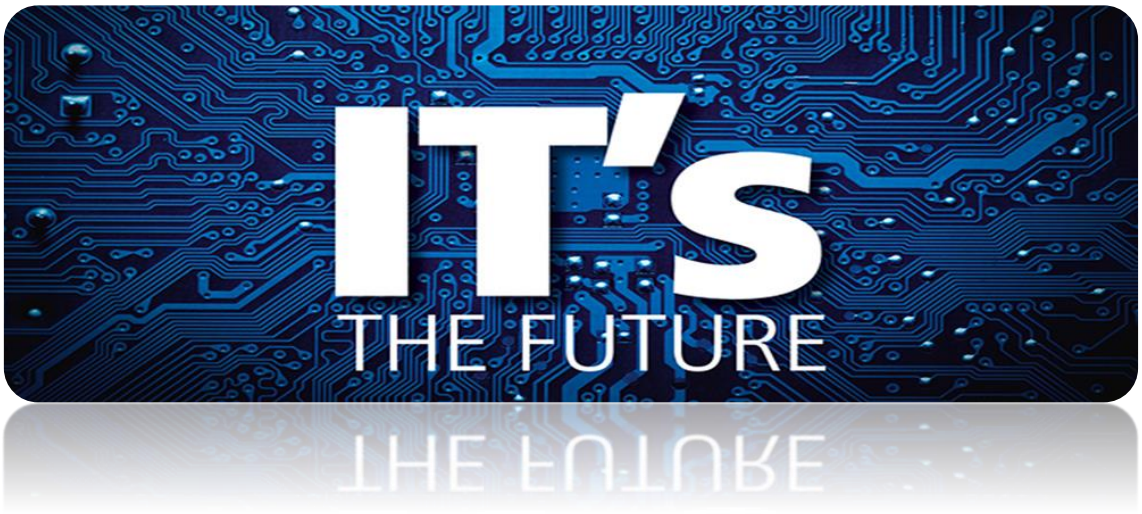
• U C •
FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Relatório do Primeiro Trabalho Prático

Entropia, Redundância e Informação Mútua



Licenciatura em Engenharia Informática

2018/2019 – 1º Semestre

Trabalho Realizado no âmbito da cadeira de Teoria da Informação, por:

- *João Filipe Carnide de Jesus Nunes - 2017247442*
- *Miguel Paulo Martins Marques - 2017266263*
- *Paulo Tomás Falcão de Almeida Cardoso – 2017249716*

18-10-2018

Índice

Exercício 1

Introdução:

No primeiro exercício é-nos pedido que implementemos uma rotina para visualizar o histograma de ocorrências dos símbolos de uma dada fonte de informação P, com um alfabeto A={a1,…,an}. Para tal, implementámos as funções: faz_freq_abs(), calcula_freq_abs(), e, recorrendo à função de MATLAB histogram(), dependendo do tipo de fonte que nos seja fornecido, uma imagem, um ficheiro de texto ou uma música, visualizámos o seu histograma. Adicionalmente, criámos as funções le_im, le_texto ou le_musica, que, novamente consoante o tipo da fonte, definem o alfabeto mais básico possível para esta, modelam as fontes de informação e transformam-nas num único vector, entre outras funcionalidades.

Código:

```
function [f_a,vector]=faz_freq_abs(orig,alfabeto)
    [l,c]=size(orig);
    if(c==1)
        f_a= calcula_frequencia_abs(orig,alfabeto);
        vector = orig;
    else
        new = zeros(l*c,1);
        k=0;
        for i=1:l
            for j=1:c
                new(k+j,1) = orig(i,j);

            end
            k = k+c;
        end
        vector = new;
        f_a = calcula_frequencia_abs(new,alfabeto);
    end
```

```
function f_a =  calcula_frequencia_abs(a,alfabeto)
    f_a=zeros(size(alfabeto));
        for n=1:length(alfabeto)
            for m=1:length(a)
                if (alfabeto(n)==a(m))
                    f_a(n)=f_a(n)+1;
                end
            end
        end
```

Breve Comentário:

Após correr o programa, verificámos que os histogramas criados estavam de acordo com os valores esperados. Testámos as rotinas nas variadas fontes de informação a nós fornecidas, e os resultados mantiveram-se consistentes. Tomámos a decisão de eliminar os espaços, pontuação e newlines do nosso ficheiro de texto, após a colocação de uma dúvida na aula e consequente esclarecimento por parte do professor responsável. Sem esta pequena correcção, os valores não se alteravam substancialmente, tendo apenas algumas ocorrências extra nas gamas mais baixas (nomeadamente o caracter 10, NEWLINE, do código de codificação de texto ASCII, ou até o caracter 32, [space]).

Exercício 2

Introdução:

No segundo exercício pede-se que programemos uma rotina para calcular o limite mínimo teórico para o número médio de bits por símbolo para a compressão da nossa determinada fonte. Assim sendo, recorremos aos conhecimentos leccionados nas aulas teóricas da cadeira, das quais retirámos a fórmula:

$$H = -\sum_{i=1}^{n} p_i \cdot \log_{2(}p_{i)}$$

Desta forma, definimos a função calcula_entropia, que, recebendo os valores da frequência absoluta calculados com a função faz_freq_abs do exercício anterior, calcula os valores da frequência relativa da nossa fonte, ou seja, a probabilidade de ocorrência dos símbolos da fonte segundo a definição frequêncista de probabilidade, e que, seguidamente, utiliza esses valores para calcular a entropia, ou seja, o limite desejado.

Contudo, com o intuito de conseguir utilizar esta rotina aqui desenvolvida, calcula_entropia(), numa fase posterior do trabalho, modificámo-la de modo a que esta calcule os valores da entropia convencional quando recebe apenas um argumento de entrada, (por meio da variável nargin predefinida pelo MATLAB), ou de forma a que calcule os valores da entropia conjunta se receber dois parâmetros de entrada.

Código:

```
function H = calcula_entropia(dado1,dado2)
    if nargin==1
            l= sum(dado1);
           freq_rel=dado1/l;
            H = 0;
            for i=1:length(dado1)
                if(freq_rel(i,1) ~= 0)
                H = H + freq_rel(i,1)*log2(1/freq_rel(i,1));
                end
            end

    elseif nargin==2
        H = calcula_entropia_intersec(dado1,dado2);
    end

end
```

Breve Comentário:

Numa fase inicial do nosso trabalho, os valores eram bastante dispares com as nossas estimativas. Após uma cuidadosa análise corrigimos um lapso de cálculo que utilizava valores errados para o cálculo das ocorrências totais de cada símbolo, e consequentemente deturpava o valor da probabilidade do dado símbolo. Os valores obtidos após a correção mantiveram-se então dentro das nossas espectativas.

Exercício 3

Introdução:

Utilizando as rotinas criadas nos exercícios anteriores, para cada uma das nossas fontes determinámos o histograma, ou seja, a distribuição estatística, e o número médio de bits por símbolo, a nossa entropia. No entanto, este exercício, de acordo com as especificações fornecidas, teve uma ligeira alteração no alfabeto utilizado para os ficheiros áudio, dependente agora do número de bits de quantização da nossa fonte. Como anteriormente já havíamos retirado os acentos e pontuação do nosso texto, decidimos importar um novo alfabeto para explorar as funcionalidades disponíveis no MATLAB. Importamos assim um documento de texto criado por nós, contendo todos os elementos do alfabeto que queremos considerar, o ficheiro Alfabeto_BaseDados.txt. Esta abordagem difere da precedente, na medida em que anteriormente removemos todos os valores desnecessários da nossa fonte à posteriori. Este novo documento contém apenas as 52 letras (maiúsculas e minúsculas), do abecedário inglês.

Código:

```
function faz_histogramas_ex3(kid,homer,homerBin,l,espacamento,ab,alfa_t)
    figure('name','Distribuição Estatística das fontes dadas','NumberTitle','off');
    subplot(3,2,1);
    Imz= 0:1:260;
    histogram(kid,Imz);
    title('kid.bmp');
    subplot(3,2,2);
    histogram(homer,Imz);
    title('homer.bmp');
    subplot(3,2,3);
    histogram(homerBin,Imz);
    title('homerBin.bmp');
    subplot(3,2,4);
    histogram(l,espacamento);
    title('guitarSolo.wav');
    subplot(3,2,5);
    histogram(ab,alfa_t);
    title('english.txt');
```

Resultados:

Ficheiro Fonte	Entropia	Taxa de Compressão
kid.bmp	6.9541	13.07%
homer.bmp	3.4659	56.68%
homerBin.bmp	0.6448	91.94%
english.txt	4.2280	25.83%
guitarSolo.wav	7.3580	8.03%

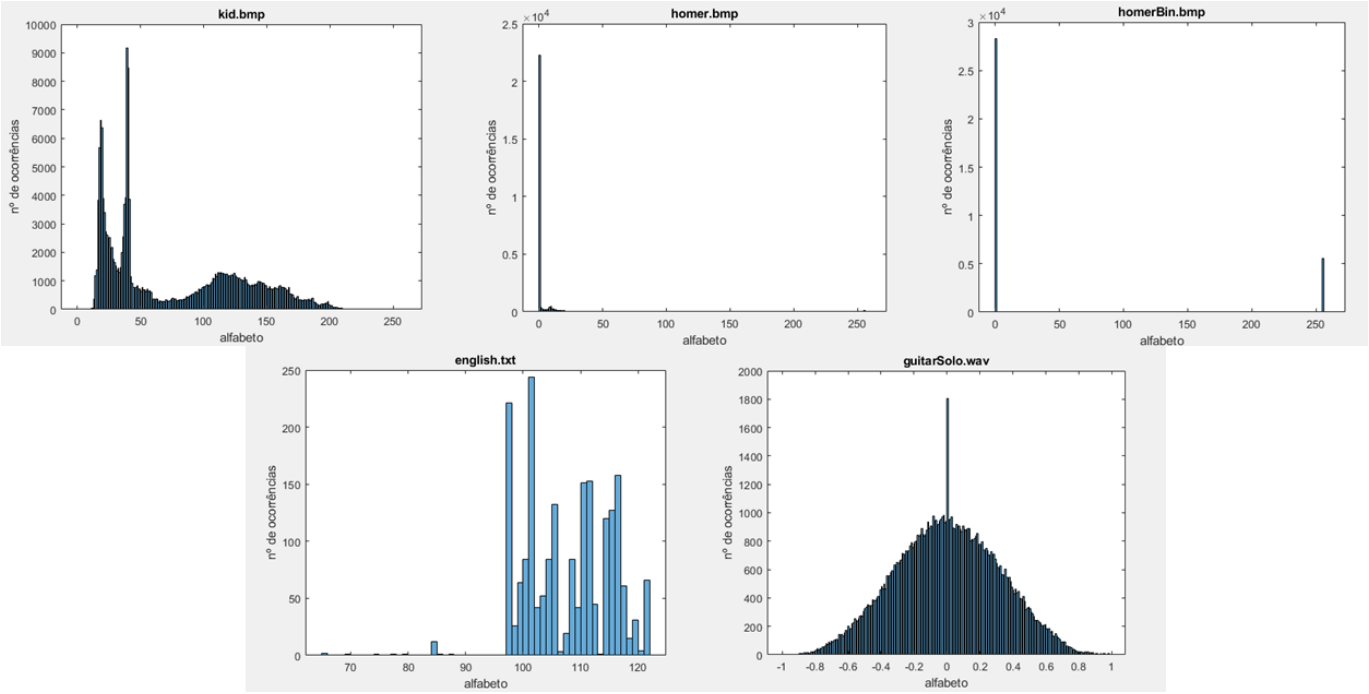


Imagem 1 – Histogramas do exercício 3

Análise:

Os resultados obtidos foram bastante promissores, e em conformidade com o expectável.

O cerne por detrás das diferenças obtidas entre os vários valores de entropia é perceptível ao comparar a distribuição das várias fontes. A imagem “kid.bmp” apresenta uma entropia elevada, e logicamente uma dispersão de resultados relativamente grande, que se traduz numa gama vasta de cinzentos. Por outro lado, a fonte “homer.bmp”, mais simples, possui um espectro reduzido de cinzentos, bastante branco e preto, detendo assim uma distribuição de resultados menos dispersa. Segue então que a sua entropia será mais pequena que a da imagem “kid.bmp”, e, por conseguinte, a imagem “homerBin.bmp”, correspondente a uma binarização de “homer.bmp”, terá uma entropia ainda mais pequena, o que se verifica nos nossos resultados. A incerteza será assim tão menor quanto mais pequena a gama de valores representados na imagem. Uma fonte com uma menor dispersão de resultados terá uma incerteza inferior, ou seja, a surpresa ao averiguar que uma imagem binaria tem muitos valores brancos é pequena, dado que esta só pode ter dois valores possíveis, ou preto, ou branco.

Desta forma, o mesmo raciocínio aplica-se às diferentes fontes, sejam elas de música ou de texto. A faixa “guitarSolo.wav” tem um histograma com uma dispersão de valores muito elevada, o que significa que terá uma incerteza grande e uma taxa compressão baixa.

Questão: Será possível comprimir cada uma das fontes de forma não destrutiva (lossless)?

É, de facto, possível uma compressão não destrutiva das nossas fontes, na qual a compressão máxima (onde a reconstrução dos dados é exata) será o valor da entropia, valor este que representa, por definição, o limite mínimo teórico do número médio de bits utilizados para representar cada símbolo na compressão.

Podemos assim concluir que seria exequível comprimir os valores de uma forma não destrutiva com uma maior eficiência. Seria, presumivelmente, possível atingir taxas de compressão entre 8.03% e 91.94% dependendo da fonte, ganhos estes substanciais.

Em todas as fontes, exceptuando o ficheiro "english.txt", a entropia máxima, tendo em conta que os alfabetos têm sensivelmente uma dimensão de 256, ou seja, N=256, será:

$$\log_2 N = \textbf{\textit{Entropia Máxima}}$$

Isto, é:

$$\log_2 256 = 8$$

Por outro lado, o alfabeto da nossa fonte de texto é menor, dado que apenas consideramos as letras do abecedário Inglês (26 letras), maiúsculas e minúsculas (26*2=52 letras). N=52.

Decorre:

$$\log_2 52 \approx 5.70$$

O limite máximo do número médio de bits para o ficheiro "english.txt" é 5.70 bits.

Assim, a equação utilizada para calcular a taxa de compressão teórica será:

$$TaxaCompressao = \frac{EntropiaMaxima - Entropia}{EntropiaMaxima} * 100$$

Exercício 4

Introdução:

Utilizando a rotina de Huffman fornecida, a função hufflen(), calculamos de forma equivalente ao primeiro/terceiro exercícios o número médio de bits por símbolo para cada fonte.

Servimo-nos da seguinte fórmula:

$$\boldsymbol{l = \sum_{i=0}^{n} P(x_i) * hufflen(x_i)}$$

Definimos então a rotina calcula_numero_medio (), que recebe por parâmetro de entrada o valor da frequência absoluta calculado no primeiro exercício. Servindo-se da função hufflen, que também recebe a frequência absoluta e devolve o valor huff, que na nossa equação corresponde a $\sum_{i=0}^{n} hufflen(xi)$, multiplica este valor pelo valor da probabilidade, prob, $\sum_{i=0}^{n} P(a_i)$.

Para calcular a variância, utilizamos a rotina var(), predefinida pelo MATLAB.

Código:

```
function [med,v] = calcula_numero_medio(f_a)
    soma = sum (f_a);
    prob = double(f_a/soma);
    huff = hufflen(f_a);
    v = var(huff,f_a);
    med = prob'*huff;
end
```

Resultados:

Ficheiro Fonte	Número Médio de Bits	Variância
homerBin.bmp	1.0000	0.0000
homer.bmp	3.5483	13.1968
kid.bmp	6.9832	2.0984
english.txt	4.2524	1.1086
guitarSolo.wav	7.3791	0.7563

Análise:

Na implementação deste algoritmo, limitámo-nos a aplicar a fórmula acima mencionada. Usámos a função sum(), incutida no MATLAB, de modo a obter o número total de ocorrências, para, na instrução seguinte, criarmos um vetor com as respectivas probabilidades (basicamente, a partir da frequência absoluta criámos a frequência relativa). Depois utilizámos a função dada hufflen.m para calcular o número médio de bits e com esse valor, por fim, calculamos o número médio de bits, recorrendo à multiplicação de matrizes, onde multiplicámos a transposta da matriz das probabilidades com a função que a função hufflen.m nos devolveu, de maneira a obtermos um único número, correspondente ao valor médio de bits da fonte fornecida.

Questão: Será possível reduzir a variância?

Relativamente a este tópico, para verificar se é possível reduzir a variância é necessário analisar o algoritmo fornecido, denominado hufflen.m, que recebe como parâmetro de entrada as ocorrências de cada símbolo de uma dada fonte e que devolve o número de bits necessário para codificar cada símbolo. Após uma breve análise do algoritmo, decidimos que a melhor abordagem seria fazer uma série de testes, e analisá-los com o intuito de verificar se a variância do número de bits necessários para cada símbolo era mínima. O primeiro dos testes que fizemos foi o mesmo que havia previamente sido explicado na aula teórica, onde possuímos cinco símbolos e as suas ocorrências correspondem a 40, 2 0,20,10 e 10. Na aula analisámos duas possibilidades para o número de bits de cada símbolo sendo um 1,2,3,4,4 e o outro sendo 2,2,2,3,3 para 40, 20,20,10 e 10 respetivamente. Após submeter a função hufflen.m a este teste o seu output foi 2,2,2,3,3. O que nos mostrou que, desde o início, o algoritmo estava de feito de maneira a que a variância fosse mínima. Mas ainda assim, para confirmarmos a nossa hipótese, fizemos mais alguns testes, e fizemos também a árvore binária para os exemplos dados (1,0,4,2,0,1 e 10,40,20,10) que comprovaram também ser impossível reduzir a variância.

Exercício 5

Introdução:

Neste exercício é-nos pedido que utilizemos a mesma estrutura aplicada no exercício 3), mas desta vez admitindo que cada símbolo é na realidade uma sequência de dois símbolos contíguos.

Desta maneira criámos a rotina agrupado(), que recebe como parâmetros de entrada a fonte e o alfabeto. Utilizando a função predefinida do MATLAB ndgrig(), constituímos uma matriz novo_alfabeto, com todas as permutações possíveis do alfabeto original duas a duas ($_2^nP$), com n o número de elementos do nosso alfabeto original. Posteriormente, moldámos a matriz fonte, com recurso a função reshape(), para definir uma nova matriz, agrup, de 2 linhas por n/2 colunas, que corresponde assim à nova matriz de elementos da nossa fonte agrupados dois a dois. Assim, utilizando a rotina de MATLAB bsxfun(), verificamos as ocorrências de cada permutação do nosso novo alfabeto na matriz agrup, recebendo uma nova matriz lógica. Esta função executa elemento a elemento o parâmetro @eq, ou seja, verifica, se os valores da permutação ocorrem na matriz agrup. Assim, executando uma operação logica entre ambas as colunas da nossa matriz, e aplicando um sum() à mesma, obtemos o número de vezes que cada permutação se verifica em agrup. Seguidamente, utilizamos a rotina desenvolvida no exercício 2), calcula_entropia(), para obter o valor da entropia agrupada que procurávamos.

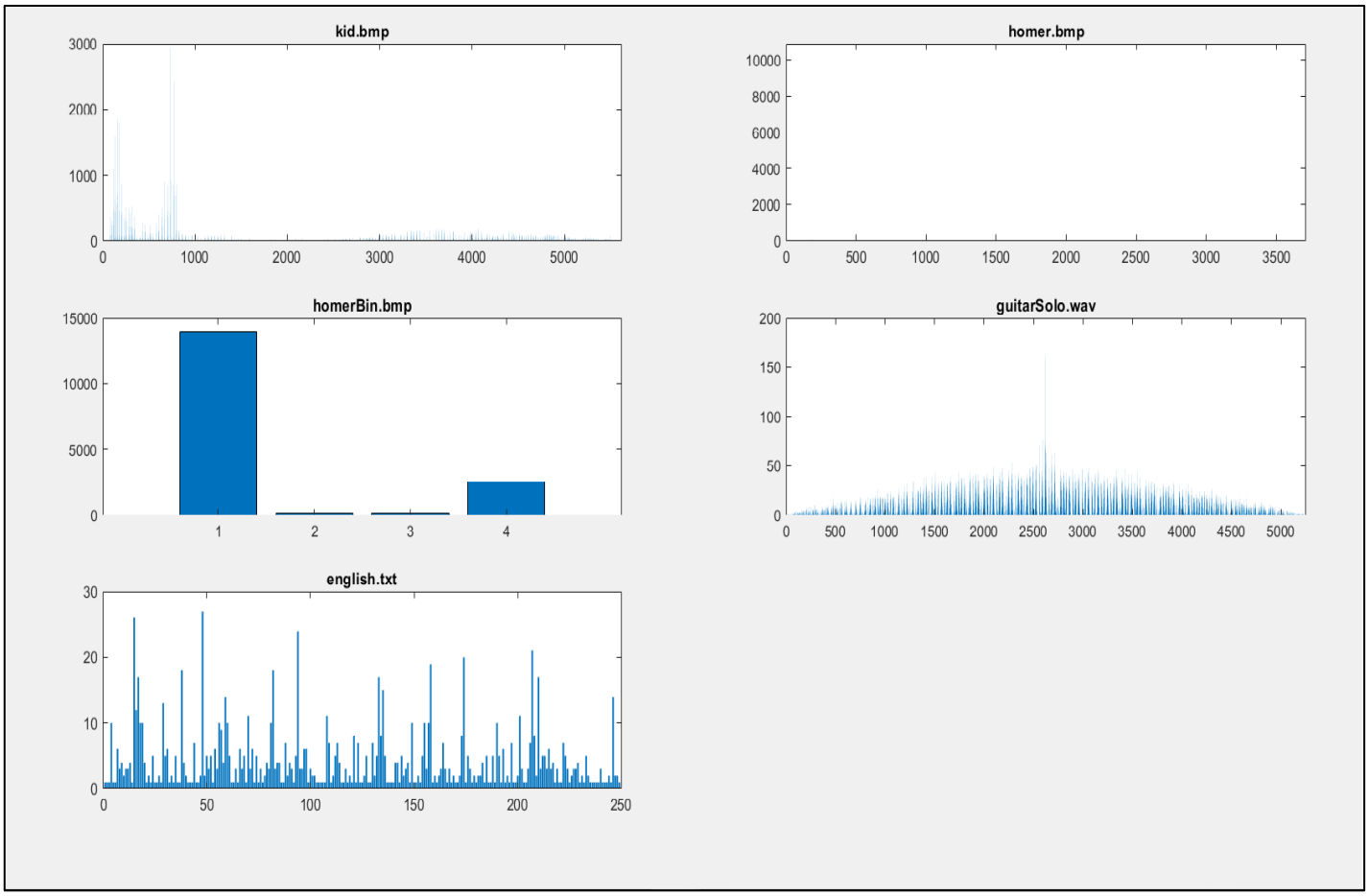
Mediante a indicação do professor responsável, decidimos que, se a nossa matriz fonte tivesse um número ímpar de elementos, desprezaríamos o último valor, procedendo assim com os nossos cálculos.

Código:

```
function [H,f_a] = agrupado(fonte,alfabeto)
    if (mod(length(fonte),2)==1)
        fonte = fonte(1:length(fonte)-1);
    end
    [a,b] = ndgrid(alfabeto);
    novo_alfabeto = [b(:),a(:)];
    [x,y] = size(novo_alfabeto);
    ocorre = zeros(1,x);
    agrup = reshape(fonte,2,[]);
    agrup = agrup';
    aux = zeros(1,y);
    for l=1:x
        aux = novo_alfabeto(l,:);
        logic = bsxfun(@eq, aux,agrup);
        soma = logic(:,1)&logic(:,2);
        %soma = logic(:,1)+logic(:,2);
        %ind = find(soma<2);
        %soma(ind) = 0;
        ocorre(1,l) = sum(soma);
    end
    inf = find(ocorre>0);
    f_a = ocorre(inf);
    H = calcula_entropia(f_a');
    H = H/2;
```

Resultados:

Ficheiro Fonte	Entropia Agrupada
homerBin.bmp	0.3978
homer.bmp	2.4127
kid.bmp	4.9091
english.txt	3.6521
guitarSolo.wav	5.7808



Análise:

Após executar o programa, os resultados obtidos estavam dentro do conforme, nomeadamente, os valores da entropia de cada amostra haviam descido quando comparados com os valores calculados no exercício 3). No entanto, o custo desta técnica é uma acrescentada complexidade algorítmica que torna a execução do programa mais lenta. Por isso, e tendo em conta que agrupamos os nossos símbolos como uma sequência de apenas dois valores contíguos, o ganho foi visível, mas não estratosférico. Segue assim que sequências maiores produzirão ganhos mais substanciais, mas sempre acrescentando à complexidade dos algoritmos e ao custo de tempo de processamento. Caberá então ao programador definir um bom compromisso entre a eficiência no cálculo de uma entropia mais elevada, e consequente compressão menos eficaz, mas mais rápida, ou o contrário.

Exercício 6.A

Introdução:

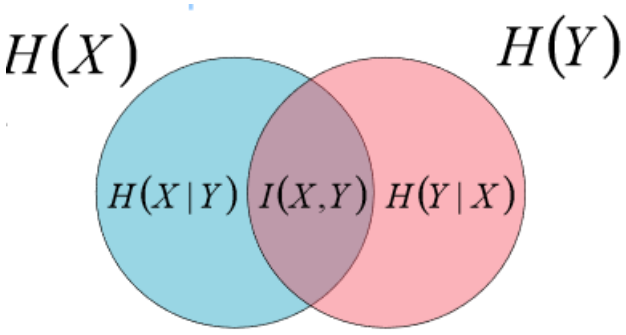
Nesta questão foi-nos pedido que considerássemos dois trechos de música. À semelhança do que acontece com comunicações em ambientes ruidosos, ou por exemplo com a aplicação "Shazam", teríamos então de identificar se o clip de música que designamos de query partilha alguma informação com o áudio que definimos como a target, aplicando o método da informação mútua.

Conforme fora explicitado no enunciado, e à semelhança dos métodos desenvolvidos no "trabalho prático zero", teríamos então de recorrer a uma janela deslizante (com a query o sinal a pesquisar), que percorreria a música onde queríamos efectuar a nossa pesquisa, a target. A cada conjunto de valores tomado pela janela determinar-se-ia a informação mútua entre a query e essa mesma janela, para que, no final, obtivéssemos um vetor com as informações mútuas de todas as janelas que haviam percorrido a target.

Sendo esta a primeira abordagem a este conceito, foi-nos dado o alfabeto, [0 1 2 3 4 5 6 7 8 9 10], a query, [2 6 4 10 5 9 5 8 0 8], e a target, [6 8 9 7 2 4 9 9 4 9 1 4 8 0 1 2 2 6 3 2 0 7 4 9 5 4 8 5 2 7 8 0 7 4 8 5 7 4 3 2 2 7 3 5 2 7 4 9 9 6], com um step igual a 1, e, desta forma, deveríamos calcular a informação mútua de 41 janelas (1 + (Tamanho target - Tamanho janela)/ step). O output a obter fora-nos fornecido no enunciado.

Com estes dados criámos uma rotina em MATLAB que calcula a informação mútua de cada janela, infor_mutua(), que tem como parâmetros de entrada a query, o target, o alfabeto e o step, sendo que o número de resultados é calculado dentro desta função. Consequentemente, esta devolve um vetor com os resultados da informação mútua de cada janela.

Para o cálculo destes valores utilizámos a fórmula leccionada nas aulas:



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

*Sendo o nosso X a query e o Y cada uma cada janelas da Target.

Para calcular a entropia individual de cada janela e da query utilizámos a função já definida no exercício 3), calcula_entropia(). Para a entropia da intersecção da janela com a query, servimo-nos da mesma calcula_entropia(), cujo código alterámos para determinar ora a entropia normal, ora a da intersecção, recorrendo à variável de MATLAB "nargin" para diferenciar os casos (como havíamos referido na introdução ao exercício 2). Se a rotina receber, assim, somente um parâmetro de entrada, calcula a entropia convencional de uma única matriz; recebendo dois, calculará a entropia da intersecção.

Código:

```
function mut = infor_mutua(query,target,alfabeto,step)
    janela=zeros(1,length(query));
    %Calcular a entropia do query, que e sempre a mesma
    f_a_query = faz_freq_abs(query,alfabeto);
    H_X=calcula_entropia(f_a_query');
    %calcular o numero de resultados, ou seja o numero de elementos de mut
    n_res= 1+floor((length(target)-length(query))/step);
    mut= zeros(n_res,1)';
    move = 1;
        for k=1:step:(length(target)-length(query)+1)
            %percorrer a janela e igualar os valores aos da target
            janela(:)= target(k:length(query)-1+k);
            %calcular a entropia da janela que vai mudando de cada vez que mudamos a janela;
            %f_a_janela=histc(janela,unique(janela));
            f_a_janela=histc(janela,unique(janela));
            H_Y=calcula_entropia(f_a_janela');
            %calcular a entropia da intersecao
            H_XY=calcula_entropia(query,janela);
            %colocar o resultado no vector
            mut(move)= H_X-(H_XY-H_Y);
            move = move +1;
        end
end
```

```
function H = calcula_entropia_intersec(query,janela)
    dados = bitsll(query,16)+janela;
    [f_a,waste] = histc(dados,unique(dados));
    H = calcula_entropia(f_a');
end
```

Exercício 6.B

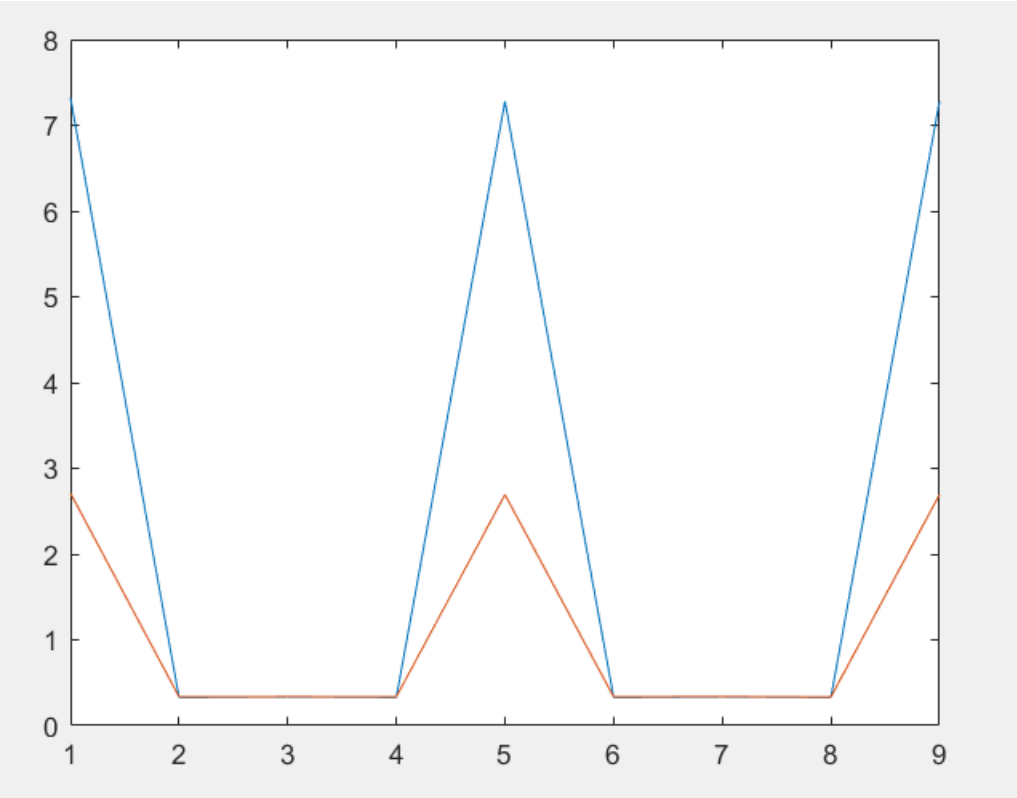
Introdução:

Entrando neste exercício num contexto mais prático, e, portanto, a nosso ver menos abstracto, pedia-se que definíssemos uma fonte WAV chamada GuitarSolo.wav como query, e que da mesma forma considerássemos dois targets distintos, o target01 – repeat.wav e o target02 – repeatNoise.wav, com um step de um quarto do tamanho do GuitarSolo.wav. Todas estas fontes se tratam de ficheiros WAV com 8 bits de quantização.

Utilizando desta vez trechos reais de música, em oposição às matrizes genéricas da alínea anterior, conseguimos mais facilmente compreender as parecenças com aplicações como o "Shazam".

Assim, é fácil deduzir a relação entre os valores obtidos para a informação mútua e as parecenças sonoras das músicas que estamos a comparar. Quanto mais semelhantes os trechos forem aos nossos ouvidos, mais próximo o valor da informação mútua do valor da entropia, e mais distante consoante a existência de ruído e outras distorções.

Resultados:



target01 – repeat.wav

target02 – repeatNoise.wav

Ficheiro	Vector de Informação Mútua								
repeat.wav	7.3207	0.3289	0.3354	0.3297	7.2789	0.3289	0.3355	0.3293	7.2806
repeatNoise.wav	2.7113	0.3354	0.3379	0.3366	2.6934	0.3363	0.3388	0.3340	2.6946

Análise:

Após uma breve audição podemos constatar que tanto o target01 – repeat.wav assim como o target02 – repeatNoise.wav são muito parecidos com o som original, mas percebemos facilmente que o target02 – repeatNoise.wav tem muito ruído em comparação com a query e com target01 – repeat.wav. Logo, após analisar os resultados, verificamos que a nossa query tem maior informação mútua com target01 – repeat.wav do que com target02 – repeatNoise.wav, assim como seria de esperar. Ocorre também um pico na informação mútua de quatro em quatro janelas, o que é consistente com a ocorrência da query na target.

Exercício 6.C

Introdução:

Para finalizar este primeiro trabalho prático fizemos um simulador de identificação de música, tal como pedido, em que a nossa query será o ficheiro WAV GuitarSolo, que já tínhamos trabalhado em exercícios anteriores. Temos assim variadas amostras a considerar como targets no nosso simulador, os trechos: Song01, Song02, Song03, Song04, Song05, Song06, Song07, Song08 e Song09 (todas estas ficheiros WAV).

Para a resolução deste problema, começámos por importar as fontes com a função predefinida do MATLAB audioread(), que aceita como parâmetro de entrada uma String com o nome do ficheiro fonte, de formato WAV. Esta função devolve a amplitude (podendo ter entre uma e duas colunas, se estiver respectivamente no formato mono ou stereo) e a taxa de amostragem, dada em Hertz.

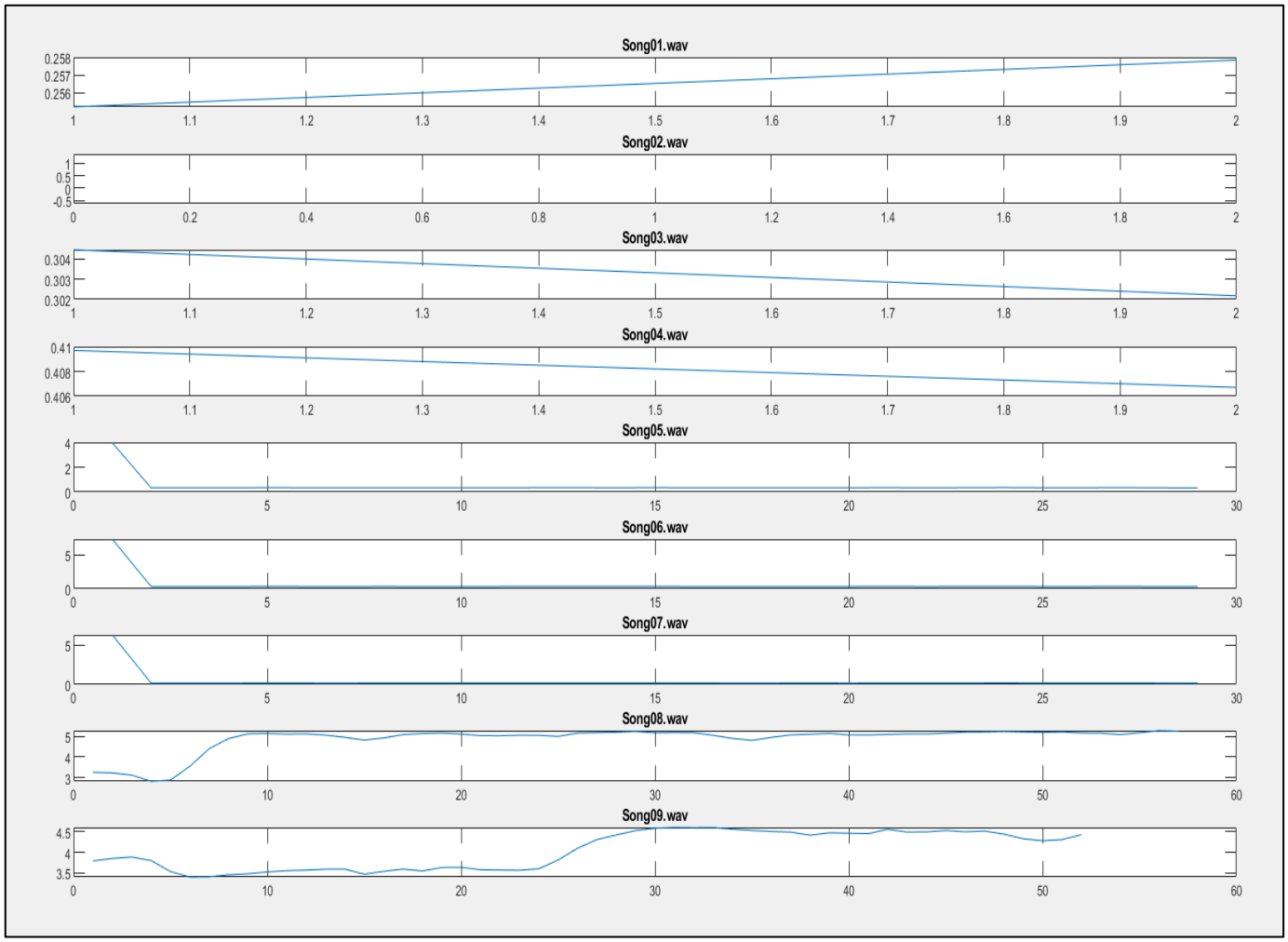
Destas duas variáveis devolvidas pelo audioread, apenas utilizámos a amplitude, pois é única necessária para determinar a informação mútua. Ulteriormente, criámos um vetor que contém, em cada célula, o respectivo valor da amplitude. É de salientar que, neste projecto, optámos por considerar apenas a coluna da esquerda nos casos em que as músicas analisadas se encontravam em formato stereo.

A rotina que desenvolvemos para resolver este exercício, simulador(), aceita como parâmetros de entrada a query (GuitarSolo.wav), o seu alfabeto, determinado no exercício 3), e, por fim, o vector de amplitudes mencionado anteriormente. Como "return" ou parâmetros de saída a função devolve um vetor com os valores máximos da informação mútua calculada para cada uma das nove músicas, todos eles ordenados por ordem decrescente, como especificado.

A nível de algoritmia do código, escolhemos "reciclar" a função usada no exercício 6b), designada infor_mutua(), e no local do segundo parâmetro de entrada colocámos a amplitude de cada uma das células. Para conseguir manipular os valores armazenados em células, e sobre eles executar as rotinas já desenvolvidas nos restantes exercícios, convertemos para forma vectorial cada célula, recorrendo à função pré-existente do MATLAB cell2mat. E, por fim, para ordenar decrescentemente os valores da informação mútua máxima, aplicámos as funcionalidades max() e sort(), também elas já implementadas pelo MATLAB.

Decidimos, após breves tentativas com diferentes alternativas, na determinação da evolução da informação mútua para cada um dos ficheiros, apresentar os nossos resultados na forma de gráfico, ou seja, usando uma função plot(), ao contrário do que seria talvez mais convencional, um histograma, conseguindo através das funções histogram() ou bar(). Tomámos esta decisão porque achámos que a visualização dos resultados, na generalidade das situações, era mais perceptível e minimalista.

Resultados:



Posição	Nome do Ficheiro	Informação Mútua Máxima
1ª	Song06.wav	7.3384
2ª	Song07.wav	6.3131
3ª	Song08.wav	5.2736
4ª	Song09.wav	4.6026
5ª	Song05.wav	3.9618
6ª	Song04.wav	0.4097
7ª	Song02.wav	0.3777
8ª	Song03.wav	0.3045
9ª	Song01.wav	0.2578

Análise:

Visualizando os gráficos obtidos, podemos facilmente compreender a evolução da informação mútua de todas as músicas, à excepção da Song02. Na realidade, esta é, na nossa opinião, a forma de representação mais autêntica, dado que este trecho em particular tem apenas um valor de informação mútua, e portanto a sua “progressão” é, para todos os efeitos, inexistente. Uma representação alternativa, nomeadamente em forma de histograma resultaria num gráfico com uma barra assinalada, ou seja, um erro de representação.

Como seria de esperar a Song06 é a que possui maior informação mutua, pois se a analisarmos com atenção, os primeiros segundos são deveras idênticos à nossa query, o que nos deixa estabelecer uma comparação, como é feito no enunciado, entre o nosso simulador e um programa tipo "Shazam", na qual na nossa mini-base de dados compreenderia estas nove músicas, e a música gravada pelo utilizador, da qual este pretenderia obter a identificação, seria a query, GuitarSolo.wav. O programa funcionaria de forma idêntica, e indicaria que a fonte original seria, nada mais nada menos que, a Song07.wav. Perante os restantes resultados, constatámos que os resultados que se sucedem pertencem às músicas semelhantes à Song07.wav, com um nível de ruído sucessivamente superior. No fundo da nossa tabela, temos sons que não possuem praticamente informação em comum com GuitarSolo.wav, algo que podemos facilmente constatar com uma audição rápida, sendo a última faixa a Song01.wav. Nas Song08.wav e Song09.wav, visto que estamos a trabalhar com fontes wav de 16 bits de quantização, seria de esperar uma entropia elevada, uma vez que são fontes com um alfabeto superior à da nossa query, que só possui 8 bits de quantização. Logo, é expectável que a informação mútua destas duas faixas seja também elevada, pois, mesmo que estas fontes pareçam distintas da nossa query, possuem ainda assim mais informação total do que as restantes fontes do simulador. Por outras palavras, para uma fonte codificada com 8 bits de quantização, uma entropia relativamente alta, por exemplo de 3.400, será uma entropia baixa para uma fonte codificada com 16 bits.

Conclusão

Chegando assim ao términus do nosso projecto, consideramos importante tecer algumas breves considerações sobre o trabalho desenvolvido.

Numa fase prévia do nosso trabalho, após a primeira obtenção de todos os resultados pedidos no enunciado, reparámos que o código que havíamos desenvolvido, apesar de aparentemente correto, era extremamente ineficiente, demorando mais de uma hora a determinar todos os resultados pretendidos. Assim, procedemos no sentido de optimizar ao máximo os algoritmos criados e de perceber quais as demoras desnecessárias e quais os "apêndices" supérfluos que poderíamos eliminar. Conseguimos diminuir substancialmente o tempo de processamento do programa desenvolvido, substituindo algoritmos fortemente dependentes em ciclos "for", por funções predefinidas de MATLAB como a função histc(), entre outras. Por outro lado, decidimos não melhorar o código desenvolvido no exercício 5), apesar de usufruirmos de bastante tempo para o fazer. Tomámos esta decisão numa tentativa de alcançar um ideal de "diversidade algorítmica" para este exercício. Passamos a explicar: escolhemos este exercício em específico porque, após uma cuidada análise, concluímos que a resolução deste decorre dos mesmos moldes utilizados posteriormente no problema 6). Assim, considerámos que repetir a mesma estratégia já conseguida noutra porção do trabalho seria redundante e derivativo, não muito diferente de uma abordagem "copy paste" que de pouco nos serve como engenheiros informáticos. Desta forma, para demonstrar uma capacidade de resolver problemas recorrendo a diferentes soluções, e cientes de que posteriormente, num exercício mais complexo e demorado como a alínea 6.C), havíamos mostrado a capacidade de atingir não só uma solução, como, mais propriamente, uma solução óptima, ou tão próximo disso quanto possível para nós, tomámos a decisão de manter este exercício na sua forma original. Em suma, queríamos apresentar soluções alternativas para exercícios semelhantes, para melhor transmitir o nosso domínio da linguagem de programação em questão e também da matéria abordada.