

# Entropia, Redundância e Informação Mútua

---

Trabalho prático nº1

## Introdução

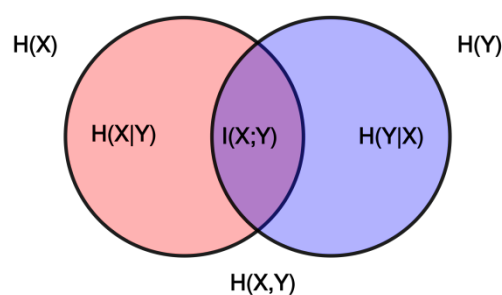
Neste trabalho pretendemos adquirir sensibilidade para as questões fundamentais de teoria de informação, mais concretamente, informação, redundância, entropia e informação mútua.

O conceito de informação e o conceito de incerteza estão interligados. Uma mensagem tem uma determinada quantidade de informação quanto maior for o seu grau de incerteza ou de imprevisibilidade, isto é quanto maior for a abundância de palavras (redundância), maior a "desordem". A esta "desordem" chamamos Entropia, que corresponde ao número médio de bits para codificar uma fonte de informação e pode ser calculada a partir da seguinte fórmula:

$$H(A) = \sum_{i=1}^n P(a_i) i(a_i) = - \sum_{i=1}^n P(a_i) \log_2 P(a_i)$$

Interessa-nos também entender o conceito de informação mútua, que é uma medida de quantidade de informação que uma variável aleatória contém acerca da outra, ou seja, da quantidade de informação partilhada entre uma e outra variáveis. A informação mútua pode ser obtida recorrendo à seguinte expressão:

$$I(X, Y) = I(Y, X) = H(Y) - H(Y|X) = H(X) - H(X|Y) = H(X) + H(Y) - H(X, Y)$$



Definidos os conceitos fundamentais à realização do trabalho, procedemos então à resolução dos exercícios propostos na ficha prática nº1.

## Main

Conforme pedido no enunciado foi criado um script principal que permite executar todos os exercícios pedidos. O programa pede sempre um exercício e o utilizador deve indicá-lo. Os resultados são indicados posteriormente e o programa volta a pedir ao utilizador um novo exercício. Caso o utilizador queira sair do programa apenas tem de escrever “exit” e o programa termina.

## Exercício 1

No primeiro exercício é pedido para calcular o número de ocorrências de cada símbolo, dado um alfabeto e uma fonte. Para tal, criei uma função *createHistogram(P,A)*, presente no script *createHistogram.m*. Esta função recebe como argumentos uma fonte e um alfabeto e devolve o histograma de ocorrência de símbolos. Esta função recorre a uma outra função (*hist(P,A)*) presente no script *hist.m* em que conta o número de ocorrências em P dos símbolos do alfabeto A.

Código:

```
function count = hist(P, A)
    s = size(A);
    if (s(1) == 1)
        A = transpose(A);
        s = size(A);
    end
    count = zeros(1, s(1));
    alf = intersect(unique(P), A);
    h = size(alf);
    h = h(1);

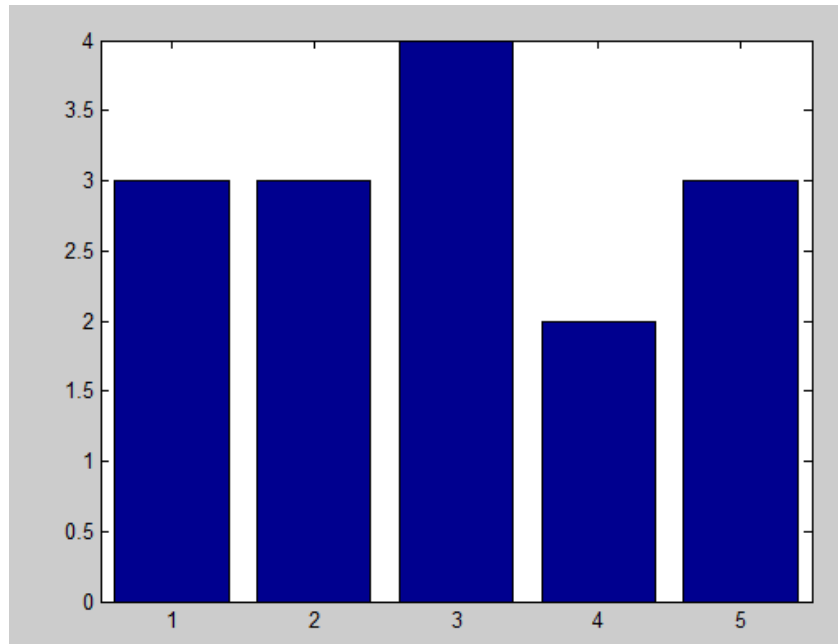
    for i=1:h
        x = alf(i, :);

        if (s(2) > 1)
            count(i) = sum(all(bsxfun(@eq, x, P), 2));
        else
            count(i) = length(find(P == x));
        end
    end

end
```

Primeiro começamos com a obtenção do tamanho do alfabeto  $A$  para saber quantos símbolos precisamos de encontrar. A seguir criamos uma matriz com o tamanho de  $A$ . Com as duas linhas a seguir pretendemos saber quantos elementos distintos existem e quais são. Usando esse número criamos um ciclo for para procurar cada elemento em  $P$  da seguinte maneira: a cada iteração do ciclo a função *find* encontra o  $x$  que queremos e devolve as posições onde ele se encontra em  $P$ . Para sabermos quantos existem em  $P$  recorremos à função *length* e guardamos esse valor em *count*. Voltando à função *createHistogram(P,A)*, ela recebe a matriz com os resultados do algoritmo anterior e faz a representação gráfica da matriz recorrendo à função *bar*.

Para testar o código, inserimos um vetor de valores aleatórios de 1 a 5, por exemplo: : [1 2 3 4 5 1 2 3 5 1 3 5 2 3 4], bem como o alfabeto [1 2 3 4 5], obtendo [3 3 4 2 3] e o seguinte histograma de ocorrências:



Histograma de ocorrências

## Exercício 2

Pretende-se, neste exercício, determinar o limite mínimo teórico para o número médio de bits por símbolo de uma fonte de informação fornecida como parâmetro. Como tal, desenvolvi uma função *entropia(P,A)* que se encontra no script *entropia.m* e que recebe, tal como no exercício anterior, um alfabeto e uma fonte de informação. Para a resolução do exercício foi aplicada a formula da Entropia enunciada anteriormente:

$$H(A) = \sum_{i=1}^n P(a_i) i(a_i) = - \sum_{i=1}^n P(a_i) \log_2 P(a_i)$$

onde n corresponde à cardinalidade do alfabeto e P(a) à probabilidade de ocorrência de cada símbolo.

**Código:**

```
function ent = entropia(P, A)

    s = size(A);

    if (s(1) == 1)
        A = transpose(A);
        s = size(A);
    end

    ent = 0;
    total = length(P);

    alf = intersect(unique(P), A);
    h = size(alf);
    h = h(1);

    for i=1:h
        x = alf(i, :);

        if (s(2) > 1)
            f = sum(all(bsxfun(@eq, x, P),2));
        else
            f = length(find(P == x));
        end

        if f > 0
            prob = f / total;
            ent = ent + (prob * log2(prob));
        end
    end

    ent = -ent;

end
```

Para testar o código criado, voltámos a inserir os valores do exercício anterior como argumentos. Sendo assim, a nossa fonte, *P*, corresponde a [1 2 3 4 5 1 2 3 5 1 3 5 2 3 4] e o nosso alfabeto, *A*, ao vetor [1 2 3 4 5], sendo que o output apresentado na janela foi o valor 2.2892 correspondente à Entropia. Como os valores inseridos são bastante simples, podemos facilmente verificar, a partir da fórmula que nos permite calcular a Entropia, que este valor está efetivamente correto.

### Exercício 3

Neste exercício, o objetivo é elaborar um programa que determine a entropia dos ficheiros fornecidos pelo professor (*Lena.bmp*, *CT1.bmp*, *Binaria.bmp*, *saxriff.wav*, *Texto.txt*), aproveitando o código criado para os Exercícios 1 e 2.

Precisamos por definir primeiramente a fonte e o alfabeto para cada tipo de ficheiro (imagem, áudio, texto). No caso das imagens, a fonte é obtida através da função *imread()* que retorna uma matriz com as intensidades de cada pixel da imagem. Já o alfabeto, sabemos que cada pixel é codificado por valores entre 0 e 255, sendo o parâmetro a inserir na posição deste argumento, 0:255. No caso da imagem *Binaria.bmp* usamos como alfabeto [0 255] visto que são as únicas cores presentes na imagem. Relativamente aos ficheiros de áudio, utilizamos a função *wavread()* que devolve três valores: amostras de áudio, frequência de amostragem e número de bits por amostra. A nível do alfabeto, o próprio enunciado referencia como devemos defini-lo (  $-1:d ; 1-d$ , onde  $d = \frac{1-(-1)}{2^{n^{\circ} \text{ de bits}}}$  ). Por fim, em relação ao ficheiro de texto, a fonte é obtida a partir das funções *fopen()* e *fscanf()* que devolvem o conteúdo do ficheiro sem espaços. Como alfabeto, assumimos todas as letras do alfabeto, de A a Z, tendo em conta letras maiúsculas e minúsculas.

### Código:

```
%lena.bmp
imagem = imread('dados/Lena.bmp');
imagem = imagem(:);
createHistogram(imagem, 0:255);
title('Lena.bmp');
disp(entropia(imagem, 0:255));
pause;

%ct1.bmp
imagem = imread('dados/CT1.bmp');
imagem = imagem(:);
createHistogram(imagem, 0:255);
title('CT1.bmp');
disp(entropia(imagem, 0:255));
pause;

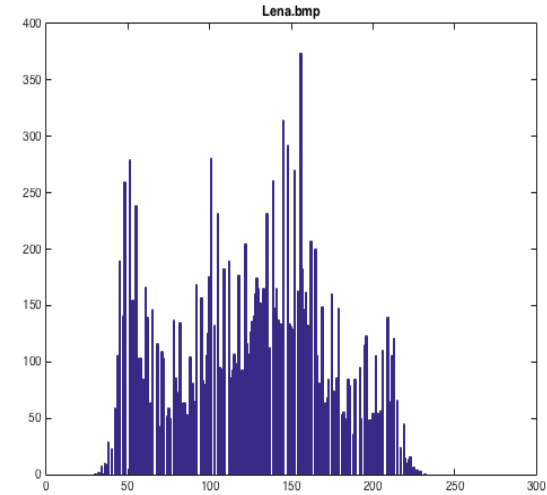
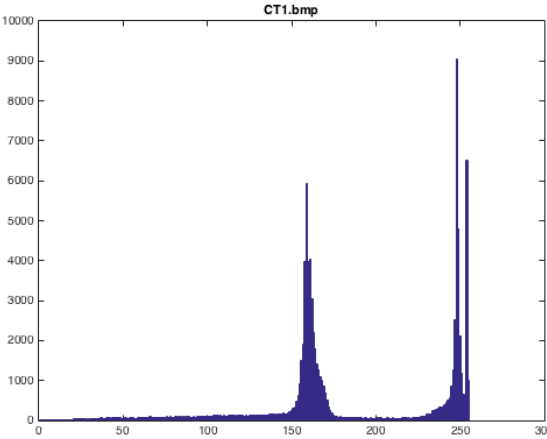
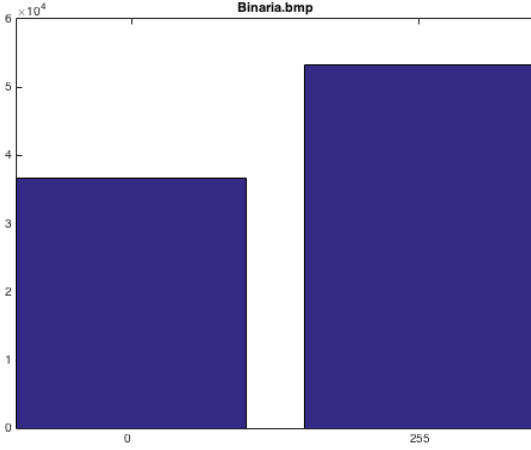
%binaria.bmp
imagem = imread('dados/Binaria.bmp');
imagem = imagem(:);
createHist(imagem, [0 255]);
title('Binaria.bmp');
disp(entropia(imagem, [0 255]));
pause;

%saxriff.wav
wav = audioread('dados/saxriff.wav');
wav = wav(:);
quant= 8; % este valor pode ser alterado
d = 1 / (2^quant);
alfabeto = -1:d:1;
createHistogram(wav, alfabeto);
axis([0 100 0 20000]);
title('saxriff.wav');
disp(entropia(wav,alfabeto));
pause;

%Texto.txt
file = fopen('dados/Texto.txt','rt');
text = fscanf(file, '%s');
fclose(file);
text = text(:);
Alphabet = ['a':'z' 'A':'Z'];
createHist(text, Alphabet);
title('texto.txt');
disp(entropia(text, Alphabet));
```

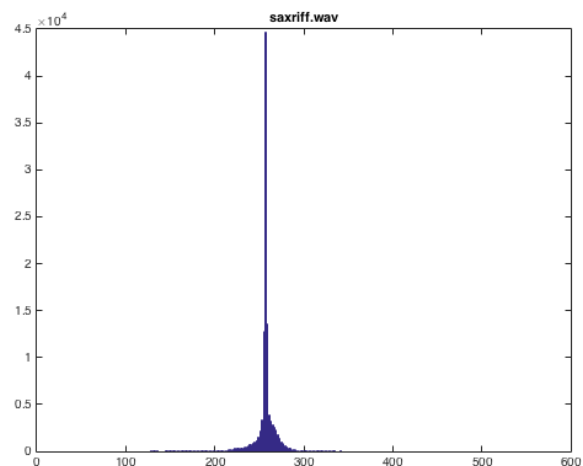
Como é possível observar a partir do código, para a resolução deste exercício utilizamos as funções relativas aos exercícios 1 e 2 que nos permitem obter o histograma de ocorrências de cada símbolo e o valor de entropia de cada ficheiro.

Resultados:

Nome do Ficheiro	Histograma de Ocorrências	Entropia
Lena.bmp		6,9153
CT1.bmp		5,9722
Binaria.bmp		0,9755

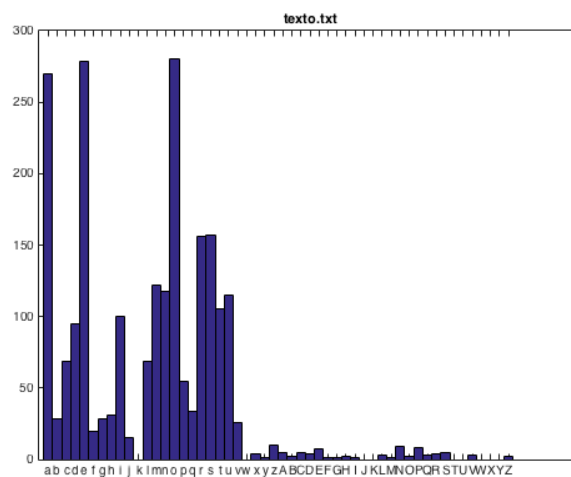


saxriff.wav



3,5356

Texto.txt



3,9547

### Análise dos resultados:

Observando atentamente os gráficos, é fácil de retirar conclusões. Em primeiro lugar os resultados das imagens. Todas elas encontram-se a preto e branco. Isto quer dizer que a matriz que é retornada representa a intensidade de cada pixel, é uma matriz  $n \times m$ . Em imagens coloridas a função *imread* retorna a intensidade de cada pixel em RGB. Logo, para imagens “cinzentas” (*grayscale images*)  $R = G = B$ , o que significa que cada pixel tem a mesma intensidade de vermelho, verde e azul (RGB). Logo, os gráficos vão apresentar diversas variações de cinzento. Estas afirmações são confirmadas pelos histogramas de ocorrências que foram obtidos.

Na imagem "**Lena.bmp**" verifica-se uma grande variedade de tons de cinzento, razão pela qual se verifica uma variedade tão grande de elementos. Por outro lado, a imagem "**CT1.bmp**" possui menos variedade de cinzentos e maior afluência de tons de preto demonstrada pelo elevado número de ocorrências referentes aquela cor. Já em relação à imagem "**Binaria.bmp**", que é uma imagem binarizada, sabemos que existem apenas duas cores: preto e branco. Como tal, só existem dois símbolos que estão perfeitamente visíveis nas barras do gráfico correspondente. Quanto ao ficheiro de áudio, "**saxriff.wav**" é um som bastante simples com apenas 3 notas tocadas, que são visíveis no histograma de ocorrências como sendo os valores mais altos. Por fim, quanto ao ficheiro de texto "**Texto.txt**", como sabemos, existem diversos caracteres que aparecem com uma determinada frequência, uns com maior frequência que outros.

Quanto à pergunta feita no enunciado sobre a compressão de fontes de forma não-destrutiva. Essa compressão é possível. A máxima compressão possível é dada pela entropia calculada, pois é o número médio de bits por símbolo.

## Exercício 4

Neste exercício, pretende-se obter o número de bits necessários à representação de cada símbolo dos alfabetos dos 4 ficheiros usando as rotinas de codificação de Huffman. Para esse efeito recorri ao código fornecido pelo professor e código usado nos exercícios anteriores. A matriz resultante da ocorrência de símbolos realizada no exercício 1 é aplicada na função *hufflen()* que retorna um vetor com o numero de bits em que estão representados o numero de bits necessários à codificação de cada símbolo.

## Código:

```
%Lena.bmp
imagem = imread('dados/Lena.bmp');
imagem = imagem(:);
h = hist(imagem, 0:255);
huf = huffman(h);
disp(entropyHuffman(h, huf));

%CT1.bmp
imagem = imread('dados/CT1.bmp');
imagem = imagem(:);
h = hist(imagem, 0:255);
huf = huffman(h);
disp(entropyHuffman(h, huf));

%binaria.bmp
imagem = imread('dados/Binaria.bmp');
imagem = imagem(:);
h = hist(imagem, [0 255]);
huf = huffman(h);
disp(entropyHuffman(h, huf));

%saxriff.wav
wav = audioread('dados/saxriff.wav');
wav = wav(:);
quant= 8; % este valor pode ser alterado
d = 1 / (2^quant);
alfabeto = -1:d:1;
h = hist(wav, alfabeto);
huf = huffman(h);
disp(entropyHuffman(h, huf));

%Texto.txt
file = fopen('dados/Texto.txt');
text = fscanf(file, '%s');
fclose(file);
text = text(:);
Alphabet = ['a':'z' 'A':'Z'];
h = hist(text, Alphabet);
huf = huffman(h);
disp(entropyHuffman(h, huf));
```

Este código também usa uma nova fórmula de calcular a entropia de acordo com a codificação de Huffman que consiste em dividir cada elemento do vetor pela soma de todos os elementos, multiplicar por cada elemento do vetor retornado pela função *huffman()* e finalmente somar todos os elementos do vetor em causa. Fonte de informação e alfabeto para cada ficheiro foram obtidos da mesma maneira do que nos exercícios anteriores.

## Resultados:

Nome do Ficheiro	Nº Médio de Bits
Lena.bmp	6,9425
CT1.bmp	6,0075
Binaria.bmp	1
saxriff.wav	3,5899
Texto.txt	4,2173

## Análise dos resultados:

Obteve-se no geral valores relativamente próximos de entropia, como seria de esperar. Tal não se nota no Binaria.bmp, pois não há outra possibilidade senão um bit para o branco e outro para o preto. Há uma grande diferença no comprimento dos códigos encontrados para cada exemplo, o que é natural, pois esperam-se códigos menores para os símbolos mais comuns e maiores para símbolos menos comuns.

A variância pode ser reduzida. Quando for formado um novo conjunto de probabilidades decrescentes houver probabilidades iguais as que resultam de agrupamentos devem ser colocadas o mais alto possível de modo a reduzir a variância. Assim, reduz-se a variância dos comprimentos das palavras de código, mas o comprimento médio mantém-se. **Vantagens:** o ritmo de produção de bits é mais uniforme e há uma maior resistência a erros do canal, na descodificação.

## Exercício 5

Neste exercício é pedido para voltar a calcular as entropias de cada um dos ficheiros, mas desta vez passamos a considerar agrupamentos de símbolos dois a dois. Através dos slides das aulas teóricas sabemos que o novo tamanho dos alfabetos será dado pelo quadrado do tamanho do alfabeto original. Por exemplo, para um alfabeto original  $A = [0\ 1\ 2\ 3]$ , fazendo agrupamento de 2 símbolos obteríamos um  $A' = [00\ 01\ 02\ 03\ 11\ 10\ 12\ 13\ 22\ 20\ 21\ 23\ 33\ 30\ 31\ 32]$ .

A função responsável por calcular as entropias encontra-se no script “*ex5.m*”.

## Código:

```
%Lena.bmp
imagem = imread('dados/Lena.bmp');
imagem = imagem(:);
imagem = vec2mat(imagem, 2);
alf = getpairs(0:255);
createHistogram(imagem, alf);
disp(entropia(imagem, alf) / 2);
axis([0 3000 0 100]);
title('Lena.bmp');
pause;

%CT1.bmp
im = imread('dados/CT1.bmp');
im = im(:);
im = vec2mat(im, 2);
alf = getpairs(0:255);
createHistogram(im, alf);
disp(entropia(im, alf) / 2);
axis([0 8000 0 4000]);
title('CT1.bmp');
pause;

%Binaria.bmp
im = imread('dados/Binaria.bmp');
im = im(:);
im = vec2mat(im, 2);
alf = getpairs([0 255]);
createHistogram(im, alf);
disp(entropia(im, alf) / 2);
title('Binaria.bmp');
pause;

%saxriff.wav
wav = audioread('dados/saxriff.wav');
wav = wav(:);
wav = vec2mat(wav, 2);
quant = 7;
d = 1 / (2^quant);
alf = getpairs(-1:d:1);
createHistogram(wav, alf);
disp(entropia(wav, alf) / 2);
axis([0 2000 0 1000]);
title('saxriff.wav');
pause;

%texto.txt
Alphabet = ['a':'z' 'A':'Z'];
textFile = fopen('dados/Texto.txt');
text = fscanf(textFile, '%s');
fclose(textFile);
text = text(:);
text = vec2mat(text, 2);
alf = getpairs(Alphabet);
createHistogram(text, alf);
disp(entropia(text, alf) / 2);
axis([0 250 0 25]);
title('Texto.txt');
```

Neste exercício usamos duas novas funções: *vec2mat* e *getpairs*. A primeira converte o vetor em matriz com duas colunas e tem como objetivo assegurar o agrupamento de símbolos. A segunda função devolve o alfabeto que é dado pelo quadrado do tamanho do alfabeto original através da função *meshgrid*. A fonte de informação para cada ficheiro é obtida da mesma maneira do que nos exercícios anteriores.

## Resultados:

Fonte de Informação	Entropia
Lena.bmp	5.3689
CT1.bmp	4.5339
Binaria.bmp	0.5382
Saxriff.wav	2.9100
Texto.txt	3.1717

## Análise dos resultados:

Tal como mostram os resultados geralmente o agrupamento de símbolos é vantajoso uma vez que permite baixar o valor da entropia. A desvantagem do agrupamento de símbolos encontra-se no facto dos requisitos de memória serem bastante mais altos sendo o algoritmo mais complexo. Para as imagens é muito frequente que pixéis consecutivos tenham a mesma cor, enquanto que pixéis consecutivos de cor diferente são muito menos comuns. Assim, usando o agrupamento de símbolos, consegue-se reduzir a entropia, pois guarda-se o pixel dois a dois.

## Exercício 6

No exercício 6 vamos recorrer à informação mútua entre duas variáveis de acordo com a seguinte fórmula:

$$I(X, Y) = H(X) + H(Y) - H(X, Y)$$

Para tal foi criada uma função que executa o deslizamento (*slidingWindow.m*) ao longo do target saltando *step by step* chamando, a cada iteração, a função que calcula a informação mútua entre a *query*

dada e a atual janela no target (*mutualInf.m*). Para efeitos de teste, o exemplo fornecido no enunciado foi usado e o resultado está de acordo com a resposta esperada.

## Código:

*SlidingWindow.m*:

```
function [mutualInfVector] = slidingWindow(query, target, alphabet, step)
    s = length(target) - length(query) + 1;

    mutualInfVector = zeros(1, length(1:step:s));
    for i=1:step:s
        mutualInfVector(ceil(i / step)) = mutualInf(query, target(i: i + length(query) - 1), alphabet);
    end

    mutualInfVector = transpose(mutualInfVector);
    %disp(mutualInfVector);
end
```

*mutalInf.m*:

```
function [inf] = mutalInf(X, Y, alf)
    % Calcula a informacao mutua entre X e Y.
    offset = - min(min(X), min(Y)) + 1;
    matrix = zeros(length(alf));

    for i=1:length(X)
        matrix(X(i) + offset, Y(i) + offset) = matrix(X(i) + offset, Y(i) + offset) + 1;
    end

    probConj = matrix ./ sum(sum(matrix));
    x = sum(matrix) ./ sum(sum(matrix));
    y = sum(matrix') ./ sum(sum(matrix));

    inf = calcEnt(x) + calcEnt(y) - calcEnt(probConj);
end

function [h] = calcEnt(x)
    x = nonzeros(x);
    h = -sum(x .* log2(x));
end
```

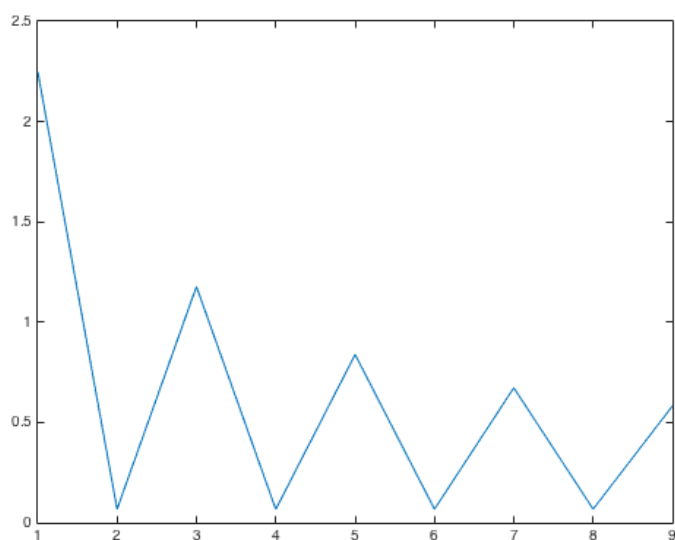
*ex6mminf.m*

```
function [mmi, values] = ex6mminf(file)
    % Calcula a informacao mutua maxima entre o ficheiro 'file' e o ficheiro
    % saxriff.wav

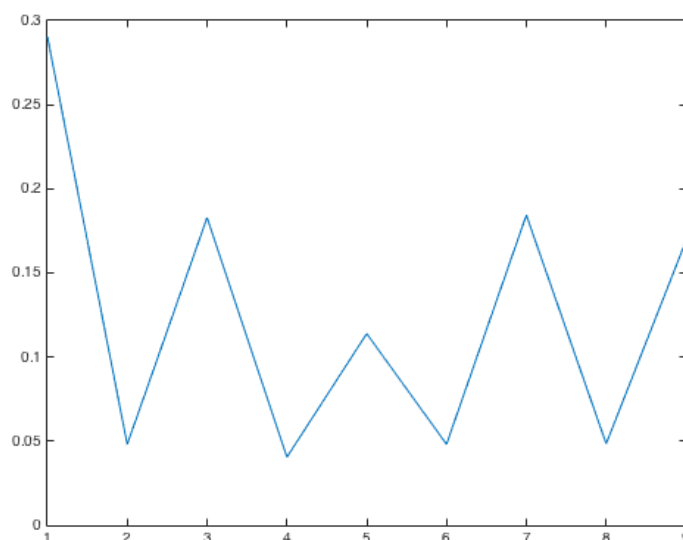
    query = audioread('dados/saxriff.wav');
    query = query(:);
    quant = 7;
    d = 1 / (2^quant);
    alf = -1:d:1;
    query = query*2^quant;

    wav = audioread(file);
    wav = wav(:);
    wav = wav*2^quant;
    r = slidingWindow(query, wav, alf, floor(0.25 * length(query)));
    values = r;
end
```

Já no **exercício b)** a função acima descrita é usada para estudar as semelhanças entre dois ficheiros áudio semelhantes e o ficheiro saxriff.wav. Recuperando o resultado do exercício 3, o ficheiro saxriff.wav tem uma entropia de 3.5356. No target 1 atingimos uma entropia máxima de 2.2489, ou seja, é neste ponto que o áudio é bastante semelhante. Já para o target 2 como há muito ruído a entropia será extremamente baixa pelo que o target 1 será o som mais próximo do som original.



Target01 – repeat.wav



Target02 – repeatNoise.wav

### Tabela de resultados

Target01	Target02
2.2489	0.2905
0.0680	0.0481
1.1771	0.1828
0.0670	0.0405
0.8388	0.1139
0.0676	0.0482
0.6727	0.1844
0.0679	0.0485
0.5824	0.1691



Relativamente ao **exercício c)** a mesma função (ex6mminf.m) foi aplicada a cada uma destas fontes e os resultados apresentados. A função *sortrows* foi usada para ordenar os resultados de acordo com as exigências do enunciado.

*Ex6c.m:*

```
r = zeros(7, 2);

for i=1:7
    [max_r, values] = ex6mminf(sprintf('dados/Song0%d.wav', i));
    r(i, 1) = max_r;
    r(i, 2) = i;
end

sortrows(r, -1);

for i=1:7
    fprintf('Song0%d.wav -> %f\n', r(i, 2), r(i, 1));
end
```

**Tabela de resultados:**

Song01.wav	0.077952
Song02.wav	0.114060
Song03.wav	0.092564
Song04.wav	0.112439
Song05.wav	0.519376
Song06.wav	3.535599
Song07.wav	3.535599

Conclui-se assim que os ficheiros Song06.wav e Song07.wav são os que têm um trecho mais semelhante com o ficheiro saxriff.wav, usando o critério de informação mútua.