linha de código cria uma matriz com quatro nomes distintos:

```
>>nomes=['Joao ';'Arminda';'Joaquim';'Lio '] [enter]
```

Ao primeiro registo da matriz nomes foi adicionado 3 espaços e ao último 4 espaços. O Matlab considera o espaço como um caractere e a utilização de espaços serve exactamente para garantir que a matriz nomes possui o mesmo numero de colunas. No entanto, em vez de um vector de string poderia estabelecer-se uma estrutura diferente: uma célula de string. Neste caso não é necessário definir o mesmo número de caracteres. Seguem os exemplos:

```
>>nomes={ 'Joao','Arminda','Joaquim','Lio' } [enter]
nomes = 'Joao' 'Arminda' 'Joaquim' 'Lio'
```

```
>>nomes={ 'Joao';'Arminda';'Joaquim';'Lio' } [enter]
nomes =
'Joao'
'Arminda'
'Joaquim'
'Lio'
```

Ao contrário dos parêntices curvos, agora a utilização de chaveta ({}) define a variável como célula. Outra alternativa envolve a utilização da função **cell()**. Para aceder ao primeiro elemento da célula basta escrever:

```
>>nomes{1} [enter]
ans = 'Joao'
```

Tal como as estruturas, as células são capazes de lidar com diferentes tipos de dados simultaneamente. Mais concretamente, um dado do tipo célula pode ser visto como uma matriz capaz de conter diversos formatos de dados simultaneamente. Por exemplo:

```
celula{1,1}=rand(1,5); [enter]
celula{1,2}='TITD'; [enter]
celula{2,1}=eye(3,3); [enter]
```

Um sintaxe alternativo, mas que leva ao mesmo resultado, apresenta-se em baixo:

```
celula(1,1)={rand(1,5)}; [enter]
celula(1,2)={'TITD'}; [enter]
celula(2,1)={eye(3,3)}; [enter]
```

Se `celula` é um dado do tipo *cell array* conforme definido na caixa anterior, o Matlab permite dois tipos de acesso à célula: com parêntice curvos e com chaveta. Ou seja tanto `celula{1,1}` como `celula(1,1)` são indexações válidas. No entanto produzem resultados diferentes. Veja-se o seguinte exemplo:

```
>> B=celula{1,1} [enter]
B =
7.5774e-001 7.4313e-001 3.9223e-001 6.5548e-001 1.7119e-001
```

e

```
>> A=celula(1,1) [enter]
A = [1x5 double]
>> A{1} [enter]
ans =
7.5774e-001 7.4313e-001 3.9223e-001 6.5548e-001 1.7119e-001
```

O primeiro caso retorna o conteúdo da célula e o segundo retorna a sub-célula dessa mesma célula.

Um elemento particular de um vector, ou matriz, dentro de um *cell array* pode ser acedido facilmente. Por exemplo para aceder ao quarto elemento do vector associado à variável `celula` escreve-se:

```
>> celula{1,1}(1,4) [enter]
ans = 6.5548e-001
```

Para finalizar, a estrutura de um *cell array* pode ser observado, graficamente, utilizando a função **cellplot()**. Por exemplo a instrução `cellplot(celula)` tem como resultado a seguinte figura:

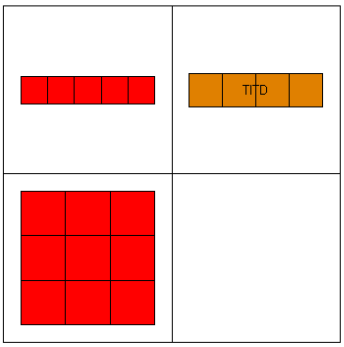


Figura 1.17: Resultado da função `cellplot()`.

1.10 Interfaces gráficas e o ambiente GUIDE

Nesta última secção apresenta-se o ambiente GUI[6] do Matlab. Este ambiente permite o desenvolvimento de plataformas mais intuitivas de interligação entre um programa desenvolvido para Matlab e o utilizador humano.

Para tornar mais transparente a forma como este tipo de interfaces são desenvolvidas apresenta-se a construção, passo-a-passo, da um programa que simula a robustez de um sistemas de codificação de canal R3 sobre um canal binário simétrico. O aspecto final que se propões encontra-se ilustrado[7] na figura 1.18.

Existe um emissor e um receptor. O emissor envia símbolos de 1 bit por um canal (binário simétrico) com uma determinada probabilidade de erro. Essa probabilidade pode ser seleccionada, de entre um conjunto pré-determinado de valores, a partir de um *menu* do tipo *pop-up*. À mensagem a transmitir é adicionada redundância e quando o utilizador pressa o botão *Transmitir...*, a mensagem passa pelo canal e chega ao descodificador que, por sua vez, trata de informar o receptor sobre o teor da mensagem enviada.

[6] GUI é o acrónimo de **G**raphical **U**ser **I**nterface

[7] Um agradecimento especial ao João e Rita Coelho pela permissão na utilização dos desenhos...

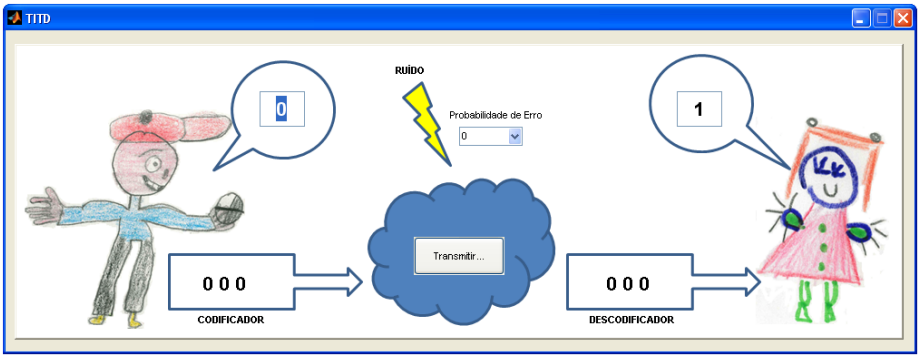


Figura 1.18: Aspecto da interface gráfica proposta para um programa que simula o comportamento da codificação de canal no aumento da fiabilidade da transmissão.

Como já se referiu, no decorrer das páginas que se seguem apresenta-se todos os passos necessários para realizar este pequeno programa. Para começar executa-se o ambiente GUI no Matlab escrevendo na linha de comandos:

```
>> guide
```

A janela de diálogo, apresentada na figura 1.19, é exibida ao utilizador.

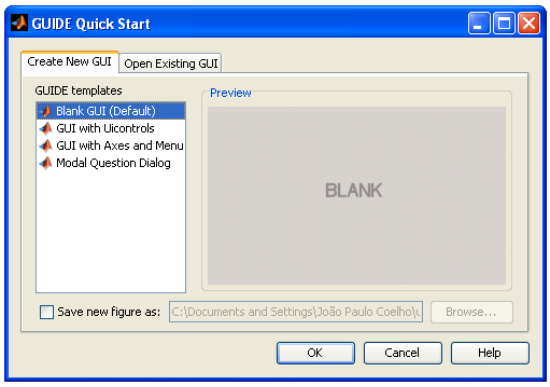


Figura 1.19: Janela de introdução após a execução do comando guide.

Para além do padrão de interface que aparece por defeito, é possível escolher alguns modelos que se apresentam pré-preenchidos. O programa que será desenvolvido ao longo desta secção segue o modelo `Blank GUI` que leva ao *front-end* principal do GUIDE cujo aspecto se apresenta na figura 1.20.

Na janela principal do programa distinguem-se duas zonas: à esquerda um conjunto de blocos com as ferramentas e objectos disponíveis e à direita a zona de trabalho onde os objectos poderão ser distribuídos.

Cada objecto possui uma lista de propriedades que podem ser alteradas a partir do `Poperty Inspector`. Para activar o `property inspector`, relativo a

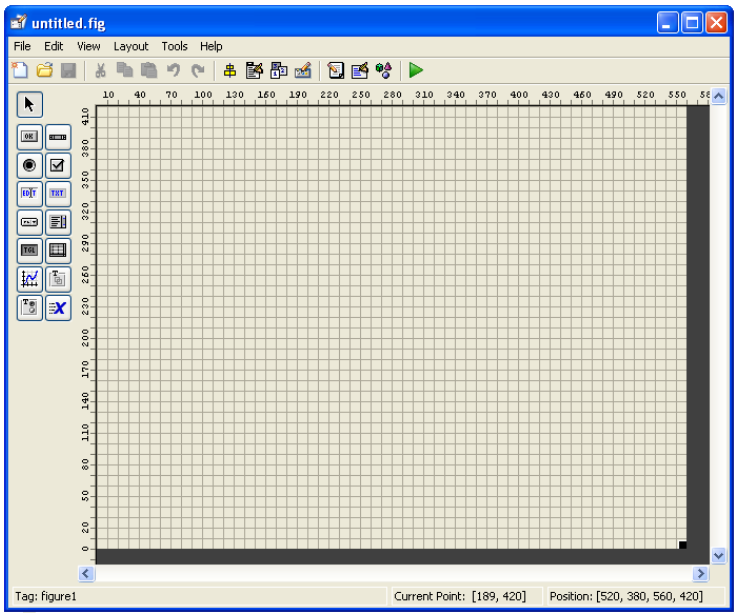


Figura 1.20: Resultado da função `cellplot()`.

um objecto colocado no espaço de trabalho, basta fazer duplo *clique* em cima desse objecto. Ou então premir o botão do lado direito do *rato* e seleccionar, no menu que aparece, o item `Property Inspector`. A figura 1.21 apresenta o procedimento para aceder às propriedades do modelo da figura.

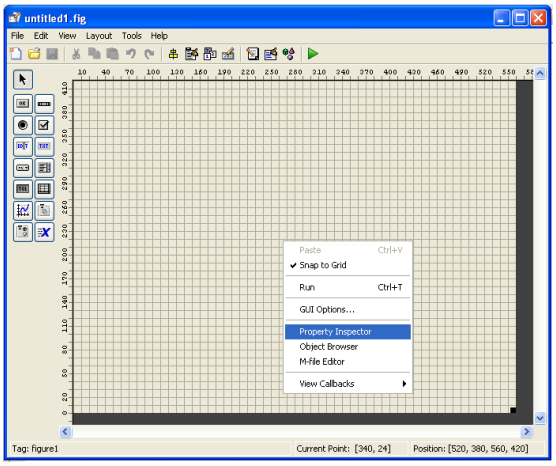


Figura 1.21: *Menu* para acesso ao `property inspector`.

Na figura 1.22 apresenta-se um excerto das opções, de uma lista substancialmente mais extensa, que podem ser alteradas no `Property Inspector`.

O programa para Matlab que esta secção se propõe apresentar pode ser dividido em duas partes. Uma refere-se ao *front-end*, ou seja à forma como

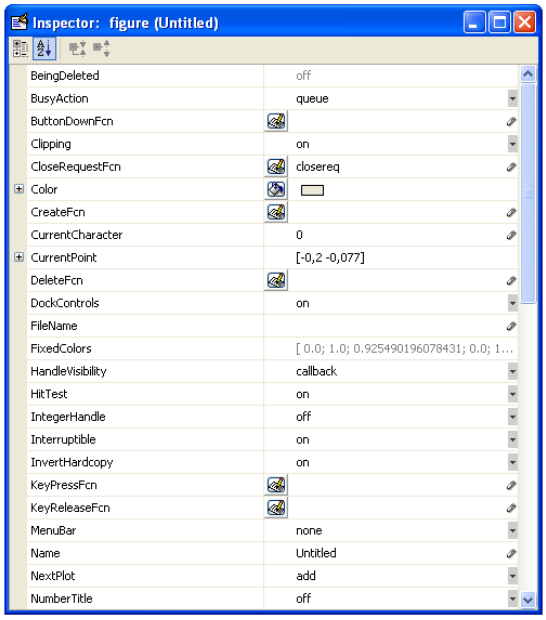


Figura 1.22: Lista de algumas das opções que podem ser alteradas no `Property Inspector`.

a informação entre o programa e o utilizador comunicam, e o *back-end* que envolve o modo de operação do programa. Ou seja como o *front-end* responde às solicitações do utilizador.

Começa-se por definir o *front-end* e depois apresenta-se o código associado ao *back-end*. Enceta-se esta descrição pela definição das propriedades associadas à janela principal. Para isso, depois do `property inspector` activo, alteram-se os campos apresentados na figura 1.23.

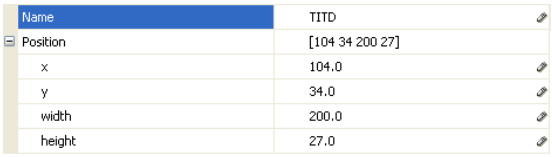


Figura 1.23: Definição das propriedades associadas à janela do programa TITD.

A alteração destas propriedades leva a uma redefinição das dimensões da área de trabalho. Observando a figura 1.18 verifica-se a existência de uma figura de fundo. Assim, depois de ter re-dimensionado a janela do programa passa-se à integração da imagem de fundo. Para isso é necessário carregar a imagem[8] a partir do ficheiro utilizando o seguinte comando executado no *workspace*[9]:

```
>> A=imread('GUIDE.tif');
```

[8] Tenha atenção de que a figura GUIDE.TIF deve estar no directório de trabalho
[9] Mantenha o GUIDE activo!

Agora no GUIDE adiciona-se um objecto do tipo `push button`. A figura 1.24 ilustra o procedimento. Para isso basta o objecto a partir das caixas à esquerda do espaço de trabalho e, utilizando o *rato* mantendo o botão esquerdo premido, descrever no ambiente a zona destinado a esse objecto.

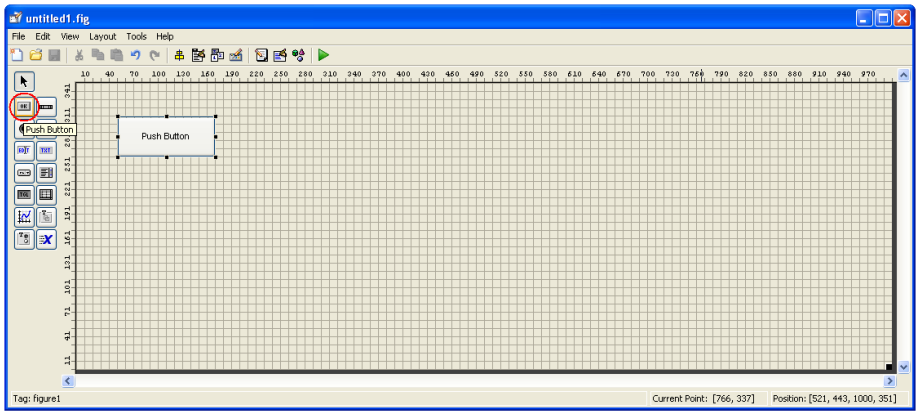


Figura 1.24: Resultado da função `cellplot()`.

Abra o `property inspector` relativo a esse novo objecto criado é altere os campos ilustrados na figura 1.25.

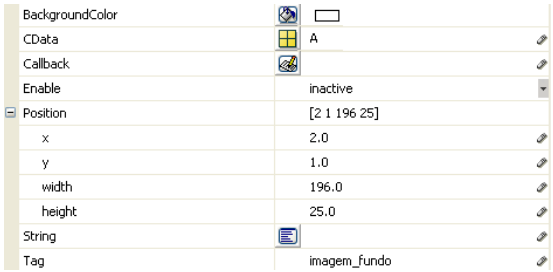


Figura 1.25: Propriedades a alterar no primeiro objecto criado.

O aspecto que irá obter, após as alterações introduzidas, encontra-se ilustrado na figura 1.26. Aproveita-se para acrescentar mais um objecto `push button` que será o botão `Transmitir` ilustrado na figura 1.18. Para que esse objecto passe a ter as características pretendidas, abra a lista de propriedades desse objecto e proceda às alterações apresentadas na figura 1.27.

Após a definição das propriedades desse objecto a interface gráfica apresenta o aspecto ilustrado na figura 1.28. Agora acrescenta-se, à zona de trabalho, um objecto de outro tipo. Trata-se de um `pop-up menu` e servirá para o utilizador escolher, de entre uma lista de valores, a probabilidade de erro do canal.

As propriedades do `pop-up menu` devem ser alteradas de acordo com os valores apresentados na figura 1.29.

Passa-se agora a acrescentar, na interface para o utilizador que nos encontramos a criar, três *string*. Uma com a palavra *codificador*, outra com a palavra

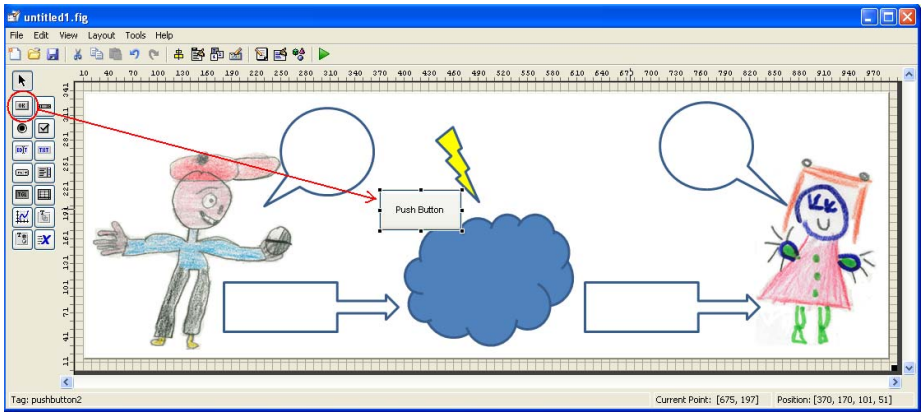


Figura 1.26: Aspecto da interface gráfica após a inserção da imagem de fundo e da colocação de um objecto `push button`.

Position		[90 6,5 20 3,2]
x		90.0
y		6.5
width		20.0
height		3.2
String		Transmitir...
Tag		Transmitir
TooltipString		Transmite BIT do João para a Rita

Figura 1.27: Propriedades relativas ao botão *Transmitir...*

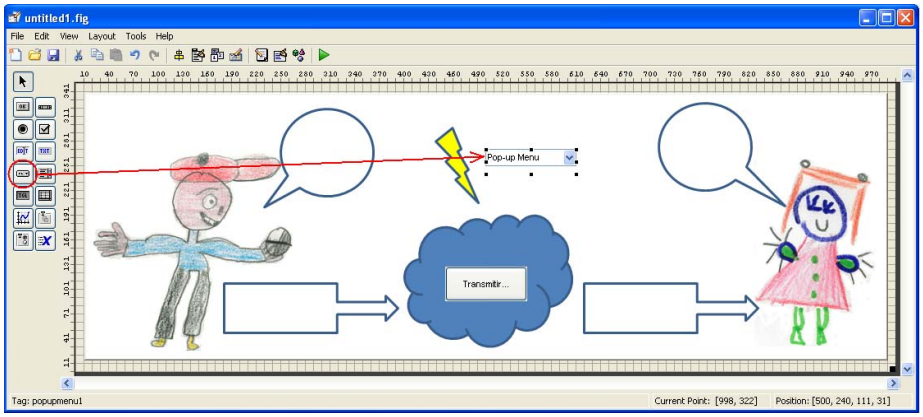


Figura 1.28: Após a descrição do botão *Transmitir*, a colocação de um `pop-up menu` no ambiente de trabalho.

descodificador e a terceira com a palavra *ruído*. Para esse efeito selecciona-se na paleta de ferramentas o objecto `static text` e dispõem-se, de forma arbitrária sobre o espaço de trabalho, três objectos. Cada um desses objectos conterá uma das três palavras referidas. A figura 1.30 ilustra este procedimento.

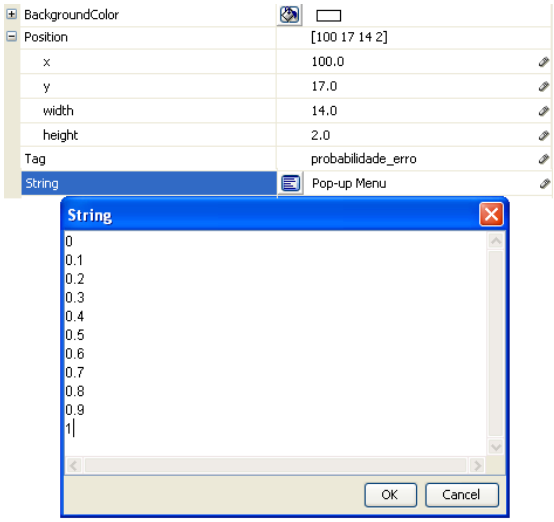


Figura 1.29: Propriedades a alterar no `pop-up menu`.

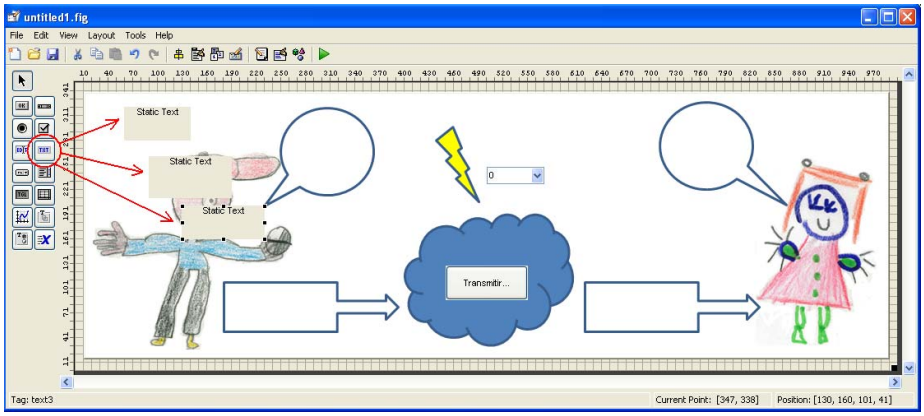


Figura 1.30: A interface já com o `pup-up menu` posicionado e com a adição de três objectos do tipo `static text`.

As propriedades a modificar, para cada um objectos `static text`, encontram-se ilustradas nas figuras 1.31 a 1.33. Após a actualização das propriedades desses objectos, a interface gráfica deve possuir um aspecto idêntico ao da figura 1.34. Ainda no mesmo ambiente, adicionam-se três novos objectos do tipo `static text`. Neste caso um dos objectos conterá a frase *probabilidade de erro* e os restantes dois serão responsáveis por apresentar as mensagens enviada e recebida pelos codificador/descodificador. Para cada um desses três objectos estabelecem-se as propriedades definidas nas figuras 1.35 a 1.37. Para concluir a interface basta adicionar mais dois objectos. Um será responsável por conter o bit que o utilizador quer enviar e o outro o valor lógico do bit recebido. Os objectos referidos são do tipo `Edit Text` e podem ser vistos na figura 1.38. As suas propriedades encontram-se ilustradas nas figuras 1.39 e 1.40.

BackgroundColor	
Position	[35,4 2 28,2 1,15]
x	35.40000000000003
y	2.0
width	28.200000000000003
height	1.15
String	CODIFICADOR
FontWeight	bold
Tag	codificador

Figura 1.31: Propriedades a alterar para o objecto `static text` que irá comportar a palavra *codificador*.

BackgroundColor	
Position	[123,8 1,338 28,2 1,15]
x	123.8
y	2.0
width	28.200000000000003
height	1.15
String	DESCODIFICADOR
FontWeight	bold
Tag	descodificador

Figura 1.32: Propriedades a alterar para o objecto `static text` que irá comportar a palavra *descodificador*.

BackgroundColor	
FontWeight	bold
Position	[84 23 10,2 1,2]
x	84.0
y	23.0
width	10.200000000000003
height	1.2
String	RUÍDO
Tag	ruido

Figura 1.33: Propriedades a alterar para o objecto `static text` que irá comportar a palavra *ruído*.

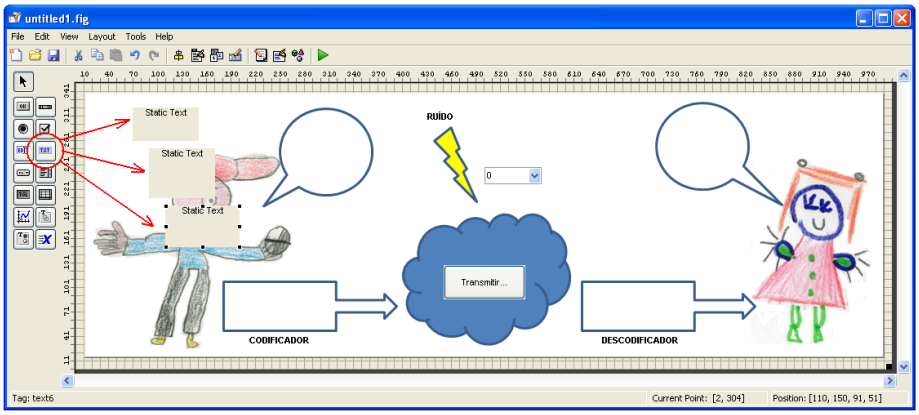


Figura 1.34: Interface já com as palavras *codificador*, *descodificador* e *ruído* posicionadas e com a adição de mais três string.

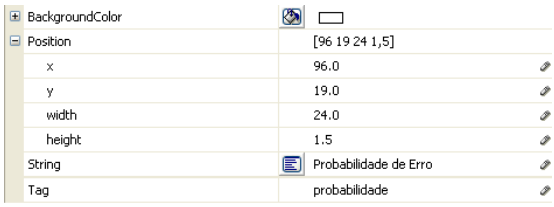


Figura 1.35: Propriedades a alterar para o objecto `static text` que irá comportar a frase *Probabilidade de Erro*.

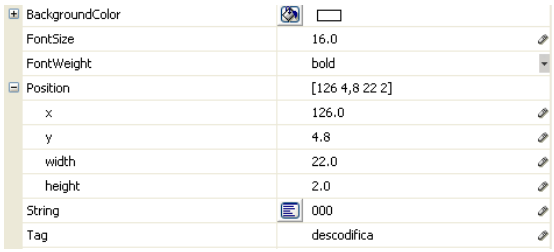


Figura 1.36: Propriedades a alterar para o objecto `static text` que irá apresentar a mensagem à entrada do descodificador.

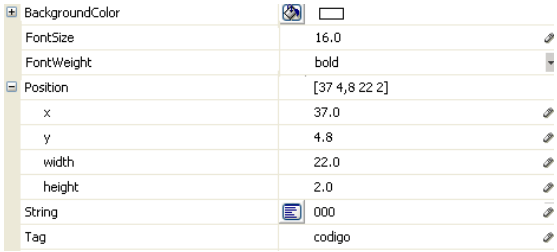


Figura 1.37: Propriedades a alterar para o objecto `static text` que irá apresentar a mensagem à saída do codificador.

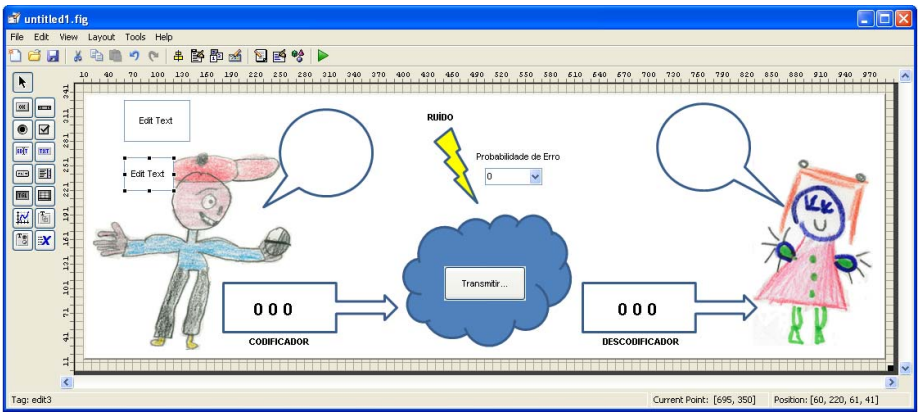


Figura 1.38: Interface já com todos os objectos `static text` colocados. Adição de dois objectos do tipo `Edit Text`.

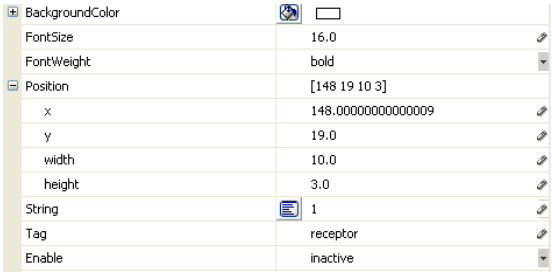


Figura 1.39: Propriedades relativas ao `Edit Text` do receptor.

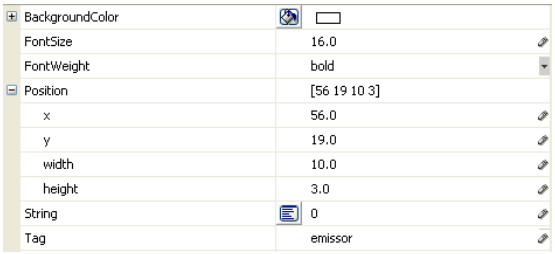


Figura 1.40: Propriedades relativas ao `Edit Text` do emissor.

Após terem sido seguidos todos os passos documentados, a interface gráfica com o utilizador encontra-se concluída e o seu aspecto final pode ser observado na figura 1.41.

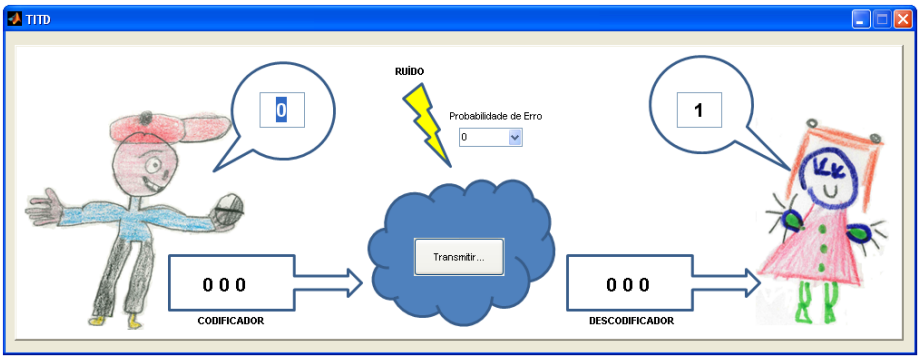


Figura 1.41: Aspecto final da interface gráfica.

O leitor irá notar que, assim que gravar a interface gráfica no disco do computador, o Matlab cria automaticamente um modelo de um ficheiro M. É nesse ficheiro que a componente relativa ao *back-end* do programa será inserido. Antes de apresentar o programa propriamente dito abre-se um pequeno parentices para falar sobre a forma como o Matlab gere as propriedades dos objectos inseridos na interface gráfica. Todos os objectos possuem um ponteiro (*handle*) sendo possível atribuir-lhes, de forma opcional, eventos (callback). Os ponteiros são

necessários para aceder e alterar as propriedades de um objecto. Por exemplo se **hnd** for o ponteiro para o `Edit Text` do emissor a execução da instrução:

```
get(hnd,'String')
```

retorna o conteúdo do campo *string* associado ao objecto com identificador **hnd**.

Numa interface gráfica, os ponteiros de todos os objectos são guardados numa estrutura designada por *handles*.

Como já se disse é possível adicionar eventos a cada objecto. Esses eventos são comportamentos que podem ser programados através da definição de uma dada rotina. O *back-end* é executado em função da activação desses eventos. Por exemplo no programa que nos encontramos a desenvolver apenas um objecto necessita de *callback*. O botão transmitir. O código que deve ser inserido, dentro do M-File sob a designação `Transmitir_Callback` encontra-se descrito a seguir.

```
% --- Executes on button press in Transmitir.
function Transmitir_Callback(hObject, eventdata, handles)
m=str2num(get(handles.emissor,'String'));
if isempty(m),
   m=0;
end
m=m>=1;
% Actualiza string...
set(handles.emissor,'String',num2str(m));
% Acrescenta redundância
G=[1 1 1];
c=m*G;
% Mostra o vector código
set(handles.codigo,'String',num2str(c));
% Verifica quais os bit a trocar...
% Adquire valor da probabilidade de erro
P=0:0.1:1;
indice=get(handles.probabilidade_erro,'Value');
Pe=P(indice); % ---------Probabilidade de erro
e=rand(1,3)<Pe;
d=soma_mod2(c,e);
% Escreve valor no descodificador
set(handles.descodifica,'String',num2str(d));
% Descodifica (por maioria)
inx=find(d==1);
if length(inx)>=2,
   m=1;
else
   m=0;
end
% Escreve mensagem no receptor.
set(handles.receptor,'String',num2str(m));
```

Com isto conclui-se o presente documento. No entanto deixa-se aqui a nota

de que muitos dos assuntos foram tratados de forma superficial. Muito mais poderia ser dito acerca de cada uma das situações endereçadas. A prova disso é o elevado número de livros editados sobre o Matlab e a sua utilização sobre os mais variados pontos de vista. No entanto, e como elemento introdutório, o domínio dos conceitos introduzidos neste documento fornecem as competências mínimas para que sejam capazes de implementar programas e rotinas que permitam resolver os problemas que serão endereçados ao longo da unidade curricular de **Teoria da Informação**.

Capítulo 2

Introdução à Teoria das Probabilidades

"The probability of life originating from accident is comparable to the probability of the unabridged dictionary resulting from an explosion in a printing shop."

–Edwin Grant Conklin

A Teoria da Informação, enquanto disciplina, partilha muito dos seus conceitos com diversas áreas do conhecimento. A estatística e a teoria das probabilidades é uma delas. Por esse motivo, antes de apresentar os matérias relativas aos sistemas de codificação estudados em teoria da informação, apresentam-se um conjunto de conceitos muito importantes relativos à teoria das probabilidades. Deixa-se aqui a nota que, neste documento, apenas se analisam sistemas que possam ser descritos por funções de distribuição discretos, i.e. processos estocásticos com um espaço amostral finito e numerável.

2.1 Introdução

Ganhar o Totoloto é o sonho de muita gente. A escolha dos seis números, que podem resultar na chave vencedora, é feita das mais diversas formas: datas de aniversário, números observados durante um sonho, etc. Se bem que não tendo hábito de jogar resolvi uma vez, há já alguns anos atrás, apostar um conjunto de chaves obtidas pela análise da frequência de saída de cada um dos 49 números utilizados na extracção do número vencedor (na altura eram só 45!). O gráfico da frequência relativa da ocorrência de cada número, designado por **histograma**, possuía um aspecto semelhante aquele que se apresenta na figura

2.1.

Cada uma das barras do histograma representa o número de vezes que um número particular foi extraído relativamente ao número total de semanas de jogo. Ou seja se n_y representar o número de vezes que, em N semanas, o número y for extraído então a sua frequência relativa ν_y é dada por:

$$\nu_y = \frac{n_y}{N} \tag{2.1}$$

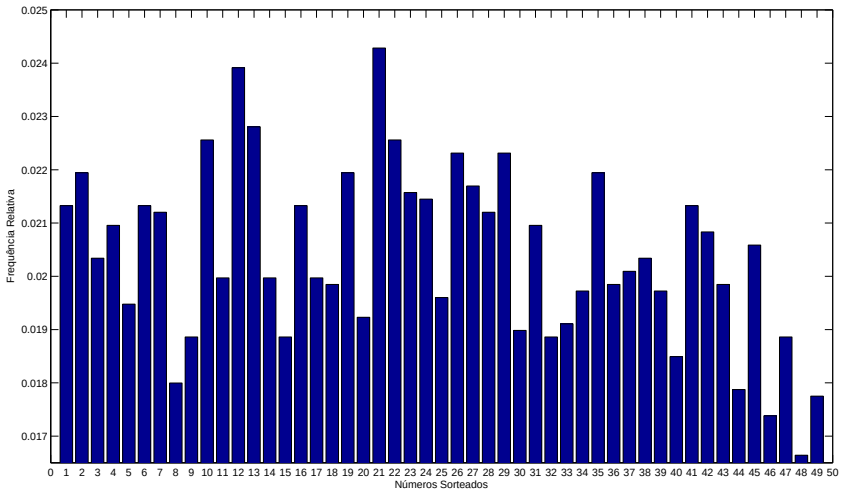


Figura 2.1: Histograma, relativo a extracção dos números do Totoloto, desde Março 1997 a Fevereiro de 2011 (*fonte da informação*: Jogos Santa Casa).

De acordo com a informação observada preenchi um conjunto de chaves de acordo com as frequências relativas. Por exemplo uma chave que envolvia os seis números mais frequentes, outra que envolvia os seis números menos frequentes, etc. Com base no histograma apresentado uma possível chave seria 10, 12, 13, 21, 22 e 29. Apostei, ao todo, dez chaves diferentes. Após a extracção oficial dos números, comparei as minhas dez apostas com o resultado da extracção e conclui, com alguma decepção, que não tinha acertado num único número... *o que é que se passou*?

O problema é que tinha tentado inferir o comportamento do sistema que gera os números do Totoloto através de uma amostra eventualmente demasiado pequena. Ou seja tentei descobrir o modo de operação de um sistema estocástico[1] tendo apenas acesso a um conjunto limitado de observações desse mesmo sistema. O que, pelo resultado das minhas apostas, se comprova não ser a mesma coisa...

No caso do Totoloto, sendo o sistema de extracção aleatório com uma *distribuição* uniforme, a *probabilidade* de, num determinado *concurso*, sair **qualquer**

[1]Estocástico e o contrário de determinístico. Um sistema determinístico responde sempre da mesma forma à mesma excitação sendo, por isso, previsível. Já num sistema estocástico não é possível conhecer, com certeza, a sua resposta a uma dada excitação.

um dos números é igual a 1/49 (aproximadamente 2%)[2]. Na secção que se segue definem-se três conceitos importantes introduzidos nesta última frase: experiência aleatória, probabilidade e distribuição.

2.2 Probabilidades: Axiomas e Propriedades

Na engenharia, e na ciência em geral, a abordagem empírica é essencial. Por norma, se uma determinada experiência for repetida sob condições semelhantes, esperam-se resultados semelhantes. No entanto nem sempre as coisas se passam assim. Com frequência a repetição de uma experiência, sob condições muito próximas, não resulta no mesmo resultado experimental. Este tipo de experiências são designadas por aleatórias. Numa experiência aleatória existe sempre incerteza relativa ao desfecho dessa experiência. Apenas se pode estimar, com uma determinada *probabilidade*, qual deverá ser o resultado.

Existem duas formas importantes, a partir das quais, é possível estimar a probabilidade da ocorrência de um dado *evento*:

Probabilidade *a priori*: Se um evento pode ocorrer de n maneiras diferentes, de um total de N, e se todas são igualmente prováveis, então a probabilidade do evento é n/N.

Probabilidade *a posteriori*: Se, após n repetições de uma experiência, se observou a ocorrência de um dado evento p vezes, então a probabilidade de ocorrência desse evento é p/n. Nesta estratégia o número de repetições deve ser elevada. Com efeito quando o número de experiência tende para infinito a frequência de ocorrência de um dado evento tende para o valor da sua probabilidade.

No contexto da teoria das probabilidades é importante definir, de forma explícita, alguns dos termos introduzidos nos parágrafos anteriores entre os quais se salientam:

Experiência Uma experiência refere-se a uma acção que pode levar a um qualquer resultado de entre um possível conjunto. Considera-se uma experiência aleatória uma situação com as seguintes características:

- Pode repetir-se indefinidamente nas mesmas condições;
- Tem um resultado imprevisível;
- Os resultados individuais são imprevisíveis mas, quando tomados no seu conjunto, observa-se alguma regularidade estatística.

Resultado Como o próprio nome indica, o resultado refere-se à observação gerada pela execução de uma experiência.

Espaço Amostral Admite-se que todo o desfecho, de uma determinada experiência, se encontra identificada num conjunto Ω designado por espaço amostral. Se ω for o resultado de uma dessas experiências então $\omega \in \Omega$.

Evento Um evento consiste num conjunto cujos valores pertencem a Ω ou seja um evento E é um sub-conjunto de Ω.

[2]Com esta distribuição a **incerteza**, quanto ao possível número extraído, é **máxima**. O conceito de incerteza e informação aparecerá brevemente no contexto da teoria da informação.

Probabilidade Define-se probabilidade como uma medida numérica quantitativa, tomada no intervalo $[0, 1]$, associada a um resultado face ao espaço amostral da experiência. Esse valor varia de acordo com o evento E segundo uma determinada função designada por *função massa de probabilidade*[3].

Na secção 2.2.1 apresenta-se um conceito nuclear à teoria das probabilidades: a *variável aleatória*. Adicionalmente estabelece-se a relação entre essa variável e um *evento* estatístico. Já na secção 2.2.2 define-se probabilidade da ocorrência de um evento no contexto de variáveis discretas. Introduz-se também o conceito de função massa de probabilidade [4]. Finalmente na secção 2.2.3 apresentam-se um conjunto de axiomas e propriedades associadas à teoria das probabilidades.

2.2.1 Eventos e Variáveis Aleatórias

Na secção anterior definiu-se evento estatístico como sendo o conjunto de resultados de uma experiência. Formalmente, qualquer subconjunto do espaço amostral é um evento. Qualquer evento que consista num único resultado do espaço amostral é designado por *evento simples*. Eventos que envolvam mais do que um resultado são chamados de *eventos compostos*. Por exemplo se um sistema digital envia sequências de três bit então, para o receptor, o espaço amostral é $\Omega = \{000, 001, 010, 011, 100, 101, 110, 111\}$. Dentro deste espaço podem definir-se vários subconjuntos. Cada subconjunto é um evento. Por exemplo o evento "todos os bit enviados são 0" refere-se a um evento simples dado por $E = \{000\}$. Já o evento, "pelo menos um dos bit transmitidos é 1", é definido pelo evento composto $E = \{001, 010, 011, 100, 101, 110, 111\}$.

Admitindo um espaço amostral $\Omega = \{\omega_1, \cdots, \omega_N\}$ o conjunto E é considerado um evento se $E \subset \Omega$. A figura 2.2 ilustra essa situação para três eventos E_1, E_2 e E_3 parcialmente sobrepostos.

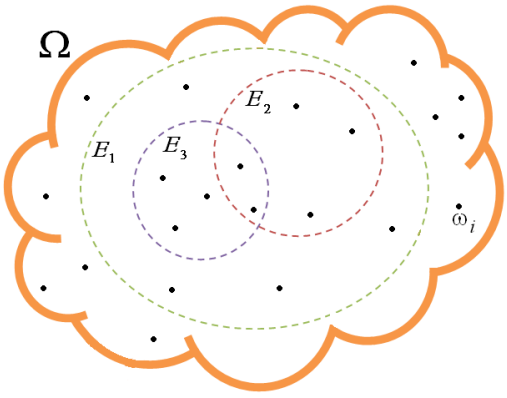


Figura 2.2: Partição do espaço amostral Ω com N elementos, de ω_1 a ω_N, em três subconjuntos E_1, E_2 e E_3.

Frequentemente associa-se a um evento ou espaço amostral uma variável com contra-domínio em $\mathbb{R}$ denominada por **variável aleatória**. Efectivamente, uma

[3] Esta função relaciona-se apenas com variáveis aleatórias discretas. No caso de variáveis contínuas o conceito entre probabilidade e função densidade de probabilidade é diferente.

[4] Frequentemente chamada também de função densidade de probabilidade discreta

variável aleatória é uma **aplicação** que faz o mapeamento do espaço amostral Ω num espaço alternativo.

Assim, admita-se uma experiência aleatória com espaço Ω. Seja ω_i um elemento qualquer pertencente a Ω. Define-se a variável aleatória $X(\omega_i)$ como sendo uma função que atribui um valor real x_i a cada amostra $\omega_i \in \Omega$. Os seja efectua o mapeamento do espaço amostral na recta real. A figura 2.3 ilustra este conceito.

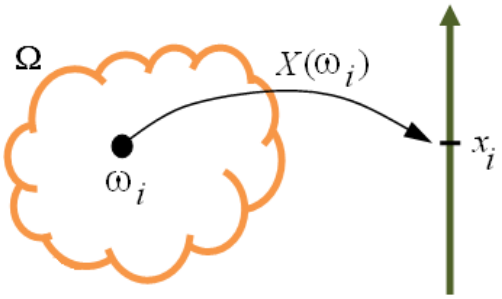


Figura 2.3: A variável aleatória $X(\omega)$ como uma função que transforma qualquer $\omega_i \in \Omega$ em $x_i \in \mathbb{R}$.

Frequentemente, por simplicidade de notação, a variável aleatória $X(\omega)$ é designada simplesmente por X.

Se um evento é um sub-conjunto do espaço amostral e se a cada elemento do espaço amostral pode ser atribuído um valor real então o mapeamento do evento pela variável aleatória X será um sub-intervalo do contra-domínio dessa variável aleatória. Esta definição pode ser sintetizada por:

$$E = \{\omega : X(\omega) = x, \;\; x \in X\} \tag{2.2}$$

ou seja E é o conjunto de todos as ocorrências ω que possuam imagem em X. Por exemplo imagine o seguinte universo de alunos de uma turma:

$$\Omega = \{\text{João Ló, Alda Neiva, João Oliveira, Maria Pereira, Ana Costa}\} \tag{2.3}$$

Imagine que se registaram as alturas desses alunos. Define-se a variável aleatória $X(\omega_i)$ como sendo a "estatura do aluno ω_i". Para o presente espaço amostral, o valor de X, em cm, é:

$$X(\Omega) = \{168, 161, 175, 155, 162\} \tag{2.4}$$

Descreve-se agora o evento E como sendo "altura superior a 160 cm e sexo feminino". O conjunto agora é:

$$E = \{\text{Alda Neiva, Ana Costa}\} \tag{2.5}$$

Os valores da variável aleatória X associada a este evento são:

$$X(E) = \{161, 162\} \tag{2.6}$$

2.2.1.1 Tipos de variáveis

As variáveis, num qualquer problema, podem ser distinguidas como pertencendo a um de dois tipos: contínuas ou discretas. Uma variável é contínua, sobre um intervalo arbitrário, se esta puder tomar qualquer valor dentro desse intervalo. Por exemplo se x representar a tensão eléctrica aos terminais de uma tomada de fornecimento de energia eléctrica, então esta pode tomar qualquer valor entre [5] -325.3 V e 325.3 V. Por outro lado, uma variável discreta é aquela que apenas pode tomar um número finito de valores entre um intervalo também finito. Por exemplo o número de bit armazenados numa *pen drive* de 4 GiB[6]. Neste caso x pode ser qualquer valor num conjunto com $8 \times 4 \times 2^{30}$ elementos[7].

Tanto as variáveis contínuas como discretas não carecem de definição num intervalo fechado. Por exemplo o número cumulativo de bit transferidos numa rede de comunicação de dados não tem limite superior. Note-se, no entanto, que para a unidade curricular de Teoria da Informação, apenas estamos interessados em variáveis discretas com domínio finito.

Tendo sido definido o conceito de *evento* e *variável aleatória* apresenta-se, na próxima secção, o conceito de *probabilidade* e de *função massa de probabilidade*.

2.2.2 Definição de Probabilidade

Existem várias formas de definir o conceito de probabilidade. Numa perspectiva clássica, a probabilidade de ocorrência do evento $E \subset \Omega$, caracterizado por um conjunto finito numerável de elementos, designa-se por $P(E)$ e é igual à razão entre o cardinal de E e o cardinal de Ω. Ou seja,

$$P(E) = \frac{\#E}{\#\Omega} \tag{2.7}$$

Estando completamente identificados os conjuntos E e Ω, o valor da probabilidade é calculado *a priori* sem requerer qualquer experiência. A probabilidade de um evento E verifica ainda as seguintes propriedades:

- $0 \leq P(E) \leq 1$ para $E \subset \Omega$

  Se $P(E) = 1$ então a ocorrência de E é certa ou seja a probabilidade de ocorrer E, como resultado de uma experiência com espaço Ω, é de 100%. Do mesmo modo um acontecimento E é impossível se $P(E) = 0$

- Definindo um conjunto N partições disjuntas de Ω, $E_1, \cdots, E_N$, de tal forma que $\Omega = E_1 \vee \cdots \vee E_N$, então:

$$\sum_{i=1}^{N} P(E_i) = 1 \tag{2.8}$$

[5] A tensão eficaz (RMS) na rede eléctrica em Portugal é igual a 230 V (alterada, em 1984 por razões de compatibilidade com os restantes países Europeus, de 220 V para 230 V). A esse valor eficaz corresponde uma tensão (monofásica) de pico, em módulo, igual a $230\sqrt{2} \approx 325.3$ V.

[6] Desde 1998 a Comissão Electrotécnica Internacional (IEC) aprovou uma lista de prefixos para múltiplos binários. O *kilo* é um prefixo SI que representa um factor $\times 1000$. Contudo, em binário, 1KByte não representava 1000 mas sim 1024 bytes (2^{10}). Por esse motivo o que era conhecido por 1KByte é agora 1KiByte (Ki representa o *kilo* binário ou *kibi*). O valor de 1KByte passou a ser 1000 bytes de acordo com a norma SI.

[7] Ignorando o espaço reservado pela FAT e admitindo a possibilidade de escrita de bit individuais.

Como se disse logo no início do capítulo, a probabilidade pode ser avaliada *a posteriori* pela análise da frequência relativa de um evento, a partir de um número suficientemente elevado de experiências. Quando o número de experiências tende para infinito, a frequência relativa do evento tende para o valor da sua probabilidade de ocorrência. Este fenómeno é designado por **lei dos grandes números** e pode ser colocada da seguinte forma:

$$P(E) = \lim_{N\to\infty} \nu_N(E) \tag{2.9}$$

onde

$$\nu_N(E) = \frac{n_E}{N} \tag{2.10}$$

se refere à frequência relativa da ocorrência do evento E em N experiências e n_E representa o número de vezes que o evento E ocorreu em N experiências.

De modo a ilustrar ambas as definições de probabilidade imagine-se que se pretende determinar a probabilidade de, numa transmissão de três bit[8], observar dois bit idênticos. De acordo com a expressão (2.7) o valor dessa probabilidade pode ser calculada pela razão do cardinal do conjunto E: "exactamente dois bit consecutivos são idênticos", que é dado pelos elementos $\{001, 011, 100, 110\}$ pelo cardinal do conjunto definido pelo espaço amostral "envio de três bit" ou seja $\Omega = \{000, 001, 010, 011, 100, 101, 110, 111\}$. Assim, neste caso,

$$P(E) = \frac{\#E}{\#\Omega} = \frac{4}{8} = 0.5 \tag{2.11}$$

Imagine-se agora que um hipotético observador, colocado a jusante do canal de transmissão, desconhece Ω. Este observador pretende calcular a probabilidade do evento E, "recepção de dois bit idênticos seguidos" estando, no entanto, impedido de recorrer a (2.11). Para isso o observador vai registando os ternos de valores recebidos. A cada nova recepção, o observador recalcula o valor da probabilidade do evento E ocorrer a partir da expressão (2.9). A figura 2.4 mostra uma possível evolução do valor estimado da probabilidade em função do número de observações efectuadas. Na caixa de texto que se segue apresenta-se a sequência de comandos executados em Matlab para obter a referida figura.

```
clear all;clc
N=1000; %-------------------Número de observações,
for k=1:N,
   B=rand(1,3)>0.5; %-------------Transmite um terno de bit
   E(k)=abs(sum(diff(B))); %---- 1 se existirem apenas dois
                           %---- bit iguais seguidos
                           %---- 0 caso contrário
   P(k)=sum(E)/k;
end
plot(1:N,P,1:N,0.5*ones(N,1),':');
xlabel('Observação');ylabel('Valor estimado da probabilidade')
```

[8] Admite-se que a probabilidade de transmitir um '0' é igual à probabilidade de transmitir o valor lógico '1'

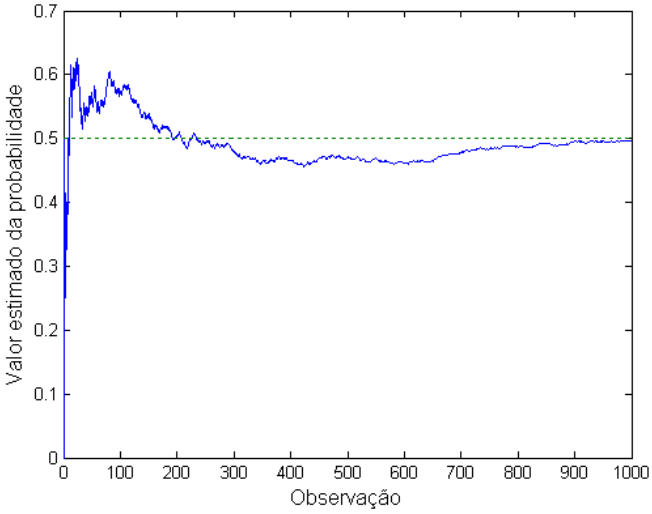


Figura 2.4: Evolução do valor estimado da probabilidade em função do número de ensaios.

Conforme se pode ver, a variância[9] na estimativa da probabilidade vai diminuindo com o número de observações N e convergindo para o valor real da probabilidade que, como se viu, é 0.5.

No caso de variáveis aleatórias discretas define-se uma quantidade relacionada com a definição anterior de probabilidade. Essa quantidade é derivada de uma função, $f_X(x)$, denominada por **função massa de probabilidade** [10] e formalizada da seguinte forma:

$$f_X(x) : x \in X, \;\; f_X(x) = P\left(X = x\right) \tag{2.12}$$

Para variáveis aleatórias discretas a relação, entre a função massa de probabilidade, para um dado valor x da variável X, e o valor da probabilidade de X tomar o valor x é de igualdade. Note que a função $f_X(x)$ se encontra definida para qualquer valor de x mesmo os que não têm imagem em Ω. Neste caso $f_X(x) = 0$ se $x \notin X(\Omega)$.

De modo a ilustrar este conceito retoma-se novamente ao problema da transmissão de três bit através de um canal de comunicação. Já se viu que o espaço amostral é dado por:

$$\Omega = \{\omega_0, \omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6, \omega_7\} \tag{2.13}$$

onde $\omega_0 = 000$, $\omega_1 = 001$, $\omega_2 = 010$, $\omega_3 = 011$, $\omega_4 = 100$, $\omega_5 = 101$, $\omega_6 = 110$ e $\omega_7 = 111$.

O evento E, definido verbalmente por "exactamente dois bit consecutivos são idênticos", corresponde ao conjunto $E = \{\omega_1, \omega_3, \omega_4, \omega_6\}$.

[9] O conceito de variância estatística será introduzido adiante.
[10] Do Inglês *probability mass function*.

Define-se agora a variável aleatória $X(\omega)$ como sendo "o número de **pares** de bit consecutivos idênticos numa mensagem enviada". Neste caso o contra-domínio de X é o conjunto $\{0, 1, 2\}$. Por exemplo $X(\omega_0) = 2$, $X(\omega_1) = 1$ e $X(\omega_2) = 0$. O mapeamento de todos os valores do espaço amostral leva ao seguinte conjunto:

$$X(\Omega) = \{2, 1, 0, 1, 1, 0, 1, 2\} \tag{2.14}$$

A figura 2.5 ilustra este mapeamento para alguns elementos de Ω.

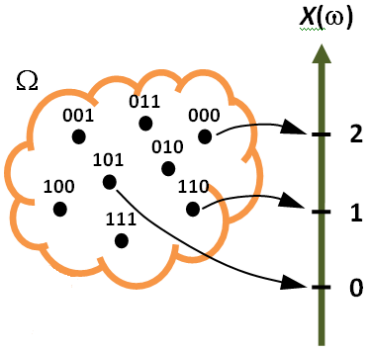


Figura 2.5: Exemplo do mapeamento de alguns elementos do espaço amostral Ω para a recta real $X(\Omega)$.

No problema em análise admite-se que cada observação ω_i em Ω tem igual probabilidade de ocorrer. Ou seja,

$$P(\omega_i) = \frac{1}{8}, \quad i = 0, \cdots, 7 \tag{2.15}$$

Para o evento E a probabilidade é, como já se viu, igual a:

$$P(E) = \frac{1}{2} \tag{2.16}$$

Relativamente à variável aleatória $X(\omega)$ a função massa de probabilidades é definida da seguinte forma:

$$f_X(x_i) = \begin{cases} \frac{2}{8} & \text{para } x_i = 0 \\ \frac{4}{8} & \text{para } x_i = 1 \\ \frac{2}{8} & \text{para } x_i = 2 \end{cases} \tag{2.17}$$

Os valores das probabilidades $P(X = x_i)$ para cada uma dos três possíveis valores de x_i, são obtidos pelo processo de contagem associada ao conjunto representado na expressão (2.14).

Neste caso o valor de $P(E)$ é igual a $P(X = 1)$ ou seja o valor da probabilidade da ocorrência do evento E é igual à probabilidade de serem transmitidos **1 par** de bit idênticos na mensagem, i.e.

$$P(E) = f_X(1) \tag{2.18}$$

No entanto, de forma mais geral, a probabilidade de um evento E, definido por N elementos $\{\omega_1, \cdots, \omega_N\}$, relaciona-se com a função massa de probabilidade por:

$$P(E) = \sum_{x_i = X(\omega_i)} f_X(x_i), \quad i = 1, \cdots, N \tag{2.19}$$

Por exemplo se a variável aleatória X fosse definida como "valor, em BCD, da soma módulo 2 dos bit da mensagem segundo a ordem recebida"[11], então:

$$X(\Omega) = \{0, 1, 3, 2, 2, 3, 1, 0\} \tag{2.20}$$

A função massa de probabilidade é agora:

$$f_X(x_i) = \begin{cases} \frac{2}{8} & \text{para } x_i = 0 \\ \frac{2}{8} & \text{para } x_i = 1 \\ \frac{2}{8} & \text{para } x_i = 2 \\ \frac{2}{8} & \text{para } x_i = 3 \end{cases} \tag{2.21}$$

A probabilidade da ocorrência de E é igual à probabilidade de X ser igual a 1 mais a probabilidade de X ser igual a 2 ou seja[12],

$$P(E) = \sum_{x_i=1, x_1=2} f_X(x_i) = \frac{2}{8} + \frac{2}{8} = \frac{1}{2} \tag{2.22}$$

Com este exemplo também se demonstra que existe sempre alguma subjectividade na escolha da variável aleatória para um dado problema. Ou seja o mesmo problema pode ser resolvido através da definição de diferentes variáveis aleatórias.

Para finalizar deixa-se aqui o reparo de que, muitas vezes, a função massa de probabilidade é representada graficamente. Por exemplo, para as funções (2.17) e (2.21), o aspecto gráfico das funções de probabilidade encontra-se ilustrados na figura 2.6.

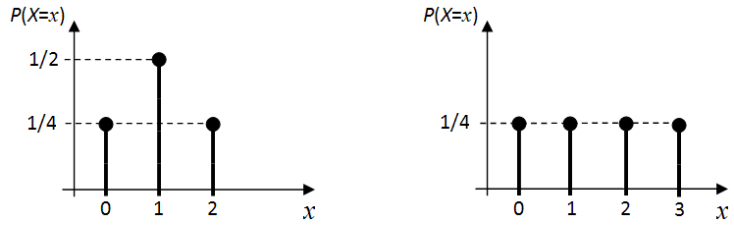


Figura 2.6: Representação gráfica da função massa de probabilidade. À esquerda o aspecto da função (2.17) e à direita a função (2.21).

No preâmbulo deste capítulo referiu-se que o conhecimento da teoria, axiomas e propriedades relativas à análise de probabilidades é de extrema importância no contexto da Teoria da Informação. Por esse motivo a secção que se segue apresenta os principais axiomas e propriedades associados à teoria das probabilidades.

2.2.3 Axiomas e Propriedades

Começa-se por se supor a existência de três eventos A, B e C. A probabilidade de ocorrência do evento A designa-se por $P(A)$ e este valor é frequentemente conhecido por **probabilidade marginal**.

[11] Admitindo a transmissão do tipo *big endian*, ou seja os bit são enviados segundo a ordem de maior significância, a primeira diferença, módulo 2, de 011 resulta em 10. O primeiro dos dois bit é calculado por 0+1=1 e o segundo bit através de 1+1=0.

[12] Caso não seja claro esta afirmação basta compara os conjuntos (2.14) com (2.20).

Tendo em consideração a definição de probabilidade, descrita na secção anterior, os seguintes axiomas confirmam-se sempre:

A probabilidade da ocorrência de um evento A resulta em:

$$0 \leq P(A) \leq 1 \tag{2.23}$$

A probabilidade da não ocorrência de um evento A é:

$$P(\overline{A}) = 1 - P(A) \tag{2.24}$$

A probabilidade de um acontecimento impossível é zero:

$$P(\varnothing) = 0 \tag{2.25}$$

A probabilidade de um acontecimento certo é um:

$$P(\mathbb{I}) = 1 \tag{2.26}$$

Se B for, tal como A, um evento então designa-se por **probabilidade conjunta** entre A e B a probabilidade da ocorrência simultânea dos eventos A e B. Esta quantidade é representada do seguinte modo:

$$P\left(A \wedge B\right) \tag{2.27}$$

A **lei das probabilidades totais** permite ainda definir $P(A)$ como:

$$P(A) = P(A \wedge B) + P(A \wedge \bar{B}) \tag{2.28}$$

Outra noção importante refere-se à **probabilidade condicional**. A probabilidade de um evento A, condicionado à ocorrência de um segundo evento B, denota-se por:

$$P\left(A|B\right) \tag{2.29}$$

A probabilidade condicional encontra-se ligada às probabilidades marginal e conjunta através do célebre teorema de **Bayes**:

$$P\left(A|\,B\right) = \frac{P\left(A \wedge B\right)}{P\left(B\right)} \tag{2.30}$$

onde, por convenção, se $P\left(B\right) = 0$ então $P\left(A|B\right) = 0$.

A regra de Bayes pode ainda ser apresentada, de forma alternativa, do seguinte modo:

$$P\left(B|\,A\right) P\left(A\right) = P\left(A|\,B\right) P\left(B\right) \tag{2.31}$$

ou ainda, para o caso de três eventos A, B e C como:

$$P\left(A|B \wedge C\right) = \frac{P(A|C)P(B|A \wedge C)}{P(B|C)} \tag{2.32}$$

Esta igualdade pode ser facilmente demonstrada recorrendo a 2.31 e à seguinte regra:

$$\begin{aligned} P\left(A \wedge B \wedge C\right) &= P\left(A\right) P\left(B|\,A\right) P\left(C|\,A \wedge B\right) \\ &= P\left(C\right) P\left(B|\,C\right) P\left(A|\,B \wedge C\right) \end{aligned} \tag{2.33}$$

designada normalmente por **regra da cadeia**.

Um conceito intimamente ligado à regra de Bayes é o de independência estatística entre dois eventos. Assim, se A é estatisticamente **independente** de B, então:

$$P\left(A|B\right) = P\left(A\right) \tag{2.34}$$

Outra forma de dizer que A é independente de B consiste em escrever:

$$P\left(A \wedge B\right) = P\left(A\right) \cdot P\left(B\right) \tag{2.35}$$

Em alguma literatura assiste-se à notação $A \perp B$ para referir que A é estatísticamente independente de B.

A noção de distribuição condicional pode ser estendida a mais do que uma variável. Por exemplo a probabilidade condicional de A dado B e C é descrita por $P(A|B \wedge C)$. Deste modo também o conceito de independência estatística pode ser dilatada a mais do que duas variáveis. Assim, se $P(A|B \wedge C)$ não depende de B,

$$P(A|B \wedge C) = P(A|C) \tag{2.36}$$

Diz-se assim que A é **condicionalmente independente** de B dado C. A partir desta definição é possível estabelecer as seguintes igualdades:

$$P\left(A \wedge B|\, C\right) = P\left(A|\, C\right) \cdot P\left(B|\, C\right) \tag{2.37a}$$
$$P\left(B|\, A \wedge C\right) = P\left(B|\, C\right) \tag{2.37b}$$

Finaliza-se esta secção apresentando a definição de probabilidade da união de dois eventos. Se, mais uma vez, A e B forem dois eventos quaisquer então:

$$P\left(A \vee B\right) = P\left(A\right) + P\left(B\right) - P\left(A \wedge B\right) \tag{2.38}$$

A expressão anterior se resume a:

$$P\left(A \vee B\right) = P\left(A\right) + P\left(B\right) \tag{2.39}$$

no caso de A e B serem eventos **mutuamente exclusivos**. Ou seja a ocorrência de A impede a ocorrência de B e vice-versa. Note que se for impossível a ocorrência simultânea dos eventos A e B então,

$$P(A \wedge B) = 0 \tag{2.40}$$

A expressão 2.38 pode ser estendida para a união de um número arbitrário de eventos. Por exemplo para o caso de três eventos A, B e C observa-se que:

$$\begin{aligned} P\left(A \vee B \vee C\right) = {} & P\left(A\right) + P\left(B\right) + P\left(C\right) - P\left(A \wedge B\right) - P\left(A \wedge C\right) \\ & - P\left(B \wedge C\right) + P\left(A \wedge B \wedge C\right) \end{aligned} \tag{2.41}$$

2.3 Momentos Estatísticos

Pela própria definição, o valor preciso de uma experiência aleatória é impossível de ser conhecido. Ou seja é imprevisível. No entanto é possível estabelecer uma

estimativa para o seu valor com base num número suficientemente grande de observações. O valor mais utilizado nessa estimativa é a média aritmética.

Se $x_1, x_2, \cdots, x_N$ forem os resultados de um conjunto de N experiências aleatórias, o valor médio das observações, designado por $\bar{x}$, é calculado por:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{2.42}$$

Por exemplo imagine que se lança uma bola e se mede a distância que esta atinge. O resultado relativo a dez ensaios é apresentado na tabela 2.1.

Tabela 2.1: Distância atingida pelo lançamento de uma bola.

Ensaio	1	2	3	4	5	6	7	8	9	10
distância/m	10.5	11.8	7.7	10.9	10.3	8.7	9.6	10.3	13.6	12.8

A média das observações é 10.6 m [13]. O que significa que, se nos fosse pedido uma estimativa do valor da distância atingida pela bola no 11º ensaio o valor mais razoável seria o valor da média, i.e. 10.6 m.

Admita-se agora o lançamento de uma segunda bola substancialmente mais leve do que a primeira. O resultado de dez lançamentos encontram-se na tabela 2.2.

Tabela 2.2: Distância atingida pelo lançamento de uma segunda bola.

Ensaio	1	2	3	4	5	6	7	8	9	10
distância/m	4.5	4.8	4.6	4.7	31	4.4	4.6	38	4.6	4.8

O valor médio neste caso é idêntico ao obtido usando os valores da tabela 2.1. No entanto os valores tabelados não são tão consistentes quanto os obtidos no conjunto de ensaios anterior. Efectivamente, observando a tabela 2.2 identificam-se dois valores bastante diferentes dos restantes. Fala-se dos ensaios 5 e 8. A discrepância desses valores, relativamente aos restantes, pode dever-se a fenómenos externos incontroláveis. Por exemplo um golpe de vento, no instante em que a bola é lançada, segundo a direcção do movimento, poderia transportar a bola para uma distância muito superior.

O importante a concluir deste exemplo é que o valor médio nem sempre é uma boa estimativa para o resultado de um processo aleatório. Por isso mesmo o valor médio deve ser sempre acompanhado por uma medida complementar que reflicta a confiança no valor médio como estimativa. Essa medida designa-se por variância e é calculada como sendo o valor médio do desvio quadrático dos resultados das experiências conduzidas. Algebricamente descreve-se como:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2 \tag{2.43}$$

Uma medida relacionada com a variância é o desvio-padrão sendo calculado por:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{2.44}$$

[13] Apresentada com três algarismos significativos por coerência com os valores tabelados.

Relativamente à variância, o desvio-padrão têm a vantagem de deixar inalteradas as unidades com que o valor da variável aleatória é registada. Por exemplo o desvio padrão e a variância, associada aos dois conjuntos de ensaios tabelados em 2.1 e 2.2, apresentam-se na tabela 2.3.

Tabela 2.3: Média, variância e desvio-padrão dos valores apresentados nas tabelas 2.1 e 2.2.

Ensaio	Média/m	Variância/m^2	Desvio-padrão/m
Tabela 2.1	10.6	3.18	1.78
Tabela 2.2	10.6	161	12.7

Como se pode constatar, apesar do valor médio ser idêntico existe uma forte discrepância entre os desvio-padrão (e variância). O desvio no primeiro conjunto de ensaios é muito menor do que o obtido no segundo conjunto. O que significa que o valor médio, no primeiro conjunto de ensaios, "encaixa" nas observações com um erro **médio**[14]do erro em torno dos 2 m. A incerteza no segundo conjunto de dados e substancialmente maior conforme se pode concluir a partir do seu valor de desvio-padrão.

O valor médio no segundo conjunto de ensaios não é representativo do que de facto acontece quando se atira a bola. Essa falta de representatividade deve-se, sobretudo, a dois ensaios cujas distâncias diferem substancialmente das restantes. Esses ensaios, embora com valores elevados, acontecem com menor frequência do que os restantes. Por isso um valor mais representativo poderia ser obtido a partir de uma média ponderada sendo dado maior peso às observações mais prováveis e menor peso às observações menos prováveis. Efectivamente essa operação é designada por esperança estatística, $\mathfrak{E}(X)$, e é descrita matematicamente por:[15].

$$\mathfrak{E}(X) = \sum_{i=1}^{N} f_X(x_i) \cdot x_i \tag{2.45}$$

onde X se refere à variável aleatória tal que $X(\Omega) = \{x_1, \cdots, x_N\}$ com função massa de probabilidade $f_X(x_i)$.

Comparando a expressão (2.42) com a expressão (2.45) verifica-se que a média aritmética não é mais do que a esperança de uma variável aleatória admitindo que todos os seus valores têm igual probabilidade de ocorrência, i.e.

$$f_X(x_i) = \frac{1}{N}, \quad \forall i \tag{2.46}$$

A esperança, enquanto operador sobre a variável aleatória X, goza da propriedade de linearidade. Efectivamente, se $X_1, X_2, \cdots, X_n$ forem n variáveis aleatórias e $k_1, \cdots, k_n$ forem n constantes então:

$$\mathfrak{E}(k_1X_1 + k_2X_2 + \cdots + k_nX_n) = k_1\mathfrak{E}(X_1) + k_2\mathfrak{E}(X_2) + \cdots + k_n\mathfrak{E}(X_n) \tag{2.47}$$

[14]Efectivamente trata-se da raíz quadrada do erro quadrático frequentemente designado por erro RMS (root-mean square)

[15]Esta expressão só é válida para variáveis discretas descritas por funções massa de probabilidade

A esperança estatística é também designada por primeiro **momento estatístico**. Em estatística, define-se o momento de ordem p de uma variável aleatória X, com densidade de probabilidade $f_X(x)$, por:

$$\mathfrak{E}(X^p) = \sum_{i=1}^{N} x_i^p \cdot f_X(x_i) \tag{2.48}$$

Note-se que o valor médio quadrático de uma variável aleatória não é mais do que o momento de ordem dois.

Define-se ainda **momento central de ordem** p de uma variável aleatória X como,

$$\mathfrak{E}\Big(\big(X - \mathfrak{E}(X)\big)^p\Big) = \sum_{i=1}^{N} \big(x_i - \mathfrak{E}(X)\big)^p \cdot f_X(x_i) \tag{2.49}$$

Como facilmente se pode confirmar, se $p = 2$ e se $f_X(x_i) = \frac{1}{N}$, então o momento central de segunda ordem dá pelo nome de variância.

2.4 Acontecimentos com Resultados Binários

Na unidade curricular de Teoria da Informação, tem particular interesse o caso em que a saída de um processo estocástico é uma sequência finita de variáveis aleatórias binárias. Ou seja variáveis que apenas podem tomar dois valores distintos. Isto porque as técnicas de codificação de canal e codificação de fonte que serão estudadas admitem que a informação é representada por um alfabeto contendo apenas dois símbolos: o zero '0' e o um '1'. Por essa razão, nesta secção, apresenta-se a função de probabilidade mais importante neste contexto: a função binomial.

2.4.1 Variável de Bernoulli

Antes de introduzir a função binomial descreve-se um tipo específico de variável aleatória designada por variável de Bernoulli. Uma variável de Bernoulli, ou variável aleatória binária, é uma variável que apenas pode, num determinado instante, estar num dos seus dois estados possíveis designados por '0' e '1'. Por exemplo o valor lógico à saída de uma porta AND é descrito por uma variável deste tipo.

Seja X uma variável de Bernoulli definida por:

$$X = \{x\} \tag{2.50}$$

onde $x \in \{0, 1\}$ dependendo se um dado evento E ocorre ou não. Define-se a variável X, de forma alternativa, recorrendo à **função indicadora** $I_E(r)$:

$$x = I_E(r) = \begin{cases} 1 & \text{se } r \in E \\ 0 & \text{se } r \notin E \end{cases} \tag{2.51}$$

onde r se refere ao resultado de uma experiência aleatória. Por exemplo se o evento E for "o resultado do lançamento de um dado é 6" (ou seja $E = \{6\}$) e

se o resultado de um dado lançamento r for 4 então $I_E(r) = 0$ pois $r \notin E$ e logo $X = 0$. Por outro lado, se o evento E fosse "o resultado do lançamento de um dado é um número par", o que implica que $E = \{2, 4, 6\}$ e se o resultado de um dado lançamento r for 4 então $I_E(r) = 1$ o que leva a que $X = 1$.

Admita-se ainda que a probabilidade de $I_E(r)$ ser '1' é ρ. Como x é uma variável binária, então a probabilidade de x não ser '1', ou seja ser $I_E(r) = 0$, é $1 - \rho$. Com esta informação descreve-se a função massa de probabilidade de X como sendo:

$$f_X(x) = \begin{cases} \rho & \text{se } x = 1 \\ 1 - \rho & \text{se } x = 0 \end{cases} \tag{2.52}$$

O valor esperado de X, para um dado evento E, pode assim ser calculado por:

$$\begin{aligned} \mathfrak{E}(X) &= \sum_{x=0}^{1} x \cdot f_X(x) \\ &= \rho \end{aligned} \tag{2.53}$$

A sua variância também pode ser calculada segundo a definição, expressa por (2.49), como:

$$\begin{aligned} \sigma^2 = \mathfrak{E}\Big(\big(X - \mathfrak{E}(X)\big)^2\Big) &= \mathfrak{E}\Big(\big(X - \rho\big)^2\Big) \\ &= \mathfrak{E}\big(X^2 - 2\rho X + \rho^2\big) \\ &= \mathfrak{E}(X^2) - 2\rho^2 + \rho^2 \\ &= \mathfrak{E}(X^2) - \rho^2 \end{aligned} \tag{2.54}$$

Como $\mathfrak{E}(X^2) = \rho$ fica[16],

$$\begin{aligned} \sigma^2 &= \rho - \rho^2 \\ &= \rho(1 - \rho) \end{aligned} \tag{2.55}$$

2.4.2 Distribuição Binomial

A distribuição binomial refere-se à distribuição do número de sucessos, numa sequência de N experiências **binárias independentes**, cada qual com uma probabilidade de ocorrência ρ. Desta definição segue que a distribuição binomial está intimamente ligada à variável de Bernoulli. Efectivamente uma distribuição binomial refere-se à execução de N experiências de Bernoulli, com resultados $r_1, \cdots, r_N$, e pode ser considerada como a soma dessas N variáveis de Bernoulli,

$$X = I_E(r_1) + \cdots + I_E(r_N) \tag{2.56}$$

Se uma variável aleatória X segue uma distribuição binomial, a sua função massa de probabilidade é dada por:

$$f_X(N, k, \rho) = \left(\begin{array}{c} N \\ k \end{array} \right) \rho^k (1 - \rho)^{n-k} \tag{2.57}$$

[16] A demonstração desta igualdade fica como exercício.

onde[17],

$$\left(\begin{array}{c} N \\ k \end{array} \right) = \frac{N!}{k! \cdot (N-k)!} \tag{2.58}$$

se refere ao valor das combinações dos N elementos em conjuntos de k elementos.

O valor de $f_X(k, N, \rho)$ indica a probabilidade de serem obtidos **exactamente** k sucessos em N experiências com probabilidade de ocorrência ρ. A figura 2.7 ilustra o aspecto de $f_X(k, N, \rho)$, em função de k, admitindo $N = 20$ e para três valores distintos de ρ: 0.1, 0.5 e 0.9.

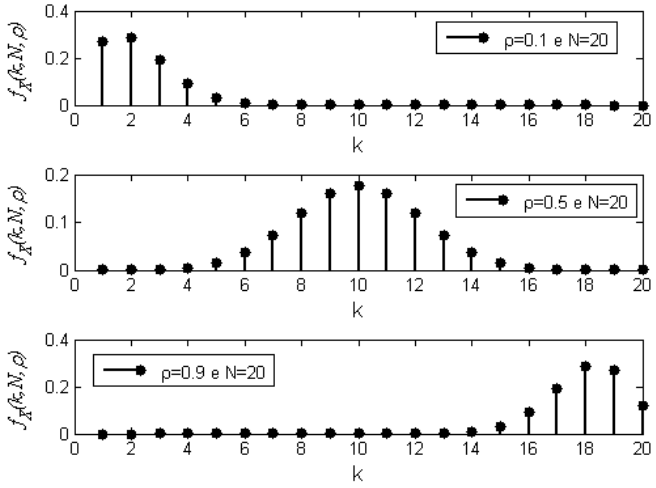


Figura 2.7: Aspecto da função massa de probabilidade, descrita pela equação (2.57), admitindo $N = 20$ e para três valores distintos de ρ: 0.1, 0.5 e 0.9.

As caixas de texto que se seguem apresentam o código para Matlab usado para gerar a figura 2.7.

```
N=20;k=1:N;
rho=[0.1,0.5,0.9];
for i=1:3,
   f_x(i,:)=binomial(N,k,rho(i));
end
subplot(3,1,1);
stem(k,f_x(1,:));legend('\rho = 0.1  e  N = 20');
xlabel('k');ylabel('f_X(k, N, \rho)');
subplot(3,1,2);
stem(k,f_x(2,:));legend('\rho = 0.5  e  N = 20');
xlabel('k');ylabel('f_X(k, N, \rho)');
subplot(3,1,3);
stem(k,f_x(3,:));legend('\rho = 0.9  e  N = 20');
xlabel('k');ylabel('f_X(k, N, \rho)');
```

[17] As combinações de N elementos k a k também pode ser representado por C_k^N.

onde

```
function f_x=binomial(N,k,rho)
f_x=combinacao(N,k).*rho.^ k.*(1-rho).^(N-k)
```

e

```
function C=combinacao(N,k)
C=factorial(N)./(factorial(k).*factorial(N-k));
```

Note ainda que o cálculo do factorial de n é uma operação computacionalmente intensiva. No entanto, a carga operacional pode ser aliviada atendendo à aproximação de Stirling dada por:

$$n! \approx n^n e^{-n} \sqrt{2\pi n} \tag{2.59}$$

Voltando à expressão (2.57), esta pode ter a seguinte interpretação: pretendem-se k sucessos e $N-k$ insucessos. No entanto os k sucessos podem ocorrer de muitas maneiras diferentes ao longo das N experiências. Efectivamente existem C_k^N maneiras diferentes desses sucessos poderem acontecer.

A figura 2.8 apresenta o aspecto da distribuição binomial agora em função da probabilidade ρ e admitindo $N = 2$ e $k = 1$.

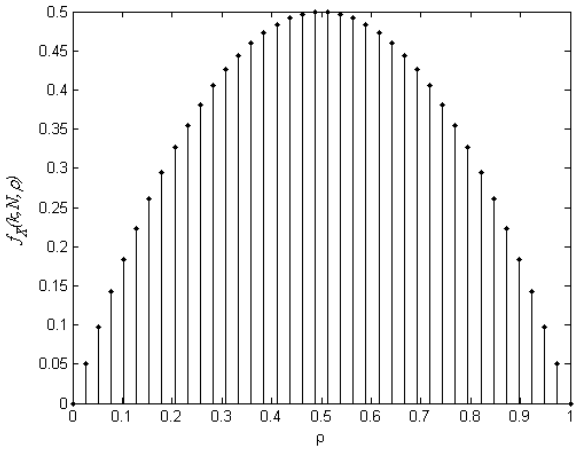


Figura 2.8: Aspecto da função massa de probabilidade, apresentada na equação (2.57) ($N = 20$ e para $\rho = \{0.1, 0.5, 0.9\}$.

Este gráfico pode ser entendido tendo por base uma situação concreta. Por exemplo qual a probabilidade de, numa sequência de dois bit, um deles ter o valor lógico '1'. Se a probabilidade do emissor gerar o bit '1' for 0.5 então a probabilidade de se obter um, e apenas um, dos bit a '1' é máxima e igual a 0.5. Para valores de ρ acima ou abaixo de 0.5 verifica-se uma diminuição da probabilidade de ocorrer **exactamente** um único bit a '1' numa sequência de dois bit. Para os casos extremos em que $\rho = 0$ e $\rho = 1$ a probabilidade disso acontecer é zero. No primeiro caso porque o emissor não consegue gerar um bit

'1' e no segundo caso porque não consegue gerar um bit '0'. Fique atento ao perfil deste gráfico pois será novamente observado, adiante, quando se falar em entropia no contexto da teoria da informação.

Mais uma vez, por razões pedagógicas, apresenta-se na caixa de texto que se segue o código para Matlab que pemitiu traçar o gráfico da figura 2.8:

```
N=2;k=1;
rho=linspace(0,1,40);
for i=1:40,
   f_x(i,:)=binomial(N,k,rho(i));
end
stem(rho,f_x);xlabel('\rho');
ylabel('f_X(k,N,\rho)');
```

É possível definir a média e variância de uma variável binomial. O primeiro momento estatístico de uma variável aleatória X, que siga uma distribuição binomial de parâmetros k, N e ρ, é dado por:

$$\mathfrak{E}(X) = N \cdot \rho \tag{2.60}$$

e a sua variância é:

$$\sigma^2 = N \cdot \rho \cdot (1 - \rho) \tag{2.61}$$

Finalmente, em termos de notação, uma variável aleatória X que siga uma distribuição binomial com parâmetros N e ρ, é representada por:

$$X \rightarrow \ Bin(n, \rho) \tag{2.62}$$

Com isto se dá por concluído o segundo capítulo deste documento. Estando na posse dos conceitos essenciais de estatísticas e análise de probabilidades o capítulo que se segue trata de uma parte fundamental da teoria da informação que é a codificação de canal: uma forma de aumentar a fiabilidade da transmissão de informação sobre canais não-fiáveis. Como se verá, a matéria estudada neste capítulo irá revelar-se essencial para a completa compreensão da matéria do capítulo 3.

Capítulo 3

Codificação de Canal

"If you understand what you don't understand help me understand." [1]

– David MacKay

Este capítulo trata de um dos problemas fundamentais da Teoria da Informação: a comunicação fiável através de um canal de comunicação não-fiável. Este problema é contornado através da adição, à mensagem a transmitir, de informação redundante. Essa informação permite ao receptor detectar a existência de erros e, eventualmente, poder corrigi-los. A esta estratégia de controlo de erros é dado o nome de codificação de canal. Este capítulo explora sobretudo uma forma particular de codificação designada por código de bloco.

3.1 Introdução

O pai da Teoria da Informação é Claude Shanon. Enquanto funcionário da AT&T publicou, em 1948, o artigo "A Mathematical Theory of Communication"[6]. Nesse artigo equacionou e resolveu, do ponto de vista teórico, a maior parte dos problemas relativos à transmissão da informação. Em suma o artigo tratava sobre a problemática da **comunicação fiável** sobre **canais não-fiáveis**. Toda a teoria foi desenvolvida em abstracto e não particularizada para um tipo específico de sistema de comunicação. Ou seja, independentemente da forma de comunicar a teoria é aplicável. Na sua forma mais simples um sistema de comunicação possui sempre um **emissor**, um **receptor** e um canal de **comunicação**.

- Por exemplo a comunicação verbal entre duas pessoas: A voz de uma

[1] Frase proferida, pelo Dr. MacKay, no decorrer de uma palestra (disponível *on-line* em `http://videolectures.net/mlss09uk_mackay_it`). Um exemplo perfeito de um sistema de detecção e correcção de erros...

(emissor) propaga-se através do ar (canal de comunicação) até aos ouvidos da outra (receptor).

- O MODEM[2] de um computador (emissor) envia uma string de bits modulada, através de um par de fios de cobre (canal de comunicação), até ao MODEM instalado num outro computador (receptor).
- Uma sonda espacial (emissor) envia imagens de um planeta longínquo através do vácuo (canal de comunicação) até ao planeta Terra (receptor).
- Um ficheiro é guardado num suporte magnético, por exemplo um disco rígido (canal de comunicação), sendo acedido num instante posterior.
- Durante o procedimento de *swap file* o conteúdo numa área da memória do computador é escrita num disco rígido (canal de comunicação) e, mais tarde, esse mesmo conteúdo voltará a ser carregado em memória. Neste caso o emissor e o receptor é a mesma entidade. No entanto a emissão e a recepção ocorrem em instantes temporais distintos.

Independentemente do canal em questão, todos eles possuem a propriedade de subverter a mensagem. Coloca-se um sinal a transmitir e o sinal que se recebe não é exactamente igual ao que se emite. Por exemplo na transmissão *wireless* os *quanta* são absorvidos e dispersos pelo meio que atravessam. No caso de comunicação por cabo, o ruído térmico, elementos parasitas armazenadores de energia (condensadores e indutores), campos magnéticos variantes no tempo que induzem tensões que se sobrepõem ao sinal de interesse entre muitos outros fenómenos descaracterizam o sinal emitido. O ideal associado a uma transmissão fiável pode resumir-se ao seguinte: o sinal enviado deve ser igual ao sinal recebido. A figura 3.1 ilustra o problema da comunicação fiável. Se M se referir à mensagem enviada pelo emissor e r for a quantidade de ruído adicionada à mensagem pelo canal de comunicação então a mensagem recebida pelo receptor é $\hat{M} = M + r$ (uma versão alterada da mensagem original).

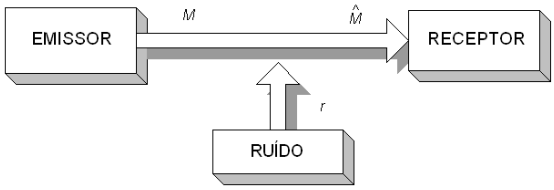


Figura 3.1: Modelo genérico de transmissão de uma mensagem do emissor, para o receptor, através de um canal imperfeito.

Para que a mensagem que deixa o emissor seja igual à que chega ao receptor, i.e. $\hat{M} = M$ é necessário que $r = 0$. No entanto a redução de r nem sempre é possível ou fácil de conseguir. Uma alternativa consiste em aumentar a "potência" do sinal a transmitir. Se $M \gg r$ então $M + r \approx M$ e logo $\hat{M} \approx M$. Daqui se conclui que uma das maneiras de aumentar a fiabilidade da comunicação consiste em transmitir a mensagem M sobre um sinal com uma potência muito maior do que a potência do sinal de ruído. A figura de mérito

[2]MODEM é o acrónimo de MOdulator-DEModulator, um sistema que permite transmitir informação binária sobre a rede telefónica.

que ilustra a diferença relativa entre as duas potências designa-se por relação sinal/ruído. A relação sinal/ruído, que se designará por SNR[3], é normalmente expressa em decibel (dB) a partir da seguinte expressão:

$$SNR_{dB} = 10 \times \log_{10}\left(\frac{P_{sinal}}{P_{ruido}}\right) \tag{3.1}$$

Por exemplo se $P_{sinal} = P_{ruido}$ então $SNR_{dB} = 0\,dB$. Se, por outro lado, $P_{sinal} = 100 \times P_{ruido}$ então $SNR_{dB} = 20\,dB$. Finalmente se a potência do ruído é 100 vezes superior à potência do sinal, o $SNR_{dB} = -20\,dB$. Conclui-se assim que a relação sinal/ruído deve ser o maior possível. Na eventualidade da potência do sinal não poder ser controlada basta aumentar a potência do sinal para que a transmissão seja realizada de forma mais fiável. No entanto esta estratégia possui diversos problemas entre os quais os limites físicos do canal de comunicação (por exemplo não é possível transmitir um sinal com uma potência de 1KW por um cabo de cobre de 0.5 mm de diâmetro) e a energia despendida durante a transmissão (o aumento da potência do sinal enviado por um satélite envolve custos económicos elevados).

Se for impossível a eliminação da fonte de ruído e se não for viável o aumento do SNR da fonte restam duas alternativas:

Alteração física do canal de comunicação: Envolve alterar a estrutura física do canal de comunicação. (ex. blindagens eléctricas).

Alteração do formato da mensagem: A mensagem enviada e alterada de modo a transportar consigo mecanismos que permitam ao receptor perceber se existiu um erro e, se possível, corrigi-lo.

A teoria da informação, enquanto disciplina, trata apenas da segunda das abordagens referidas. Para isso o sistema de comunicação ilustrado na figura 3.1 e expandido de modo a englobar dois blocos adicionais: um de codificação e outro de descodificação. A figura 3.2 apresenta essa nova estratégia.

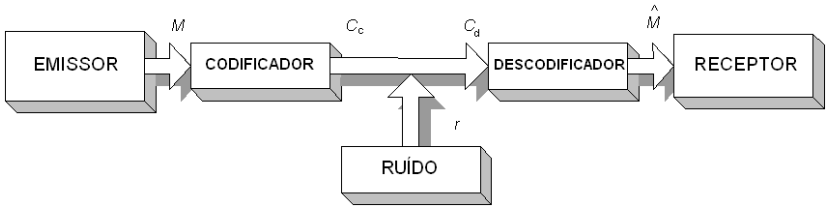


Figura 3.2: Aumento da fiabilidade de um canal de comunicação recorrendo a um par codificador/descodificador.

O emissor gera uma mensagem M. O codificador pega nessa mensagem e adiciona-lhe redundância transformando a mensagem original em C_c. Esta nova mensagem é enviada através do canal de comunicação sendo alterada, devido aos fenómenos já referidos, para $C_d = C_c + r$. Finalmente o descodificador explora a redundância de C_d e gera a mensagem $\hat{M}$ para o receptor. Se a comunicação ocorreu sem erros então $\hat{M} = M$.

[3]Do Inglês *Signal-to-Noise Ratio*

Adiante irá falar-se sobre as diversas formas como o codificador pode alterar a mensagem original. Para já deixa-se aqui o reparo que, se a mensagem M for uma sequência (string) binária com m bit o codificador expande essa sequência para n bit com $n > m$. É essa informação adicional que é acrescentada à mensagem que irá permitir ao descodificador perceber se existiu um erro na transmissão e, em alguns casos, recuperar essa informação.

3.2 Modelo do Canal: o canal binário simétrico

Dependendo do tipo de sinal envolvido no transporte da mensagem assim o canal desenvolve diferentes comportamentos. Por exemplo uma sequência de bit enviados sobre um par entrançado de fios de cobre terão diferentes aspectos se forem enviados em banda base (como acontece com a rede de dados Ethernet) ou através de modulação (como acontece com a transmissão de sinais digitais através da rede telefónica). Nesta unidade curricular analisa-se com algum cuidado um modelo do canal de transmissão designado por **canal binário simétrico**. O diagrama associado a este tipo de modelo encontra-se representado na figura 3.3.

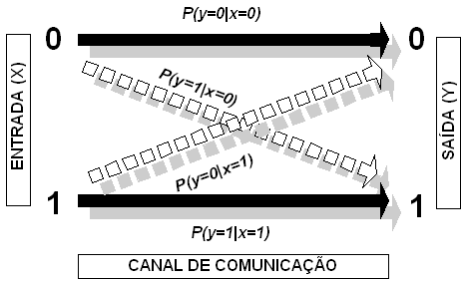


Figura 3.3: Modelo para um canal binário.

Neste modelo a transmissão de um bit ao longo de um canal pode ser alterado. Essa alteração é estocástica e apenas se admite o conhecimento da distribuição de probabilidade do canal. Se o modelo é simétrico a probabilidade de ocorrência de um erro se o bit enviado é '1' é igual à probabilidade de erro para o caso do bit enviado ser '0'. No diagrama representado na figura 3.3 essas probabilidades são, respectivamente, $P(y = 0|x = 1)$ e $P(y = 1|x = 0)$ que, na suposição de simetria, possuem o mesmo valor.

3.3 Caso de estudo: fiabilidade de um disco rígido

Para tornar mais claro o problema da transmissão fiável sobre canais reais admite-se a seguinte situação [4]:

Um determinado processo de fabrico de discos rígidos é capaz de armazenar informação com uma taxa de erro de 10%. Admite-se que o processo de lei-

[4] Adaptado do exemplo apresentado pelo Dr. MacKay na palestra sobre Teoria da Informação disponível *on-line*.

tura/escrita pode ser modelado por um canal binário simétrico com o seguinte aspecto:

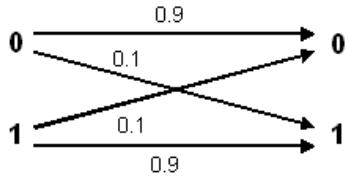


Figura 3.4: Modelo do sistema de leitura/escrita de um disco rígido.

Questão 1 Suponha-se que 10000 bit são armazenados no disco rígido. No instante seguinte o mesmo número de bit são lidos. Quantos bit se **espera** encontrar errados?

R:

A distribuição estatística subjacente é binomial com parâmetros $n = 10000$ e $\rho = 0.1$. Se a variável aleatória X representar o numero de bit errados lidos do disco rígido, o valor esperado de erros é, pela definição:

$$\begin{aligned} E[X] &= n \times \rho \\ &= 10000 \times 0.1 = 1000 \,\text{bit} \end{aligned} \tag{3.2}$$

Com um desvio padrão de:

$$\begin{aligned} \sigma_X &= \sqrt{n \times \rho \times (1-\rho)} \\ &= \sqrt{10000 \times 0.1 \times 0.9} = 30 \,\text{bit} \end{aligned} \tag{3.3}$$

Admitindo que o valor esperado está, dentro de um certo grau de certeza, a uma distância de dois desvios-padrão então $X = 1000 \pm 60$ bit.

Questão 2 Para poder ser um produto comercializável qual deverá ser o valor máximo da probabilidade de erro para um disco de 1 GB (gigabyte S.I.)?

R:

A resposta a esta questão é subjectiva. Depende do critérios de qualidade. Admitindo que o comprador efectua uma operação de leitura/escrita de 1 GB por dia, e se se pretende que o tempo esperado sem problemas de operação seja de cinco anos então apenas se admite um bit de erro em $8 \times 10^9 \times 365 \times 5 = 1.46 \times 10^{13}$ ou seja uma probabilidade de erro em torno de $1/10^{13}$. Se forem vendidos, não um, mas um milhão de discos a probabilidade deve diminuir para garantir que todos os clientes ficam satisfeitos dentro do horizonte temporal definido. Ou seja o número de bit transaccionados são agora $8 \times 10^9 \times 365 \times 5 \times 1 \times 10^6 \approx 10^{19}$ a probabilidade de erro em 1 bit deverá estar perto de $1/10^{19}$. Na prática espera-se uma probabilidade de erro entre os dois valores calculados. Concretamente, o padrão industrial impõe, como fasquia, um erro igual ou inferior a $1/10^{18}$.

O derradeiro problema consiste em transformar o disco com probabilidade de erro igual a 10% e transformá-lo, num disco que cumpra os padrões de fabrico internacionais. A forma melhorar a prestação consiste em dotar o disco com um codificador/descodificador. Existe uma miríade de formatos para esses mecanismos. Vamos começar por analisar um dos método mais simples: o registo dos bit com redundância.

Uma das formas de minimizar os erros de transmissão envolve redundância. Ou seja a repetição da informação a transmitir. Uma das formas mais comuns de codificação segundo esta estratégia é a redundância R3. A mensagem enviada pela fonte é triplicada pelo codificador antes de ser enviada, através do canal, para o receptor. Quando o emissor quer enviar '0' o codificador envia pelo canal a sequência '000' e, quando quer enviar '1' o codificador transmite '111'. Uma mensagem $M = 01101$ enviada pelo emissor passa a ser transmitida pelo codificador como $C_c = 000111111000111$.

No outro extremo do canal de comunicação encontra-se o descodificador. Este transforma a sequência recebida C_d e transforma-a na mensagem $\hat{M}$ para o receptor. Ou seja efectua uma tarefa de inferência. Quando uma codificação R3 é utilizada o descodificador agrupa os bits da mensagem recebida em grupos de 3 e utiliza um sistema de descodificação por "votação". Se em cada grupo existirem mais '1' do que zero então esses três bit são substituídos por um único bit com valor lógico '1'. Se, por outro lado, existirem dois ou mais bit a '0' no grupo então o descodificado substitui esse terceto por '0'. Na tabela que se segue apresenta-se o exemplo de descodificação de algumas situações:

Codificado	Descodificado
000	0
110	1
010	0

Efectivamente a escolha do bit por esta estratégia segue uma estratégia de máxima verosimilhança. Por exemplo para a segunda linha da tabela 3.3 calcule-se as probabilidades do valor transmitido ser '0' e '1':

$$P(M = 0|C_d = 110) = \frac{P(C_d = 110|M = 0)P(M = 0)}{P(C_d = 110)} \tag{3.4}$$

onde

$$P(C_d = 110) = P(C_d = 110|M = 0)P(M = 0) + P(C_d = 110|M = 1)P(M = 1) \tag{3.5}$$

Como,

$$P(C_d = 110|M = 0) = 0.1 \times 0.1 \times 0.9 = 0.009 \tag{3.6}$$

e

$$P(C_d = 110|M = 1) = 0.9 \times 0.9 \times 0.1 = 0.081 \tag{3.7}$$

Admitindo ainda que a probabilidade do emissor enviar '0' é igual à probabilidade de enviar '1' obtém-se:

$$P(C_d = 110) = 0.009 \times 0.5 + 0.081 \times 0.5 = 0.045 \tag{3.8}$$

o que leva a:

$$\begin{aligned} P(M=0|C_d=110) &= \frac{0.009 \times 0.5}{0.045} \\ &= 0.1 \end{aligned} \tag{3.9}$$

Agora para $M = 1$,

$$P(M=1|C_d=110) = \frac{P(C_d=110|M=1)P(M=1)}{P(C_d=110)} \tag{3.10}$$

o que leva a:

$$\begin{aligned} P(M=1|C_d=110) &= \frac{0.081 \times 0.5}{0.045} \\ &= 0.9 \end{aligned} \tag{3.11}$$

Com esta estratégia, para que haja erro em um bit é necessário que dois ou mais bit sejam alterados ao longo da transmissão. Considerando ainda o problema do disco rígido, atendendo a que a probabilidade de erro é 10%, uma estratégia de codificação R3 aplicada ao sistema reduz essa probabilidade para cerca de 3%. Esse cálculo é feito da seguinte forma: Se X for a ocorrência de erro no bit recebido e se $\{E_2, E_3\}$ forem dois eventos em que o primeiro reflecte a existência de dois bit alterados pelo canal e o segundo todos os bit foram alterados pelo canal então:

$$\begin{aligned} P(X) &= P(E_2) + P(E_3) \\ &= \left(\begin{array}{c} 3 \\ 2 \end{array} \right) 0.1^2 0.9 + \left(\begin{array}{c} 3 \\ 3 \end{array} \right) 0.1^3 0.9^0 \\ &= 0.027 + 10^{-3} = 0.028 \quad (2.8\%) \end{aligned} \tag{3.12}$$

Se bem que a probabilidade de erro ainda esteja muito acima do valor desejado verificou-se uma diminuição em cerca de 1/3 o valor inicial do erro. No entanto não é possível conseguir "alguma coisa" por "nada". O que se ganha em robustez perde-se em taxa de transmissão. Agora existem três vezes mais bits para transmitir do que na situação original.

Questão Qual deverá ser o valor de i (ímpar) para um sistema de codificação Ri de modo a que se atinja uma probabilidade de erro igual ou inferior a 10^{-18}?

R:

Viu-se anteriormente que, para $i = 3$ a probabilidade de erro consistia na probabilidade de se alterarem 2 ou mais bit. Aumentando i para 5 agora a probabilidade de erro consiste na probabilidade de se alterarem 3 ou mais bit. Para um número arbitrário ímpar i trata-se de calcular a probabilidade de se alterarem $(i + 1)/2$ bits. Neste caso a probabilidade de erro é:

$$P(E) = \sum_{k=\frac{(i+1)}{2}}^{i} C_k^i \rho^k (1-\rho)^{(i-k)} \tag{3.13}$$

Vamos utilizar o Matlab para este procedimento. Para isso executa-se um ciclo de operações (3.13) onde se varia o valor de i de 3 até, por exemplo 71. Para cada valor de i calcula-se a probabilidade de erro e guarda-se o valor num vector designado por P.

```
clear all;clc
ro=0.1
ptr=1;
for n=3:2:81,
   P(ptr)=0;
   for i=(n+1)/2:n,
      C=nchoosek(n,i);
      P(ptr)=P(ptr)+C*ro^i*(1-ro)^(n-i);
   end
   ptr=ptr+1;
end
n=3:2:81;
plot(n,log(P),'k',n,log(1/10^18)*ones(length(n),1),'r:');
xlabel('Redundância (R-i)');ylabel('log(Probabilidade)');
```

O resultado da execução deste código leva ao resultado ilustrado pelo gráfico que se segue:

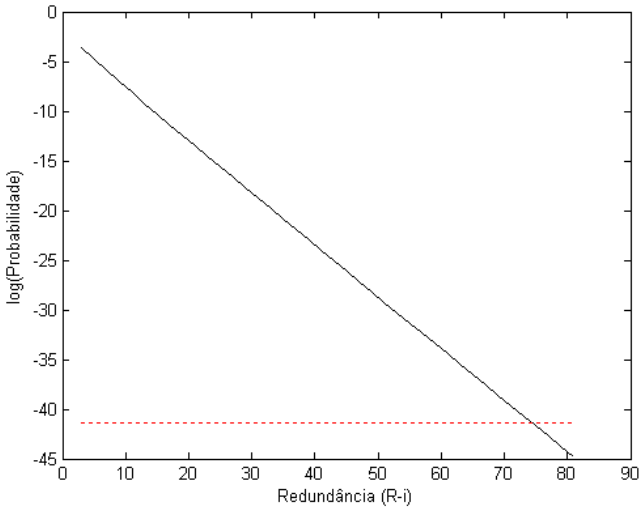


Figura 3.5: Probabilidade de erro face ao aumento da redundância.

De onde se conclui que o menor valor para i que satisfaz a condição de erro é 75. O que significa que cada bit tem que ser escrito 75 vezes. Em termos práticos o disco de 1GB apenas possui disponível para o utilizador pouco mais de 13 MB (SI). A figura que se segue representa a relação entre probabilidade de erro e razão de informação.

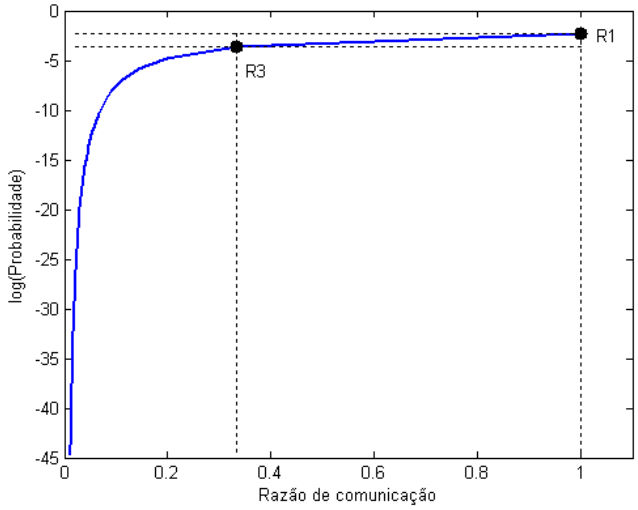


Figura 3.6: Gráfico do logaritmo da probabilidade de erro face à razão de informação.

Conforme se pode constatar a diminuição da probabilidade de erro conduz a uma diminuição da taxa de transmissão, i.e. o número de bits a transmitir aumenta. Ou seja a mesma mensagem demora mais tempo a transmitir pois possui maior número de bit. Este resultado sugere que a utilização de redundância não resolve o problema relativo à probabilidade de erro do disco rígido. Desta forma métodos alternativos devem ser explorados.

3.4 Estratégias para a codificação de canal

Existem diversos métodos que podem ser utilizados para diminuir o erro de transmissão. Alguns métodos assentam na retransmissão da informação. Ou seja se o receptor detecta um erro na mensagem pede ao emissor que retransmita. Este modo de controlo de erro é uma forma especial de redundância. Em alguns tipos de estratégias o receptor envia, de volta ao emissor, uma mensagem de confirmação da mensagem. Este método designa-se por **ARQ** - **A**utomatic **R**epeat re**Q**uest. Se o receptor não receber uma confirmação durante um intervalo de tempo (timeout) reenvia a mensagem até que receba a *frame* ou pacote de confirmação ou até que um número máximo de retransmissões seja excedido.

O problema com este tipo de estratégia reside na necessidade de existir comunicação *full-duplex* ou seja o receptor deve ser capaz de comunicar, em tempo-real, com o emissor. Em muitas situações isso não é possível. O receptor não se encontra em contacto com o emissor e, deste modo, não há forma de este requisitar nova transmissão em caso de erro. Nestes casos a própria mensagem deve transportar consigo mecanismos que dotem o receptor com a capacidade de detectar e eventualmente corrigir os erros. A esta forma de transmissão é chamado de **FEC** - **F**orward **E**rror **C**orrection. Os sinais digitais utilizam o FEC em dois métodos principais[5]:

[5]Efectivamente existem ainda os códigos Turbo computacionalmente mais eficientes do que

Códigos de Bloco Operam em "blocos" de bit recorrendo a um determinado algoritmo. A mensagem original é expandida, i.e. a sua dimensão aumenta, com informação adicional que permite a detecção e correcção de erros.

Códigos Convolucionais Tal como no caso dos códigos de bloco, os códigos convolucionais aumentam o comprimento da mensagem. No entanto, neste caso, a operação de transformação não envolve apenas a mensagem actual mas também as últimas recebidas até uma profundidade arbitrária.

A secção que se segue trata da codificação por códigos de bloco. Introduz-se o conceito de distância de Hamming e apresentam-se alguns exemplos.

A fonte discreta gera informação sob a forma de símbolos binários. O codificador de canal aceita a mensagem e adiciona-lhe redundância de acordo com uma regra pré-determinada. O descodificador de canal explora a redundância de modo a decidir quais os bits que foram efectivamente transmitidos. O objectivo do par codificador/descodificador é minimizar o efeito do ruído no canal de comunicação.

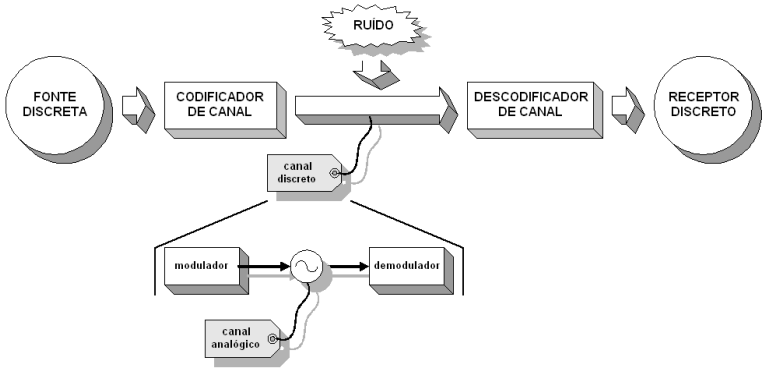


Figura 3.7: XXXXXXXXXX

Existem diversas formas de implementar a codificação e descodificação de canal. De forma grosseira pode-se classificar esses códigos em duas famílias:

- Códigos de bloco.
- Códigos convolucionais.

O que os distingue é a presença ou ausência de memória no codificador. Por exemplo num código de bloco (n, k) o codificador de canal recebe *blocos* de k bit e transforma-os em blocos de n bit. Já num código convolucional a operação de codificação

O codificador de um código convolucional opera na mensagem enviada pela fonte usando uma "janela deslizante" com duração igual à sua memória. Num código convolucional o codificador de canal aceita os bits da fonte como uma sequência contínua gerado, simultâneamente, uma sequência contínua de bit codificados.

os códigos convolucionais e que são presentemente o estado da arte em sistemas de codificação.

3.5 Códigos ARQ

Automatic **R**epeat re**Q**uest (ARQ), é um método de controlo de erro em sistemas de transmissão de dados que utiliza mensagens enviada pelo receptor indicando se uma mensagem foi correctamente recebida ou não. Definem-se ainda intervalos de tempo para que a mensagem de confirmação seja recebida pelo emissor. Se o emissor não receber uma confirmação (*acknowledge*) antes do tempo terminar este re-envia novamente a mesma mensagem. Este procedimento é repetido até que seja recebido uma confirmação do receptor ou então um número pré-definido de tranmissão tenha ocorrido.

Os códigos ARQ só podem ser usados em situações em exista uma via de comunicação entre o receptor e o emissor. Seja simultânea (*full-duplex*) ou partilhada (*half-duplex*). Numa transmissão *full-duplex* existe a possibilidade de comunicação simultânea entre o emissor e o receptor e vice-versa. Num canal *half-duplex* é necessário esperar que um dos extremos termine a transmissão para o outro poder transmitir.

Esta estratégia de controlo e correcção de erro utiliza a redundância da palavra-código apenas para detectar a existência de erros na transmissão.

3.5.1 Stop-and-Wait

Neta estratégia o emissor envia uma mensagem e activa um temporizador e aguarda por um sinal de confirmação transmitido pelo receptor. Se o sinal de confirmação chegar antes de terminar o tempo imposto pelo temporizador, o emissor envia nova mensagem re-inicializando o temporizador. Se a confirmação não chegar antes do temporizador atingir o fim da contagem o emissor re-envia novamente a mesma mensagem.

Para que o receptor detecte o erro na transmissão, o emissor envia a mensagem com redundância que permita essa detecção. No caso de serem detectados erros o receptor descarta a mensagem e aguarda pelo re-envio.

Para além da redundância, o emissor marca cada nova mensagem com um cabeçalho composto por um bit que alterna. Esta estratégia permite evitar que o receptor se confunda no caso do sinal da confirmação da mensagem anterior se perder. Se for este o caso o emissor re-envia a mesma mensagem mas o receptor está à espera de uma nova. Com a adição deste bit, se o cabeçalho da nova mensagem tiver o mesmo valor lógico do cabeçalho da mensagem anterior então o receptor sabe que se trata do reenvio da mensagem anterior e re-envia a confirmação.

Comparativamente às outras formas de ARQ, o *stop-and-wait* é ineficiente devido à baixa taxa de transmissão conseguida. De forma a melhorar o desempenho apresentam-se duas técnicas alternativas que conseguem atingir taxas de transmissão mais elevadas enviando, em cada vez, mais do que uma mensagem.

3.5.2 Go-back-N

Ao contrário do stop-and-wait, o ARQ *go-back-N* envia um número específico N de mensagens sem aguardar pela confirmação do receptor. Por outro lado o receptor vai tomando nota da próxima mensagens a receber e envia essa informação ao emissor. No caso de uma mensagem recebida não ser a esperada esta é ignoradas. No fim das N mensagens o emissor toma conhecimento de

qual a última confirmação e envia N mensagens a partir dessa. Com esta estratégia consegue-se maiores taxas de transmissão mas podem ocorrer re-envios de mensagens transmitidas sem erro. Por esse motivo uma nova alternativa foi projectada designada por *selective repeat*.

3.5.3 Selective Repeat

Ao contrário do protocolo *go-back-N*, o receptor no ARQ-SR continua a aceitar mensagens após um erro e enviando confirmação dessas mensagens. O receptor toma nota do número da última mensagem incorrecta e envia-o em cada sinal de confirmação. Assim que o emissor enviar as N mensagens re-envia as mensagens cujos números são fornecidos nos sinais de confirmação recebidos.

3.6 Códigos FEC

A filosofia subjacente aos códigos FEC é substancialmente diferente daquela definida para os códigos ARQ. Como se viu os códigos ARQ utilizam a redundância da palavra-código apenas para a detecção de erros. No caso de um erro ter sido detectado o receptor envia um pedido de retransmissão ao emissor. Por outro lado os códigos FEC assentam na redundância no código transmitido tanto para a *detecção* como a *correcção* de erros. Admite-se comunicação *simplex* entre o emissor e o receptor.

3.6.1 Códigos de Bloco Lineares

Se uma mensagem (sequência de k bit) for vista como um vector, os códigos de bloco efectuam um mapeamento desse vector num vector alternativo sobre um espaço de dimensionalidade n. Por esse motivo os códigos de bloco são referidos como códigos (n,k). Um dos exemplos é o código Hamming (7,4) que veremos adiante. Considere-se por exemplo o seguinte mapeamento:

00	11000000
01	00011110
10	01000100
11	10111011

uma mensagem $M = 1011$ será, neste caso, transmitida como $C_c = 0100010010111011$.
(compor)....

- n designa-se por comprimento do bloco.
- A palavra à saída do codificador designa-se por palavra-código [6].
- Se a fonte discreta produz uma mensagem com uma taxa de T_F bits-por-segundo (bit/s) o codificador irá reproduzir essa mensagem com uma taxa $\mathrm{T}_C = (k/n)\mathrm{T}_F$

[6] Do Inglês *codeword*

A fonte binária gera uma sequência à taxa de T_F bits-por-segundo. Este bits são agrupados pelo codificador de canal em blocos com dimensão k. A cada bloco é acrescentada $(n-k)$ bit de redundância produzindo uma palavra-código com n bit. A taxa de transmissão do codificador é $\mathrm{T}_C = (n/k)\mathrm{T}_F$ bit/s.

Um código é designado por **linear** se a soma, em módulo 2[7], de quaisquer palavras-código resulta numa terceira palavra-código que faz parte do código. Por exemplo a codificação R3 é um código linear pois.

–Melhorar - Acrescentar a necessidade de ter um vector 0000

Por exemplo a mensagem 1011 é codificada em 111000111111 e a mensagem 0001 em 000000000111. A soma módulo 2 das duas palavras-código é 111000111000. Esta última é uma palavra-código válida no sistema de codificação utilizado dado que existe repetibilidade de ordem 3 de cada bit. A *string* binária anterior é descodificada em 1010.

3.6.1.1 Estrutura e formato matricial

Num código de bloco (n, k) o codificador adiciona, aos k bit que compõem a mensagem, $n - k$ bits designados por bits de paridade ou redundância. A redundância é portanto o número de bits usados para transmitir uma mensagem para além daqueles efectivamente usados pela mensagem. A diferença entre o número de bits necessários pela mensagem e o numero de bits efectivamente transmitidos é chamado de *overhead*.

Se o código possui uma **estrutura sistemática** então a palavra-código é dividida em duas partes bem distintas. Uma parte é ocupada pela mensagem e a outra pelos bit de paridade. Na figura 3.8 representa-se esse conceito.

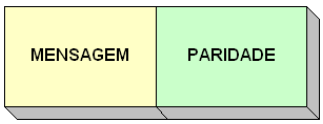


Figura 3.8: Estrutura de uma palavra-código sistemática.

Se $\{m_1, \cdot, m_k\}$ se referem aos bits relativos à mensagem M e se $\{t_1, \cdots, t_{n-k}\}$ se referem ao $(n-k)$ bit de paridade então a palavra-código é $\{c_1, \cdots, c_n\}$ onde,

$$c_i = \begin{cases} m_i, & i = 1, \cdots, k \\ t_{i-k}, & i = k+1, \cdots, n \end{cases} \tag{3.14}$$

Os valores lógicos dos bits de paridade são, de forma genérica, obtidos pela combinação linear dos bits da mensagem M, i.e.

$$t_i = \alpha_{1i} m_1 + \cdots \alpha_{ki} m_k \tag{3.15}$$

onde os coeficientes α_{ij} são definidos da seguinte forma:

$$\alpha_{ij} = \begin{cases} 1 & \text{se } t_i \text{ depende de } m_i \\ 0 & \text{caso contrario} \end{cases} \tag{3.16}$$

[7] A soma em módulo-2 e a soma binária desprezando o transporte, i.e. a operação xOr entre dois bit

Para além disso a operação '+' refere-se à adição módulo 2 (operação XOR) e $\alpha_{ij}m_i$ ao produto módulo 2 (operação AND).

Voltando à expressão 3.15, uma forma mais eficiente de a representar envolve o produto interno entre o vector $\mathbf{m} = [m_1 \cdots m_k]$ e a matriz dos coeficientes:

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_{11} & \cdots & \alpha_{1(n-k)} \\ \vdots & \ddots & \vdots \\ \alpha_{k1} & \cdots & \alpha_{k(n-k)} \end{bmatrix} \tag{3.17}$$

levando a,

$$\mathbf{t} = \mathbf{m} \cdot \boldsymbol{\alpha} \tag{3.18}$$

onde $\mathbf{t} = [t_1 \cdots t_{n-k}]$.

Se, adicionalmente, $\mathbf{c} = [c_1 \cdots c_n]$ for o vector palavra-código então,

$$\mathbf{c} = \mathbf{m} \cdot \mathbf{G} \tag{3.19}$$

onde $\mathbf{G}$ se designa por **matriz geradora** e é definida por:

$$\mathbf{G} = [\mathbf{I}_k | \boldsymbol{\alpha}] \tag{3.20}$$

ou seja é a concatenação de $\mathbf{I}_k$, a matriz identidade de dimensão $k \times k$, com a matriz dos coeficientes $\boldsymbol{\alpha}$ com dimensão $k \times (n-k)$.

Quando a matriz $\mathbf{G}$ se encontra no formato indicado pela expressão (3.20) diz-se que esta está na **forma padrão** [8]. No caso da matriz geradora, para um determinado código linear, não se encontrar na forma padrão é possível definir uma matriz $\mathbf{G}$ alternativa, na forma padrão, para um código equivalente ao primeiro. Essa segunda matriz $\mathbf{G}$ é obtida, da primeira, por operações elementares entre as linhas e colunas. As operações válidas são:

Permutação de colunas Um par de colunas da matriz $\mathbf{G}$ podem trocar de lugar. Se $\mathbf{G} = [\mathbf{g}_1 \cdots \mathbf{g}_i \cdots \mathbf{g}_n]$ para $\mathbf{g}_i$, com $i = 1, \cdots n$, vectores coluna então, por exemplo, a permutação entre as colunas 1 e i dá lugar a uma matriz $\mathbf{G}' = [\mathbf{g}_i \cdots \mathbf{g}_1 \cdots \mathbf{g}_n]$.

Permutação de linhas Outra das operações válidas é a troca de linhas. Se $\mathbf{G} = \left[\mathbf{g}_1^T \cdots \mathbf{g}_i^T \cdots \mathbf{g}_k^T\right]^T$ para $\mathbf{g}_i$, com $i = 1, \cdots n$, vectores linha então, por exemplo, a permutação entre as linhas 1 e i dá lugar a uma matriz $\mathbf{G}' = \left[\mathbf{g}_i^T \cdots \mathbf{g}_1^T \cdots \mathbf{g}_n^T\right]^T$.

Soma de linhas Uma linha de $\mathbf{G}$ pode ser substituída pela sua soma com outra linha qualquer da matriz geradora. Ou seja se $\mathbf{G} = \left[\mathbf{g}_1^T \cdots \mathbf{g}_i^T \cdots \mathbf{g}_k^T\right]^T$, $\mathbf{G}' = \left[\mathbf{g}_1^T \cdots \mathbf{g}_1^T + \mathbf{g}_i^T \cdots \mathbf{g}_k^T\right]$ é uma matriz geradora equivalente.

Um código $\mathcal{C}'$ obtido a partir de $\mathbf{G}'$, uma matriz gerador derivada, a partir de operações lineares, da matriz $\mathbf{G}$ é linearmente equivalente ao código $\mathcal{C}$. Por exemplo considere um código com a seguinte matriz geradora:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

[8] Do Inglês *standard form*

Esta matriz não se encontra na forma padrão e as palavras-código geradas por ela **não são sistemáticas**. Por isso começa-se por somar a linha 1 com a linha 2 resultando em:

$$\mathbf{G}' = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

agora a linha 2 passa a ser igual à soma das linhas 2 e 4:

$$\mathbf{G}' = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.23}$$

Nova operação de soma. Agora a linha 1 ficará com o resultado da soma entre as linhas 1 e 3.

$$\mathbf{G}' = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.24}$$

Traçando agora as colunas da matriz anterior obtém-se:

$$\mathbf{G}' = \begin{bmatrix} \underbrace{\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}}_{I_4} & \begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{matrix} \end{bmatrix} \tag{3.25}$$

Que, como se pode comprovar, se encontra na forma padrão. Note que a mínima distância de Hamming do código original e do novo código sistemático é a mesma ($d_H^* = 3$). Para além disso existe os histogramas dos pesos de Hamming de ambos os códigos é exactamente a mesma.

Define-se ainda a **matriz de teste de paridade H** como sendo:

$$\mathbf{H} = \left[\boldsymbol{\alpha}^T | \mathbf{I}_{n-k}\right] \tag{3.26}$$

Prova-se que, se a palavra-código for correcta, então:

$$\mathbf{c} \cdot \mathbf{H}^T = \mathbf{0} \tag{3.27}$$

A matriz **G** é utilizada na operação de codificação do transmissor e a matriz **H** é utilizada durante a operação de descodificação no receptor. A figura 3.9 ilustra este conceito.

No modelo de transmissão ilustrado na figura, o codificador envia o vector **c** e o descodificador recebe **d**. Numa transmissão sem erros ambas as palavras-código são idênticas. Por outro lado, no caso de terem existido erros na comunicação, existem bit nas duas palavras que são distintos. Define-se o vector de erro **e** como sendo o vector obtido pela diferença (em módulo-2) entre as strings **c** e **d**.

$$\mathbf{e} = \mathbf{d} - \mathbf{c} \tag{3.28}$$

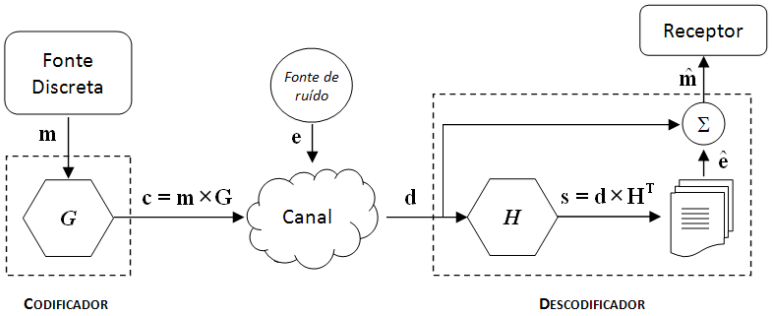


Figura 3.9: As matrizes **G** e **H** no contexto da codificação de canal.

se $\mathbf{d} = [d_1 \cdots d_n]$ e $\mathbf{c} = [c_1 \cdots c_n]$ então $\mathbf{e} = [e_1 \cdots e_n]$. Por exemplo se $\mathbf{c} = [1001]$ e $\mathbf{d} = [1\ 1\ 0\ 0]$ então $\mathbf{e} = [0\ 1\ 0\ 1]$. Os elementos a '1' no vector **e** representam os bits que sofreram mutação após a transmissão.

A jusante do canal de comunicação, o descodificador recebe **d** e recorre à expressão 3.27 para avaliar a palavra recebida:

$$\mathbf{d}\mathbf{H}^T \stackrel{?}{=} \mathbf{0} \tag{3.29}$$

Como $\mathbf{d} = \mathbf{c} + \mathbf{e}$ obtém-se:

$$\begin{aligned} \mathbf{d}\mathbf{H}^T &= (\mathbf{c} + \mathbf{e})\,\mathbf{H}^T \\ &= \mathbf{c}\mathbf{H}^T + \mathbf{e}\mathbf{H}^T \end{aligned} \tag{3.30}$$

como $\mathbf{c} \cdot \mathbf{H}^T = \mathbf{0}$ a expressão anterior resume-se a:

$$\mathbf{d}\mathbf{H}^T = \mathbf{e}\mathbf{H}^T \tag{3.31}$$

Atribui-se ao vector $\mathbf{d}\mathbf{H}^T$ a designação de **síndrome**. Efectivamente o síndrome não é mais do que o resultado do teste de paridade a um determinado vector. No caso do vector **d** o vector síndrome designa-se por $\mathbf{s_d}$ e calcula-se por:

$$\mathbf{s_d} = \mathbf{d} \cdot \mathbf{H}^T \tag{3.32}$$

Já para o vector **e** o síndrome denota-se por $\mathbf{s_e}$,

$$\mathbf{s_e} = \mathbf{e} \cdot \mathbf{H}^T \tag{3.33}$$

A igualdade apresentada na expressão (3.31) indica que o síndrome do vector recebido é igual ao síndrome do erro. Esta consideração é importante e, como veremos à frente, permite desenvolver um procedimento eficiente de descodificação.

Por exemplo admita-se um código (6,4) onde,

$$\boldsymbol{\alpha} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{3.34}$$

e logo,

$$H = \left[\begin{array}{cccc|cc} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{3.35}$$

e

$$\mathbf{G} = \left[\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right] \tag{3.36}$$

Se $\mathbf{m} = [1\ 0\ 1\ 1]$, e admitindo a inexistência de erros de transmissão,

$$\begin{aligned} \mathbf{c} = \mathbf{d} &= \mathbf{m} \cdot \mathbf{G} \\ &= \left[\begin{array}{cccccc} 1 & 0 & 1 & 1 & 1 & 0 \end{array}\right] \end{aligned} \tag{3.37}$$

O valor do síndrome é, neste caso,

$$\begin{aligned} \mathbf{s} &= \mathbf{d} \cdot \mathbf{H}^T \\ &= \left[\begin{array}{cccccc} 1 & 0 & 1 & 1 & 1 & 0 \end{array}\right] \left[\begin{array}{cc} 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{array}\right] \\ &= \left[\begin{array}{cc} 0 & 0 \end{array}\right] \end{aligned} \tag{3.38}$$

Admita-se agora a existência de um erro de transmissão no primeiro bit. Neste caso $\mathbf{d} = \left[\begin{array}{cccccc} 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right]$ e logo o síndrome é $\mathbf{s} = \left[\begin{array}{cc} 1 & 1 \end{array}\right]$. Os valores a '1' no síndrome indicam que ambos os valores de paridade são inválidos. Se, por exemplo, $\mathbf{d} = \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 1 & 0 \end{array}\right]$ o resultado é também $\mathbf{s} = \left[\begin{array}{cc} 1 & 1 \end{array}\right]$. Conclui-se assim que a informação contida no síndrome não é suficiente para que o descodificador descubra a mensagem exacta que foi transmitida.

Suponha-se que uma palavra-código $\mathbf{c}$ é transmitida por um canal. Admita-se ainda que esse canal introduz erro na mensagem enviada caracterizado por $\mathbf{e}$. Neste sentido, o código recebido pelo descodificador é $\mathbf{d} = \mathbf{c} + \mathbf{e}$. No caso de $\mathbf{d}$ não pertencer a $\mathcal{C}$, o descodificador apercebe-se da ocorrência de um erro de transmissão e pode tentar recuperar a palavra-código enviada pelo codificador. Uma possível estratégia parte da substituição da palavra-código inválida por uma pertencente ao alfabeto $\mathcal{C}$. A escolha da sequência estimada para a palavra-código enviada, $\hat{\mathbf{c}}$, pode ser feita de inúmeras formas. Uma delas parte do pressuposto que a probabilidade de terem ocorrido erro em muitos dos bit é muito menor do que a probabilidade de terem sido modificados, pelo canal, apenas um número restrito de bits. Neste sentido a escolha da palavra-código $\hat{\mathbf{c}}$ é feita no sentido de ser, de entro o universo das palavras-código válidas $\mathcal{C}$, aquela que tenha uma menor distância de Hamming à palavra-código recebida $\mathbf{d}$. Normalmente essa pesquisa é feita numa tabela, com 2^n entradas, designada por **matriz padrão**[9].

Uma forma equivalente, mas mais eficiente do ponto de vista computacional, recorre ao cálculo dos vectores de síndrome. Este método será descrito ao longo da secção 3.6.1.5.

[9] Do Inglês *standard array*

3.6.1.2 Distância de Hamming

Nesta secção define-se a distância entre duas palavras-código de um dado alfabeto. A métrica utilizada quando se fala de *string* binárias é a distância de Hamming. Pela importância que este conceito apresenta na teoria da informação seguem-se algumas definições:

Peso de Hamming $w_H(\mathbf{c})$: O peso de Hamming de uma palavra-código $\mathbf{c}$ é designado por $w_H(\mathbf{c})$ e refere-se ao número de bit a '1' na palavra $\mathbf{c}$. Por exemplo 1011 tem $w_H(\mathbf{c}) = 3$.

Mínimo peso de Hamming w_H^*: O mínimo peso de Hamming de um código de bloco é o menor peso da palavra-código não nula no universo $\mathcal{C}$ do código.

$$w_H^* = \min\{w_H(\mathbf{c}) : \mathbf{c} \in \mathcal{C} \wedge \mathbf{c} \neq \mathbf{0}\}$$

Distância de Hamming $d_H(\mathbf{c}_1, \mathbf{c}_2)$: Admitindo duas palavras-código, $\mathbf{c}_1$ e $\mathbf{c}_2$ com a mesma dimensão (numero de bit), a distância de Hamming entre essas duas palavras consiste no número de bit diferentes entre elas. Por exemplo se $\mathbf{c}_1 = 1011$ e se $\mathbf{c}_2 = 1100$ então $d_H(\mathbf{c}_1, \mathbf{c}_2) = 3$. A distância de Hamming pode ser entendida como sendo o peso de Hamming da operação XOR entre $\mathbf{c}_1$ e $\mathbf{c}_2$:

$$d_H(\mathbf{c}_1, \mathbf{c}_2) = w(\mathbf{c}_1 \text{ xor } \mathbf{c}_2) \tag{3.39}$$

- $w_H(\mathbf{c}) = d(\mathbf{0}, \mathbf{c})$
- $d_H(\mathbf{c}_1, \mathbf{c}_2) = w_H(\mathbf{c}_1 - \mathbf{c}_2)$
- $w_H(\mathbf{c}) = 0$ se, e só se, $\mathbf{c} = \mathbf{0}$

Mínima Distância de Hamming d_H^*: No universo de todas as palavras-código possíveis para uma dada codificação, a mínima distância de Hamming d_H^* e o menor valor da distância de Hamming entre quaisquer par de palavras-código (desde que $\mathbf{c}_1 \neq \mathbf{c}_2$).

Para ilustrar o conceito de distância mínima admita-e o conjunto de todas as palavras-código de 3 bit como se mostra na tabela 3.3.

Tabela 3.1: Código de 3 bit para uma mensagem com alfabeto de dimensão 8.

c	código	**c**	código
a	000	e	100
b	001	f	101
c	010	g	110
d	011	h	111

O número de combinação possíveis duas-a-duas pode ser calculado da seguinte forma:

$$\begin{pmatrix} 2^3 \\ 2 \end{pmatrix} = \frac{8!}{2!6!} = \frac{8 \times 7}{2} = 4 \times 7 = 28 \tag{3.40}$$

Todas as combinações possíveis, assim como as respectivas distâncias de Hamming, encontram-se tabeladas em 3.2.

Tabela 3.2: Todas as possíveis distâncias de Hamming para o alfabeto representado na tabela 3.3.

$\mathbf{c}_1\mathbf{c}_2$	$d(\mathbf{c}_1,\mathbf{c}_2)$	$\mathbf{c}_1\mathbf{c}_2$	$d(\mathbf{c}_1,\mathbf{c}_2)$	$\mathbf{c}_1\mathbf{c}_2$	$d(\mathbf{c}_1,\mathbf{c}_2)$	$\mathbf{c}_1\mathbf{c}_2$	$d(\mathbf{c}_1,\mathbf{c}_2)$
ab	1	bc	2	cd	1	de	3
ac	1	bd	1	ce	3	df	2
ad	2	be	2	cf	2	dg	2
ae	1	bf	1	cg	2	dh	1
af	2	bg	3	ch	1	ef	1
ag	2	bh	2	fg	2	eg	1
ah	3	gh	1	fh	1	eh	2

Por observação da tabela 3.2 verifica-se que $d_H^* = 1$. A mesma conclusão poderia ser obtida, sem a análise exaustiva de todas as combinações possíveis a partir do seguinte teorema:

Teorema 1: *Para qualquer* ***código linear*** *a mínima distância d_H^* é igual a peso mínimo w_H^*.*

Neste caso, observando a tabela 3.3 observa-se que o menor valor de $w(\mathbf{c})$ é igual a 1 confirmando o teorema anterior.

Admita-se agora que o código é constituído apenas pelas palavras-código d, f, g. De acordo com o lema anterior prevê-se que $d_{min} = 2$. Para o comprovar listam-se todas as combinações das palavras anteriores tomadas duas-a-duas.

Tabela 3.3: Todas as possíveis distâncias de Hamming entre os códigos d, f e g.

$\mathbf{c}_1\mathbf{c}_2$	$d(\mathbf{c}_1,\mathbf{c}_2)$
df	2
dg	2
fg	2

`Questão: Porque é que o teorema não é aplicável ao código formado por` d, f, g `e` h`?`

A aplicação do teorema anterior, a códigos de bloco lineares, permite a determinação mais rápida, em termos computacionais, da distância d_H^* quando comparada com a pesquisa exaustiva utilizando a definição. Efectivamente a procura da distância mínima é um problema classificado como *NP-Hard*[10].

Outro teorema evidencia a relação entre a matriz de paridade $\mathbf{H}$ e o mínimo peso de Hamming:

Teorema 2: *O mínimo peso de Hamming, w_H^*,* de um código de bloco linear é igual ao menor número de colunas linearmente dependentes da matriz de paridade $\mathbf{H}$

[10] O tempo de execução aumenta de forma não-polinomial com a dimensão do problema

O conhecimento de d_H^* de um código de bloco linear é importante pois esse valor indica-nos a capacidade desse código detectar erros. Efectivamente, o número máximo de bit errados (n_d) que podem ser detectados, num código com d_H^* é:

$$n_d = d_H^* - 1 \tag{3.41}$$

Por exemplo considere-se novamente o código estabelecido na tabela 3.1. O valor de d_H^* neste caso é 1 o que implica, de acordo com a igualdade (3.41), $n_\epsilon = 0$ ou seja é impossível, para o receptor, detectar a existência mesmo de um bit errado. Facilmente se comprova esta afirmação. Imagine-se que se transmite o código 011 relativo a d. Admita-se que um dos bit é alterado pelo canal de comunicação. Por exemplo a mensagem recebida pelo receptor é 001. Este converte a mensagem recebida em b não se apercebendo do erro. Considere-se agora o código composto pelas palavras-código d, f e g. Viu-se que a mínima distância de Hamming é, neste caso, igual a 2. De acordo com a expressão (3.41) este código permite a detecção de um bit errado. Para o comprovar imagine-se que se envia novamente d. O erro em um bit pode levar o receptor a ler uma de três possíveis mensagens: 111, 001 e 010. Como nenhuma delas pertence ao alfabeto do código, o receptor não consegue descodificar a mensagem, sinalizando a ocorrência de erro.

Uma das características de um código FEC é a capacidade, não só de detectar erro nas mensagens, mas também possuir a capacidade de os corrigir. O número máximo de bits que podem ser corrigidos, n_c, com um código também depende de d_H^*. Mais concretamente,

$$n_c = \left\lfloor \frac{d_H^* - 1}{2} \right\rfloor \tag{3.42}$$

onde $\lfloor \cdot \rfloor$ se refere ao arredondamento, por defeito, ao inteiro mais próximo. A partir desta expressão pode-se concluir que, para que um código possua a capacidade de corrigir, pelo menos um bit errado, a distância de Hamming mínima deve ser superior ou igual a 3. Ambos os códigos apresentados não admitem correcção de erro. O primeiro porque $d_H^* = 1$ e o outro porque $d_H^* = 2$. Por isso mesmo vamos inventar um código, para o alfabeto apresentado na tabela 3.1, de modo a que a distância mínima seja 3.

3.6.1.3 Como inventar um código?

O código apresentado na tabela 3.1 possui $d_H^* = 1$. Por este motivo é necessário aumentar esse valor para $d_H^* = 3$. Os códigos que obrigam a $d_H^* = 1$ são b, c e e. O que significa que é necessário aumentar o peso de Hamming para esses códigos. Como existem três códigos nessa situação é necessário adicionar, no mínimo, três bit de redundância. Se se adicionassem apenas 2 bit de redundância estes teriam que ser obrigatoriamente 11 para os três casos o que seria manifestamente impossível à luz da expressão (3.18).

Atribui-se, de forma arbitrária, os seguintes bit de paridade aos códigos b, c e e. Apenas se teve o cuidado de garantir que $w_H(b) = w_H(c) = w_H(e) = 3$ e que as linhas de $\mathbf{t}$ eram linearmente independentes. Deste modo obteve-se:

Tabela 3.4: Todas as possíveis distâncias de Hamming para o alfabeto representado na tabela 3.3.

c	mensagem	paridade
b	001	110
c	010	011
e	100	101

ou seja, para este código, a matriz $\boldsymbol{\alpha}$ é igual a:

$$\boldsymbol{\alpha} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{3.43}$$

Com base nesta matriz, e atendendo à expressão (3.18) obtém-se as seguintes palavras-código:

Tabela 3.5: Todas as possíveis distâncias de Hamming para o alfabeto representado na tabela 3.3.

c	mensagem	paridade	**c**	mensagem	paridade
a	000	000	e	100	101
b	001	011	f	101	110
c	010	110	g	110	011
d	011	101	h	111	000

Como se pode comprovar, mesmo por inspecção visual, a distância mínima de Hamming é 3.

3.6.1.4 Relação entre distância de Hamming e matriz de teste da paridade

A partir da matriz de teste da paridade $\mathbf{H}$ é possível determinar a distância mínima de Hamming para um código linear. Para demonstrar essa particularidade admita-se uma palavra-código arbitrária $\mathbf{c}_i \in \mathcal{C}$. Atendendo à condição expressa em 3.27 conclui-se que a combinação linear das colunas de $\mathbf{H}$ apontadas pelos elementos não-nulos de $\mathbf{c}_i$ é igual a zero. Por exemplo se $\mathbf{c} = 011101$ e se,

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{3.44}$$

então a soma das colunas 2, 3, 4 e 6 de $\mathbf{H}$ deve ser zero. Esta afirmação pode ser facilmente comprovada. Observe-se que,

$$\begin{aligned} 010 + 001 + 110 + 101 &= 011 + 110 + 101 \\ &= 101 + 101 \\ &= 000 \end{aligned} \tag{3.45}$$

Por outro lado, se o código for linear, a distância mínima de Hamming é igual ao menor peso de Hamming, do universo de todas as palavras-código, com excepção do vector nulo. Seja $\mathbf{c}_k$ a palavra-código pertencente a $\mathcal{C}$ com o menor peso de Hamming. Seja esse peso igual a d_H^*. Pela definição de peso de

Hamming, o número de bits, em $\mathbf{c}_k$, diferentes de zero é igual a d_H^*. Conclui-se assim que a distância mínima de Hamming é igual ao *menor* número de colunas linearmente **dependentes** de **H**. Assim sendo evidenciam-se as seguintes situações [4]:

- A distância mínima de Hamming é **um** se existir uma coluna a zero em **H**. Nesta situação é claro que existe, no alfabeto de $\mathcal{C}$, uma palavra-código com peso de Hamming igual a um.
- A distância mínima de Hamming é **dois** se existirem duas colunas idênticas em **H**. Tal como no item anterior, este facto indica a existência de uma palavra-código com peso de Hamming igual a dois no alfabeto de $\mathcal{C}$

O que acabou de ser dito pode ser resumido pelo seguinte teorema [7]:

Teorema 3: *A distância mínima de Hamming de um código linear, d_H^*, tem valor igual a p se, e só se, todos os conjuntos formados por $p-1$ colunas da matriz de teste de paridade* **H** *forem linearmente* ***independentes*** *e algum(s) conjuntos formados por p colunas de* **H** *são linearmente* ***dependentes****.*

Em álgebra linear, o *rank* de uma matriz é o número de linhas (ou colunas) linearmente independentes dessa matriz[11]. Desta forma, o teorema anterior garante que o *rank* da matriz **H** deve ser superior, ou igual, a $p-1$. Ou seja [7],

$$\text{rank}(\mathbf{H}) \geq p-1 \tag{3.46}$$

A partir desta expressão é possível determinar um majorante para a distância mínima de Hamming a partir de:

$$d_H^* \leq rank(\mathbf{H}) + 1 \tag{3.47}$$

Por exemplo para a matriz **H** apresentada em (3.44) o valor do rank é 3. O que significa que o maior valor possível para d_H^* é 4. Efectivamente a mínima distância de Hamming para esse código é 3. Basta ver que não existe nenhuma coluna a zero, não existem duas colunas iguais e a soma, por exemplo, das colunas 1, 2 e 4 dá, como resultado, o vector zero.

3.6.1.5 Descodificação

Nesta parte da matéria vamos analisar o que acontece após interferência e como é que o descodificador pode lidar com essa perturbação. Na secção anterior verificou-se que o processo de codificação envolve apenas produtos matriciais, sendo por isso uma operação **linear**. No entanto o processo de descodificação é **não-linear** e a forma mais comum é discutida nos parágrafos que se seguem.

Normalmente, quando um código **d** é recebido pelo descodificador nada pode ser dito a respeito da mensagem original **c**. A tarefa do descodificador assenta num conjunto de suposições, inferências, acerca da mensagem em função de **d**. Ou seja a afirmação feita pelo descodificador, de que **d** se refere à mensagem **c**, é feita de forma condicional: admitindo a inexistência de erros, admitindo a existência de um erro, ..., admitindo a existência de n erros.

[11] O *rank* de uma matriz tomado ao longo das linhas ou colunas tem o mesmo valor.

A inferência sobre o número provável de erros é feita analisando a redundância enviada em $\mathbf{c}$. Por norma se $\mathbf{d} \in \mathcal{C}$ o descodificador assume que não houve erros na transmissão. Caso isso não aconteça o descodificador *pode* tentar desvendar qual terá sido a verdadeira mensagem enviada. Existem várias formas de executar essa tarefa. Um dos métodos envolve a escolha, de entre o alfabeto $\mathcal{C}$, da palavra-código que mais se aproxima de $\mathbf{d}$. Normalmente essa aproximação é estimada segundo a distância de Hamming. Ou seja o processo de descodificação, em caso de erro, escolhe, como sequência real $\mathbf{d}'$, o vector $\mathbf{v} \in \mathcal{C}$ com menor distância de Hamming do vector recebido:

$$\mathbf{d}' = \arg\min_{\mathbf{v}} \quad d_H(\mathbf{d}, \mathbf{v}) \tag{3.48}$$

Para ilustrar este método recorde o código apresentado na secção 3.6.1.3. Imagine que se envia o caractere b pelo canal de comunicação e um erro ocorreu em um dos bits. Por exemplo enviou-se 001011 e foi recebido 011011. Como a palavra-código recebida não pertence ao alfabeto do código, o descodificador gera um erro. Agora, para o corrigir, vai calcular a distância da palavra recebida a todas as outras que fazem parte do código. Os resultados encontram-se apresentados na tabela 3.6.

Tabela 3.6: Todas as possíveis distâncias de Hamming, à palavra 011011, para o alfabeto representado na tabela 3.3.

$\mathbf{c}$	d_H	$\mathbf{c}$	d_H
(x,a)	4	(x,e)	4
(x,b)	1	(x,f)	3
(x,c)	3	(x,g)	2
(x,d)	2	(x,h)	3

Neste caso a palavra-código mais próxima de 011011, na métrica de Hamming, é 001011 que corresponde exactamente à mensagem enviada[12].

Vamos agora explorar, de forma mais efectiva, a rotina que é necessário implementar, no lado do descodificador, para que este leve a cabo a sua tarefa. Para derivar o modo de operação do descodificador começa-se por introduzir o conceito de **classe lateral**[13]. Admita-se, para já, a transmissão de um código $\mathbf{c}$ através de um canal de comunicação que adiciona ruído $\mathbf{e}$. O código $\mathbf{d}$ é aquele que chega ao descodificador para ser traduzido na mensagem $\mathbf{m}$. A figura 3.9 ilustra este conceito.

Se a dimensão da palavra-código é n então pode dizer-se que o vector $\mathbf{e}$ pertence a um conjunto, que designaremos por E, com 2^n elementos. Ou seja,

$$E = \{\mathbf{e}_i\}, \quad i = 0, \cdots, 2^n - 1 \tag{3.49}$$

Por exemplo se $n = 3$ então,

$$E = \{000, 001, 010, 011, 100, 101, 110, 111\} \tag{3.50}$$

Se o alfabeto de $\mathcal{C}$ possuir q símbolos então, por acção de perturbações no canal,

[12] O que já era de esperar dado que o código projectado possui a capacidade de corrigir até um erro na palavra-código transmitida

[13] Do Inglês *coset*

qual deverá ser a dimensão do alfabeto do código recebido $\mathbf{d}$?

Cada uma das palavras-código $\mathbf{c}_i$ com $i = 1, \cdots, q$ poderá ser corrompida por uma dos 2^n possíveis vectores de erro. Desta forma poderíamos ser tentados a pensar que o alfabeto de $\mathbf{d}$ seria $q \times 2^n$ o que não faz sentido porque com n bit apenas se conseguem 2^n sequências distintas. Por isso, se D for o conjunto de todas as possíveis palavras-código, então a sua dimensão é 2^n.

Admita-se uma palavra de erro arbitrária $\mathbf{e}_i$ das 2^n possíveis. O conjunto de todas as palavras-código D_i que podem ser geradas à entrada do descodificador são dadas por:

$$D_i = \mathbf{e}_i + \mathcal{C} = \{\mathbf{e}_i + \mathbf{c}_j\}, \quad j = 1, \cdots, q \tag{3.51}$$

onde $i = 0, \cdots, 2^n - 1$.

Este conjunto designa-se por **classe lateral** de $\mathbf{e}_i$. Dentro desse conjunto, o elemento com menor peso de Hamming é designado por **líder dessa classe lateral**[14]. O líder da classe lateral i, que se irá designar por $\mathbf{a}_i$ é dado por.

$$\mathbf{a}_i = \arg\min_{D_i} \quad w_H(D_i) \tag{3.52}$$

Apresentam-se agora algumas considerações importantes:

- Uma classe lateral pode possuir mais do que um líder. Nesse caso a escolha para líder da classe é feita de forma arbitrária;
- O alfabeto $\mathcal{C}$ é uma classe lateral trivial e o seu líder é o vector $\mathbf{0}$. Ou seja $\mathbf{a}_0 = [0 \ \cdots \ 0]$.
- Existem, ao todo, 2^{n-k} classes laterais. Efectivamente existem 2^k palavras-código mas 2^n possíveis palavras-código com erro. Como cada classe lateral possui 2^k palavras-código existem $2^n/2^k = 2^{n-k}$ classes laterais.
- As classes laterais formam, entre sí, conjuntos disjuntos. Ou seja não existem elementos comuns entre duas quaisquer classes laterais.
- A união de todas as classes laterais fornece o alfabeto D:

$$D = D_0 \vee \cdots \vee D_{2^{n-k}-1} \tag{3.53}$$

Vamos ilustrar a forma como se obtém as diferentes classes laterais através de um exemplo. Para isso vamos supor um código de bloco com $k = 2$, $n = 4$ e com a seguinte matriz geradora [7]:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{3.54}$$

Para as 4 possíveis mensagens $\mathbf{m} = \{00, 01, 10, 11\}$ existe o seguinte alfabeto $\mathcal{C}$. Cada palavra-código foi obtida, usando cada uma das mensagens, recorrendo à expressão 3.19:

$$\mathcal{C} = \begin{Bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{Bmatrix} \tag{3.55}$$

[14] *coset leader* em Inglês

Existem, ao todo, quatro classes laterais. A primeira é, como já se disse, o próprio $\mathcal{C}$. Neste caso $\mathbf{a}_0 = [0\ 0\ 0\ 0]$. Para encontrar a próxima classe lateral começa-se por pesquisar pelo seu líder. Para isso procura-se, em $\mathcal{C}$, um elemento que **não** lhe pertença com o **menor** peso de Hamming possível. Qualquer elemento do conjunto:

$$\{0001, 0010, 0100, 1000, 0011, 0110, 0111, 1010, 1100, 1101, 1001, 1111\} \quad (3.56)$$

é um possível candidato a líder. No entanto, do conjunto apresentado, apenas quatro apresentam $w_H = 1$:

$$\{0001, 0010, 0100, 1000\} \quad (3.57)$$

qualquer um desse quatro vectores pode ser o líder da próxima classe lateral. A escolha de qual é irrelevante. A selecção de um em detrimento dos outros apenas vai alterar o valor da palavra-código descodificada caso tenham ocorrido mais erros do que aqueles que são possíveis de corrigir.

Tomando por exemplo $\mathbf{a}_1 = [0\ 0\ 0\ 1]$ obtém-se a classe lateral $D_1 = \mathbf{a}_1 + \mathcal{C}$:

$$\mathbf{a}_1 + \mathcal{C} = \begin{cases} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{cases} \quad (3.58)$$

Repete-se novamente o mesmo procedimento mas agora a escolha do líder da próxima classe não poderá existir no conjunto formado pela reunião de $\mathcal{C}$ com $\mathbf{a}_1 + \mathcal{C}$. O conjunto dos elementos que não pertencem à união são:

$$\{0010, 1000, 0011, 0110, 0111, 1100, 1101, 1001\} \quad (3.59)$$

Desta vez existem dois potenciais líderes: 0010 e 1000. Admitindo agora que $\mathbf{a}_2 = [0\ 0\ 1\ 0]$ obtém-se para $D_2 = \mathbf{a}_2 + \mathcal{C}$ o seguinte conjunto de vectores:

$$\mathbf{a}_2 + \mathcal{C} = \begin{cases} 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{cases} \quad (3.60)$$

Três classes laterais já foram definida. Falta agora determinar o líder para $\mathbf{a}_3$. Neste caso o líder deve ser escolhido do conjunto de elementos que não pertençam ao conjunto $\{\mathcal{C} \vee (\mathbf{a}_1 + \mathcal{C}) \vee (\mathbf{a}_2 + \mathcal{C})\}$. Neste caso:

$$\{1000, 0011, 0110, 1101\} \quad (3.61)$$

A única escolha para $\mathbf{a}_3$ é $[1\ 0\ 0\ 0]$. Os restantes elementos desse conjunto farão parte da mesma classe lateral de $\mathbf{a}_3$ conforme se comprova pelo resultado da seguinte operação.

$$\mathbf{a}_3 + \mathcal{C} = \begin{cases} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{cases} \quad (3.62)$$

O processo de descodificação utiliza as classes laterais para determinar, através de um procedimento de *table lookup* qual o valor mais próximo, em

caso de erro, do código recebido. Para isso começa-se pela formação de uma matriz, designada por **matriz padrão**, cujas linhas são as classes laterais e os líderes encontram-se na coluna mais à esquerda. Ou seja,

$$\begin{bmatrix} \mathbf{a}_0 & \mathbf{c}_1 & \cdots & \mathbf{c}_{2^k-1} \\ \mathbf{a}_1 & \mathbf{a}_1+\mathbf{c}_1 & \cdots & \mathbf{a}_1+\mathbf{c}_{2^k-1} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{a}_{2^{n-k}-1} & \mathbf{a}_{2^{n-k}-1}+\mathbf{c}_1 & \cdots & \mathbf{a}_{2^{n-k}-1}+\mathbf{c}_{2^k-1} \end{bmatrix} \tag{3.63}$$

Com base nesta matriz o algoritmo de descodificação executa os seguintes passos:

1. Dado um vector $\mathbf{d}$ recebido, localiza esse símbolo na matriz.
2. O vector de erro é assumido como sendo o **líder** da linha onde esse vector se encontra.
3. O vector descodificado é assumido como sendo o valor do **primeiro** elemento da coluna onde esse vector se encontra.

De modo a mostrar este procedimento voltemos novamente ao exemplo do código (4,2) apresentado anteriormente. Para essa situação a matriz padrão tem o seguinte aspecto:

$$\begin{bmatrix} 0000 & 0101 & 1011 & 1110 \\ 0001 & 0100 & 1010 & 1111 \\ 0010 & 0111 & 1001 & 1100 \\ 1000 & 1101 & 0011 & 0110 \end{bmatrix} \tag{3.64}$$

Imagine-se que se recebe $\mathbf{d} = [1\ 0\ 0\ 1]$. Este vector situa-se na terceira linha, terceira coluna da matriz. Nesta situação o descodificador presume que $\mathbf{e} = \mathbf{a}_2$ e que $\mathbf{c} = [1\ 0\ 1\ 1]$. Observe que o vector descodificado depende da escolha que efectuamos para os líderes. Por exemplo se o valor recebido pelo descodificador fosse $\mathbf{d} = [1\ 1\ 1\ 1]$, o valor previsto para o erro seria $\mathbf{e} = [0\ 0\ 0\ 1]$ e estimava-se $\mathbf{c} = [1\ 1\ 1\ 0]$. No entanto, se para $\mathbf{a}_1$ se tivesse escolhido o vector 0100 para líder e não 0001, o valor descodificado para o mesmo $\mathbf{d}$ seria $\mathbf{c} = [1\ 0\ 1\ 1]$. Esta indecisão apenas indica que, para a palavra 1111 existe mais do que um código em $\mathcal{C}$ com o mesmo valor em termos de distância de Hamming. Efectivamente, por mera inspecção visual, se nota que na primeira linha da matriz $\mathbf{A}$ existem dois vectores que distam 1 bit de 1111. Para terminar deixam-se aqui um par de afirmações importantes:

- O processo de descodificação, pela mínima distância de Hamming, pressupõe que a probabilidade de ocorrência de erro é inversamente proporcional ao peso de Hamming de um dado vector de erro.
- Os líderes na matriz padrão são os vectores de erro com **maior probabilidade** de ocorrência.
- O descodificador tem a seu cargo a tarefa de particionar os 2^n possíveis palavras-código em 2^{n-k} conjuntos disjuntos.

- O vector resultante da diferença entre dois vectores da mesma classe lateral pertence ao alfabeto $\mathcal{C}$. Por exemplo admita-se, para uma determinada classe lateral, um líder $\mathbf{e}$. Seja $\mathbf{c}_i$ e $\mathbf{c}_j$ duas palavras código pertencentes a $\mathcal{C}$. Os elementos $\mathbf{x}_i = \mathbf{c}_i + \mathbf{e}$ e $\mathbf{x}_j = \mathbf{c}_j + \mathbf{e}$ pertencem à mesma classe lateral. A diferença $\mathbf{x}_i - \mathbf{x}_j$ é igual a $\mathbf{c}_i + \mathbf{e} - \mathbf{c}_j - \mathbf{e} = \mathbf{c}_i - \mathbf{c}_j$. Se o código é linear então $\mathbf{c}_i - \mathbf{c}_j \in \mathcal{C}$ o que comprova a afirmação inicial.

De cada vez que um vector é recebido, o descodificador deve procurar na tabela, a linha e coluna em que esse elemento se encontra. No entanto o número de pesquisas que o descodificador deve fazer aumenta exponencialmente com n. Por esse motivo, este método não tende a ser implementado na prática. No entanto existe uma possibilidade de acelerar o processo de pesquisa através da estimativa do erro calculado pelo síndrome. Este método designa-se por **descodificação pelo síndrome** e será apresentado em seguida.

Descodificação pelo Síndrome

Para explicar o processo de descodificação pelo síndrome começa-se por arbitrar um código de bloco (n, k) **linear** com matriz de paridade $\mathbf{H}$. Já se viu que, se um vector pertencer a $\mathcal{C}$, o seu síndrome é o vector $\mathbf{0}$. Por outro lado, se o vector recebido pelo descodificador não pertencer ao alfabeto do código, então este pode ser decomposto em:

$$\mathbf{d} = \mathbf{c} + \mathbf{e} \tag{3.65}$$

O cálculo do síndrome de $\mathbf{d}$ é então igual a:

$$\begin{aligned} \mathbf{d}\mathbf{H}^T &= (\mathbf{c} + \mathbf{e})\mathbf{H}^T \\ &= \mathbf{c}\mathbf{H}^T + \mathbf{e}\mathbf{H}^T \\ &= \mathbf{e}\mathbf{H}^T \end{aligned} \tag{3.66}$$

ou seja o síndrome de $\mathbf{d}$ é **igual** ao síndrome de $\mathbf{e}$. Introduz-se agora o seguinte lema [4]:

Lema 1: Sejam $\mathbf{y}$ e $\mathbf{z}$ duas palavras-código. A igualdade dos síndromes de $\mathbf{y}$ e $\mathbf{z}$, i.e. $\mathbf{s_y} = \mathbf{s_z}$, só acontece se, e só se, $\mathbf{y}$ e $\mathbf{z}$ se encontrarem na mesma classe lateral. Ou seja se $\mathbf{y} - \mathbf{z} \in \mathcal{C}$.

Este lema diz-nos que se conhecermos o síndrome de $\mathbf{d}$ então conhecemos o síndrome do erro $\mathbf{e}$. Pelo lema anterior, tanto $\mathbf{d}$ como $\mathbf{e}$ pertencem à mesma classe lateral. Adicionalmente $\mathbf{e}$ é o líder dessa classe lateral. Desta forma, conhecendo $\mathbf{d}$ e tendo uma estimativa para $\mathbf{e}$ determina-se $\mathbf{c}$ por $\mathbf{c} = \mathbf{d} - \mathbf{e}$.

Em baixo apresenta-se o algoritmo que permite, de forma sistemática, descodificar o valor $\mathbf{d}$, de acordo com a mínima distância de Hamming, utilizando apenas os valores dos líderes e respectivos síndromes.

Algorítmo 1

1. Cálculo do síndrome de $\mathbf{d}$;
2. No vector padrão encontrar o líder cujo síndrome seja igual ao de $\mathbf{d}$. Se o síndrome de $\mathbf{a}_i$ for igual a $\mathbf{s_d}$ então o erro de transmissão estimado $\hat{\mathbf{e}}$ é assim igual ;

3. Estimar $\mathbf{c}$ a partir de:

$$\hat{\mathbf{c}} = \mathbf{d} - \hat{\mathbf{e}} \tag{3.67}$$

A figura 3.10 apresenta um diagrama esquemático do processo de descodificação pelo síndrome conforme enumerado no algoritmo anterior.

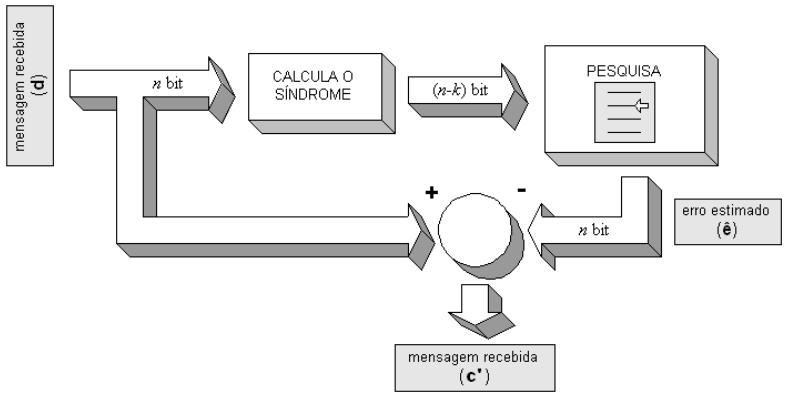


Figura 3.10: Estrutura "feedforward" do descodificador por síndrome para um código linear.

Só para encerrar esta secção uma última observação. A partir do valor do síndrome é possível estimar qual o vector mais provável associado ao erro. Conhecendo esse vector, e conhecendo também a mínima distância de Hamming do código, é possível definir uma estratégia que indique se o valor descodificado $\bar{\mathbf{c}}$ é efectivamente igual à mensagem enviada $\mathbf{c}$ ou não. Se o peso de Hamming do vector $\mathbf{e}$ for superior ao valor de n_c [15], então não há garantia de que a sequência descodificada coincida com a *string* transmitida.

3.6.1.6 "Empacotamento de esferas" e códigos perfeitos

O problema da correcção de erros pode ser posto da seguinte forma: *Dado n e d_H^* qual o número máximo possível de palavras-código*? Esta questão pode ser respondida de várias formas. A mais comum é derivada do conceito de **limite de Hamming**. Por razões que se tornarão claras já a seguir, este limite também é designado por *empacotamento de esferas* ou *limite de volume*.

Começa-se por definir o domínio $\mathbb{B}^n$ de todas as *strings* binárias com n bit. Em termos geométricos, cada um dos vectores pertencentes a $\mathbb{B}^n$ corresponde ao vértice de um hipercubo. Por exemplo, se $n = 3$, a distribuição espacial das 8 possíveis palavras-código binárias encontra-se representada na figura 3.11.

A relação entre a distância euclideana e a distância de Hamming, para quaisquer dois vértices $\mathbf{v_1}$, $\mathbf{v_2}$ é dada por:

$$d_{euclid} = \sqrt{d_H(\mathbf{v}_1, \mathbf{v}_2)} \tag{3.68}$$

Admita-se agora a existência de uma esfera, com centro num vector arbitrário $\mathbf{v} \in \mathbb{B}^n$, e com raio r bit. As figuras 3.12, 3.13 e 3.14 ilustram este conceito, admitindo como centro o vértice 001, para três valores distintos de r: 1, 2 e 3 bit.

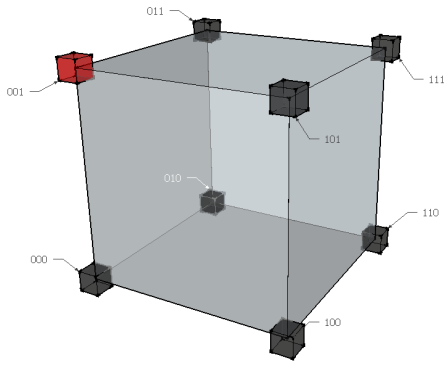


Figura 3.11: Representação dos 8 possíveis vectores pertencentes ao domínio $\mathbb{B}^3$. A cada vértice corresponde uma *string* diferente.

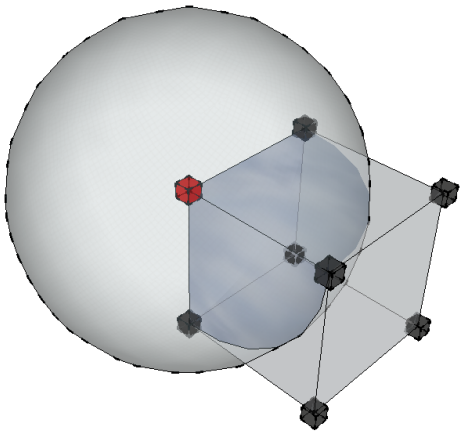


Figura 3.12: Representação de uma esfera, no espaço euclideano, com raio igual a 1 bit e centro no vértice 001 do cubo representado na figura 3.11.

Conforme se pode ver na figura 3.12, com $r = 1$ a **superfície** da esfera contém os vectores 000, 011 e 101. Para $r = 2$ são agora os vectores 010, 100 e 111 que se encontram sobre a casca esférica. Finalmente, e como se pode observar da figura 3.14, para $r = 3$ apenas o vector 110 é tangente à esfera.

A partir desta esfera define-se o conceito de **volume de Hamming**. O volume de Hamming refere-se ao número de vectores contidos no interior da esfera de raio r[16]. Por exemplo, no caso da figura 3.12, o volume de Hamming é 4. Já na figura 3.13 o volume é 7 e, para a situação ilustrada na figura 3.14 o volume de Hamming é igual a todo o espaço $\mathbb{B}^3$.

O volume de Hamming refere-se ao conjunto de todos os vectores pertencentes a $\mathbb{B}^n$ que distem de r bit do vector $\mathbf{v}$. Ou seja o volume da esfera de Hamming de raio r, no espaço $\mathbb{B}^n$, denotado por $v_H(\mathbf{v}, n, r)$, é dado pelo cardinal do conjunto:

$$v_H(\mathbf{v}, n, r) = \#\{\mathbf{v}_i \in \mathbb{B}^n : d_H(\mathbf{v}, \mathbf{v}_i) \leq r\} \tag{3.69}$$

[15]O que indica a existência de mais erros do que aqueles que o código consegue corrigir.
[16]Incluindo aqueles que estão sobre a superfície.

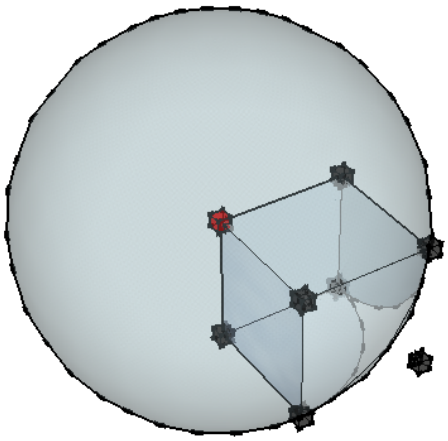


Figura 3.13: Esfera com centro em 001 e raio igual a 2 bit.

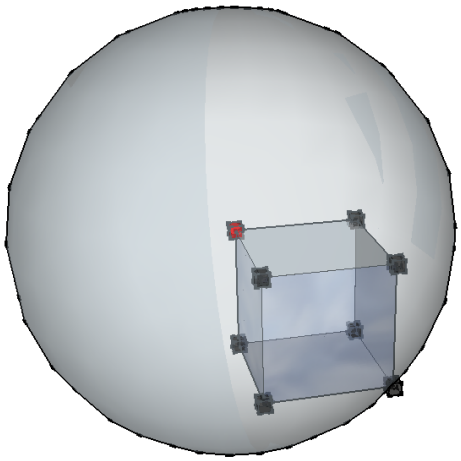


Figura 3.14: Esfera com centro em 001 e raio igual a 3 bit.

No caso de um código binário com n bit, o volume de Hamming, relativo à esfera de raio 1 bit centrada em $\mathbf{v}$, consiste em todas as string binárias com uma diferença de 1 bit, ou menos, relativamente a $\mathbf{v}$. Esse número consiste no número total de formas distintas de $\mathbf{v}$ com zero bit diferentes adicionada ao número total de formas distintas de $\mathbf{v}$ com um bit diferente. Para $r = 0$ apenas o próprio $\mathbf{v}$ existe sendo contabilizando como um elemento. Já para $r = 1$ o número de variantes de $\mathbf{v}$ pode ser calculado por:

$$\begin{pmatrix} n \\ 1 \end{pmatrix} \tag{3.70}$$

ou seja o número de combinações possíveis obtidas por alteração de 1 bit. Por exemplo para o caso de $n = 3$, a avaliação da expressão anterior leva ao resultado

3. Ainda neste contexto, para $r = 1$, o volume de Hamming é dado por:

$$v_H(\mathbf{v}, 3, 1) = 1 + \left(\begin{array}{c} 3 \\ 1 \end{array} \right) = 4 \tag{3.71}$$

Se $r = 2$ o volume de Hamming é composto por sete palavras-código. Este valor é calculado da seguinte forma:

$$v_H(\mathbf{v}, 3, 2) = 1 + \left(\begin{array}{c} 3 \\ 1 \end{array} \right) + \left(\begin{array}{c} 3 \\ 2 \end{array} \right) = 7 \tag{3.72}$$

Finalmente, para $r = 3$, o volume é obtido, por:

$$v_H(\mathbf{v}, 3, 3) = 1 + \left(\begin{array}{c} 3 \\ 1 \end{array} \right) + \left(\begin{array}{c} 3 \\ 2 \end{array} \right) + \left(\begin{array}{c} 3 \\ 3 \end{array} \right) = 8 \tag{3.73}$$

que é exactamente o número total de palavras-código definidas pelo domínio $\mathbb{B}^3$.

É fácil demonstrar que, para o caso genérico dum código binário (n, k), o volume para a esfera de Hamming com raio r, centrada em $\mathbf{v}$, é dado por:

$$v_H(\mathbf{v}, n, r) = \sum_{i=0}^{r} \left(\begin{array}{c} n \\ i \end{array} \right) \tag{3.74}$$

Considere-se agora o caso de um código binário $\mathcal{C}$ de parâmetros (n, k) com 2^k palavras-código distintas. Admite-se que uma mensagem m com dimensão n atravessa um canal de comunicação imperfeito. Como não se sabe quais os bit que poderão ser sujeitos a erro, o domínio do código que chega ao descodificador é $\mathbb{B}^n$. Ou seja o conjunto de chegada possui um total de 2^n elementos.

Imagine-se agora (hiper)esferas centradas nas 2^k palavras-código. Cada uma das esferas pode ter um raio arbitrário r_i para $i = 1, \cdots, 2^k$. No entanto vamos forçar a que o raio das esferas seja igual e tal que não exista contacto entre hiperesferas. A figura 3.15 representa a ideia para o caso de dois vectores num plano. De forma geral o valor do raio da hiperesfera deve satisfazer o critério:

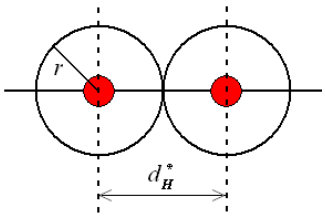


Figura 3.15: Representação das hiperesferas em torno de duas palavras-código com distância d_H^* entre elas.

$$r < \frac{d_H^*}{2} \tag{3.75}$$

ou, visto que a distância de Hamming é sempre um número inteiro, uma outra forma de descrever a relação anterior é através da inequação,

$$r \leq \frac{d_H^* - 1}{2} \tag{3.76}$$

Observe-se que o raio da esfera de Hamming é expressa em bit e logo o termo à direita da desigualdade deve ser forçado a ser um inteiro. A forma de efectuar essa operação é através de arredondamento. No entanto o arredondamento deve ser feito sempre em direcção ao menor inteiro caso contrário, o excesso no valor do raio, irá obrigar à intersecção das esferas. Basta pensar no seguinte exemplo: Se $d_H^* = 4$ então r deveria ser $3/2$. Arredondando por excesso no limite $r = 2$ o que implica que existe, pelo menos, um par de esferas com um ponto comum o que invalida o pressuposto de não-sobreposição. Por este motivo o raio das esferas deve obedecer a:

$$r \leq \left\lfloor \frac{d_H^* - 1}{2} \right\rfloor \tag{3.77}$$

de modo a que não exista sobreposição de esferas.

Se estão recordados, esta expressão não é nova e foi já apresentada na secção 3.6.1.2 na equação (3.42) e referia-se ao número de erros que um código com distância d_H^* era capaz de corrigir. Uma consequência importante desta afirmação pode ser sumariada da seguinte forma:

*Num código capaz de **corrigir** até r erros, as esferas de Hamming, de raio r centradas nas palavras-código, não se sobrepõem.*

Para além disso note-se que a soma do volume de cada esfera de Hamming, na condição em que o raio é dado por (3.77), será sempre menor, ou igual, ao número de *strings* do espaço $\mathbb{B}^n$. Ou seja,

$$\sum_{i=1}^{2^k} v_H(\mathbf{c}_i, n, r) \leq 2^n \tag{3.78}$$

como o volume da esfera é independente do centro fica:

$$v_H(\mathbf{c}, n, r) \sum_{i=1}^{2^k} \leq 2^n \tag{3.79}$$

atendendo à expressão (3.80), a inequação anterior toma o seguinte aspecto:

$$2^k \cdot \sum_{j=0}^{r} \binom{n}{j} \leq 2^n \tag{3.80}$$

O valor 2^k refere-se ao cardinal do conjunto $\mathcal{C}$. Exprimindo a desigualdade anterior em função desta quantidade obtém-se:

$$\#\mathcal{C} \leq \frac{2^n}{\sum_{j=0}^{r} \binom{n}{j}} \tag{3.81}$$

Esta expressão possui a seguinte interpretação:

O número de palavras-código possíveis com um código de n bit capaz de corrigir até r bit de erro é, no máximo, $2^n \cdot \left(\sum_{j=0}^{r} C_j^n\right)^{-1}$.

Note-se que esta expressão indica o número **máximo** possível de palavras-código que podem ser formadas mas não diz quais são. Este valor é designado por limite de Hamming.

Por exemplo considere o código de 3 bit. Admitindo um alfabeto composto pelas 8 possíveis palavras-código a distância mínima de Hamming é 1 bit. Para que o código tenha a capacidade de corrigir até 1 erro apenas se podem estabelecer, no máximo, 2 palavras-código. Este numero pode ser obtido pelo limite de Hamming considerando $r = 1$ e $n = 3$:

$$\#\mathcal{C} = \frac{8}{1+3} = 2 \tag{3.82}$$

Por exemplo os pares $\{000, 111\}$ ou $\{001, 110\}$, entre outros, podem ser utilizados.[17]

Um código que verifique a inequação (3.81) com igualdade, ou seja que atinja o limite de Hamming, é chamado de **código perfeito**. Ou seja se,

$$\#\mathcal{C} \cdot \sum_{j=0}^{r} \left(\begin{array}{c} n \\ j \end{array} \right) = 2^n \tag{3.83}$$

o código é chamado de perfeito.

Um código de bloco é perfeito se toda a palavra-código se encontra à distância r de qualquer outra palavra-código possível. Geometricamente o mesmo é dizer que o conjunto das esferas de Hamming, centradas nas palavras-código, cobrem todas as *strings* pertencentes ao domínio $\mathbb{B}^n$.

Note ainda que, para que o código seja perfeito, o valor da distância mínima de Hamming deve ser ímpar. Efectivamente basta analisar a expressão (3.77) para perceber porque é que têm que ser assim.

De seguida apresentam-se alguns exemplos de códigos perfeitos. Por exemplo um código $(3, 3)$ com $d_H^* = 1$ é perfeito visto que, neste caso, $r = 0$ e,

$$8 \cdot 1 = 2^3 \tag{3.84}$$

Outro exemplo de código perfeito é o código $(7, 4)$ com $d_H^* = 3$. Neste caso $r = 1$ e,

$$\begin{aligned} 2^4 \cdot (1+7) &= 2^7 \\ 2^4 \cdot 2^3 &= 2^7 \end{aligned} \tag{3.85}$$

Este último código é designado por Hamming (7,4) e será apresentado, de forma mais pormenorizada, na subsecção 3.6.2.3.

Efectivamente o universo dos códigos perfeitos é muito restrito. Para além dos códigos de Hamming, apenas os códigos de **repetição**[18] e os códigos de **Golay** são perfeitos. Este último será apresentado na subsecção 3.6.2.4.

Podem juntar-se ainda ao universo destes três tipos de códigos os chamados códigos triviais. Os **códigos triviais** são aqueles compostos por todas as

[17] No entanto, das quatro possibilidades, apenas o par $\{000, 111\}$ fornece um código linear.
[18] Com n ímpar.

possíveis palavras com n bit, como aconteceu no primeiro exemplo dado, ou então códigos com apenas uma palavra.

Antes de encerrar esta secção define-se ainda o conceito de **código quase-perfeito**[19]. Um código quase-perfeito e aquele em que as esferas de Hamming de raio r são disjuntas e as esferas de raio $r+1$ cobrem todo o espaço eventualmente com algumas sobreposições. Por exemplo o código de Golay $(24, 12)$ com $d_H^* = 7$ é um exemplo.

3.6.1.7 Entropia, Informação e Capacidade

Nesta subsecção apresenta-se um conceito nuclear à teoria da informação: o conceito de entropia. A entropia refere-se ao valor médio da incerteza associado a uma mensagem e está intimamente ligada à definição quantitativa de informação.

Apresenta-se também o teorema fundamental da teoria da informação proposto por Shannon. Este teorema estabelece um limite teórico para a transmissão de informação através de um canal imperfeito. O limite identificado é designado por capacidade de um canal e será também objecto de descrição no decorrer desta subsecção.

Entropia

Em teoria da informação os conceitos de *informação* e *incerteza* descrevem a forma como um processo selecciona um, ou mais símbolos, de um dado alfabeto finito. Por exemplo um dispositivo debita uma sequência de bits. Quando, no instante presente, se recebe um determinado bit, existe *incerteza* sobre qual o valor lógico do próximo bit. Uma vez esse bit recebido a incerteza diminui e diz-se que foi recebida *informação*. Informalmente, diz-se que **a informação é um decréscimo na incerteza**.

É intuitivo perceber que a incerteza no envio de um determinado símbolo depende da dimensão do alfabeto da fonte. Por exemplo no caso anterior se apenas se recebe um bit de cada vez apenas existe incerteza se o próximo bit é 1 ou 0. Por outro lado se fossem recebidos dois bit de cada vez a incerteza na palavra seria maior pois o número de símbolos diferentes é agora 4: $\{00, 01, 11, 10\}$. Diz-se que a incerteza no primeiro caso é de 2 símbolos e no segundo caso de 4 símbolos, i.e. de 1 bit e 2 bit respectivamente.

Como se deve suspeitar, se cada símbolo for representado por 3 bit existem agora um total de 8 símbolos diferentes no alfabeto. A relação que existe entre a dimensão do alfabeto e o número de bits que o representam é dada por:

$$n_s = 2^{n_b} \tag{3.86}$$

onde n_s se refere ao número de símbolos e n_b o número de bit.

Se a incerteza é proporcional ao número de símbolos, e se a expressão (3.86) relaciona o número de símbolos com o número de bits, então diz-se que uma fonte discreta possui uma incerteza Γ de n_b bits e calcula-se por:

$$\Gamma = \log_2 (n_s) \tag{3.87}$$

[19] Do Inglês *quasi-perfect codes*.

Vamos admitir que existe completa independência entre a sequência de bits enviados por uma fonte. O mesmo é dizer que esta se comporta como um processo estocástico com uma determinada distribuição de probabilidade. Se admitirmos que essa função distribuição é uniforme então a probabilidade de surgir, como próximo bit, o valor lógico '1' é igual à probabilidade de surgir '0'. Pode-se dizer que a incerteza é então de 1 bit o que pode ser comprovado pela substituição, em (3.87), da variável n_s por 2.

Admita-se que a função probabilidade de massa não é uniforme. Imagine-se que a probabilidade de obter o valor lógico '1' é duas vezes superior à probabilidade da fonte gerar o valor '0'. Nesta nova situação a incerteza diminui e essa diminuição pode ser entendida da seguinte forma. Suponha que se encontra a jusante do canal de comunicação e o seu objectivo é prever qual o valor lógico do próximo bit enviado. Admita-se que o método que usa, para efectuar as previsões, consiste em afirmar que o valor lógico, que será enviado, é sempre o mesmo e igual a '1'. Se a função probabilidade for uniforme, ao fim de um número elevado de ensaios, existirão tantos acertos como erros pela parte do observador. Se, por outro lado, a função de probabilidade for não-uniforme, por exemplo se $f_{\mathcal{C}}(m = \text{`1'}) = 2 \times f_{\mathcal{C}}(m = \text{`0'})$ então, ao fim de um número suficientemente elevado de previsões, o número de vezes que acertamos é duas vezes superior às que erramos. Ou seja a incerteza diminuiu. Por este motivo existe necessidade de incorporar esse conhecimento na expressão (3.86). Vejamos como.

No caso de uma função distribuição uniforme, a probabilidade da fonte gerar um dos n_s símbolos é $1/n_s$. Ou seja, dado que o cardinal do alfabeto da fonte, $\mathcal{C} = \{\mathbf{c}_1, \cdots, \mathbf{c}_{n_s}\}$, é n_s então a probabilidade de esta fornecer o símbolo $\mathbf{c}_i$ é:

$$f_{\mathcal{C}}(\mathbf{c}_i) = \frac{1}{n_s} \tag{3.88}$$

Neste sentido a expressão (3.87) pode ser reescrita da seguinte forma:

$$\begin{aligned} \Gamma_i &= -\log_2\left(\frac{1}{n_s}\right) \\ &= -\log_2\Big(f_{\mathcal{C}}(\mathbf{c}_i)\Big) \end{aligned} \tag{3.89}$$

A incerteza **média** de um código com n_s símbolos é obtida através do momento estatístico de ordem um por:

$$\sum_{i=1}^{n_s} f_{\mathcal{C}}(\mathbf{c}_i) \cdot \Gamma_i \tag{3.90}$$

A esta quantidade Shannon designou por **entropia**[20] e designou-a pela letra H[21]. Ou seja,

$$H = -\sum_{i=1}^{n_s} f_{\mathcal{C}}(\mathbf{c}_i) \cdot \log_2\Big(f_{\mathcal{C}}(\mathbf{c}_i)\Big) \tag{3.91}$$

A unidade de entropia é medida em bit/símbolo.

[20] Devido à quantidade homónima derivada da termodinâmica.
[21] Não confundir com **H** da matriz de paridade.

Na figura 3.16 apresenta-se o aspecto da entropia para um código binário ($n_s = 2$) em função da probabilidade p de ocorrência de um dos símbolos (por exemplo o símbolo '1'). Em baixo mostra-se o código, para Matlab, executado para a obter.

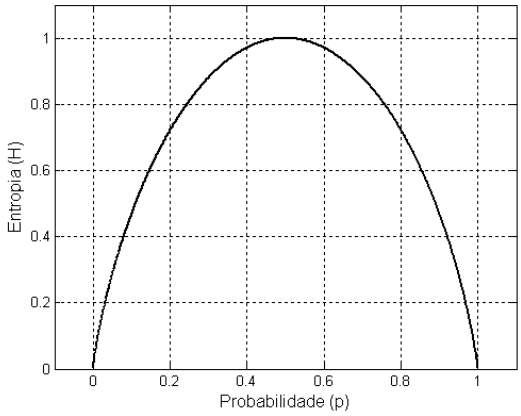


Figura 3.16: Aspecto da entropia H em função da probabilidade p para um código com dois símbolos.

```
>> p=linspace(eps,1-eps,1000);
>> H=-(p.*log2(p)+(1-p).*log2(1-p));
>> plot(p,H);xlabel('Probabilidade (p)');
>> ylabel('Entropia (H)');grid on
>> axis([-.1 1.1 0 1.1])
```

Relativamente à entropia H, de um código com n_s símbolos, tecem-se as seguintes considerações:

- $H = 0$ se, e só se, todos os valores de $f_{\mathcal{C}}(\mathbf{c}_i)$ são nulos com excepção de um que possui o valor unitário;
- Para um determinado alfabeto $\mathcal{C}$ com n_s símbolos, a entropia é máxima, e igual a $\log_2(n_s)$, quando o valor de $f_{\mathcal{C}}(\mathbf{c}_i) = 1/n_s$;
- $f_{\mathcal{C}}(\mathbf{c}_i) \cdot \log_2\Big(f_{\mathcal{C}}(\mathbf{c}_i)\Big) = 0$ para $f_{\mathcal{C}}(\mathbf{c}_i) = 0$. Efectivamente,[22]

$$\lim_{f_{\mathcal{C}}(\mathbf{c}_i)\to 0} \quad f_{\mathcal{C}}(c_i) \cdot \log_2\left(f_{\mathcal{C}}(\mathbf{c}_i)\right) = 0 \tag{3.92}$$

- A entropia de uma fonte com n_s símbolos é sempre inferior, ou igual, ao logaritmo de n_s. Ou seja,

$$H \leq \log_2(n_s) \tag{3.93}$$

[22] Conforme se pode confirmar pelo teorema de L'Hopital: $\lim_{x\to 0} \frac{g(x)}{h(x)} = \lim_{x\to 0} \frac{g(x)'}{h(x)'}$ se $g(x)$ e $h(x)$ forem diferenciáveis e $g(0)' \neq 0$.

Entropia Conjunta e Entropia Condicional

Para além da entropia (marginal) definida anteriormente, em teoria da informação descreve-se ainda **entropia conjunta** e **entropia condicional**. Começemos por analisar a primeira. Para isso imagine-se a situação ilustrada na figura 3.17. Existem dois emissores e cada um deles possui alfabetos de comprimentos m e n respectivamente. Associado ao primeiro emissor existe uma função probabilidade de massa $f_X(\mathbf{x}_i)$ e ao segundo emissor uma função $f_Y(\mathbf{y}_j)$

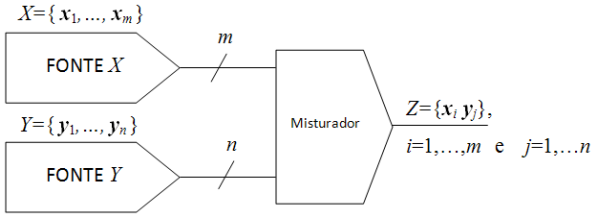


Figura 3.17: Estrutura do descodificador por síndrome para um código linear.

Define-se, para cada uma das duas fontes, o valor (marginal) da entropia como:

$$\begin{aligned} H_X &= -\sum_{i=1}^{m} f_X(\mathbf{x}_i) \cdot \log_2\Big(f_X(\mathbf{x}_i)\Big) \\ H_Y &= -\sum_{j=1}^{n} f_Y(\mathbf{y}_j) \cdot \log_2\Big(f_Y(\mathbf{y}_j)\Big) \end{aligned} \tag{3.94}$$

Admita-se que o sinal proveniente de ambas as fontes é fundido num terceiro sinal Z. Cada símbolo de Z é composto pela concatenação, de um dos m símbolos da fonte X, com um dos n símbolos da fonte Y. Ou seja, $\mathbf{z}_k = \{\mathbf{x}_i\mathbf{y}_j\}$. O número de símbolos do alfabeto Z é, nesta situação, $m \times n$. Qual o valor da entropia relativa à fonte Z?

A probabilidade de gerar o símbolo $\mathbf{z}_k$ é igual à probabilidade de ter sido gerado o símbolo $\mathbf{x}_i$ pela fonte X e, simultâneamente, a fonte Y gerar o símbolo $\mathbf{y}_j$. Ou seja $f_Z(\mathbf{z}_i)$ coincide com a probabilidade conjunta $f_{X\wedge Y}(\mathbf{x}_i \wedge \mathbf{y}_j)$. Por isso a entropia da fonte Z é igual è entropia conjunta dada por:

$$H_{X\wedge Y} = -\sum_{i=1}^{m}\sum_{j=1}^{n} f_{X\wedge Y}(\mathbf{x}_i \wedge \mathbf{y}_j) \cdot \log_2\Big(f_{X\wedge Y}(\mathbf{x}_i \wedge \mathbf{y}_j)\Big) \tag{3.95}$$

Se ambas as fontes, X e Y, forem **estatisticamente independentes** então:

$$f_{X\wedge Y}(\mathbf{x} \wedge \mathbf{y}) = f_X(\mathbf{x}) \cdot f_Y(\mathbf{y}) \tag{3.96}$$

Nesta situação a expressão (3.95) fica:

$$\begin{aligned} H_{X\wedge Y} &= -\sum_{i=1}^{m}\sum_{j=1}^{n} f_X(\mathbf{x}_i) \cdot f_Y(\mathbf{y}_j) \cdot \log_2\Big(f_X(\mathbf{x}_i) \cdot f_Y(\mathbf{y}_j)\Big) \\ &= -\sum_{i=1}^{m}\sum_{j=1}^{n} f_X(\mathbf{x}_i) \cdot f_Y(\mathbf{y}_j) \cdot \Big(\log_2\Big(f_X(\mathbf{x}_i)\Big) + \log_2\Big(f_Y(\mathbf{y}_j)\Big)\Big) \end{aligned} \tag{3.97}$$

que, depois de se organizarem os somatórios, se resume a:

$$H_{X\wedge Y} = H_X + H_Y \tag{3.98}$$

Prova-se que [6], independente da relação estatística entre X e Y, se têm sempre:

$$H_{X\wedge Y} \leq H_X + H_Y \tag{3.99}$$

Para elém disso, a entropia conjunta é sempre superior a qualquer das entropias marginais. Ou seja,

$$H_{X\wedge Y} \geq \max\{H_X, H_Y\} \tag{3.100}$$

Por exemplo a fonte X possui um alfabeto de dois símbolos, $(x_1, x_2) \rightarrow (0, 1)$. A probabilidade de gerar o bit '1' e o dobro da probabilidade de gerar o bit '0'. Ou seja $f_X(x_2) = 2 \cdot f_X(x_1) = 2/3$. Por outro lado o emissor Y possui um alfabeto de 4 símbolos, $(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4) \rightarrow (00, 01, 10, 11)$ com probabilidade de massa uniforme. A entropia da fonte X é:

$$\begin{aligned} H_X &= -\sum_{i=1}^{2} f_X(x_i) \cdot \log_2\Big(f_X(x_i)\Big) \\ &= -f_X(x_1) \cdot \log_2\Big(f_X(x_1)\Big) - f_X(x_2) \cdot \log_2\Big(f_X(x_2)\Big) \\ &= -\frac{1}{3}\log_2\left(\frac{1}{3}\right) - \frac{2}{3}\log_2\left(\frac{2}{3}\right) \\ &\approx 0.92 \text{ bit} \end{aligned} \tag{3.101}$$

O valor médio da incerteza para a fonte X é ligeiramente inferior a 1 bit. Agora para a fonte Y tem-se:

$$\begin{aligned} H_Y &= -\sum_{i=1}^{4} f_Y(\mathbf{y}_i) \cdot \log_2\Big(f_Y(\mathbf{y}_i)\Big) \\ &= -\frac{1}{4} \cdot \log_2\left(\frac{1}{4}\right) \sum_{i=1}^{4} 1 \\ &= 2 \text{ bit} \end{aligned} \tag{3.102}$$

uma incerteza de 2 bit (um valor já esperado devido à distribuição uniforme das probabilidades o que implica, como já se disse, uma incerteza máxima).

Supondo que as fontes X e Y são independentes, a entropia do sinal Z é igual a 2.92 bit (quase 3 bit de incerteza).

Admita-se, por outro lado, que as duas fontes se encontram relacionadas estatisticamente conforme se mostra na tabela 3.7 [23]:

A entropia conjunta é agora:

$$H_{X\wedge Y} = -\sum_{i=1}^{2}\sum_{j=1}^{4} f_{X\wedge Y}(x_i \wedge \mathbf{y}_j) \cdot \log_2\Big(f_{X\wedge Y}(x_i \wedge \mathbf{y}_j)\Big) \tag{3.103}$$

[23] Sendo probabilidades, os valores deste tipo de tabelas encontram-se balizadas pelo intervalo $[0, 1]$. Para além disso, a soma de todos os valores da tabela deve ser 1.

Tabela 3.7: Probabilidades conjuntas entre X e Y.

	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$	$\mathbf{y}_4$
x_1	0.09	0.03	0.21	0.29
x_2	0.01	0.25	0.1	0.02

Desdobrando os somatórios fica:

$$\begin{aligned}H_{X\wedge Y} = &-f_{X\wedge Y}(x_1\wedge\mathbf{y}_1)\cdot\log_2\Big(f_{X\wedge Y}(x_1\wedge\mathbf{y}_1)\Big)-f_{X\wedge Y}(x_1\wedge\mathbf{y}_2)\cdot\log_2\Big(f_{X\wedge Y}(x_1\wedge\mathbf{y}_2)\Big)-\\ &f_{X\wedge Y}(x_1\wedge\mathbf{y}_3)\cdot\log_2\Big(f_{X\wedge Y}(x_1\wedge\mathbf{y}_3)\Big)-f_{X\wedge Y}(x_1\wedge\mathbf{y}_4)\cdot\log_2\Big(f_{X\wedge Y}(x_1\wedge\mathbf{y}_4)\Big)-\\ &f_{X\wedge Y}(x_2\wedge\mathbf{y}_1)\cdot\log_2\Big(f_{X\wedge Y}(x_2\wedge\mathbf{y}_1)\Big)-f_{X\wedge Y}(x_2\wedge\mathbf{y}_2)\cdot\log_2\Big(f_{X\wedge Y}(x_2\wedge\mathbf{y}_2)\Big)-\\ &f_{X\wedge Y}(x_2\wedge\mathbf{y}_3)\cdot\log_2\Big(f_{X\wedge Y}(x_2\wedge\mathbf{y}_3)\Big)-f_{X\wedge Y}(x_2\wedge\mathbf{y}_4)\cdot\log_2\Big(f_{X\wedge Y}(x_2\wedge\mathbf{y}_4)\Big)\end{aligned}$$

Substituindo pelos valores tabelados obtém-se:

$$\begin{aligned}H_{X\wedge Y} = &-0.09\cdot\log_2(0.09)-0.03\cdot\log_2(0.03)-0.21\cdot\log_2(0.21)-0.29\cdot\log_2(0.29)-\\ &0.01\cdot\log_2(0.01)-0.25\cdot\log_2(0.25)-0.1\cdot\log_2(0.1)-0.01\cdot\log_2(0.01)\\ =&2.47 \text{ bit}\end{aligned}$$

confirmando-se assim a diminuição na entropia conjunta. A caixa de texto que se segue apresenta uma possível forma de calcular, com o Matlab, o valor da entropia conjunta dado os valores das probabilidades conjuntas.

```
f_X_Y=[0.0900 0.0300 0.2100 0.2900
0.0100 0.2500 0.1000 0.0200];
H=0;
for i=1:2,
   for j=1:4,
      H=H-f_X_Y(i,j)*log2(f_X_Y(i,j));
   end
end
disp(['Entropia Conjunta = ' num2str(H)])
```

Uma forma *vectorizada* de resolver o mesmo problema apresenta-se em baixo:

```
f_X_Y=[0.0900 0.0300 0.2100 0.2900
0.0100 0.2500 0.1000 0.0200];
H=-ones(1,2)*(f_X_Y.*log2(f_X_Y))*ones(4,1);
disp(['Entropia Conjunta = ' num2str(H)])
```

Em teoria da informação define-se ainda **entropia condicional**. A entropia condicional, também designada por **equívoco**, quantifica o que resta da entropia de uma variável aleatória dado que o valor de uma segunda variável aleatória é conhecida. Sejam Y e X essas variáveis aleatórias. A entropia condicional de Y, dado o conhecimento de X, descreve-se como $H_{Y|X}$ e calcula-se por:

$$H_{Y|X} = -\sum_{i=1}^{m}\sum_{j=1}^{n} f_{Y\wedge X}(\mathbf{y}_j\wedge\mathbf{x}_i)\cdot\log_2\Big(f_{Y|X}(\mathbf{y}_j|\mathbf{x}_i)\Big) \tag{3.104}$$

ou ainda,

$$H_{Y|X} = -\sum_{i=1}^{m}\sum_{j=1}^{n} f_{Y|X}(\mathbf{y}_j|\mathbf{x}_i)\cdot f_X(\mathbf{x}_i)\cdot \log_2\Big(f_{Y|X}(\mathbf{y}_j|\mathbf{x}_i)\Big) \tag{3.105}$$

onde $f_{Y|X}(\mathbf{y}_j|\mathbf{x}_i)$ se refere à probabilidade condicional.

Como veremos na subsecção que se segue, o cálculo da entropia condicional é sempre necessária quando existe dependência entre duas fontes. O caso mais paradigmático é aquele em que se pretende determinar a entropia introduzida por um canal de comunicação não-ideal. Observe que, se X e Y forem estatisticamente independentes:

$$H_{Y|X} = H_Y \text{ e } H_{X|Y} = H_X \tag{3.106}$$

De modo a ilustrar o conceito de probabilidade condicional imagine-se uma fonte X que envia, de forma sequencial, dados a partir de um alfabeto com **dois** símbolos x_1 e x_2 com função de probabilidade de massa $f_X(x)$. Admita-se que a informação é enviada por um canal binário simétrico com probabilidade de erro p. A entropia gerada pelo canal de comunicação é calculada por:

$$H_{Y|X} = -\sum_{i=1}^{2}\sum_{j=1}^{2} f_{Y|X}(x_j|x_i)\cdot f_X(x_i)\cdot \log_2\Big(f_{Y|X}(x_j|x_i)\Big)$$

para $Y = \{y_1, y_2\}$ com $y_1 = x_1$ e $y_2 = x_2$ obtém-se:

$$\begin{aligned} H_{Y|X} = &- f_X(x_1)\Big(f_{Y|X}(x_1|x_1)\cdot\log_2\Big(f_{Y|X}(x_1|x_1)\Big) + f_{Y|X}(x_2|x_1)\cdot\log_2\Big(f_{Y|X}(x_2|x_1)\Big)\Big) - \\ &f_X(x_2)\Big(f_{Y|X}(x_1|x_2)\cdot\log_2\Big(f_{Y|X}(x_1|x_2)\Big) + f_{Y|X}(x_2|x_2)\cdot\log_2\Big(f_{Y|X}(x_2|x_2)\Big)\Big) \end{aligned}$$

Como $f_{Y|X}(x_1|x_1) = f_{Y|X}(x_2|x_2) = 1-p$ e $f_{Y|X}(x_1|x_2) = f_{Y|X}(x_2|x_1) = p$ a expressão anterior toma a seguinte forma:

$$\begin{aligned} &H_{Y|X} = \\ &- f_X(x_1)\Big((1-p)\cdot\log_2(1-p) + p\cdot\log_2(p)\Big) + f_X(x_2)\Big(p\cdot\log_2(p) + (1-p)\cdot\log_2(1-p)\Big) \end{aligned}$$

o que leva a:

$$H_{Y|X} = -(1-p)\cdot\log_2(1-p)\Big(f_X(x_1)+f_X(x_2)\Big) - p\cdot\log_2(p)\Big(f_X(x_1)+f_X(x_2)\Big)$$

como $f_X(x_1) + f_X(x_2) = 1$ fica,

$$H_{Y|X} = -(1-p)\cdot\log_2(1-p) - p\cdot\log_2(p) \tag{3.107}$$

ou seja a incerteza introduzida pelo canal, por cada bit transmitido pela fonte, é igual à incerteza de uma fonte X com dois símbolos $\{x_1, x_2\}$ com função probabilidade de massa:

$$f_X(x) = \begin{cases} p & x = x_1 \\ 1-p & x = x_2 \end{cases} \tag{3.108}$$

A relação, entre a entropia condicional e a entropia conjunta, encontra-se descrita pela seguinte igualdade:

$$H_{X\wedge Y} = H_X + H_{Y|X} \tag{3.109}$$

designada por **regra da cadeia** para a entropia.

Um conceito intimamente ligado à entropia é a informação. Como já foi dito a informação não é mais do que a diminuição da entropia. Na subsecção que se segue explora-se este tema introduzindo ainda o conceito de informação mútua.

Informação e Informação Mútua

Anteriormente definiu-se entropia de uma fonte discreta como sendo o valor esperado (médio ponderado) da incerteza. Já se viu que, associado ao conceito de incerteza, existe o conceito de **informação**. Efectivamente, informação é definida como sendo um decréscimo na entropia. Definindo I como a quantidade de informação então:

$$I = H_{antes} - H_{depois} \tag{3.110}$$

onde H_{antes} se refere ao valor da entropia, no receptor, antes da recepção da mensagem e H_{depois} à entropia, também no receptor, após a recepção da mensagem.

A incerteza, depois de receber a mensagem, deve-se a possíveis erros durante a transmissão. Por exemplo admita-se que uma fonte discreta, com uma função de probabilidade uniforme, envia um bit. Nesta situação a probabilidade do bit enviado ser '0' é igual à probabilidade desse bit ser '1'. Assim, antes da mensagem ser enviada, o receptor possui uma incerteza, H_{antes}, de 1 bit[24] antes de receber a mensagem. Suponha-se ainda que essa informação é enviada por um canal binário simétrico com probabilidade de erro igual a 0.01. Neste caso, mesmo depois da mensagem ter chegado ao receptor, este têm ainda incerteza em relação ao seu conteúdo. Essa incerteza depende da probabilidade de erro do canal de acordo com:

$$\begin{aligned} H_{depois} &= -0.99 \cdot \log_2(0.99) - 0.01 \cdot \log_2(0.01) \\ &= 0.0807 \quad \text{bit/símbolo} \end{aligned} \tag{3.111}$$

O que significa que a informação recebida seria inferior à informação enviada. Por outras palavras, foi enviado um bit/símbolo de informação e apenas se foi recebido,

$$\begin{aligned} I &= 1 - 0.0807 \\ &= 0.919 \quad \text{bit/símbolo} \end{aligned} \tag{3.112}$$

Observe que, no caso extremo de uma probabilidade de erro igual a 50%, nenhuma informação seria recebida. Efectivamente a entropia *antes* seria igual à entropia *depois* o que implicaria $I = 0$.

[24] $H = -0.5 \cdot \log_2(0.5) - 0.5 \cdot \log_2(0.5) = 1$

No exemplo anterior H_{depois} não é mais do que a entropia condicional $H_{Y|X}$ estando associada a variável aleatória X ao símbolo emitido pela fonte e Y ao símbolo recebido pelo receptor. Deste modo,

$$I_{X,Y} = H_X - H_{X|Y} \tag{3.113}$$

onde $I_{X,Y}$ designa-se por **informação mútua** [25]. A informação mútua $I_{X,Y}$ mede a dependência entra as variáveis X e Y. Ou seja qual a redução na incerteza de uma variável que pode ser atingida pelo conhecimento da outra. Por outras palavras, a informação mútua define a quantidade de informação que a variável Y contém sobre X. Pela definição de informação mútua retiram-se as seguintes propriedades:

- Se o canal é ideal então $I_{X,Y} = I_Y$;
- Se X e Y são estatisticamente independentes então $I_{X,Y} = 0$;
- A informação mútua é sempre positiva (ou nula): $I_{X,Y} \geq 0$;
- A informação mútua é simétrica. Efectivamente,

$$I_{X,Y} = H_X - H_{X|Y} = H_Y - H_{Y|X} = I_{Y,X} \tag{3.114}$$

- Verificam-se ainda as seguintes igualdades:

$$I_{X,Y} = H_Y + H_{Y|X} \tag{3.115}$$

$$I_{X,Y} = H_X + H_Y - H_{X\wedge Y} \tag{3.116}$$

A informação mútua entre $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_m\}$ e $Y = \{\mathbf{y}_1, \cdots, \mathbf{y}_n\}$ pode ser descrita, com base nas expressões (3.91) e (3.105), da seguinte forma:

$$I_{X,Y} = \sum_{i=1}^{m}\sum_{j=1}^{n} f_{X\wedge Y}(\mathbf{x}_i \wedge \mathbf{y}_j) \cdot \log_2\left(\frac{f_{X\wedge Y}(\mathbf{x}_i \wedge \mathbf{y}_j)}{f_X(\mathbf{x}_i) \cdot f_Y(\mathbf{y}_j)}\right) \tag{3.117}$$

Na subsecção que se segue apresenta-se outro conceito nuclear da teoria da informação. Trata-se da definição de capacidade do canal, ou seja a máxima taxa de transmissão a que a fonte pode pode debitar informação admitindo uma comunicação fiável.

Capacidade de um Canal

Antes de falar sobre a capacidade de um canal de comunicação vamos introduzir o conceito de **razão de informação**.

Considere-se, para isso, dois códigos distintos: Um código (4,2) e um código (8,3). Qual dos dois códigos é mais *informativo*. No primeiro caso enviam-se quatro bits de dados por cada dois bits de informação. No segundo caso envia-se uma *string* de 8 bits por cada mensagem de 3 bit. O *overhead* no primeiro caso é de $2/4 = 0.5$ e no segundo é $5/8 = 0.625$ bits. O que significa que metade dos

[25] Shannon designou esta quantidade por *razão de transmissão*.

bits no primeiro caso não transportam informação e no segundo mais de 60% são redundância. Por este motivo pode-se dizer que o conteúdo informativo do código (4,2) é superior à do código (8,3). Define-se assim a *razão de informação* R como sendo a razão entre o número de bits da mensagem (k) pelo número de bits do código (n) , i.e.

$$R = \frac{k}{n} \quad \text{bit/símbolo} \tag{3.118}$$

Voltemos novamente à situação do código R3. Este código possui $n = 3$ e $k = 1$ e a distância mínima de Hamming é igual a 3. Por isso possui a capacidade de corrigir um erro. Se a informação é enviada por um canal binário simétrico com probabilidade de erro ϵ a probabilidade da mensagem enviada pela fonte, ser descodificada de forma incorrecta, é igual à probabilidade de existirem dois ou mais bits alterados durante a transmissão. Se a variável aleatória x se referir ao número de erros ocorridos durante a transmissão, para que a mensagem não seja correctamente descodificada é necessário que $x \geq 2$. A probabilidade de isso acontecer é facilmente calculada por:

$$\begin{aligned} P(x \geq 2) &= P(x = 2) + P(x = 3) \\ &= \binom{3}{2} \epsilon^2 (1 - \epsilon) + \binom{3}{3} \epsilon^3 \\ &= 3 \cdot \epsilon^2 (1 - \epsilon) + \epsilon^3 \end{aligned} \tag{3.119}$$

Se $\epsilon = 0.01$, a probabilidade do receptor receber a mensagem errada passa de 0.01 para 0.00028. Como se pode imaginar, aumentando a redundância diminui-se a probabilidade de erro. A figura 3.18 mostra um gráfico onde se apresenta a evolução da probabilidade de erro face à razão de informação.

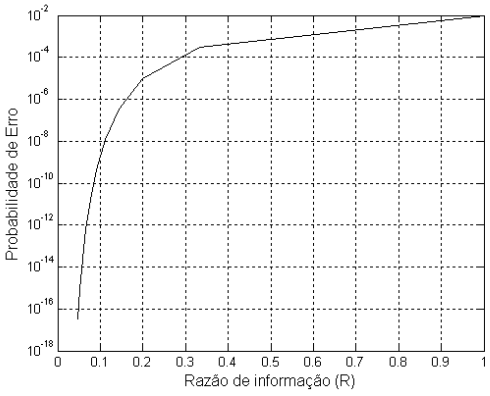


Figura 3.18: Estrutura do descodificador por síndrome para um código linear.

O código, em Matlab, utilizado para gerar a figura 3.18 apresenta-se na caixa subsequente.

A figura anterior sugere que atingir um erro nulo implica uma razão de informação também nula. Seríamos, desta forma, tentados a pensar que qualquer código se encontraria à esquerda de uma recta que passa pela origem do referencial conforme se mostra na figura 3.19.

```
e=0.01;
n=3:2:21;
for i=1:length(n),
   x=0.5*n(i)-0.5*3+2;
   P_e(i)=0;
   for j=x:n(i),
      P_e(i)=P_e(i)+nchoosek(n(i),j)*e^j*(1-e)^(n(i)-j);
   end
end
R=1./n;
semilogy([1 R],[e P_e]);
xlabel('Razão de informação (R)');
ylabel('Probabilidade de Erro');
```

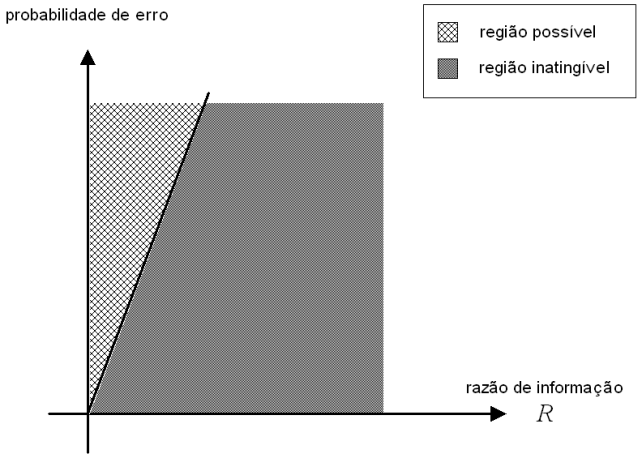


Figura 3.19: Estrutura do descodificador por síndrome para um código linear.

Uma das maiores descobertas em teoria da informação desfaz essa suposição referindo a existência de um valor de R mínimo, para a transmissão fiável de informação, superior a zero. O que implica que a fronteira entre o que é possível alcançar e o que não é possível não é uma recta que passa pela origem mas um ponto alternativo C designado por **capacidade do canal**.

A capacidade de um canal de transmissão refere-se à taxa máxima a que a informação pode circular nesse canal com erro aceitável. Existem vários factores que afectam a capacidade de um canal. Entre eles se destacam:

- **Limite de Banda**. Normalmente o canal de comunicação possui um comportamento do tipo passa-baixo. Assiste-se assim a uma atenuação e distorção do sinal enviado em função do seu conteúdo espectral.
- **Atenuação**. Refere-se às perdas no sinal transmitido. A atenuação varia com diversos factores entre os quais a distância física entre a fonte e o receptor.
- **Ruído**. Tensões e correntes parasitas, induzidas ao longo do canal de comunicação, devido à presença de campos magnéticos variantes no tempo.

Por simplicidade de tratamento, todos estes efeitos são tidos em consideração na definição da probabilidade p de alteração de um bit, durante a sua passagem pelo canal de comunicação. No entanto a capacidade de um canal, ou seja o limite máximo de informação que pode ser transmitido, não depende exclusivamente de p. Basta pensar que na década de 90, do século anterior, era comum aceder à Internet, pela linha telefónica, utilizando MODEMS com taxas de transferência até 56 kbps e, um pouco mais tarde, sobre o mesmo canal mas agora com ADSL é possível a transmissão de uma maior quantidade de informação por unidade de tempo.

A definição formal de capacidade de um canal encontra-se sumariada no teorema 4 [5]:

Teorema 4: *Para qualquer canal discreto sem memória, a capacidade de um canal é dada por:*

$$C = \max_{f_X(\mathbf{x})} I_{X,Y} \tag{3.120}$$

Esta expressão pode ser interpretada da seguinte forma: A capacidade de um canal é o valor mais elevado da informação mútua que pode ser atingido utilizando uma distribuição específica $f_X(\mathbf{x})$ para a fonte discreta X.

A capacidade de um canal binário simétrico, com probabilidade de erro p, é dada por:

$$\begin{aligned} C &= 1 - H_{Y|X} \\ &= 1 + p \cdot \log_2(p) + (1-p) \cdot \log_2(1-p) \end{aligned} \tag{3.121}$$

Por exemplo, um canal simétrico com probabilidade de erro igual a 0.02[26] possui uma capacidade aproximadamente igual a 0.86 bit. O mesmo é dizer que, se forem esperados erros em 2% do tempo, então é possível transmitir mensagens com um conteúdo informativo de até 86% de informação continuando a ser possível a sua descodificação com precisão arbitrária.

Dependendo do valor de p, a capacidade do canal pode ir de 0 a 1 bit. Se $p = 0$, $H_{Y|X} = 0$ bit e logo $C = 1$ bit. Por outro lado, se $p = 1/2$ então $H_{Y|X} = 1$ bit o que leva a $C = 0$ bit. Uma capacidade igual a 1 bit significa que não é necessária a adição de redundância para uma transmissão fiável e uma capacidade igual a 0 bit indica que é impossível enviar qualquer informação pelo canal.

A forma de minimizar o erro de transmissão consiste em adicionar, à mensagem original, informação redundante que permita a detecção e correcção de eventuais erros. Relativamente aos bits de paridade contidos na mensagem apresenta-se o seguinte teorema:

Teorema 6: *Para qualquer $\epsilon > 0$ e $R < C$, para valores elevados de n existe um código de comprimento n e razão maior, ou igual, a R de tal forma que a máxima probabilidade de erro, por bloco, é inferior a ϵ.*

Este teorema identifica a existência de um par codificador/descodificador capaz de transmitir uma mensagem, por um canal imperfeito, com probabilidade

[26] Na prática este valor é extremamente elevado. Actualmente muitos dos canais operam com probabilidades entre 10^{-7} e 10^{-15} [1].

tão pequena quanto se queira. No entanto, para diminuir a probabilidade de erro, o codificador deve gerar códigos com comprimentos cada vez maiores. Este facto obriga a maiores atrasos e requisitos computacionais mais exigentes.

No caso de ser permitida a transmissão com erro, até um valor máximo μ, introduz-se o seguinte teorema:

Teorema 7: *Se for aceitável uma probabilidade de erro, por bit, igual a μ, então podem ser atingidas razões até $R \leq R_\mu$ com,*

$$R_\mu = \frac{C}{1 - H_{Y|X}} \tag{3.122}$$

e onde $H_{Y|X}$ se refere à entropia do canal com probabilidade de erro igual a μ, i.e. $H_{Y|X} = -\mu \cdot \log_2(\mu) - (1-\mu) \cdot \log_2(1-\mu)$

Uma implicação dos teoremas citados anteriormente é o teorema 8 descrito da seguinte forma:

Teorema 8: *Razões de informação superiores a R_μ não são atingíveis.*

Graficamente, os teoremas anteriores definem regiões específicas no plano cujos eixos são a razão de informação R e a probabilidade de erro. A figura 3.20 identifica essas zonas.

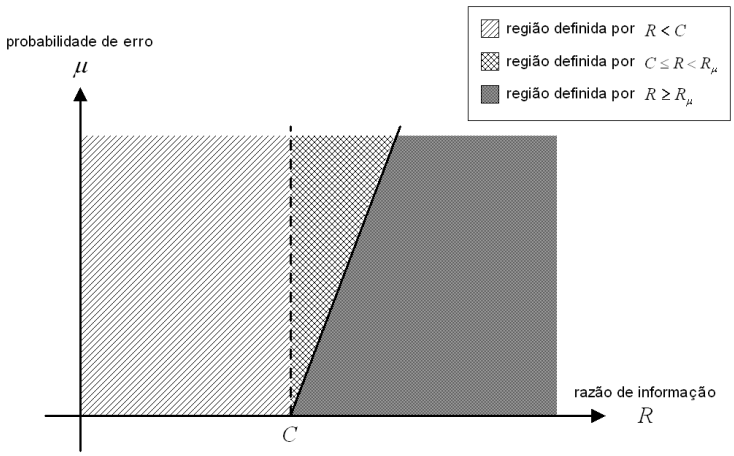


Figura 3.20: Estrutura do descodificador por síndrome para um código linear.

A secção que se segue explora, de forma particular, alguns dos tipos mais comuns de códigos de bloco começando com o célebre código de Hamming[27] e terminando nos códigos cíclicos.

[27] Após a distância de Hamming, distância mínima de Hamming, volume de Hamming, limite de Hamming finalmente o código de Hamming...

3.6.2 Análise de alguns códigos de bloco

Até ao momento apenas se falou, em abstracto, sobre os códigos de canal. Nesta secção apresentam-se alguns exemplos de códigos de bloco lineares extensivamente utilizados. Antes de prosseguir deixa-se aqui um reparo acerca das características ideais que um código deve satisfazer:

- Baixo valor para n de modo a reduzir atrasos;
- Elevado valor para R para fazer um uso eficiente do canal;
- Elevado d para ser capaz de corrigir um número elevado de erros.

No entanto estes objectivos são incompatíveis. Por isso mesmo não existe um código óptimo mas sim um conjunto de códigos que dão preferência a algumas característica em detrimento das restantes.

Nesta secção apresentam-se alguns paradigmas de códigos de bloco. Esses serão divididos em dois tipos distintos: códigos apenas de detecção e códigos de detecção e correcção. Relativamente ao primeiro tipo distingue-se a adição do bit de **paridade**, o **checksum**, e os **códigos cíclicos (CRC)**. De entre os códigos capazes de corrigir erros apresenta-se os códigos de **Hamming** e **Golay**.

3.6.2.1 Paridade e Checksum

A introdução do bit de paridade é uma das técnicas de detecção de erro mais simples. A uma sequência de bits é adicionado mais um bit definido a partir de uma combinação linear de todos os bits da palavra original. Com esta estratégia aumenta-se, em uma unidade, a distância de Hamming. Por isso, em códigos com $d_H^* = 1$, a adição de um bit de paridade faz com que d_H^* passe a ser igual a 2. Como se sabe um código com $d_H^* = 2$ permite a detecção de, até, um erro de transmissão. Observe ainda que a adição de bit de paridade a outro tipo de códigos é comum. Como veremos à frente, o código de Hamming e de Golay estendido envolvem a adição deste tipo de redundância ao código original.

Se uma mensagem $\mathbf{m}$ a ser transmitida tem dimensão $n-1$ bit então o código $\mathbf{c}$, obtido de $\mathbf{m}$ por adição de um bit de paridade possui n bit. Trata-se pos isso de um código de bloco $(n, n-1)$ com $d_H^* = 1$.

Se $\mathbf{m} = [b_1 \ \cdots \ b_{n-1}]$ então $\mathbf{c} = [b_1 \ \cdots \ b_{n-1} \ t_1]$ onde o bit de paridade t_1 é calculado por:

$$t_1 = b_1 + \cdots + b_{n-1} \tag{3.123}$$

A operação em questão é, mais uma vez, a soma módulo 2. O valor de t_1 é sempre tal que o número de bits a '1' em $\mathbf{c}$ é sempre par. Por exemplo se $\mathbf{m} = [1\,0\,1\,1]$ então $t_1 = 1$ e logo $\mathbf{c} = [1\,0\,1\,1\,1]$ de onde se observa que, de facto, o número de bits a '1' em $\mathbf{c}$ é um número **par**.

Portanto a matriz geradora para este código possui o seguinte formato:

$$\mathbf{G} = [\mathbf{I}_{n-1}|\mathbf{1}_{n-1}] \tag{3.124}$$

onde $\mathbf{1}_{n-1}$ se refere a um **vector coluna**, de dimensão $n-1$, com todos os seus elementos iguais à unidade.

O *modus operandis* deste código de bloco é muito simples. Admitindo que d_H^* de $\mathbf{m}$ é igual a 1 então, após a adição do bit de paridade, o código possui distância mínima igual a dois bits. Se, durante a tranmissão do código, ocorrer

um erro em um dos bits (incluindo o bit de paridade), o descodificador irá detectar falta de coerência entre o valor lógico do bit de paridade e o número de bits a '1' da mensagem. Efectivamente este código permite detectar erro na mensagem se existir mais do que um erro de transmissão. Concretamente, se o número de erros ocorridos for **ímpar**, o bit de paridade possui a capacidade de detectar esse tipo de acidente.

No entanto não é possível ao descodificador saber qual ou quais foram os bits que sofreram alteração após a transmissão. Ou seja este tipo de código não permite correcção automática.

Na prática, este tipo de técnica de detecção de erros é utilizada principalmente em sistemas de comunicação ARQ. No domínio dos computadores identifica-se a utilização do bit de paridade nos barramentos de dados **SCSI** e **PCI** ou, na já praticamente extinta, **porta série** (RS232).

Outra técnica de detecção de erro muito utilizada é designada por **checksum**. Admitindo-se uma mensagem com $m \times p$ bits, o procedimento de cálculo do checksum começa por fraccionar essa *string* em m sub-*strings* com p bits. A seguir o codificador executa a soma binária (com transporte) entre essas palavras desprezando os últimos bits de transporte. No final aplica ao resultado da soma o complemento para um (ou complemento para dois) e associa essa string à mensagem inicial. Desta forma a palavra código terá, ao todo, $n = (m+1) \times p$ bits.

De modo a ilustrar o procedimento imagine-se uma mensagem composta por 32 bits: 10110001.10000110.01001100.10100011. Admita-se que o codificador divide a palavra em bytes procedendo, a seguir, à sua soma conforme se representa a seguir:

$$\begin{array}{r} 10110001 \\ 10000110 \\ 01001100 \\ +\ 10100011 \\ \hline 00100110 \end{array} \tag{3.125}$$

Aplicando o complemento para um obtém-se, como *checksum*, a string - 11011001. Deste modo o codificador envia, pelo canal de comunicação, a seguinte palavra-código:

$$10110001.10000110.01001100.10100011.11011001 \tag{3.126}$$

No outro extremo o descodificador particiona a palavra-código recebido em bytes e efectua a soma de todas as parcelas. No caso de não ocorrência de erros de transmissão, o resultado deverá ser 11111111. No presente exemplo:

$$\begin{array}{r} 10110001 \\ 10000110 \\ 01001100 \\ 10100011 \\ +\ 11011001 \\ \hline 11111111 \end{array} \tag{3.127}$$

Admitindo um erro no quarto bit do segundo byte o checksum no descodifi-

cador é:

$$\begin{array}{r} 10110001 \\ 10010110 \\ 01001100 \\ 10100011 \\ 11011001 \\ +\quad\quad\quad\quad \\ \hline 11101111 \end{array} \tag{3.128}$$

O cálculo do checksum é a técnica de detecção de erro utilizada no protocolo de comunicação TCP/IP. Para além da sua utilização no nível mais baixo do protocolo[28], o checksum pode ser encontrado, por exemplo, nos cabeçalhos TCP. Neste caso o checksum é calculado sobre todo o conjunto de dados e enviado no cabeçalho juntamente com outra informação.

3.6.2.2 Códigos Cíclicos

O *checksum*, como estratégia de codificação, possui diversas falhas. Por exemplo o descodificador não é capaz de se aperceber da ocorrência de um erro quando a mensagem é modificada por inversão ou troca de grupos. Falha também quando se adicionam ou removem grupos nulos.

Uma técnica de codificação alternativa, mais poderosa do que as anteriores, é designada por CRC - Cyclic Redundancy Check. Efectivamente esta estratégia de codificação é utilizada numa miríade de aplicações. Entre elas se destacam:

- Redes de computadores (o protocolo Ethernet utiliza CRC-32);
- Unidades de armazenamento de dados (discos rígidos SATA e cartões de memória SD por exemplo);
- Protocolo WEP de autenticação em redes de comunicação de dados sem fios (wireless);
- Em algumas aplicações informáticas como é o caso do WinZip® (ver figura 3.21).

Figura 3.21: Erro de CRC detectado pelo programa de compressão de dados WinZip.

A técnica de codificação por CRC é particularmente útil no caso de erros aleatórios ou "*burst*" de erros. Efectivamente um código CRC com dimensão n são capazes de detectar:

- Todos os erros em um ou dois bit;

[28] Por exemplo nos pacotes Ethernet.

- Qualquer número ímpar de erros;
- Até n erros consecutivos na palavra-código[29].

No entanto não protege contra a alteração intencional dos dados (razão da falha de segurança do protocolo WEP). Ou seja é possível definir uma mensagem alternativa com o mesmo CRC.

Para além disso, e tal como os dois métodos anteriores, a sua implementação em *Hardware* é bastante simples.

A técnica de detecção de erro por CRC deve o seu nome ao facto de este gerar um **código cíclico** que posteriormente é enviado para o receptor (ou eventualmente lido de qualquer suporte de dados).

Códigos cíclicos

Um código $\mathcal{C}$ é designado por cíclico se, para qualquer palavra-código $\mathbf{c} \in \mathcal{C}$: $\mathbf{c} = [c_1 \ \cdots \ c_n]$, a palavra código,

$$\mathbf{c}' = [c_n \ c_1 \ \cdots \ c_{n-1}] \tag{3.129}$$

obtida por rotação de um bit à direita do código $\mathbf{c}$, é também uma palavra-código pertencente ao alfabeto de $\mathcal{C}$.

Por indução prova-se que a palavra-código obtida, por um número arbitrário de rotações à direita, ainda pertence a $\mathcal{C}$. Observe que para uma palavra código de n bits, uma rotação à direita resulta numa segunda palavra-código idêntica à rotação à esquerda de $n-1$ bits. Por este motivo um código é cíclico se qualquer rotação à direita ou esquerda resulta numa palavra-código válida.

Esta definição não significa que todas as palavras-código devem ser formadas apenas por rotação. Significa que qualquer palavra-código pode ser formada, a partir de qualquer outra palavra-código, pela combinação das operações de **rotação** e **soma** [3].

Como exemplo de um código cíclico tem-se o código de repetição R3. Um segundo exemplo segue do código obtido por adição de paridade a uma mensagem de dois bits. Efectivamente o alfabeto nesta situação é $\mathcal{C} = \{000, 011, 101, 110\}$. Como veremos adiante, a técnica de codificação por bit de paridade é um caso particular do código CRC de um bit.

Sequências binárias e polinómios

Antes de apresentar a estrutura de um codificador CRC introduz-se a ligação entre uma sequência binária e um polinómio.

Um polinómio $P(x)$ é qualquer função dada pela soma algébrica de factores do tipo x^k com $k \in \mathbb{N}$ cada qual ponderado por um coeficiente $a_k \in \mathbb{R}$. Ou seja,

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdot + a_n x^k \tag{3.130}$$

Uma forma alternativa de representar o polinómio consiste em apresentar um vector constituído apenas pelo seus coeficientes. Neste caso a posição relativa

[29] Designados por "burst".

de um coeficiente no vector é importante. É utilizando esta estratégia de representação que o Matlab lida com polinómios. Por exemplo o polinómio:

$$P(x) = 3x^2 + 2x + 1 \tag{3.131}$$

É representado em Matlab pelo vector: O cálculo das raízes deste polinómio,

```
>> P_x=[3 2 1];
```

ou seja todos os valores de x que conduzem a $P(x) = 0$ pode ser efectuado recorrendo à função:

```
>> roots(P_x)
ans =
-0.3333 + 0.4714i
-0.3333 - 0.4714i
```

No caso dos coeficientes do polinómio apenas poderem tomar o valor 0 ou 1, ou seja $a_k \in \{0, 1\}, \forall k \in \mathbb{N}$, a representação vectorial de um polinómio coincide com uma string binária. Pode-se assim associar um polinómio a uma mensagem **m**, palavra-código **c** ou redundância de uma sequência binária codificada. Por exemplo a palavra-código $\mathbf{c} = [1\ 0\ 1\ 1\ 0]$ pode ser vista como representando o polinómio $x^4 + x^2 + x$.

Estrutura do codificador

Num código de bloco genérico, qualquer palavra-código **c** é formada a partir da mensagem **m** recorrendo à matriz geradora **G**. Ou seja, como já foi referido anteriormente,

$$\mathbf{c} = \mathbf{m} \cdot \mathbf{G} \tag{3.132}$$

No caso de um código cíclico todas as palavras-código, representadas como um polinómio $c(x)$, são geradas a partir de um polinómio gerador $g(x)$ com base no polinómio associado à mensagem $m(x)$, de acordo com a seguinte igualdade:

$$c(x) = q(x) \cdot g(x) \tag{3.133}$$

onde $q(x)$ é o polinómio obtido a partir do quociente da divisão polinomial entre $m(x)x^{n-k}$ e $g(x)$.

Repare que se **m** for uma sequência binária com k bits, tendo associado um polinómio $m(x)$, então a palavra binária, obtida a partir da concatenação de p bits à direita de **m**, possui um polinómio associado igual a $m(x)x^p$. A figura 3.21 ilustra este conceito.

Admita-se $m(x)$ como sendo o polinómio associado à mensagem **m** e $g(x)$ o polinómio gerador. Relativamente a este último, a *ordem do polinómio gerador deve ser igual à do número desejado de bits de redundância.* Se o código e a mensagem tiverem dimensões n e k respectivamente, então o número de bits de redundância é $n - k$. Num código CRC os bits de redundância são calculado como sendo os coeficientes do polinómio $r(x)$, obtido do resto da divisão entre

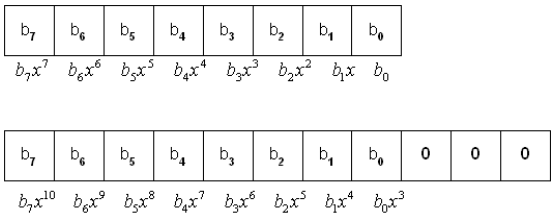


Figura 3.22: Efeito da multiplicação da adição, à direita, de três bits a zeros num byte de informação. Efeito no polinómio associado.

$m(x)x^{n-k}$, e o polinómio gerador $g(x)$. Ou seja,

$$\begin{array}{cc} m(x)x^{n-k} & \left| \frac{g(x)}{q(x)} \right. \\ r(x) & \end{array} \tag{3.134}$$

O polinómio $q(x)$ obtido no quociente é descartado. Note ainda que a ordem do polinómio $r(x)$ é sempre **menor** do que a ordem do polinómio gerador $g(x)$.

Polinómio gerador

A definição da estrutura de um polinómio gerador é um problema complexo cuja descrição ultrapassa o objectivo da presente unidade curricular. No entanto deixam-se aqui alguns reparos acerca de $g(x)$:

- O coeficiente de ordem zero deve ser unitário;
- O coeficiente de ordem mais elevada deve ser unitário;

  Ou seja $g(x)$ deve ter a forma:

$$g(x) = 1 + \sum_{i=1}^{n-k-1} g_i \cdot x^i + x^{n-k} \tag{3.135}$$

- O resto da divisão de $g(x)$ por $x^n - 1$ deve ser zero.

Note-se que quanto maior for a ordem do polinómio melhor é a capacidade de detectar erros. Adicionalmente, supondo que $x^n + 1$ é o produto de p polinómios irredutíveis, existem ao todo 2^p possíveis códigos cíclicos de comprimento n (eventualmente com diferentes dimensões de redundância). Por exemplo o polinómio $x^7 + 1$ pode ser factorizado da seguinte forma:

$$x^7 + 1 = (x + 1)(x^3 + x^2 + 1)(x^3 + x + 1) \tag{3.136}$$

Existem assim, 8 possíveis códigos cíclicos com $n = 7$:

- Código (7,7) com $g(x) = 1$
- Código (7,6) com $g(x) = x + 1$
- Código (7,4) com $g(x) = x^3 + x^2 + 1$

- Código (7,4) com $g(x) = x^3 + x + 1$
- Código (7,2) com $g(x) = (x+1)(x^3 + x^2 + 1) = x^4 + x^2 + x + 1$
- Código (7,2) com $g(x) = (x+1)(x^3 + x + 1) = x^4 + x^3 + x^2 + 1$
- Código (7,1) com $g(x) = (x^3 + x + 1)(x^3 + x^2 + 1) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$
- Código (7,0) com $g(x) = x^7 + 1$

Define-se ainda um polinómio $h(x)$ tal que:

$$g(x) \cdot h(x) = x^n - 1 \tag{3.137}$$

A este polinómio é dado o nome de **polinómio de teste** do código $\mathcal{C}$.

Palavras-código e divisão polinomial

Através de um exemplo concreto irá demonstra-se a metodologia subjacente à criação de um código CRC. Para isso admita-se uma fonte que gera mensagens com um comprimento de 1 byte. Quatro bits de redundância são adicionados à mensagem antes do seu envio pelo canal de transmissão. O polinómio gerador utilizado consiste no CRC-4-ITU dado por $x^4 + x + 1$. Admita-se que a fonte pretende enviar a mensagem [0 1 1 1 0 1 0 1]. O polinómio $m(x)$ tem portanto a seguinte estrutura:

$$m(x) = x^6 + x^5 + x^4 + x^2 + 1 \tag{3.138}$$

e logo $m(x)x^4$ é dado por $x^{10} + x^9 + x^8 + x^6 + x^4$. O polinómio de redundância $r(x)$ é obtido a partir do resto da divisão:

$$\begin{array}{l} x^{10} + x^9 + x^8 + x^6 + x^4 \\ \quad r(x) \end{array} \quad \Big| \underline{x^4 + x + 1} \\ \quad q(x) \tag{3.139}$$

Para efectuar a divisão basta considerar os coeficientes dos polinómios passando a relação anterior a ser definida da seguinte forma:

$$1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0 \quad \Big| \underline{1\ \ 0\ \ 0\ \ 1\ \ 1} \tag{3.140}$$

A operação de divisão realizada passo-a-passo é apresentada de seguida. O procedimento é semelhante à da divisão convencional e as diferenças e produtos são realizados em módulo 2.

$$\begin{array}{rccccccccccc|l} & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & \underline{1\ \ 0\ \ 0\ \ 1\ \ 1} \\ - & 1 & 0 & 0 & 1 & 1 & & & & & & & 1 \\ \hline & 0 & 1 & 1 & 1 & 0 & & & & & & & \end{array} \tag{3.141}$$

$$\begin{array}{rccccccccccc|l} & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & \underline{1\ \ 0\ \ 0\ \ 1\ \ 1} \\ & & 1 & 1 & 1 & 0 & 0 & & & & & & 1\ \ 1 \\ - & 1 & 0 & 0 & 1 & 1 & & & & & & & \\ \hline & & 0 & 1 & 1 & 0 & 1 & & & & & & \end{array} \tag{3.142}$$

$$
\begin{array}{ccccccccccc|ccccc}
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
\cline{12-16}
 & 1 & 1 & 1 & 0 & 0 & & & & & & 1 & 1 & 1 & & \\
 & & 1 & 1 & 0 & 1 & 1 & & & & & & & & & \\
 & - & 1 & 0 & 0 & 1 & 1 & & & & & & & & & \\
\cline{1-6}
 & & 0 & 1 & 0 & 0 & 0 & & & & & & & & &
\end{array}
\tag{3.143}
$$

$$
\begin{array}{ccccccccccc|ccccc}
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
\cline{12-16}
 & 1 & 1 & 1 & 0 & 0 & & & & & & 1 & 1 & 1 & 1 & \\
 & & 1 & 1 & 0 & 1 & 1 & & & & & & & & & \\
 & & & 1 & 0 & 0 & 0 & 0 & & & & & & & & \\
 & & - & 1 & 0 & 0 & 1 & 1 & & & & & & & & \\
\cline{1-7}
 & & & 0 & 0 & 0 & 1 & 1 & & & & & & & &
\end{array}
\tag{3.144}
$$

$$
\begin{array}{ccccccccccc|cccccccc}
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & & & \\
\cline{12-19}
 & 1 & 1 & 1 & 0 & 0 & & & & & & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
 & & 1 & 1 & 0 & 1 & 1 & & & & & & & & & & & & \\
 & & & 1 & 0 & 0 & 0 & 0 & & & & & & & & & & & \\
 & & & & & & 1 & 1 & 0 & 0 & 0 & & & & & & & & \\
 & & & & & - & 1 & 0 & 0 & 1 & 1 & & & & & & & & \\
\cline{1-10}
 & & & & & & 0 & 1 & 0 & 1 & 1 & & & & & & & &
\end{array}
\tag{3.145}
$$

De onde se conclui que o polinómio $r(x)$ é x^3+x+1. Portanto os quatro bits de redundância, a enviar em conjunto com os bits da mensagem, são [1 0 1 1].

Matriz geradora e matriz teste de paridade

Estando na posse do polinómio gerador $g(x)$, conforme expresso pela equação (3.135), é possível definir a matriz geradora $\mathbf{G}$, associada a um código equivalente não sistemático. Para isso observe que o polinómio gerador possui a seguinte representação como string binária de dimensão n:

$$
g(x) \rightarrow \left[\begin{array}{ccccccc} 1 & g_1 & \cdots & g_{n-k-1} & 1 & \underbrace{0 \quad \cdots \quad 0}_{k-1} \end{array} \right] \tag{3.146}
$$

A matriz geradora $\mathbf{G}$ é obtida do vector anterior a partir de k rotações, de um bit, à direita. Ou seja,

$$
\mathbf{G} = \left[\begin{array}{ccccccccc}
1 & g_1 & \cdots & g_{n-k-1} & 1 & 0 & 0 & \cdots & 0 \\
0 & 1 & g_1 & \cdots & g_{n-k-1} & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & g_1 & \cdots & g_{n-k-1} & 1 & 0 & \cdots \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & 0 & 1 & g_1 & \cdots & g_{n-k-1} & 1
\end{array} \right]
$$

A mesma coisa pode ser dita a respeito da matriz teste de paridade. Se $h(x)$ for o polinómio de teste de paridade então,

$$
h(x) \rightarrow \left[\begin{array}{ccccccc} 1 & h_1 & \cdots & h_{k-1} & 1 & \underbrace{0 \quad \cdots \quad 0}_{n-k-1} \end{array} \right] \tag{3.147}
$$

Mais uma vez a matriz **H** é obtida por rotações sucessivas do vector definido na expressão anterior. Agora o número de linhas é igual ao número de bits de redundância ou seja $n-k$. A estrutura da matriz **H** tem o seguinte aspecto:

$$\mathbf{H} = \begin{bmatrix} 1 & h_1 & \cdots & h_{k-1} & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & h_1 & \cdots & h_{k-1} & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & h_1 & \cdots & h_{k-1} & 1 & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 1 & h_1 & \cdots & h_{k-1} & 1 \end{bmatrix} \tag{3.148}$$

Por exemplo se $g(x) = x^3 + x + 1$ e se $h(x) = x^4 + x^2 + x + 1$, cujo produto é igual a $x^7 + 1$, então[30] :

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{3.149}$$

e

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{3.150}$$

O polinómio síndrome

O codificador envia um código que consiste na concatenação dos bits da mensagem mais os bits de redundância. Em termos polinomiais a palavra-código $c(x)$ é expressa por $m(x)x^{n-k} + r(x)$. Atendendo à expressão (3.134) o mesmo é dizer que

$$m(x)x^{n-k} + r(x) = q(x) \cdot g(x) = c(x) \tag{3.151}$$

Durante o processo de descodificação, o receptor recebe a palavra-código **d**, à qual corresponde o polinómio $d(x)$, e divide-a pelo polinómio gerador $g(x)$. Analisando o resto dessa divisão o descodificador conclui que a palavra-código recebida pertence ao alfabeto $\mathcal{C}$ se todos os coeficientes do polinómio forem nulos. Caso isso não aconteça o descodificador assinala erro de transmissão. Efectivamente, se $d(x) = c(x)$ então $d(x) = q(x) \cdot g(x)$. Dividindo $d(x)$ por $g(x)$ resulta em $q(x)$ e o resto é zero.

No caso de um código CRC define-se **síndrome** como o polinómio $s(x)$ resultante do resto da divisão de $d(x)$ pelo polinómio gerador $g(x)$. Se o síndrome $s(x)$ for igual a zero o descodificador admite que não existiu erro de transmissão. Caso o síndrome seja diferente de zeros o descodificador pode tentar recuperar o erro a partir da estimação da sequência de erro mais provável. Com efeito

[30] $(x^3 + x + 1)(x^4 + x^2 + x + 1) = x^7 + x^5 + x^4 + x^3 + x^5 + x^3 + x^2 + x + x^4 + x^2 + x + 1$ como, na base 2, $a + b = a - b$ o polinómio anterior é equivalente a $x^7 + x^5 + x^4 + x^3 - x^5 - x^3 + x^2 + x - x^4 - x^2 - x + 1 = x^7 + 1 = x^7 - 1$

observe que se $d(x) = c(x) + e(x)$, onde $e(x)$ é o polinómio de erro associado a $\mathbf{e}$, então o síndrome $s_d(x)$ de $d(x)$ é igual a:

$$\begin{aligned} s_d(x) &= d(x) \mod g(x) \\ &= \big(c(x) + e(x)\big) \mod g(x) \\ &= c(x) \mod g(x) + e(x) \mod g(x) \\ &= e(x) \mod g(x) \\ &= s_e(x) \end{aligned} \tag{3.152}$$

onde $s_e(x)$ se refere ao polinómio associado ao síndrome do erro. Atendendo à igualdade definida na expressão (3.152), uma das formas do descodificador recuperar do erro de transmissão consiste em determinar o polinómio síndrome $s_d(x)$ e pesquisar, numa tabela de correspondências entre polinómios síndrome e erros, o polinómio $e(x)$ mais provável. A tabela 3.8 representa um conjunto de possíveis correspondências.

Tabela 3.8: Tabela de correspondências entre possíveis polinómios erro e síndromes.

$e(x)$	$s(x)$
1	$1 \mod g(x)$
x	$x \mod g(x)$
x^2	$x^2 \mod g(x)$
$\vdots$	$\vdots$
$x+1$	$x+1 \mod g(x)$
x^2+1	$x^2+1 \mod g(x)$
$\vdots$	$\vdots$

3.6.2.3 Códigos de Hamming

Os códigos de Hamming representam uma família de códigos de bloco com a característica $(2^m - 1, 2^m - m - 1)$ para $m \geq 2$ tendo como denominador comum o facto da distância mínima de Hamming ser constante e igual a 3. Assim, os código de Hamming conseguem detectar erros em, até, dois bits e permitem recuperar de um bit errado. Foi exactamente esta característica que motivou o seu criador, Richard Hamming, durante os meados do século XX a desenvolvê-los.

Devido à sua simplicidade, os códigos de Hamming são largamente utilizados em memórias (RAM) para computadores.

O código de Hamming (7,4)

Em 1950 Hamming apresentou o código (7,4) capaz de encapsular mensagens de 4 bit em palavras-código de 7 bits. Como se disse no início da secção, este código possui a particularidade de ter $d_H^* = 3$ o que lhe permite detectar dois erros e corrigir um. Frequentemente, e como acontece na sua aplicação em memórias de computador, um bit de paridade é adicionado ao código original tornando-o num código (8,4) com $d_H^* = 4$. Esta extensão do código (7,4) permite detectar

até três bit errados. No entanto, e ao contrário do código (7,4), o código (8,4) não é um código perfeito.

Nesta secção o código de Hamming (7,4) será vista sobre diversas perspectivas. A primeira delas começa por estabelecer a matriz geradora e a matriz de teste de paridade.

CVamos começar pela matriz **H**. Como $d_H^* = 3$ significa que o número mínimo de colunas linearmente independentes de **H** é três. O número de bits do código é 7 e o número de bits de redundância é três. Assim sendo, a matriz **H** será uma matriz com três linhas e sete colunas. Atendendo à distância de Hamming não pode haver nenhuma coluna a **zeros**, nem nenhum par de colunas idêntico. Como o número possível de strings distintas com três bits é 8 e, para além disso, como três das suas colunas devem formar a matriz identidade, a única configuração possível é a seguinte:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.153}$$

o que leva a que,

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.154}$$

O código de Hamming (7,4) pode ser descrito, de forma alternativa, como sendo um código cíclico cujo polinómio gerador é $g(x) = x^3 + x + 1$ e o polinómio teste de paridade é $h(x) = x^4 + x^2 + x + 1$.

A terceira, e última forma, de entender como o mapeamento da mensagem para uma palavra-código é realizado resulta de um diagrama conforme se explica adiante. Imagine-se para já uma mensagem **m** composta por quatro bit $\{m_1, m_2, m_3, m_4\}$. O código de Hamming acrescenta a esses quatro, três bits adicionais $\{t_5, t_6, t_7\}$ de acordo com a seguinte sequência:

1. Três circunferências são desenhadas de modo a que todas se interceptem. O aspecto deste diagrama encontra-se representado na figura 3.23.
2. Nos espaços centrais, limitados pelas curvas, insere-se a informação da mensagem original $\{m_1, m_2, m_3, m_4\}$.
3. Os valores lógicos dos bits $\{t_5, t_6, t_7\}$ devem ser definidos de modo a que, dentro de cada círculo, o número de bits a '1' seja par.

De modo a ilustrar este procedimento considere-se a seguinte mensagem: $\mathbf{m} = [1\ 0\ 1\ 0]$. Esta sequência é representada no diagrama circular ficando com o aspecto ilustrado na figura 3.24.

Para que o número de bits a '1' na circunferência superior seja **par**, o bit de redundância t_5 deve ser igual a '0'. Da mesma forma, para que o numero de bits a '1' da circunferência à esquerda seja **par** é necessário que t_6 seja igual a '1'. Finalmente é fácil ver que t_7 também deve ser igual a '1'. A figura 3.25 resume este procedimento. Portanto a mensagem $\mathbf{m} = [1\ 0\ 1\ 0]$ codificada fica com o seguinte aspecto:

$$\mathbf{c} = [1\ 0\ 1\ 0\ 0\ 1\ 1] \tag{3.155}$$

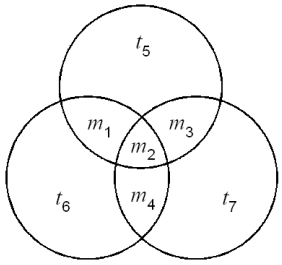


Figura 3.23: Diagrama circular para descrever o código (7,4) de Hamming.

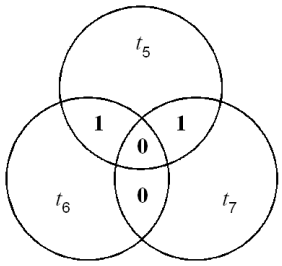


Figura 3.24: O diagrama preenchido com a mensagem $\mathbf{m} = [1\,0\,1\,0]$.

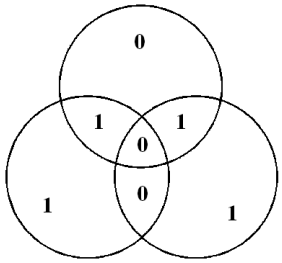


Figura 3.25: O diagrama preenchido com a mensagem $\mathbf{m} = [1\,0\,1\,0]$ e com os bits de redundância.

É possível confirmar que esta palavra-código pertence ao alfabeto do código de Hamming (7,4) recorrendo, por exemplo, ao cálculo do polinómio de síndrome. Com efeito,

$$\begin{array}{rcccccccc} & 1 & 0 & 1 & 0 & 0 & 1 & 1 & \\ - & 1 & 0 & 1 & 1 & & & & \\ \hline & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \\ & & & - & 1 & 0 & 1 & 1 & \\ \hline & & & & 0 & 0 & 0 & 0 & \end{array} \qquad \begin{array}{c|cccc} & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 & \end{array} \tag{3.156}$$

o que leva a que $s(x) = 0$.

3.6.2.4 Código de Golay

Juntamente com os códigos de Hamming (isto para além dos triviais), o código de Golay fecha a família dos poucos códigos perfeitos. Este gera palavras-código de dimensão 23 a partir de mensagens de 12 bit. É portanto um código (23,12). A distância mínima de Hamming deste código é 7 o que lhe permite detectar até seis erros e corrigir 3.

Historicamente este código foi utilizado durante as missões espaciais Voyager 1 e 2 nos finais da década de setenta, princípios da década de oitenta, do segundo milénio. As sondas não-tripuladas lançadas nessas missões, com objectivos de pesquisa científica, transmitiram para o planeta Terra centenas de fotografias (a cores) a partir dos planetas Jupiter e Saturno.

O código de Golay pode ser construído de várias formas. Uma delas deriva da utilização do polinómio:

$$g_1(x) = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1 \tag{3.157}$$

ou então,

$$g_2(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1 \tag{3.158}$$

pois ambos são factores de $x^{23} + 1$. Efectivamente,

$$x^{23} + 1 = (x + 1)g_1(x)g_2(x) \tag{3.159}$$

Tal como no código de Hamming, o código de Golay pode ser estendido num código quasi-perfeito (24,12) com a adição de um bit de paridade ao código (23,12). Neste caso a distância mínima de Hamming passa a ser igual a 8.

Capítulo 4

Exercícios

4.1 Exercícios Teorico-Práticos

Exercício 1 Uma fonte discreta envia, por um canal binário simétrico com probabilidade de erro 0.01, uma string binária composta por 4 bit.

a) Calcule as probabilidades de ocorrerem 1, 2, 3 e 4 erros na palavra enviada.

b) Calcule a probabilidade de existirem menos de 3 erros na palavra enviada.

c) Calcule a probabilidade de existirem três ou mais erros.

d) Determine a probabilidade da mensagem enviada ter sido 1011 dado que a mensagem recebida foi 0011.

Exercício 2 Uma fonte discreta gera sequências binárias a partir do alfabeto $\mathcal{C} = \{00, 01, 10, 11\}$ com função massa de probabilidade:

$$f_{\mathcal{C}}(\mathbf{c}) = \begin{cases} \frac{1}{8} & \text{se } \mathbf{c} = 00 \\ \frac{1}{4} & \text{se } \mathbf{c} = 01 \\ \frac{1}{2} & \text{se } \mathbf{c} = 10 \\ \frac{1}{8} & \text{se } \mathbf{c} = 11 \end{cases} \tag{4.1}$$

calcule a probabilidade da fonte gerar a sequência 10.00.11.11.01.

Exercício 3 Um bloco de informação é transmitido através de um canal com ruído. Sempre que seja detectado um erro a transmissão repete-se até se obter uma transmissão sem erros. Qual o espaço amostral? Defina o evento "O número de transmissões é inferior a 10".[8]

Exercicio 4 Num sistema de comunicação binário o utilizador introduz na entrada sequências de símbolos 0 ou 1 que são em seguida transmitidos e sujeitos a erro no canal de transmissão. No receptor uma decisão é tomada quando o sinal é recebido atribuindo-se, de novo, os valores 0 e 1. Se 5% for a probabilidade de erro calcule qual a entrada mais provável se à saída temos 1 admitindo que as entradas 0 e 1 têm a mesma probabilidade de ocorrência. [8]

Exercício 5 Um canal de comunicação introduz erros com uma probabilidade 0.1%. O transmissor envia cada bit três vezes e no receptor existe um descodificador que decide qual o bit recebido depois de aplicar uma regra de maioria. Qual a probabilidade de tomar a decisão incorrecta. [8]

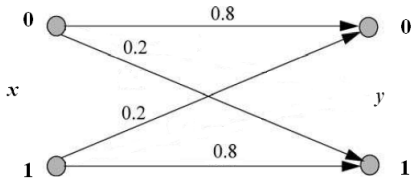
Exercício 6 Um canal digital de comunicação transmite um sinal como uma colecção de "uns" e "zeros". Admite-se que 40% dos "1" e 33% dos "0" são alterados durante a transmissão. Suponha-se que, numa mensagem, a razão entre o número de "1" e "0" é 5/3. Qual a probabilidade do sinal recebido ser idêntico ao sinal transmitido se:

a) O sinal recebido foi "1".

a) O sinal recebido foi "0".

Exercício 7 Um bloco de informação contendo 100 bit é transmitido através de um canal de comunicação que tem uma probabilidade de erro 1×10^{-3}. Determine a probabilidade de, na recepção, um bloco ter 3 ou mais erros (admite-se independência entre bits). [8]

Exercício 8 Considere o canal binário simétrico representado em baixo.



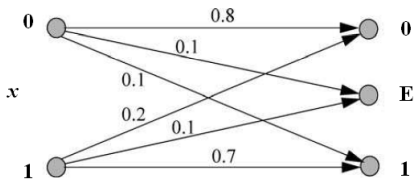
Admita ainda que a fonte gera os símbolos 0 e 1 de acordo com as seguintes probabilidades: $P(x = 1) = 0.6$ e $P(x = 0) = 0.4$.

a) Qual a probabilidade de se ter transmitido o símbolo 1 dado que se recebeu 0?

b) Qual a probabilidade de se ter transmitido o símbolo 0 dado que se recebeu 1?

c) A probabilidade de erro.

Exercício 9 Considere o canal de comunicação ilustrado na figura subsequente[2].



Os símbolos transmitidos são o 0 e o 1. Contudo podem ser recebidos três símbolos diferentes 0, 1 ou E (símbolo irreconhecível). Define-se o símbolo de entrada $x \in \{0, 1\}$ e o símbolo de saída $y \in \{0, 1, E\}$. Se a função probabilidade de massa da fonte for uniforme determine:

a) As probabilidades do símbolo à saída ser 0, 1 ou E.

b) Se for recebido um 0, qual a probabilidade de ter sido enviado um '1'?

c) Se o símbolo recebido for irreconhecível, qual a probabilidade de ter sido transmitido 1?

d) Se for recebido o símbolo 1, qual a probabilidade de ter sido transmitido 1?

Exercício 10 Um porto RS232 envia uma sequência de 7 bit, através de um canal binário simétrico com probabilidade de erro 0.01, e o receptor recebeu a combinação $d = 0001111$. Qual a probabilidade da mensagem enviada ter sido $m = 0001111$? Qual a probabilidade da mensagem enviada ter sido $m = 1001110$?

Exercício 11 Considere o sistema de detecção de erro que consiste em adicionar, a uma mensagem de 5 bit, um bit adicional de paridade. Represente as matrizes $\mathbf{G}$ e $\mathbf{H}$ e demonstre que $\mathbf{c} \cdot \mathbf{H}^T = \mathbf{0}$ se o bit de paridade for concordante com a mensagem.

Exercício 12 Admita-se um código linear com o seguinte alfabeto:

$$\mathcal{C} = \{00000, 01011, 10110, 11101\}$$

a matriz paridade é dada por:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Apresente um possível vector padrão e calcule, para cada classe lateral, o respectivo síndrome.

Exercício 13 Considere um código linear $\mathcal{C}$ cuja matriz $\boldsymbol{\alpha}$ é dada por:

$$\boldsymbol{\alpha} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

a) Escreva as matrizes geradora ($\mathbf{G}$) e de teste de paridade ($\mathbf{H}$).

b) Exprima todas as palavras do alfabeto de $\mathcal{C}$

c) Determine as distâcias de Hamming entre os vectores de $\mathcal{C}$. Qual o valor da distância mínima?

d) Quantos erros este código consegue detectar e corrigir?

Exercício 14 Considere um código linear $\mathcal{C}$ com matriz geradora:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

a) Escreva todos os elementos de $\mathcal{C}$.

b) Com base na matriz $\mathbf{H}$ determine a mínima distância de Hamming.

c) Se este código for utilizado para a detecção de erro num canal binário simétrico com probabilidade de erro igual a 0.1, calcule a probabilidade de, num vector recebido, o descodificador não detectar a existência de erro.

Exercício 15 Considere o código binário com a seguinte matriz geradora:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

a) Determine o valor para d_H^*.

b) Construa a matriz padrão.

c) Descodifique os vectores 01010 e 11110.

d) Dê um exemplo de uma palavra código que, após dois erros, possa ser corrigida e o exemplo de uma que não possa.

e) Admitindo que a informação é transmitida por um canal com probabilidade de erro igual a 0.02 determine a probabilidade da palavra descodificada ser igual à enviada.

Exercício 16 Seja $\mathcal{C}$ um código (6,3) com matriz de teste de paridade:

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

a) Qauntos erros o código tem a capacidade de corrigir?

b) Escreva a matriz $\mathbf{G}$, determine os líderes de classe e estabeleça a matriz padrão.

c) Determine o valor do síndrome para todos os líderes das classes laterais.

d) Descodifique as sequências 110010 e 101100 utilizando o método de descodificação pelo síndrome.

e) Se o código é transmitido por um canal binário simétrico com probabilidade de erro igual a 0.01, determine a probabilidade de um vector recebido ser correctamente descodificado. Calcule também a probabilidade de serem detectados erros.

Exercício 17 Coloque a seguinte matriz $\mathbf{G}$ na forma padrão:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Exercício 18 Coloque a seguinte matriz $\mathbf{G}$ na forma padrão:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Exercício 19 Quantos dos 10 valores inteiros (0 a 9) poderiam ser codificados com um código de 4 bit garantindo $d_H^* = 3$?

Exercício 20 Mostre que o código R-3 é perfeito.

Exercício 21 Demonstre a desigualdade:

$$H \leq \log_2(n_s)$$

para uma fonte discreta com n_s símbolos.

Exercício 22 Uma fonte discreta possui um alfabeto de duas palavras-código, $\{01, 10\}$. Cada um dos símbolos é enviado por um canal binário simétrico com probabilidade de erro 0.03. Sabendo que a função probabilidade de massa da fonte é dada por:

$$f_X(\mathbf{x}) = \begin{cases} 3/8 & \text{se } \mathbf{x} = 01 \\ 5/8 & \text{se } \mathbf{x} = 10 \end{cases} \tag{4.2}$$

Determine:

a) A entropia da fonte.

b) A entropia do canal.

c) A informação mútua.

Exercício 23 Demonstre que, para um canal binário simétrico com probabilidade de erro p, a entropia condicional $H_{Y|X}$ é:

$$H_{Y|X} = -p \cdot \log_2(p) - (1-p) \cdot \log_2(1-p)$$

Exercício 24 Calcule a informação mútua entre X e Y para o canal binário simétrico com $p = 0.15$ sabendo que a função probabilidade de massa de X é:

a)

$$f_X(x) = \begin{cases} 0.5 & \text{se } x = 0 \\ 0.5 & \text{se } x = 1 \end{cases}$$

b)

$$f_X(x) = \begin{cases} 0.9 & \text{se } x = 0 \\ 0.1 & \text{se } x = 1 \end{cases}$$

Exercício 25 Considere um canal de informação binário descrito pela seguinte matriz de probabilidades[1]:

$$\begin{bmatrix} 3/5 & 2/5 \\ 1/3 & 2/3 \end{bmatrix}$$

Determine:

a) A entropia do alfabeto da fonte H_X admitindo uma função de probabilidade uniforme.

b) A entropia condicional $H_{Y|X}$.

c) A informação mútua.

Exercício 26 Considere um canal de informação binário descrito pela seguinte matriz de probabilidades (canal Z):

$$\begin{bmatrix} 1 & 0 \\ q & 1-q \end{bmatrix}$$

Determine:

a) A entropia do alfabeto da fonte H_X considerando que um dos símbolos têm o dobro da probabilidade de ser gerado pela fonte.

b) A entropia condicional $H_{Y|X}$.

c) A informação mútua.

Exercício 27 Qual dos seguintes códigos são cíclicos:

a) $\{0000, 1100, 0110, 0011, 1001\}$

b) $\{00000, 10110, 01101, 11011\}$

c) Código de dimensão 4 com todas as palavras com peso de Hamming igual a 2.

Exercício 28 Dada a factorização de $x^7 - 1$ no seguinte produto irredutível de polinómios:

$$(x+1)(x^3+x+1)(x^3+x^2+1)$$

Determine todos os possíveis códigos cíclicos de dimensão 7.

Exercício 29 Considere um código (7,3) gerado pelo seguinte polinómio:

$$g(x) = x^4 + x^3 + x^2 + 1$$

Determine, assumindo $\mathbf{m} = 101$,

a) O polinómio $r(x)$.

b) A palavra-código $\mathbf{c}$.

c) Admita $\mathbf{d}$ como sendo uma rotação de um bit, à esquerda de $\mathbf{c}$. Confirme que o polinómio síndrome é zero.

[1] As linhas estão associadas à fonte e as colunas ao receptor

Exercício 30 Considere um código (15,5) gerado pelo seguinte polinómio:

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$$

a) Obtenha a palavra-código associada à mensagem $\mathbf{m} = 10101$.

b) Mostre que o polinómio $c(x) = x^{14}+x^8+x^6+x^4+1$ pode representar uma palavra-acódigo transmitida sem erros.

Exercício 31 Sabendo que:

$$x^9 + 1 = (x + 1)(x^6 + x^3 + 1)(x^2 + x + 1)$$

Determine,

a) Um polinómio gerador para um código (9,6).

b) A matriz geradora $\mathbf{G}$ desse código na forma sistemática.

Exercício 32 Admitindo a seguinte factorização do polinómio $x^7 - 1$:

$$x^7 - 1 = (x - 1)(x^3 + x + 1)(x^3 + x + 1)$$

Determine,

a) Quantos códigos cíclicos diferentes de dimensão 7 são possíveis?

b) Descreva as dimensões possíveis para um código cíclico de comprimento 7.

c) Escreva o polinómio $g(x)$ e o polinómio de teste de paridade para um código (7,4).

d) Para o código da alínea anterior descreva as matrizes $\mathbf{G}$ e $\mathbf{H}$ (na forma sistemática).

e) Escreva todas as palavras-código, para um código (7,3), utilizando um dos dois possíveis polinómios geradores de ordem 4.

Exercício 33 Determine a matriz geradora e a matriz teste de paridade do código de Hamming gerado por:

$$g(x) = x^3 + x + 1$$

Exercício 34 Mostre que qualquer código de Hamming é perfeito.

Exercício 35 Obtenha as matrizes $\mathbf{G}$ e $\mathbf{H}$ para o código de Hamming $(2^m - 1, 2^m - m - 1)$ para $m = 4$.

Exercício 36 Obtenha as matrizes $\mathbf{G}$ e $\mathbf{H}$, na forma canónica, para o código de Hamming extendido $(8, 4)$.

Exercício 37 A partir do polinómio $h(x) = x^4+x^2+x+1$ obtenha a matriz $\mathbf{H}$ e mostre que se da matriz teste de paridade do código de Hamming (7,4).

4.2 Exercícios para Matlab

Exercício 1

Desenvolva um *script* para Matlab que permita confirmar a lei dos grandes números. Para isso gere N experiências com resultado $\{0, 1\}$ (compor)

Exercício 2

Construa uma função que tenha como argumento um vector e retorne uma matriz circulante. Deve possuir como argumento de entrada um vector `x` com dimensão n e deve retornar a matriz circulante. O protótipo da função deverá ser do tipo:

```
A = matriz_circular(x)
```

Exercício 3

Construa uma função que retorne o valor das combinações de n elementos tomados k-a-k sem recorrer à função `factorial()`. Compare o resultado da sua função com o valor obtido por:

$$\left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k!\,(n-k)!} \tag{4.3}$$

utilizando explicitamente a função `factorial()`. A função deve possuir dois argumentos de entrada n e k e uma variável de saída `C` (por exemplo: `C = combinacao(n,k)`). O programa deve ainda avaliar os parâmetros de entrada e avisar o utilizador no caso dos parâmetros serem inválidos.

Exercício 4

Com base na função anterior, escreva uma função para Matlab capaz de calcular a probabilidade binomial para parâmetros arbitrários n, k e r, onde n representa o número total de experiências, k o número de sucessos em n experiências e r a probabilidade de sucesso (por exemplo: `P = binomial(n,k,r)`)

Exercício 5

Utilizando a função anterior, e para $n = 2$ e $k = 1$, trace o gráfico de $P(x)$ em função de r. Admita para r 1000 valores igualmente espaçados entre 0 e 1.

Exercício 6

Desenvolva uma função capaz de retornar a codificação **R3** de uma palavra de dimensão arbitrária. A função deve ter um argumento de entrada vectorial com elementos 0 ou 1 e um argumento de saída com a palavra-código equivalente. (por exemplo: `vector_out=codR3(vector_in)`)

Exercício 7

Repita o exercício anterior para o caso genérico de um código $\mathbf{R}n$ para n ímpar maior ou igual a 3. Ou seja a função agora, para além da mensagem, deve possuir como segundo argumento de entrada o valor de n. (por exemplo: `vector_out=codR3(vector_in,n)`)

Exercício 8

Desenvolva uma função que simule o comportamento de um canal binário

simétrico (**BSC**) sem memória. A função deve admitir um vector de entrada e deve fornecer um vector de saída com a mesma dimensão. Para além do vector de entrada a função deve transportar o valor da probabilidade de erro (r).
(por exemplo: `vector_recep=canal_binario_simetrico(vector_emi,r)`)

Exercício 9

Desenvolva uma função que gere um vector binário aleatório (X), com dimensão arbitrária n, com valor esperado de bit a '1' (E[X=1]) igual a $r \times n$. A função deve ter dois argumentos de entrada: a dimensão do vector e o valor da probabilidade r. Deve retornar um vector com n elementos, ou zeros ou uns, que satisfaça a condição de valor esperado. Teste o seu programa calculando empiricamente o valor esperado para diferentes valores de n e r.

Exercício 10

Desenvolva um conjunto de funções para Matlab que executem as seguintes operações:

- Soma módulo-2 de duas variáveis (escalares, vectores ou matrizes).
- Produto módulo-2 de duas variáveis (escalares, vectores ou matrizes).
- Rotação e *shift* de n bit à direita.
- Rotação e *shift* de n bit à esquerda.

Exercício 11

Desenvolva uma função para Matlab que retorne a distância de Hamming e o peso de Hamming de um par de strings binárias. A função deve ser do tipo:

```
[dH,wH]=HammDist(v1,v2)
```

onde `dH` é um escalar e `wH` um vector onde o primeiro elemento é o peso de `v1` e o segundo elemento é o peso de `v2`.

Exercício 12

Modifique a função do exercício anterior de modo a que agora possa ser calculada a distância de Hamming entre o vector `v1` e um vector de vectores (matriz) `v2`. Sugere-se o seguinte protótipo::

```
dist=HammDist(v1,v2)
```

onde `dist` é agora um vector com a dimensão igual ao número de vectores em `v2`.

Exercício 13

Considere o código da seguinte tabela:

c	mensagem	paridade	**c**	mensagem	paridade
a	000	000	e	100	101
b	001	011	f	101	110
c	010	110	g	110	011
d	011	101	h	111	000

Desenvolva uma função para Matlab capaz de confirmar que a alteração de apenas um bit no código de bloco pode ser detectado e corrigido pelo critério da mínima distância de Hamming.

Bibliografia

[1] J. I. Hall. Notes on coding theory. Department of Mathematics, Michigan State University.

[2] Olivier C. Ibe. *Fundamentals of Applied Probability and Random Processes.* Elsevier Academic Press, 2005.

[3] Yuan Jiang. *A practical guide to error-control coding using Matlab.* Artech House, 2010.

[4] Frank Kschischang. Ece 1501 - error control codes. University of Toronto.

[5] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[6] C. Shanon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379423, 623656, 1948.

[7] Peter Symonds. *Math32031 - Coding Theory.* Manchester University, 2009.

[8] Francisco Vaz. *Probabilidades e Processos Estocásticos para Engenharia Electrotécnica.* 2002.