# TP0 – Trabalho Laboratorial de Introdução ao NumPy e Matplotlib


## Introdução

Período de execução: 14 dias (2 aulas praticas laboratoriais)

Esforço extra-aulas: 4h

Data de Entrega: N/A (o TP0 não será avaliado)

Objetivo: pretende-se que o aluno se familiarize com as bibliotecas NumPy, SciPy e Matplotlib.

## Instruções de instalação e configuração

A seguir é apresentada uma opção para instalação e configuração, embora existem outras alternativas possíveis que não envolvem o uso de Anaconda. Adicionalmente, podem utilizar um IDE a escolha, podendo em alguns casos exigir configurações adicionais para ligação ao Python e respetivas bibliotecas.

1. Install Anaconda and use Spyder as the IDE (this is a recommendation that simplify the integration of all libraries needed in the course).
2. If you already have Anaconda, make sure it is updated to the last version (as well as the required libraries: Python, NumPy, Matplotlib, SciPy).
3. Install also the sounddevice module.

I. Windows users: Anaconda power shell (run as administrator)
II. Mac / Linux users: open terminal

```
conda install -c conda-forge python-sounddevice
```

Professor T: Paulo Carvalho, Marco Simões
Professores PLs: Paulo Carvalho, Marco Simões, Francisco Antunes

# Trabalho Prático 0 – Parte A

## Elaboração de um conjunto de scripts e funções para manipulação de som

1. Crie um script e grave-o com o nome **'mainAudio.py'**. Este script será utilizado na chamada de todas as funções da Parte A.

2. Dentro deste script, faça a leitura do ficheiro de áudio disponibilizado em formato wave **'drumloop.wav'**, utilizando a função ***read*** do scipy.io:

```
from scipy.io import wavfile

[fs, data] = wavfile.read(filename);
```

   Os parâmetros de saída são: frequência de amostragem (fs), i.e., quantas amostras do sinal foram adquiridas por segundo no processo de digitalização; e data que são os próprios valores do sinal.

   Para mais detalhes, pode consultar a ajuda do scipy: **help(wavfile.read)**

3. Escute o sinal de áudio, com recurso à função ***play*** da biblioteca sounddevice.

```
import sounddevice as sd

sd.play(data, fs)
status = sd.wait()                    # Wait until file is done playing
```

   3.1. Escute o sinal de áudio indicando uma frequência igual a fs*2
   3.2. Escute o sinal de áudio indicando uma frequência igual a fs/2

4. Crie uma função **apresentarInfo('nomeFicheiro', fs, nrBitsQuant)** que apresente no ecrã informações (nome, taxa de amostragem e nº de bits da quantização) sobre o ficheiro. Utilize a função ***print*** para mostrar esta informação na consola do Python.

   Nota: nrBitsQuant refere-se ao número de bits com o qual foi quantificado o sinal, e deverá ser inserido como argumento de entrada da função. Para obter esta informação, pode utilizar a função **data.itemsize** que retorna o número de bytes de quantitização.

   Exemplo do resultado da função:

| Informações do ficheiro |
| --- |
| Nome: drumloop.wav |
| Taxa de amostragem: 44.1 kHz |
| Quantização: 16 bits |

5. Elabore uma função chamada **visualizacaoGrafica(data, fs)** que apresente o sinal de áudio num gráfico 2D (amplitude vs. tempo).

   5.1. Para definir o eixo do tempo, deverá converter o índice de cada amostra para o respetivo valor em segundos. Para fazer a conversão deverá utilizar a variável **fs** (número de amostras adquiridas num segundo) obtida na leitura do ficheiro, como segue:

   Considere que a primeira amostra corresponde a t=0;

   O intervalo de tempo entre sucessivas amostras (i.e. o período de amostragem) será: Ts = 1/fs;

   O instante final (em segundos) será tfim= (Num_Amostras - 1) * Ts.

   5.2. Um sinal de áudio pode ser monoaural (um único canal) ou estéreo (canal direito e esquerdo).

   No caso de se tratar de um sinal monoaural, apresente o sinal num único plot.

   Caso o sinal seja stereo, apresente o canal esquerdo na parte superior e o direito na parte inferior da figura.

   5.3. Apresente as amplitudes do sinal normalizadas, i.e., o sinal deverá estar contido no intervalo [-1, +1].

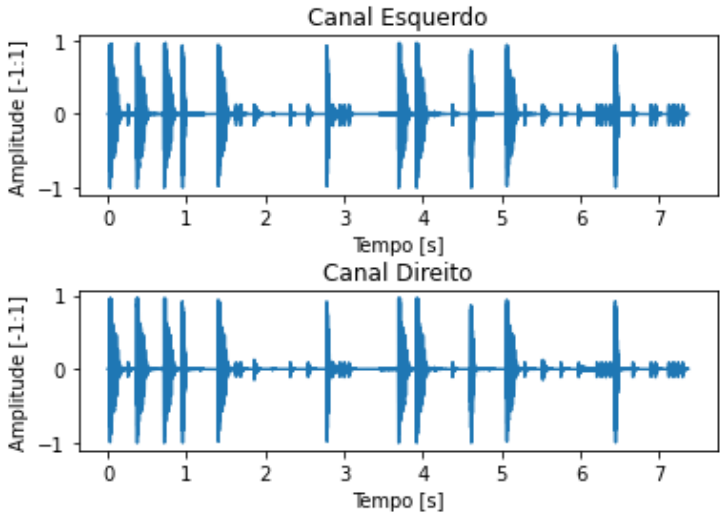
   5.4. Dar nomes aos eixos ***x*** e ***y*****,** e um título a cada canal.

   5.5. Para a implementação gráfica utilize as funções ***figure, plot***, ***subplot, xlabel***, ***ylabel***, ***title*****,** **show** e ***subplots_adjust*** da biblioteca matplotlib.pyplot. Exemplo:

```
import matplotlib.pyplot as plt

plt.figure(1)
plt.subplot(211)
plt.plot(tempo, canalDir)
plt.xlabel('Tempo [s]')
plt.ylabel('Amplitude [-1:1]')
plt.title('Canal Direito')
plt.subplots_adjust(hspace=0.5)
...
plt.show()
```

Exemplo do gráfico resultante:




5.6. Torne a função mais genérica, definindo-a como **visualizacaoGrafica(*args)**, onde o número de argumentos de entrada poderá variar entre 2 e 4, como segue: **(data, fs, tini, tfim)**

5.6.1. No caso de apenas serem introduzidos os dois primeiros argumentes **(data, fs)** o funcionamento deverá ser idêntico ao descrito no ponto 5.1.

5.6.2. No caso de serem introduzidos três argumentos **(data, fs, tini)**, o gráfico não deverá começar em 0 mais sim no valor indicado por **tini**.

5.6.3. No caso de serem introduzidos os quatro argumentos, o gráfico deverá ser restringido ao intervalo **[tini, tfim]**.

6. Execute o mesmo procedimento do ponto anterior, agora com a função ***axis*** do matplotlib.pyplot, a qual permite restringir o intervalo de visualização de forma mais simples.

```
plt.axis([Xmin, Xmax, Ymin, Ymax])
```

7. Crie uma função que adicione ruído uniforme ao sinal. Esta função deverá receber o sinal original e a amplitude do ruído e devolver o sinal original com ruído.

7.1 Para gerar o sinal de ruído recorra à função ***np.random.rand***.

7.2 A função ***rand*** gera uma matriz, do tamanho especificado, contendo valores aleatórios no intervalo [0, 1). Deverá, portanto, adaptar a escala do vetor ruído, para que seus valores fiquem contidos no intervalo [-Amp, +Amp), sendo Amp a amplitude do ruído especificada como argumento de entrada.

8. Implemente uma função para o cálculo da energia do sinal; a função em causa deverá receber o sinal, e devolver a sua energia (ou a energia de cada canal, no caso de um áudio stereo).

8.1. O cálculo da energia (E) deverá ser implementado de acordo com a equação:

$$E = \sum_{i=0}^{N-1} x_i^2$$

onde ***i*** representa cada amostra do sinal ***x***, e N o número total de amostras contido no sinal.

8.2. Utilize a função ***sum*** do numPy para o cálculo de E.
8.3. Para evitar overflows nos cálculos poderá converter os seus dados para float usando, por exemplo, o método astype (type) do numPy.

Exemplo:

```
D=data.astype(np.float32)
```

9. Elabore uma função que substitua o canal direito do sinal original por outro sinal de áudio (guitar.wav). Assuma que os dois sinais têm a mesma frequência de amostragem.
A função deverá receber o sinal original, o nome do ficheiro do novo sinal, e o instante de tempo em que deverá começar a reprodução do mesmo.
A função deverá devolver uma matriz com duas colunas, onde a primeira corresponde ao canal esquerdo do sinal original, e a segunda ao novo sinal (caso o sinal seja estéreo deverá utilizar o canal esquerdo do mesmo).

Sugestão 1: adicione zeros ao início do novo sinal, de forma a deslocá-lo para o instante desejado;

Sugestão 2: garanta que a dimensão dos dois canais é a mesma, através da adição de zeros ao final do menor deles.

9.1. Reproduza o sinal estéreo resultante. Verifique que o som de cada sinal é enviado por uma coluna diferente.
9.2. Visualize o novo sinal recorrendo à função implementada no ponto 5.
9.3. Adicione agora os sinais do canal esquerdo e direito, a fim de obter um sinal mono (uma única coluna).
9.4. Escute o resultado e verifique que o som de ambas as colunas é igual.
9.5. Visualize o novo sinal recorrendo à função anteriormente implementada.

10. Implemente uma função que devolva o contorno de amplitude de um sinal, **CA=contornoAmplitude(data, W)**, onde W (ímpar) determina a dimensão de uma janela deslizante necessária para a determinação do contorno. Aqui será implementado um método simples para o seu cálculo, como descrito a seguir:

10.1. O cálculo do contorno começa por uma operação designada por "retificação de meia-onda", definida segundo a equação:

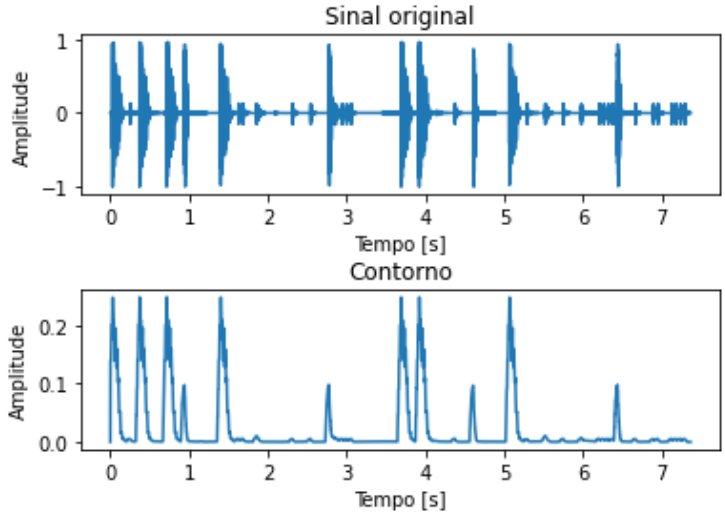
$$x_r[t_i] = \begin{cases} 0, & x[t_i] < 0 \\ x[t_i], & x[t_i] \geq 0 \end{cases}$$

Isto pode ser implementado de forma simples com uma operação do tipo: **x[x<0]=0**

10.2. De seguida, o contorno é determinado pelo cálculo da média móvel do sinal. Para isto, deverá implementar um ciclo for, onde em cada iteração calcula-se a média para cada posição da janela. Para isto, pode utilizar uma sentença do tipo:

```
CA[i] = np.mean(xr[i-np.floor(W/2) : i + np.floor(W/2)])
```

O resultado deverá ser semelhante com:




10.3. Visualize o efeito de aumentar ou reduzir o tamanho da janela.

# Trabalho Prático 0 – Parte B

## Elaboração de um conjunto de scripts e funções para manipulação de imagem

1. Crie um script e grave-o com o nome **mainImage.py**. Este script será utilizado na chamada de todas as funções indicadas a seguir.

2. Leia a imagem contida no ficheiro **polarbear.jpg**. Para tal, utilize a função ***imread*** do módulo matplotlib.image, seguindo o exemplo seguinte:

```
import matplotlib.image as mpimg

img=mpimg.imread(file)
```

   Para imagens monocromáticas (i.e., imagens em escala de cinza), o resultado será uma matriz 2D de dimensões WxH, em que W e H são o número de pixéis correspondentes à largura e à altura da imagem, respetivamente.

   Em imagens policromáticas (i.e., imagens coloridas) que utilizam o modelo de cores RGB, o resultado será uma matriz 3D de dimensão WxHx3, em que cada plano (terceira dimensão) representam separadamente o mapa de intensidades para as cores vermelhas (R, *red*), verde (G, *green*) e azul (B, *blue*), as quais sobrepostas formam todas as cores do espectro.

   Os possíveis valores de intensidade em cada plano irão depender do número de bits utilizados na gravação. Os valores de intensidade em cada plano irão variar entre 0 e np.max(img).

3. Visualize a sua matriz com recurso à função ***imshow*** do *matplotlib.pyplot*. Obterá um resultado como:

Professor T: Paulo Carvalho, Marco Simões
Professores PLs: Paulo Carvalho, Marco Simões, Francisco Antunes

Para retirar os valores dos eixos da figura, utilize a função ***axis('off')*** do *matplotlib.pyplot*.

4. Crie uma função **enhanceColor(img, fator, canal)** que realce a componente azul da imagem. Para isso, multiplique o plano de imagem relativo a esta componente (i.e., **img(:, :, canal)**) por um fator maior que um.

   A imagem abaixo foi gerada com um fator de 1.8.

   Antes de multiplicar pelo fator, transforme os dados para o tipo float32 (**np.float32**). Após aplicar a transformação, converta novamente para uint8 (**np.uint8**).

   Valores superiores a 255 deverão ser limitados a este valor máximo. Para isto pode utilizar a função ***clip***.

   Na implementação desta função não utilize ciclos.

5. Implemente uma função **imagemMosaico(img, W)**, que gere o efeito mosaico na imagem. Como argumento de entrada, deve ser indicada a largura (W=ímpar) do mosaico.

   Nota 1: Para gerar o efeito mosaico, em cada linha e coluna, os pixels no intervalo [j – floor(W/2), j + floor(W/2)] receberão todos a intensidade do pixel j. Isto deve ser aplicado a cada um dos planos R, G e B.

   Nota 2: Os mosaicos na borda da imagem podem ter dimensão inferior a W, e deverão ser tratados convenientemente.

   Nota 3: Utilize ciclos ***for*** (*nested*) para a implementação desta função.

   5.1. Visualize o resultado para diferentes valores de W.

A figura seguinte apresenta os resultados para W = 12. Nesta figura, a última coluna de mosaicos é mais estreita que as restantes.

6. Crie uma função **color2gray(imgB)**, para conversão da imagem para níveis de cinzento. Para esse efeito, deverá calcular a luminância monocromática pela combinação dos valores RGB de acordo com o standard NTSC:

$$Y[i,j] = 0.2978 \cdot R[i,j] + 0.5870 \cdot G[i,j] + 0.1140 \cdot B[i,j]$$

onde [i,j] representa cada pixel da imagem.

Nota: Como imagem de entrada, utilize a imagem com realce da componente azul obtida no ponto 4.

O resultado, apresentado na seguinte figura, será uma matriz WxH, com valores entre 0 e 255.

7. Crie uma função **imagemBin(imgG, limiar)** que binarize a imagem obtida no ponto anterior. Para tal, intensidades superiores ou iguais a um determinado limiar, indicado como argumento de entrada, devem ser convertidas para branco (i.e., 255), enquanto intensidades inferiores ao limiar devem ser convertidas para preto (i.e., 0).

   Nota: não utilize ciclos.

   7.1. Observe o efeito de mudar o limiar.

   A figura abaixo ilustra os resultados para um limiar de 85.

8. Implemente uma função **imagemCont(imgBin)** que receba a imagem binarizada obtida no ponto anterior, e devolva outra imagem evidenciando os

contornos. Para isto, transições em pixels sucessivos de pretoàbranco ou de brancoàpreto deverão receber intensidades iguais a 255, enquanto aos restantes pixels deverá ser atribuído o valor 0.

A figura abaixo ilustra o resultado obtido a partir da imagem apresentada no ponto 7.

9. Grave esta imagem em formato ***bmp*** utilizando a função ***imsave*** da package matplotlib.image.

   Verifique que a dimensão do ficheiro correspondente à imagem final, apesar de ser uma imagem binária, é substancialmente superior à do ficheiro inicial em formato jpeg. De facto, jpeg é um formato de compressão (destrutiva) que permite reduzir de forma considerável o tamanho dos ficheiros de imagem.