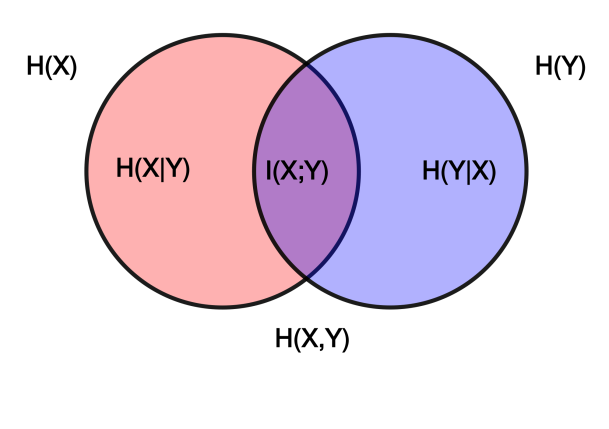
|  |
| --- |
| Faculdade de ciências e tecnologia da universidade de coimbra  2016/2017  **Teoria da informação** |
| Entropia, Redundância e Informação Mútua |
| Trabalho prático nº1 |
|  |
| **Nuno Ferreira 2013153319** |

# 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:

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:

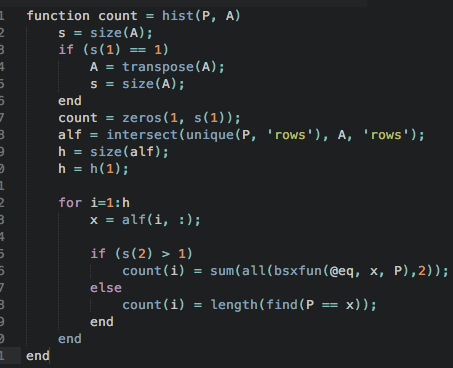


Definidos os conceitos fundamentais à realização do trabalho, procedemos então à resolução dos exercícios propostos na ficha prática nº1.

# 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.

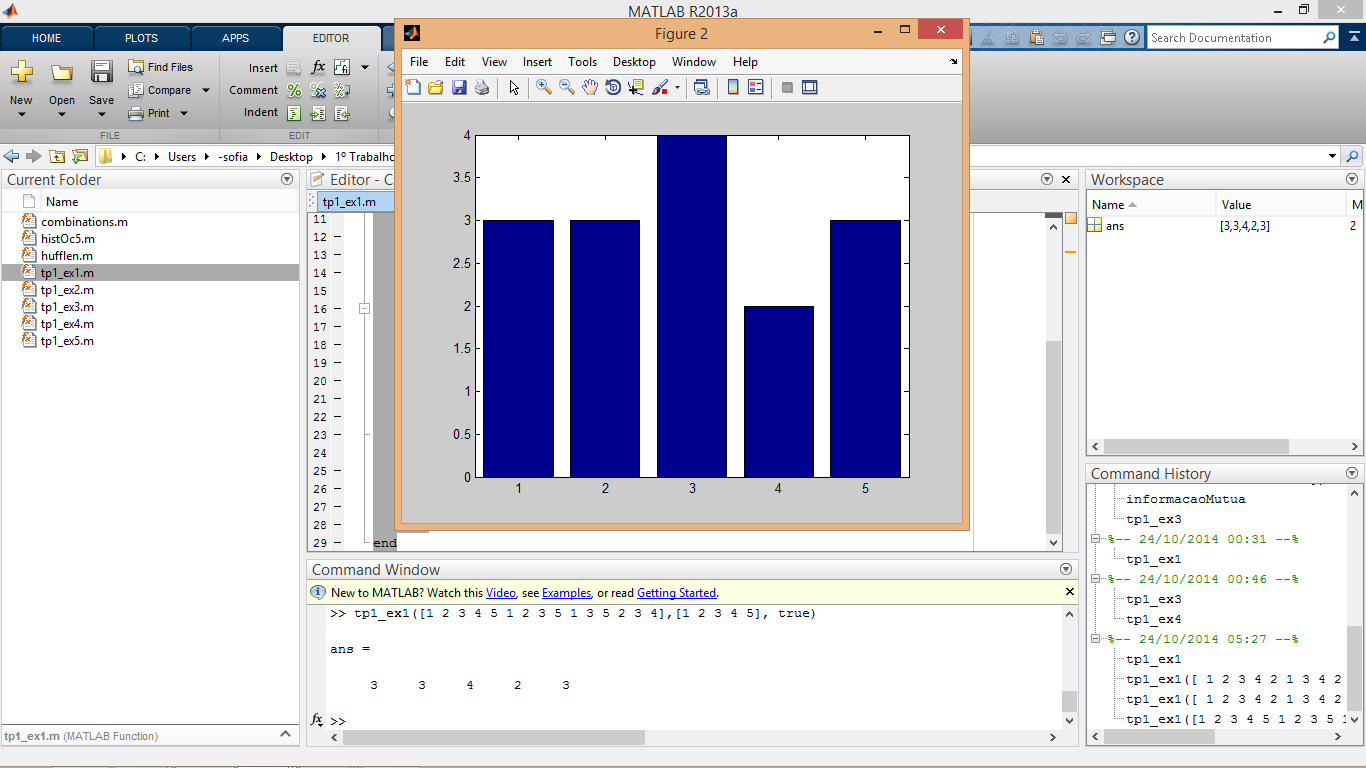
Código:



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 sabemos 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*.

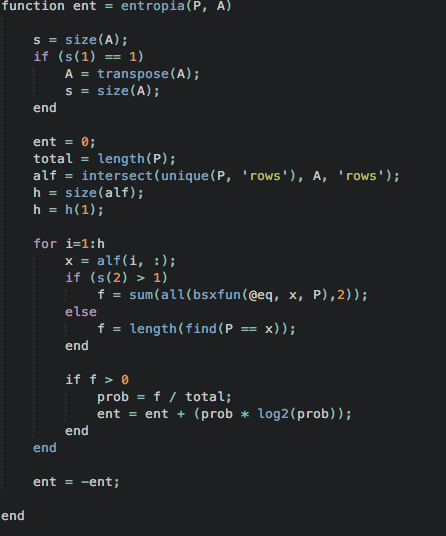
Para testar o código, inserimos um vetor de valores aleatórios de 1 a 5, por exemplo: : [1 2 3 4 5 1 2 3 5 1 3 5 2 3 4], bem como o alfabeto [1 2 3 4 5], obtendo **[3 3 4 2 3]** e o seguinte histograma de ocorrências:



# 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:

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

**Código:**

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 ). 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:**

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 |

**Analise dos resultados:**

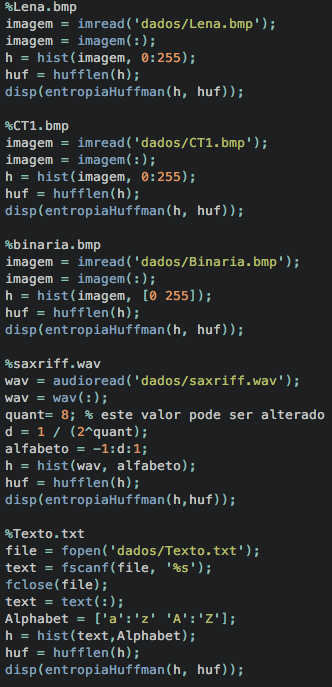
Observando atentamente os gráficos, é fácil de retirar conclusões. Em primeiro lugar os resultados das imagens. Todas elas encontram-se a preto e branco. Isto quer dizer que a matriz que é retornada representa a intensidade de cada pixel, é uma matriz n\*m. Em imagens coloridas a função *imread* retorna 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. Naturalmente que 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:

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 *hufflen()* 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:**

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.

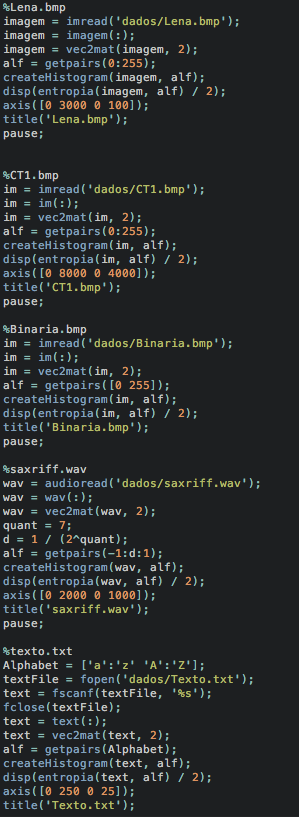
# Exercício 5

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”**.

C**ódigo:**



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.

**Análise dos resultados:**

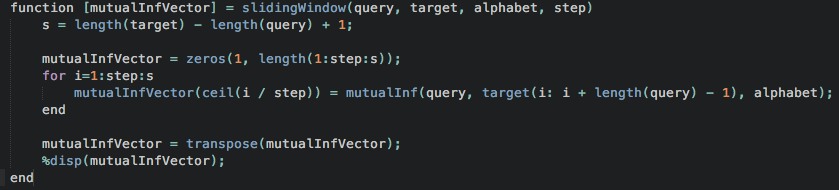
Tal como mostram os resultados acima, geralmente o agrupamento de símbolos é vantajoso uma vez que permite baixar o valor da entropia. A desvantagem do agrupamento de símbolos encontra-se no facto de os requisitos de memoria serem bastante mais altos sendo o algoritmo mais complexo. Para as imagens é 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

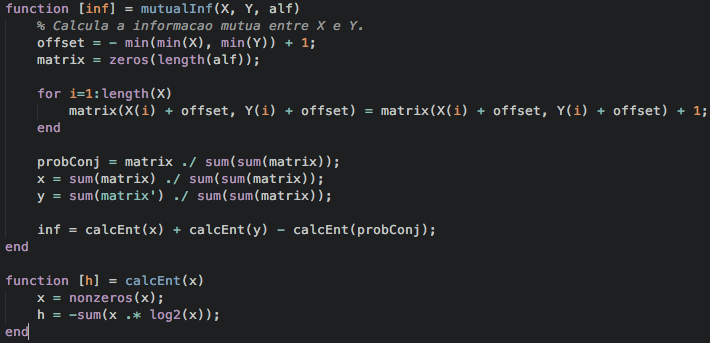
No exercício 6 vamos recorrer à informação mútua entre duas variáveis de acordo com a seguinte fórmula:

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.

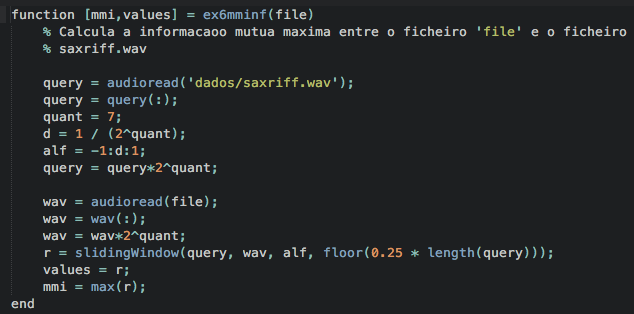
**Código:**

 *SlidingWindow.m:*

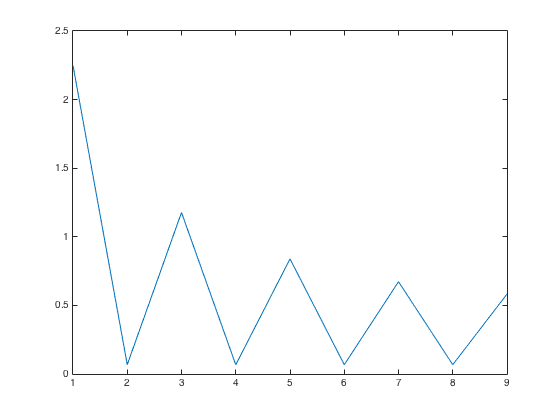
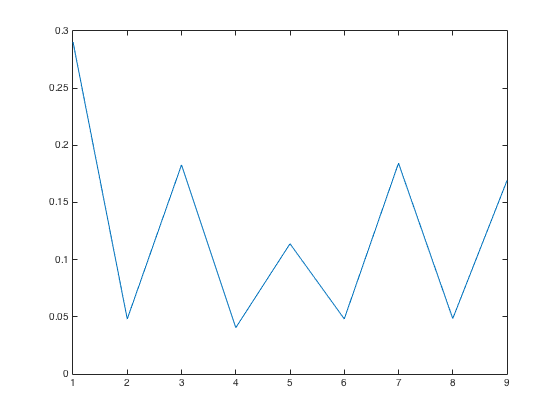
*mutalInf.m:*



ex6mminf.m



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 ruido a entropia será extremamente baixa pelo que o target 1 será o som mais próximo do som original.

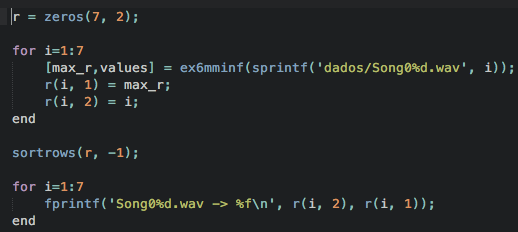


Target01 – repeat.wav Target02 – repeatNoise.wav

**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:

**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.