Manipulação de Áudio no MATLAB

Carlos Eduardo Leal de Castro

0.1 Audio Signals

Neste trabalho, utilizaremos o MATLAB como plataforma principal para manipularmos nossa pesquisa. No MATLAB, trabalharemos com o uso de vetores de números reais e com sinais no tempo-espaço. Esse termo nos diz que, embora o tempo ocorra continuamente na natureza, no mundo digital só podemos trabalhar com amostras desse tempo do mundo real. Esse processo é chamado de *"sampling"* e é o começo do processo de criação digital.

Para ilustrar, tomando o primeiro sample como sendo $0s$, o segundo como sendo $0.001s$, o terceiro como sendo $0.002s$, e assim por diante, obtemos que o período do sample é de $T_s = 0.001s$. Assim, temos que um sample é descrito a cada T_s segundos. A *Frequência* deste período é descrita como $F_s = \frac{1}{T_s} = \frac{1}{0.001} = 1000$ Hz (Hertz), o que nos diz que 1000 samples de sons do mundo real serão executados em 1 segundo.

0.1.1 Criando Sons sintéticos

Nesta seção, iremos criar um som sintético para começar os estudos. Mais especificamente, três sinais senoides (tons) de diferentes frequências serão geradas sinteticamente e a sua soma será computada e exibida.

Segue o script:

```
% somsintetico1.m

Fs = 15999; Ts = 1/Fs; %Frequência e período que iremos trabalhar
time = 0:Ts:0.1; %Definimos aqui o tempo em que o sample vai agir

Freqs = [421.875 562.5 800.9]; %Definimos aqui as frequências dos sinais
Xs = zeros(length(Freqs),length(time)); %Criamos uma matriz com zeros
                                        %em todas as entradas e com
                                        %tamanhos respectivos ao vetor de
                                        % frequências trabalhadas
                                        % e com o vetor de tempo (time).

for i = 1:length(Freqs) %Criamos aqui um sinal de áudio por frequencia
    Xs(i,:) = cos(2*pi*Freqs(i)*time);
end

x = sum(Xs);                              %A soma final dos tons
x = x ./ max(abs(x));                    %Normalizando a soma final

figure; plot(time,x); axis([0 time(end) -1 1]); %Plotando a soma (x)
xlabel('Tempo (seg)'); ylabel('Sinal ampl.');
title('Um sinal de audio simples')
sound(x)    %função que toca o vetor como som
```

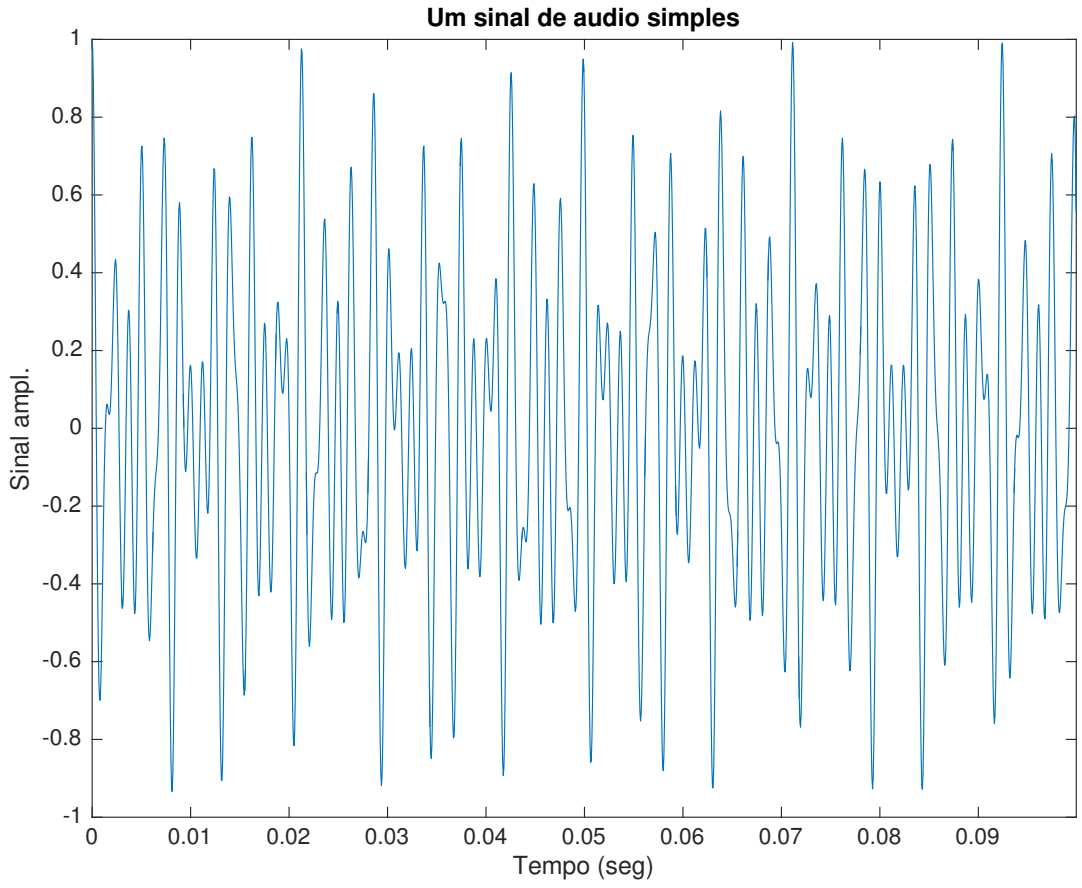


Figura 0.1.1 – Gráfico plotado pelo MATLAB ao executarmos o script acima.

Feito isso, criamos um vetor $Freqs$ 1×3 com as frequências dos sinais que iremos

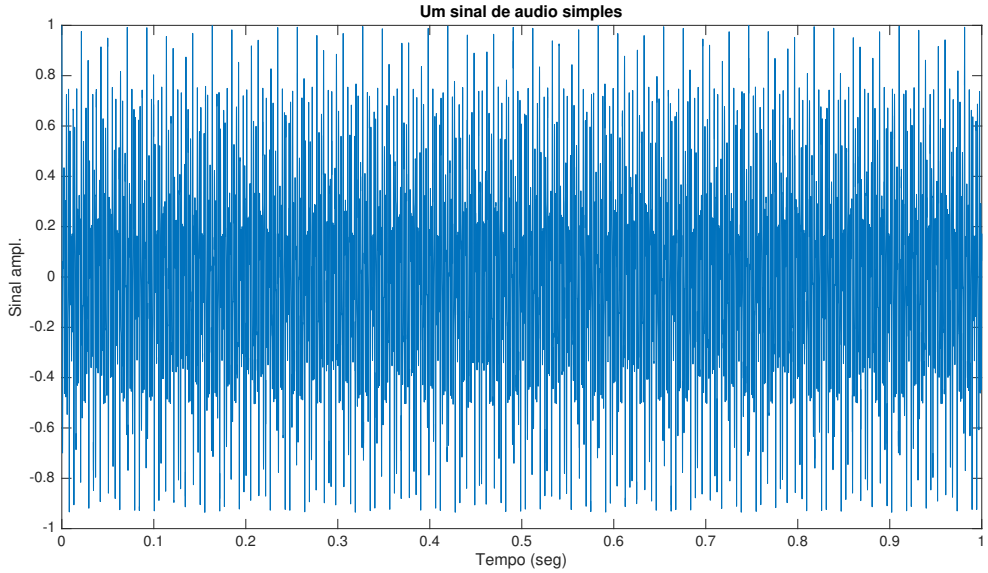
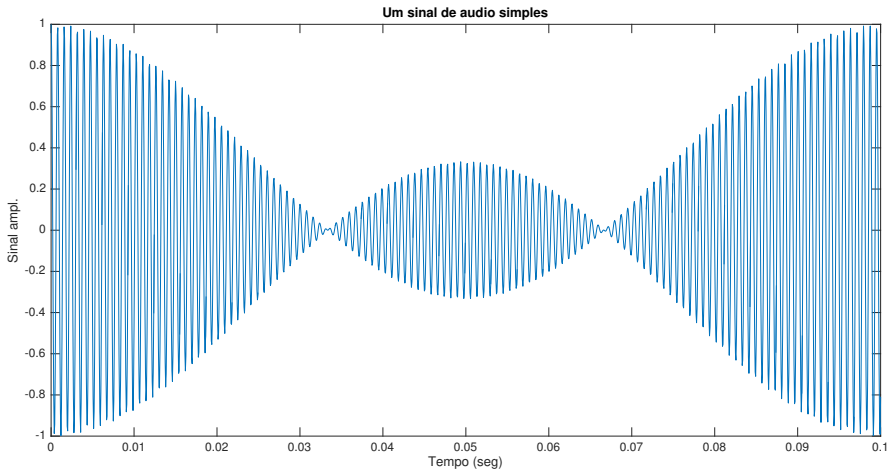


Figura 0.1.2 – Gráfico plotado pelo MATLAB ao executarmos o script acima, alterando o tempo total para 1s de execução do som.

criar. Se modificarmos os valores das entradas deste vetor, alteramos a tonalidade do som emitido pelo programa. Por exemplo, o vetor Freqs com os valores de entrada $Freqs = [421.875 \quad 562.5 \quad 800.9]$ (esses valores foram escolhidos para formarem uma combinação das frequências das notas Lá, Ré e Sol#, respectivamente), obtemos uma soma de três frequências, uma aguda, uma média aguda e uma grave. Ao alterarmos para $Freqs = [421.875 \quad 562.5 \quad 800.9 \quad 843.75]$, adicionamos a esse som uma frequência ainda mais aguda ao que já apresentava.

Ao alterarmos novamente o vetor Freqs com valores de entrada bem próximos uns dos outros, o som emitido tem aspecto de "vibrato", com o seguinte gráfico:



Definidas as frequências desejadas, criamos uma matriz Xs com zeros em todas

as entradas e com dimensão definida pelo vetor Freqs e pelo vetor time (neste caso, a matriz tem dimensão 3×16000). A partir dele, em cada entrada será calculado o valor do sinal por frequência. Este valor é calculado usando a expressão $\cos(2\pi \cdot Fs \cdot t)$. Nesta expressão, cada linha da matriz Xs utilizará o valor da entrada correspondente no vetor $Freqs$, representada na expressão acima por Fs.

Por fim, ao encontrarmos a matriz Xs, vamos somar todas as entradas de Xs criando um vetor denominado "x", com dimensão 1×16000 e normalizando-o para restringir no intervalo $[-1, 1]$. Em seguida, plotamos em um gráfico de amplitude do sinal por tempo de execução.

O MATLAB nos fornece uma função que se propõe a executar um sinal sonoro a partir de um vetor de entrada dado. Esta função é executada a partir do comando *sound(x,fs,nbits)*, tal que, de acordo com o próprio programa, a entrada "x" deve ser o vetor que está trabalhando, "fs" é a frequência a que se propõe trabalhar e "nbits" é a resolução do som que estamos produzindo, e que a maioria das plataformas suporta 8 e 16 Bits. O vetor "x" pode ter formato de matriz n $\times$ 2. Desse modo, o MATLAB irá produzir um som estéreo.

No script anterior, trabalhamos com um som mono. A seguir, criaremos um script que irá produzir um som com diferentes frequências em cada lado (esquerdo e direito do fone de ouvido/caixa de som).

```
1 -    Fs = 15999; Ts = 1/Fs; %Frequência e Período do Sampling
2 -    time = 0:Ts:0.1; %Tempo de execução do som
3
4 -    Freqs = [421.875  800.9]; %Frequências trabalhadas
5
6 -       xLeft = cos(2*pi*Freqs(1)*time)'; %Canal esquerdo
7 -       xRight = cos(2*pi*Freqs(2)*time)'; %Cabal direito
8
9 -    x = [xLeft xRight]; %Matriz 1600x2
10
11     %plotando
12 -   subplot(2,1,1); plot(time,x(:,1));
13 -   xlabel('Tempo (s)'); title('Canal esquerdo');
14 -   subplot(2,1,2); plot(time,x(:,2));
15 -   xlabel('Tempo (s)'); title('Canal direito');
16 -   sound(x);
```

O script acima, além de produzir um som diferente para cada lado do reprodutor de áudio, produziu os seguintes gráficos:

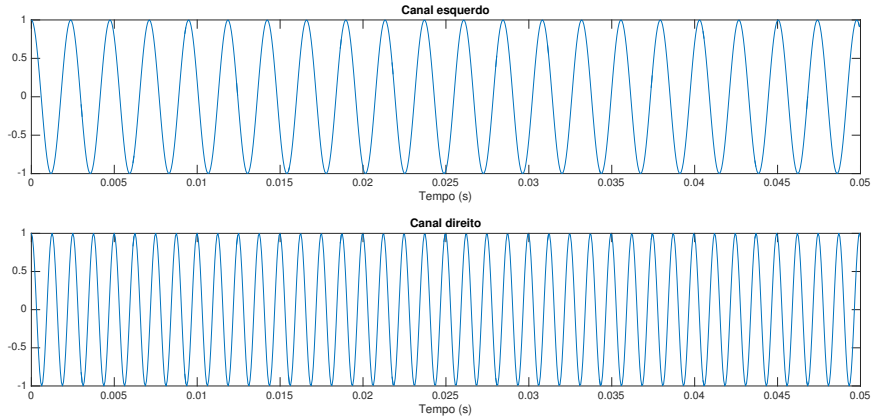


Figura 0.1.3 – Gráfico plotado pelo MATLAB ao executarmos o script acima, alterando o tempo total para 0.05s de execução do som.

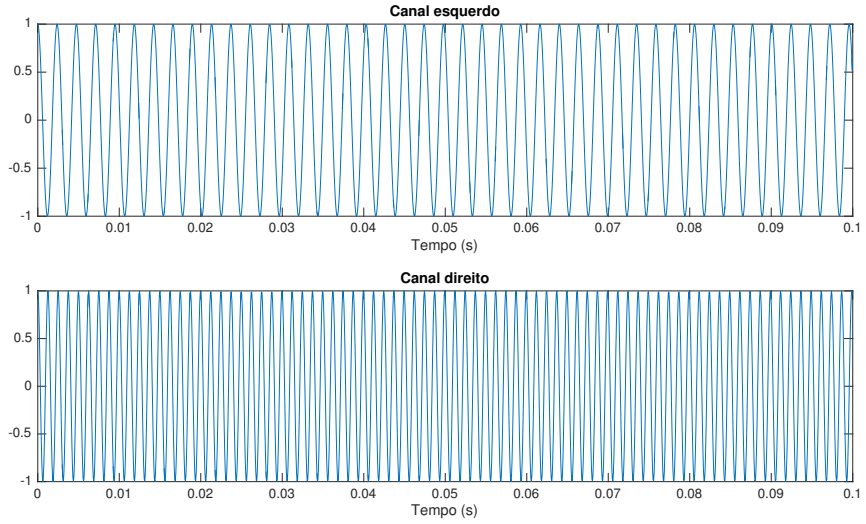


Figura 0.1.4 – Gráfico plotado pelo MATLAB ao executarmos o script acima, alterando o tempo total para 0.1s de execução do som.

De forma muito parecida com o script anterior, trabalharemos com uma frequência de $Fs = 15999$ Hz e um período de $Ts = \dfrac{1}{Fs}$, e com um vetor tempo variando de $0s$ a $0.1s$. Em seguida, criamos os vetores $xLeft = \cos(2\pi \cdot Freqs(1) \cdot t)$ e $xRight = \cos(2\pi \cdot Freqs(2) \cdot t)$. Em cada um desses vetores, utilizamos um valor de entrada do vetor Freqs para determinar o som com que iremos trabalhar (optamos por informar seus valores na forma de um vetor). Esses dois vetores irão ser inseridos em cada coluna respectiva da matriz "x" e, em seguida, plotamos a matriz "x" e pedimos para o MATLAB executar o som (com a função *sound(x)*).

0.1.2 Criando/lendo arquivos de Áudio

Os arquivos de áudio podem ser escritos em diversos formatos. Os mais comuns a serem utilizados são os arquivos .wav, que são arquivos WAVE sem compressão, .mp3 (MPEG3) com uma compressão Lossy (com muita perda de dados ao comprimir) e arquivos .flac (Free LossLess Audio Compress) com maior qualidade e compressão sem perdas significativas. O MATLAB trabalha com os três arquivos citados anteriormente e com arquivos com extensão .ogg, .mp4 e m4a (os dois últimos são arquivos de áudio e vídeo).

O MATLAB consegue facilmente ler esses arquivos usando o comando *[Y,FS] = audioread('Nome do arquivo')*. Com ele, você fará com que o programa leia o arquivo de áudio, que deverá estar previamente na pasta de scripts do MATLAB, retornará um vetor (caso o arquivo seja mono) ou uma matriz (caso o arquivo seja estéreo) na variável "Y" e lhe informará a frequência com que o arquivo foi criado, na variável "FS".

Para criarmos um arquivo de áudio do MATLAB, basta usarmos a função ***audiowrite('Nome.formato', x,FS)***, cujos valores de entrada são, respectivamente, o nome desejado para o arquivo, seguido da sua extensão, o vetor no qual o sinal do áudio está inscrito e a frequência desejada para o arquivo.

O MATLAB trabalha com as extensões WAVE(.wav) sem uso de compressão, MPEG-4 Audio(.m4a,.mp4), cujo método de compressão é "AAC", Free Lossless Audio Compression (.flac), com compressão Lossless e Ogg/Vorbis(.ogg,.oga), com método de compressão Vorbis.

Para arquivos Free Lossless (.flac), o MATLAB também guarda outras informações no arquivo, como Título, Comentários e Artista. Para guardar essas informações no arquivo, deve-se adicionar os dados da seguinte maneira: ***audiowrite('Nome.formato', x,FS,'Nome1','Valor1','Nome2','Valor2',...)***, no qual o Nome é a informação que você quer adicionar (Título, Artista ou Comentário) e Valor é o texto pretendido.

Além disso, pode-se adicionar outras informações no arquivo, como "BitsPerSample" (para arquivos .wav e .flac) e "BitRate" (para arquivos .m4a e .mp4), usado para determinar o número de kilobites por segundo. Essa informação é importante para determinar o "grau" de compressão do arquivo.

Podemos também obter informações sobre um arquivo de áudio usando a função ***audioinfo('Nome.formato')***. Essa função nos informará o método de compressão do arquivo, número de canais (mono ou estéreo), SampleRate, TotalSamples, Duração, BitsPerSample e, no caso de arquivos .flac, Título, Comentários e Artista.

A leitura de arquivos de áudio muito longos pelo MATLAB pode exigir processamento impraticável [**?**]. De acordo com o mesmo autor, um arquivo .wav contendo uma música estéreo de 3 minutos e com frequência $44100Hz$ (qualidade de CD), é lida

utilizando, aproximadamente, 200mb de memória do computador (considerando o padrão de 8 bytes).

Ao tomarmos a leitura do arquivo "elo.flac" com duração de 4 minutos e 55 segundos, e pedirmos para o MATLAB ler este arquivo, usando a função $[a, fs] = audioread('elo.flac')$, ele nos retorna uma matriz 14029056×2 e a frequência $fs = 47500$.

0.1.3 Gravando áudio

O Matlab consegue captar sons utilizando dispositivos disponíveis no computador do usuário, a partir da função *audiorecord*. Essa função cria o objeto de gravação. Nela podemos adicinar informações como frequência, número de bits utilizados, número de canais utilizados (mono ou estéreo) e dispositivo a ser utilizado. Respectivamente:

$$recObj = audiorecord(Fs,nBits,nCanais,ID).$$

Criado o objeto de gravação, o MATLAB dispõe de funções auxiliares na gravação. Entre elas, podemos listar as funções *record(recObj)*, que inicia as gravações no objeto, *recordblocking(recObj,tempo)* para iniciar uma gravação com um tempo determinado, *pause(recObj)* para pausar a gravação caso esteja em andamento, sem finalizar, *play(recObj)*, na qual toca o som gravado no objeto, *resume(recObj)* tem a função de continuar uma gravação que foi pausada anteriormente, *stop(recObj)* para parar a gravação.

Podemos também adicionar certas propriedades às gravações usando a função *set(recObj,'Prop1','ValorProp1',...)*. Entre elas, temos *Userdata*, para adicionarmos qualquer tipo de informação ao arquivo, *Tag* para adicionar tags de informações do arquivo.

0.1.4 Processamento de sinais de áudio por blocos (Short-Term)

Ao trabalharmos com arquivos de áudio, podemos de alguma maneira preferir, ou precisar, usar apenas uma parte do áudio completo. Podemos, por exemplo, estar analisando um áudio contendo uma conversa entre duas pessoas e, de repente, ocorrer uma perturbação no som (pode ser um tiro, ou algo quebrando), de modo que altere o sinal a ser analisado. Nesse caso, para obtermos uma leitura do áudio da conversa de forma precisa, precisamos quebrar a gravação em pequenos segmentos para computar os valores de forma separada [**?**].

Visto isso, seja $x(n)$ um sinal de áudio e $n = 0, 1, 2, \ldots N-1$ a duração do sample (no caso N samples). Focaremos a análise em pequenos espaços de tempo do áudio. Para isso, no script que mostraremos a seguir, criaremos uma função box, definida da seguinte maneira:

$$W : \mathbb{R} \to \mathbb{R},\ W(x) = \begin{cases} 1 & \text{se x} \in [n, m] \\ 0 & \text{se x} \in \mathbb{R} \backslash [n, m] \end{cases}$$

em que n é o tempo inicial e m o tempo final (em segundos) desejados para a limitação do áudio.

Definido o intervalo de tempo desejado, a função Box irá transformar em nulo os valores do sinal que estão fora do intervalo e manter iguais os valores dentro do intervalo. Em outras palavras, criaremos um vetor W nas dimensões do sinal, com valores binários definidos no intervalo $[n, m]$. Em seguida, multiplicaremos este vetor W com o vetor do sinal, mantendo os valores no intervalo indicado e anulando todos os demais valores.

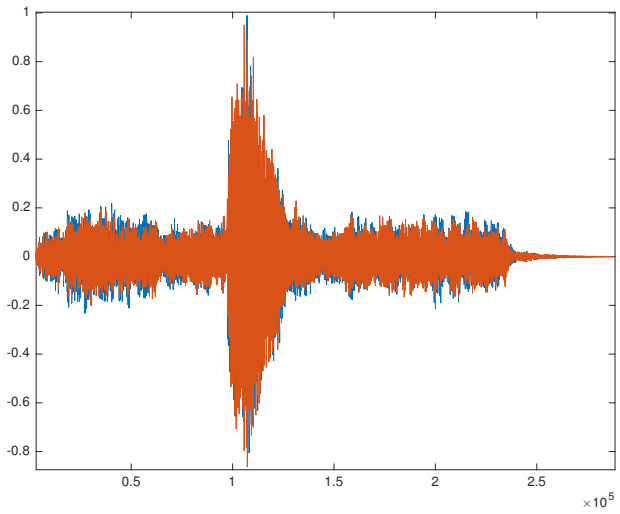
Segue script:

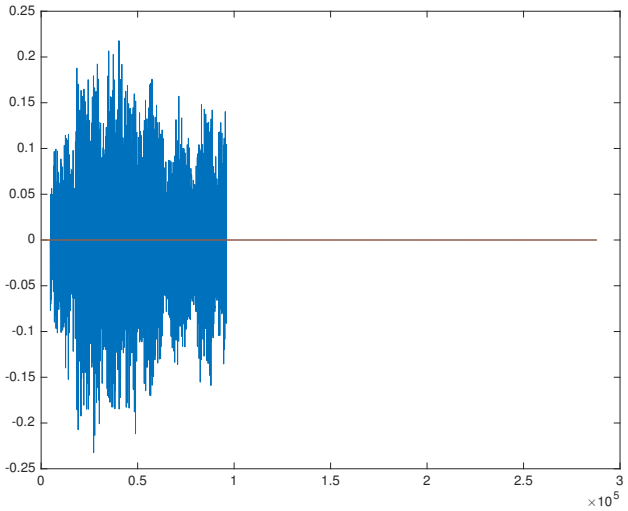
```
function ripaudio(t1,t2,audiofile)
%>> ripaudio(n,m,audiofile)
%Função utilizada para fazer um recorte do som a ser estudado. A variável
%"t1" indicará o valor inicial, em segundos, do tempo desejado. A variável
%"t2" indicará o valor final, em segundos, do tempo desejado.
%Importante lembrar que, na variável "audiofile", é necessário colocar ' no
%início e no fim, além da extensão do arquivo (.wav,.mp3,.flac, etc)

[x,fss] = audioread(audiofile);
t1 = t1*fss; %transforma o tempo inicial em frames;
t2 = t2*fss; %transforma o tempo final em frames.
W = zeros(length(x),2);
for i = t1:t2
    W(i) = 1;
end

x1 = x.*W;

plot(x1)
sound(x1,fss)
end
```

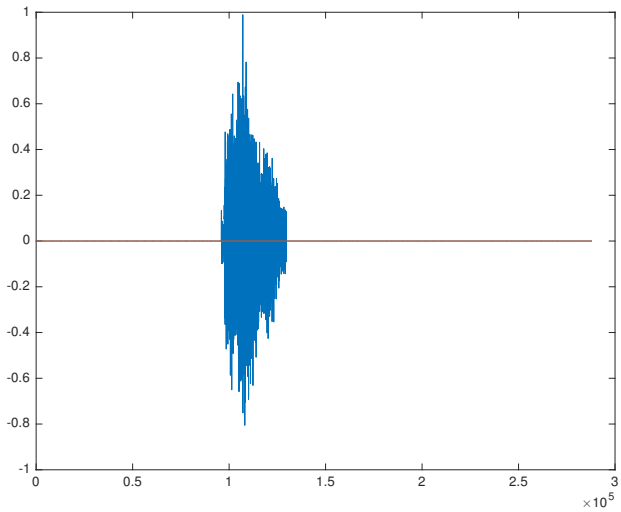
Neste script, é criada uma função em que são dados três valores: no primeiro, "t1", é pedido o valor de tempo inicial para o recorte do sinal de áudio; na variável "t2", é pedido o valor de tempo final para o corte (esses tempos são pedidos em segundos); na terceira variável, pede-se o nome e formato do arquivo de áudio, que deve estar em .mp3, .flac, .wav, entre outros. É necessário que se digite o nome do arquivo entre aspas simples, para que o MATLAB identifique o arquivo externo, e não uma variável.

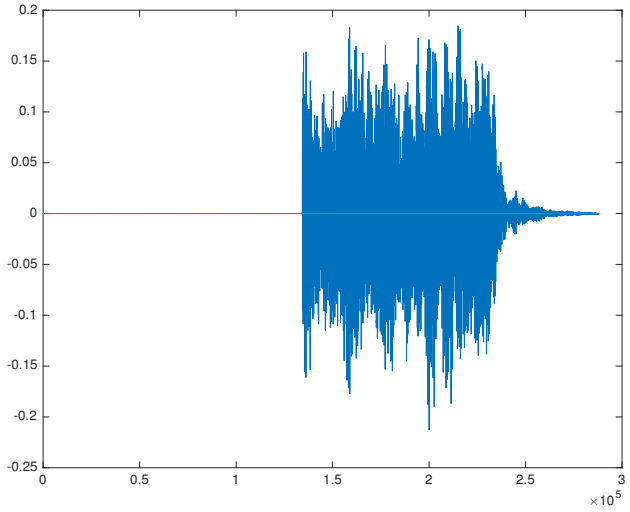
Em seguida, a função cria o vetor do sinal de áudio e a frequência em que este arquivo foi gravado. Com os tempos determinados por quem executa o programa, a função transforma de segundos para samples, com a fórmmula $t1 = t1\dot{f}ss$, onde fss é a frequência, em samples por segundo. A partir do momento em samples, calculado anteriormente, a função determina o intervalo em que a função box W deve agir. Por fim, a função cria novamente um outro vetor x_1, do mesmo arquivo de áudio, porém restrito ao momento desejado.

Seguem os gráficos dos testes executados com este script:


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0.5
1
1.5
2
2.5
×10^5


0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
0
0.5
1
1.5
2
2.5
3
×10^5


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.5
1
1.5
2
2.5
3
×10^5



Percebe-se com as imagens que o algoritmo seleciona a faixa do som desejada, definindo a faixa de tempo em que se quer trabalhar e indicando ao executar a função.

O MATLAB adotou o pacote FFTW (disponível em http://www.fftw.org/) para implementar a Transformada de Fourier. A função *fft(x)* desse pacote computa a DFT de um vetor x dado. Por definição, o tamanho do vetor do sinal de áudio é o mesmo do vetor gerado pela DFT. A função que computa a Transformada Discreta Inversa de Fourier é a função $ifft(x)$.

Como exemplo, tome o vetor $x = [-1 \;\; 2 \;\; 0 \;\; -1 \;\; 1 \;\; 2 \;\; 3 \;\; 2 \;\; 1]$ e, ao aplicarmos a função $fft(x)$, geramos o vetor $X = [9.0 \;\; -2.17+5.13i \;\; -1.06-4.41i \;\; -3.0-1.73i \;\; -2.77+0.85i] \;\; -3.0+1.73i \;\; -1.06+4.41i \;\; -2.17-5.13i]$ com os coeficientes de Fourier. Ao aplicarmos a função da inversa, obtemos novamente o vetor x.

Uma outra função no MATLAB que usaremos para aplicar DFT em sinais é a função *fftshift()*. Ela reorganiza o vetor de saída colocando os zeros do vetor inicial no centro do vetor ou matriz (depende do sinal de entrada).

0.1.5 Transformada de Fourier Short-Time

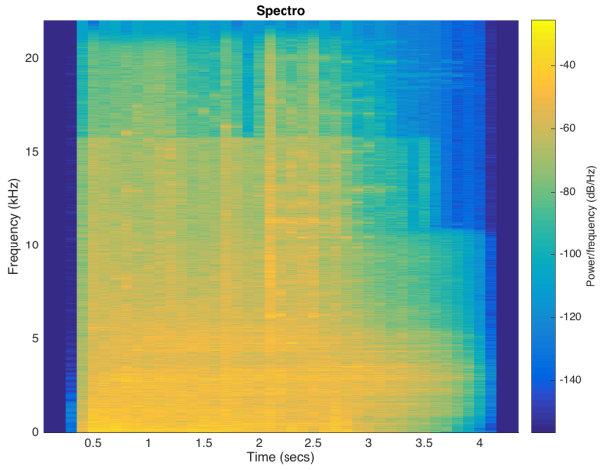
De acordo com [**?**], o objetivo da Transformada de Fourier Short-Time (Short Time Fourier Transform - STFT) é a de quebrar o sinal com que se está trabalhando em frames sobrepostos, usando a função window, e computando a DFT de cada termo. O tamanho tal que a função window dividirá o sinal é importante pois definirá a qualidade da resolução, em termos de frequência, do gerado pela DFT. Ainda de acordo com [**?**], quanto mais longo o recorte do sinal (com o auxílio da função window), maior é a resolução de frequência, enquanto com pouca acuidade em função do tempo. Por outro lado, janelas mais curtas que fornecem informações mais detalhadas com relação ao tempo levam a uma resolução de frequência ruim.

Se os coeficientes da DFT de cada frame são colocados em colunas separadas de uma matriz, a STFT poderá ser representada como uma matriz de coeficientes, em que o índice de coluna representa o tempo e o índice das linhas representará o respectivo coeficiente da DFT. A partir daí, podemos representá-la como imagem e poderá ser visualizada. Essa imagem é conhecida como o espectrograma e apresentará a evolução do sinal no domínio do tempo. No MATLAB, usaremos a função *spectrogram()* para implementar a imagem do espectrograma do sinal desejado.

O próximo script que iremos apresentar aqui é uma função criada para ler um arquivo de áudio, em um dos formatos reconhecidos pelo programa e criar seu espectrograma. Segue o script:

```
function spectrog(audio)
%Função que criará a imagem do Spectro do áudio desejado.
%
%audio = Adicione o nome do áudio que deseja estudar, seguido do seu
%formato, entre aspas simples.
%Exemplo: spectrog('som.wav')


[SIGN,Freq] = audioread(audio);

SIGN=sum(SIGN,2); %Transforma o som STEREO em MONO
SIGN = SIGN'; %Aplica a transposta, necessária para a função spectrogram

spectrogram(SIGN,0.2*Freq,0.1*Freq,Freq/2,Freq,'yaxis');
title('Spectro');
sound(SIGN,Freq);
end
```

Ao usarmos a função criada acima em um áudio captado em um acidente de carro (áudio obtido no site http://freesound.org), a função *spectrogram()* nos mostra o espectrograma do arquivo lido, como na imagem a seguir:



Observe que no eixo x da imagem acima podemos ver o domínio do tempo, em segundos, de execução do áudio, e no eixo y, a medida de frequência em função do tempo,

em kHz.

Referência

GIANNAKOPOULOS, T.; PIKRAKIS, A. *Introduction to Audio Analysis: A MATLAB Approach*. Academic Press, 2014.