# Teoria da Informação

## *Entropia, Redundância e Informação Mútua*

Leonardo Vieira – 2015236155

Tiago Gomes – 2015238615

Artur Coutinho – 2014230432

## Exercício 1.

No primeiro exercício é-nos pedido que, dada uma fonte de informação *P* e um alfabeto *A*, se determine e visualize o histograma de ocorrências dos seus símbolos. Para isso implementámos a rotina, *Histograma(p,alfa,titulo)* que recebe a fonte *p*, o alfabeto *alfa* e a string *titulo* com o nome do histograma a desenhar. Transformámos a matriz *p* num array *parray* usando mecanismos simples do *MATLAB*, para depois a usar na rotina *histc()* que devolve um array com o número de ocorrências de cada elemento de *parray* que pertença a *alfa*. Este vetor é usado para valor de retorno e na rotina *bar()* , onde o histograma vai ser desenhado. Por fim, nomeámos o histograma e os respetivos eixos.

**Código:**

```
function [dados] = Histograma(p, alfa, titulo)
    parray = p(:); %transforma matriz "p" num array "parray"
    dados = histc(parray,alfa);
    bar(dados,0.75);
    ylabel('Ocorrências');
    xlabel('Alfabeto');
    title(titulo);
end
```

## Exercício 2.

No segundo exercício, é-nos pedido que seja determinado o limite mínimo teórico para o número médio de bits por símbolo, ou seja, a entropia. A fórmula usada é a seguinte:

$$H(A) = -\sum_{k=0}^{n} P(\alpha_k) \cdot \log_2 P(\alpha_k)$$

Na rotina criada, *Entropia(dados)*, em que dados é o vetor determinado no exercício anterior e determinámos o vetor probabilidade, que contém a probabilidade de cada um dos elementos do alfabeto, usando a soma de todos os valores com a função *sum()*. São depois retirados os valores iguais a zero visto que o logaritmo de zero não é real, e por fim utilizámos a fórmula.

**Código:**

```
function entropia = Entropia(dados)
    soma=sum(dados);
    dados = dados(dados~=0);
    probabilidade = dados/soma;
    %multiplica ponto a ponto
    entropia = -sum( probabilidade .* log2(probabilidade));
end
```

# Exercício 3.

É-nos pedido para usar as rotinas desenvolvidas nos exercício 1 e 2, para determinar a distribuição estatística (histograma) e o número médio de bits por símbolo (entropia) para as seguintes fontes fornecidas: *Lena.bmp, CT1.bmp, Binaria.bmp, saxriff.wav* e *Texto.txt.*

Tal como aconselhado, utilizámos a rotina *imread()* para ler a informação dos ficheiro .bpm, *fopen()* e *fscan()* para .txt e *audioread()* para .wav. Como diferente ficheiro necessitam de diferentes alfabetos, também foi usada em alguns casos a rotina *unique()*, que devolve um array com todos os dados da fonte, mas sem repetições. Nos restantes casos (Textto.txt e saxriff.wav) foram implementadas manualmente as restrições impostas no enunciado.

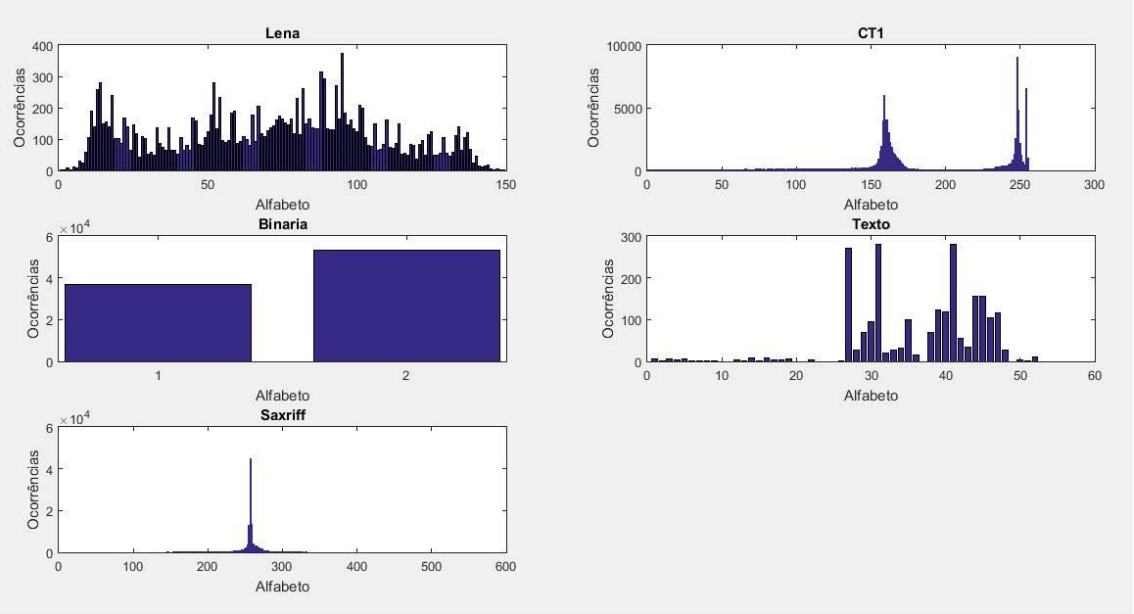


*Figura 1 - Histogramas do exercício 3*

## Será possível comprimir cada uma das fontes de forma não destrutiva?

Sim, é, sendo a compressão máxima (em que não haja perda de dados e se consiga extrair toda a informação original) o valor da entropia pois, por definição, esta é o limite mínimo teórico do número médio de bits por símbolo.

Em todos os casos a entropia máxima é 8, visto que $\log_2 256 = 8$, sendo 256 o tamanho do alfabeto. Para o cálculo da Taxa de compressão usou-se a fórmula:

$$Taxa(x) = \frac{EntropiaMax - Entropia}{EntropiaMax} \times 100$$

***Lena.bmp***

Entropia = 6.9153 Taxa de compressão: 13.56%

***CT1.bmp***

Entropia = 5.9722 Taxa de compressão: 25.35%

***Binaria.bmp***

Entropia = 0.9722 Taxa de compressão: 87.84%

***Texto.txt***

Entropia = 4.1969 Taxa de compressão: 47.54%

***Saxriff.wav***

Entropia = 3.5356 Taxa de compressão: 55.81%

A diferença de valores de entropia entre imagens faz sentido, pois na imagem *Lena.bmp* existe uma maior variação dos níveis de cinzento do que na imagem *CT1.bmp*, que por sua vez tem uma maior variação dos níveis de cinzento do que a imagem *Binaria.bmp*, pois esta última varia apenas entre 2 valores de cor (imagem binária).

## Exercício 4.

No exercício 4 utilizámos a rotina *hufflen()* dada, para calcular o número médio de bits necessários por cada símbolo da mesma maneira que no exercício 1.

Utilizámos a seguinte fórmula:

$$l = \sum_{k=0}^{n} P(\alpha_k) \cdot huffle n(x)$$

Sendo *x* o vetor com a informação que foi retornada por *Histograma().*

**Código:**

```
function H_Entropia = EntropiaHuffman(dados)
    soma=sum(dados);
    dados = dados(dados~=0);
    probabilidade = dados/soma;

    H_Entropia=sum(probabilidade .* hufflen(dados));
end
```

***Lena.bmp***

Entropia de Huffman = 6.9425

***CT1.bmp***

Entropia de Huffman = 6.0075

***Binaria.bmp***

Entropia de Huffman = 1

***Texto.txt***

Entropia de Huffman = 4.2173

***Saxriff.wav***

Entropia de Huffman = 3.5899

Os valores obtidos são muito próximos aos obtidos no exercício 3, tal como era previsto pela condição:

$$H(S) \leq l \leq H(S) + 1$$

## Será possível reduzir-se a variância?

A variância aumenta com o número de símbolos na fonte que faz com que o número de bits que é preciso para codificar um símbolo aumente, levando por isso ao aumento da variância. Para diminuir a variância, devemos ter em conta a probabilidade dos elementos quando os combinamos, de modo a que as combinações de elementos com menos probabilidade ganhem prioridade.

## Exercício 5.

No exercício 5 repetimos a estrutura do exercício 3, mas usando sequências de dois símbolos contíguos. No alfabeto usamos todas as combinações possíveis, enquanto na fonte apena os pares de símbolos contíguos. Para tal, criámos duas rotinas: *HistogramaAgrup()* e *EntropiaAgrup().* Na primeira rotina usamos algumas funções já definidas pelo *MATLAB*, como a rotina *vec2mat()* e a rotina *meshgrid(alfa)* assim como métodos de junção de matrizes para formar uma nova matiz também com duas colunas com todas as combinações possíveis dos valores presentes em alfa.
De seguida, dentro de um ciclo *for,* utilizámos a rotina *bsxfun()* que executa operações elemento a elemento, e quando usado o parâmetro *@eq*, cria uma nova matriz onde coloca a 1 a posição onde encontra elementos de *new_p* iguais aos de *par*. Depois usando *all(x, dim)* com dim=2 (linha a linha) e *sum(),* temos o numero de vezes que *par* ocorre em new_p.
Usando essa informação desenhámos o histograma e calculámos a entropia.
A rotina *EntropiaAgrup()* é igual à rotina original para calcular a entropia, com a diferença de que no final, como se quer o limite mínimo do número de bits por símbolo e nao para um par de símbolos, dividimos por 2.

**Código:**

```
function aux = HistogramaAgrup(p, alfa, titulo)
    %transforma matriz "p" num array "parray"
    parray = reshape(p',1,[]);
    new_p = vec2mat(parray, 2);
    [comb1, comb2]=meshgrid(alfa);
    AlfaComb=[comb1(:) comb2(:)];
    size_a=size(AlfaComb);
    aux=zeros(1, size_a(1));

    for i=1:size_a(1)
        par = AlfaComb(i, :);
        aux(i) = sum(all(bsxfun(@eq, par, new_p),2));
        %bsxfun -> element wise binary operations (neste caso @eq
[equals])
        %all dim = 2 -> aplica && em cada linha
    end

    bar(aux)
    ylabel('Ocorrências');
    xlabel('Alfabeto');
    title(titulo);
end



function Entropia_A = EntropiaAgrup(dados)
    soma=sum(dados);
    dados = dados(dados~=0);
    probabilidade = dados/soma;
    Entropia_A = -sum(probabilidade.*log2(probabilidade))/2;
end
```

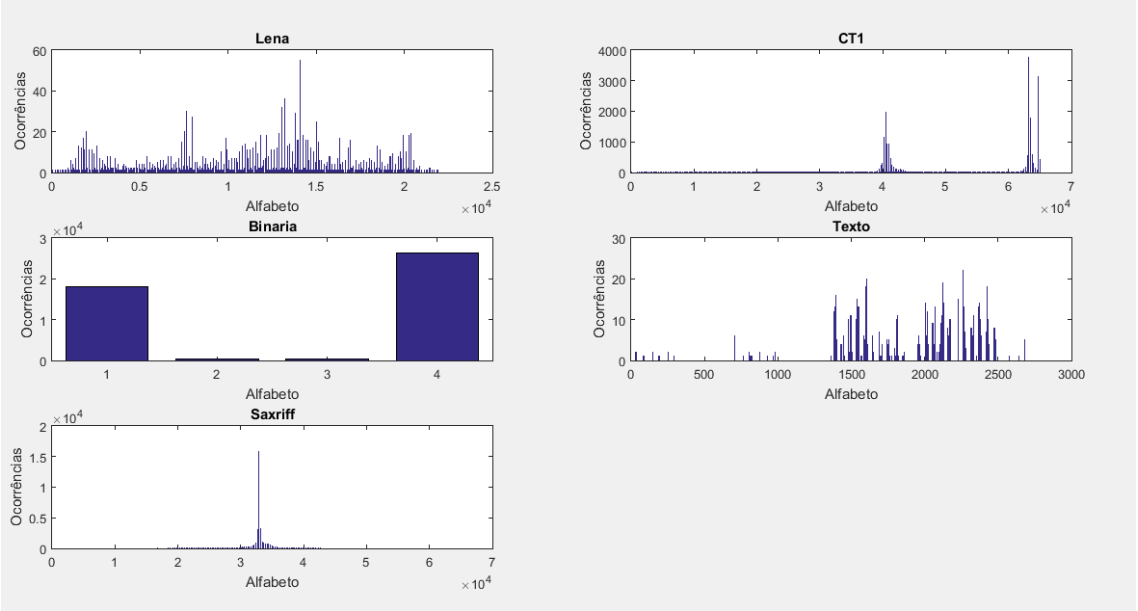


*Figura 2 - Histogramas do exercício 5*

***Lena.bmp***

Entropia Agrupada = 5.5965

***CT1.bmp***

Entropia Agrupada = 4.4813

***Binaria.bmp***

Entropia Agrupada = 0.5424

***Texto.txt***

Entropia Agrupada = 3.6576

***Saxriff.wav***

Entropia Agrupada = 2.4706

Apesar de requerer uma maior complexidade algorítmica e tornar o programa mais lento, o agrupamento de símbolos permite neste caso um valor mais baixo da entropia relativamente ao exercício 3. Logo, neste caso, podemos considerar mais vantajoso o agrupamento de símbolos.

## Exercício 6A.

Neste exercício é-nos pedido para desenvolvermos uma rotina que, dada uma query, uma janela e um alfabeto, devolva um vetor com contem a informação mútua entra a query e cada janela. Para tal usámos a seguinte formula:

$$I(x,y) = H(x) + H(y) - H(x,y)$$

Criámos também a rotina *EntropiaConj(alfa,query,janela)* e usámos a rotina *Entropia(dados)* previamente criada. Usamos a variável *plus* para evitar índices negativos na matriz *"dados"*, que vai conter as coordenadas dos valores iguais entre query e janela, sendo depois estes todos somados para uso no cálculo da entropia conjunto.

Na rotina *Infmut()* definimos a variável *s* que terá o número que podemos avançar a query, e a variável *aux* que vai conter a janela a enviar para a *EntropiaConj(alfa,query,janela).*

**Código:**

```
function dados = InfMut(query, target, alfa, step)
    s = length(target)-length(query)+1;
    dados = zeros(1,s);
    dadosq = histc(query, alfa);

    for i=1:step:s;
        janela=target(i:i+length(query)-1); %vai avançando a query
        dadosj=histc(janela, alfa);
        res = Entropia(dadosq) + Entropia(dadosj) -
EntropiaConj(alfa,query,janela);
        dados(i) = res;
    end
end



function entropia_conj = EntropiaConj(alfa,query,janela)
    size_c = size(query,1);
    plus= - min(min(query), min(janela)) +1; %evitar indices negativos
em dados
    dados=zeros(2*length(alfa));

    for i=1:size_c
        dados(query(i,1)+plus,janela(i,1)+plus) =
dados(query(i,1)+plus,janela(i,1)+plus)+1;
    end

    soma=sum(sum(dados));
    dados = dados(dados~=0);
    probabilidade = dados/soma;
    entropia_conj = -sum( probabilidade .* log2(probabilidade));
end
```

## Exercício 6.B.

Na alínea B. usámos a função *InfMut(query, target, alfa, step)* para calcular a variação da informação mútua entre os ficheiros *"saxriff.wav"*, *"target01 - repeat.wav"* e *"target02 - repeatNoise.wav"*.
Como a query se repete 3 vezes em cada target, é normal que no gráfico esteja representado 3 picos de informação mútua. Como o step é ¼ do tamanho da query, entre cada pico são visualizadas mais 3 medidas. É também visível que a linha laranja tem os picos mais pequenos, isso devido à quantidade de ruido ser maior no *"target02 - repeatNoise.wav"*.

**Resultados:**

*"target01 -repeat.wav"* – linha azul:
2.8027 0.0618 0.0793 0.0612 1.1882 0.0608 0.0781 0.0611 0.8573

*"target01 - repeatNoise.wav"* – linha laranja:
0.2647 0.0716 0.0986 0.0606 0.4727 0.0621 0.0673 0.0790 0.1576

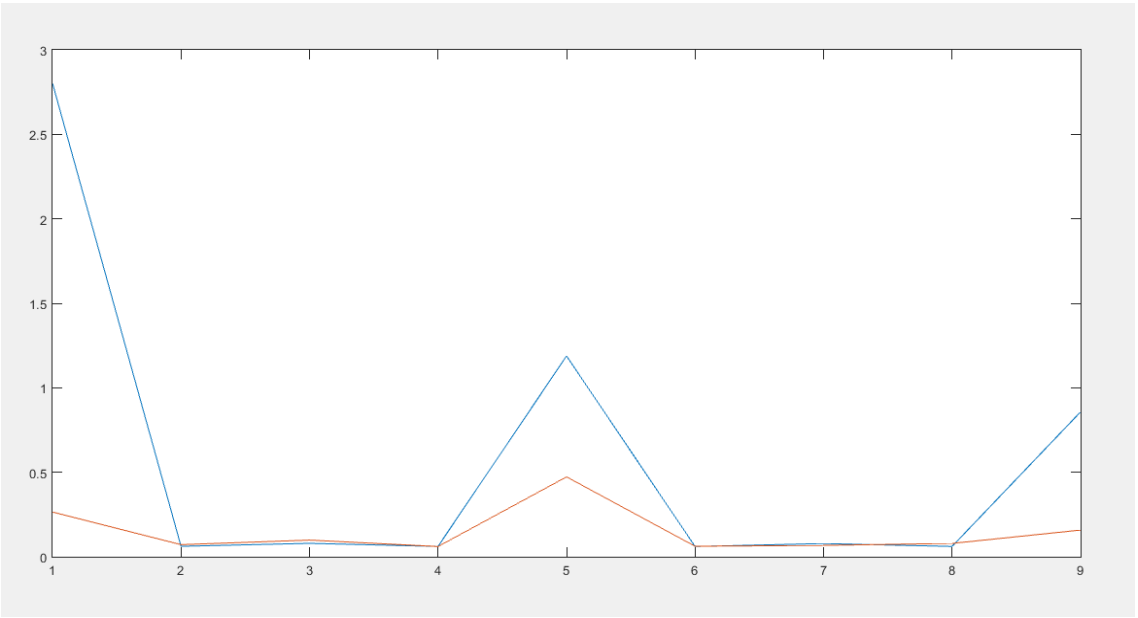


*Figura 3 - Gráfico do exercício 6.B.*

## Exercício 6.C

Assim como na alínea anterior, utilizámos rotinas desenvolvidas na alínea a) para calcular a informação máxima entre a query e Song*.wav como target. Para ordenar de forma decresceste os máximos, usámos a função *sort()* com o argumento *'descend'*. De seguida estão os resultados ordenados, assim como os gráficos que representam a evolução das informações mútuas entre a query e as 7 amostras:

**Song 7** -> 3.530989 bits

**Song 6** -> 3.530989 bits

**Song 5** -> 0.532292 bits

**Song 4** -> 0.191816 bits

**Song 2** -> 0.168576 bits

**Song 3** -> 0.190066 bits

**Song 1** -> 0.137107 bits

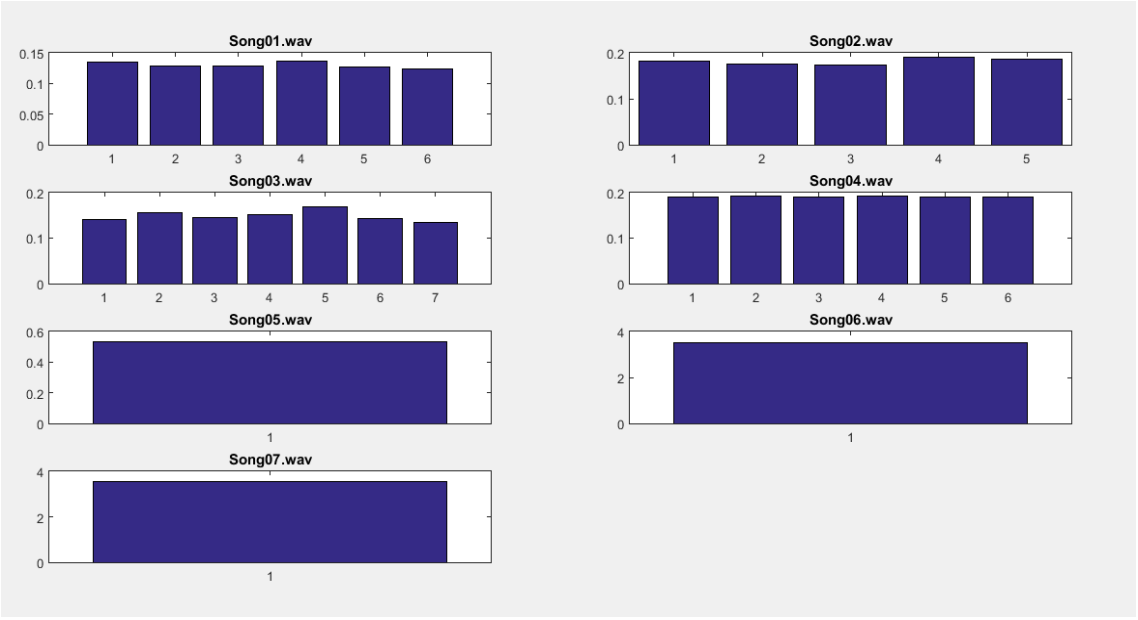


*Figura 4 - Histogramas do exercício 6.C.*