Análise Harmônica e Sinais Sonoros

MAP3121 - Exercício Programa 2 - 2018

2 de junho de 2018

1 Transformadas de Fourier

Consideremos uma função $F(x)$ 2π-periódica tabelada em $2N$ pontos uniformemente espaçados em $[0, 2\pi]$. Denotemos os pontos por $x_j = j\pi/N, j = 0, 1, .., 2N-1$. Os valores de $F(x_j)$ podem ser reais ou complexos. Na descrição que segue iremos considerar o caso mais geral e caso $F(x)$ seja real, consideramos que sua parte imaginária é nula.

A transformada discreta de Fourier de $F(x)$ associa aos valores $(F(x_0), F(x_1), ..., F(x_{2N-1}))$ os coeficientes de Fourier de F: $(a_{1-N}, a_{2-N}, ..., a_N)$ tais que:

$$F(x_j) = \sum_{k=1-N}^{N} a_k e^{ikx_j} \quad , j = 0, .., 2N-1 \tag{1}$$

No caso em que F é uma função real, os coeficientes a_k e a_{-k} são complexos conjugados ($a_k = \alpha_k + i\beta_k$ e $a_{-k} = \overline{a_k} = \alpha_k - i\beta_k$), de tal forma que

$$a_{-k}e^{-ikx_j} + a_k e^{ikx_j} = 2(\alpha_k \cos(kx_j) - \beta_k \sin(kx_j)) \tag{2}$$

representa o harmônico de ordem k de $F(x)$, com amplitude $A_k = 2\sqrt{\alpha_k^2 + \beta_k^2} = 2|a_k|$. Os coeficientes a_0 e a_N têm parte imaginária nula quando F é real.

O cálculo da transformada de Fourier é feito através da expressão:

$$a_k = \frac{1}{2N} \sum_{j=0}^{2N-1} F(x_j)e^{-ikx_j}, \quad k = 1-N, .., N \tag{3}$$

enquanto que a transformada inversa, através da qual se recupera os valores de $F(x_j)$ a partir dos coeficientes a_k, é dada por (1).

Note que se definirmos o produto interno complexo $< f, g >= \frac{1}{2N}\sum_{j=0}^{2N-1} f(x_j)\overline{g(x_j)}$, temos que as funções e^{ikx}, $k = 1-N, N$ formam um conjunto ortonormal em relação a ele e que $a_k =< F(x), e^{ikx} >$.

Observação: Note que

$$e^{-ikx_j} = e^{-ikj\pi/N} = e^{i2\pi j}e^{-ikj\pi/N} = e^{i(2N-k)j\pi/N} = e^{i(2N-k)x_j} \tag{4}$$

e portanto podemos alternativamente escrever as transformadas (1) e (3) como

$$F(x_j) = \sum_{k=0}^{2N-1} c_k e^{ikx_j} \quad , j = 0, .., 2N-1 \tag{5}$$

e

$$c_k = \frac{1}{2N} \sum_{j=0}^{2N-1} F(x_j)e^{-ikx_j}, \quad k = 0, ..., 2N-1 \tag{6}$$

com $c_k = a_k$, $k = 0, 1, .., N$ e $c_{2N-k} = a_{-k}, k = 1, ..., N-1$.

2 Transformada Rápida de Fourier (FFT)

Note que o cálculo das transformadas (direta) de Fourier (5) e sua inversa (6) requerem cada um a avaliação de $2N$ somas, cada uma com $2N$ elementos. Assim, o cálculo destas transformadas requer $O(N^2)$ operações, se feito diretamente usando estas expressões. Em 1965, Cooley e Tukey (Cooley, James W.; Tukey, John W. (1965), An algorithm for the machine calculation of complex Fourier series, Math. Comput. 19, 297-301) propuseram um algoritmo, mostrando como calcular transformadas de Fourier com apenas $O(N \log N)$ operações, trazendo grande progresso a várias aplicações computacionais. Embora as idéias envolvidas no algoritmo remontem a Gauss (veja por exemplo Heideman, M. T., D. H. Johnson, and C. S. Burrusi (1984), Gauss and the history of the fast Fourier transform, IEEE ASSP Magazine 1, (4), 14-21), foi apenas após o trabalho de Cooley e Tukey que o método realmente se efetivou.

A efetividade do algoritmo depende de $2N$ ser fatorável em vários fatores primos. Vamos aqui nos restringir ao caso em N é uma potência de 2, embora haja diversas implementações muito eficientes para transformadas de comprimento do tipo $2^l 3^m 5^n 7^p$. Vamos descrever o método para a transformada inversa (5) no caso em que $N = 2^l$. Temos que:

$$\begin{aligned} F(x_j) &= \sum_{k=0}^{2N-1} c_k e^{ikx_j} \\ &= \sum_{k=0}^{N-1} c_{2k} e^{i2k\pi j/N} + \sum_{k=0}^{N-1} c_{2k+1} e^{i(2k+1)\pi j/N} \\ &= \sum_{k=0}^{2(N/2)-1} c_{2k} e^{ik\pi j/(N/2)} + e^{i\pi j/N} \sum_{k=0}^{2(N/2)-1} c_{2k+1} e^{ik\pi j/(N/2)} \\ &= Even_j + e^{i\pi j/N} Odd_j \ , \ j = 1, .., N \end{aligned} \tag{7}$$

e

$$\begin{aligned} F(x_{j+N}) &= \sum_{k=0}^{2N-1} c_k e^{ikx_(j+N)} \\ &= \sum_{k=0}^{N-1} c_{2k} e^{i2k\pi (j+N)/N} + \sum_{k=0}^{N-1} c_{2k+1} e^{i(2k+1)\pi (j+N)/N} \\ &= \sum_{k=0}^{2(N/2)-1} c_{2k} e^{ik\pi j/(N/2)} + e^{i\pi (j+N)/N} \sum_{k=0}^{2(N/2)-1} c_{2k+1} e^{ik\pi j/(N/2)} \\ &= Even_j - e^{i\pi j/N} Odd_j \quad , \ j = 1, .., N. \end{aligned} \tag{8}$$

Vemos então que as duas somas (cada uma com 2N termos) para a avaliação de $F(x_j)$ e $F(x_{j+N})$ requerem apenas a avaliação de duas somas ($Even_j$ e Odd_j), cada uma com N termos (e mais duas adições e uma multiplicação). Note ainda que estas duas somas são do mesmo tipo que a inicial e se N é par, podem novamente ser reduzidas a somas com metade dos termos. A aplicação recursiva desta ideia, leva à transformada rápida, com $O(N \log N)$ operações (onde $\log N$ é o número de etapas em que se reduz o tamanho das somas pela metade).

De mesma maneira pode-se calcular a transformada direta (6) (deixando a divisão por $2N$ para o final):

$$\begin{aligned} c_k &= \sum_{j=0}^{2N-1} F(x_j) e^{-ik\pi j/N}, \quad k = 0, ..., 2N-1 \\ &= \sum_{j=0}^{N-1} F(x_{2j}) e^{-ik\pi j/(N/2)} + e^{-ik\pi/N} \sum_{j=0}^{N-1} F(x_{2j+1}) e^{-ik\pi j/(N/2)} \\ &= Even_k + e^{-ik\pi/N} Odd_k \ , \ k = 1, .., N \end{aligned} \tag{9}$$

e

$$c_{k+N} = Even_k - e^{-ik\pi/N} Odd_k \ , \ k = 1, .., N \tag{10}$$

Detalhamos um pouco mais como implementar um procedimento recursivo para o cálculo das transformadas na próxima seção.

FFT recursiva

Um pseudo código para fft recursiva (com $N = 2^n$) é dado por

fftrec(c,f,n,dir)

```
complexos f(0:2n-1),c(0:2n-1),even(0:n-1),odd(0:n-1),fe(0:n-1),fo(0:n-1)
logico dir
Se n=1 então
    c(0) = f(0)+f(1)
    c(1) = f(0)-f(1)
senão
    para j=0,n-1
        fe(j)=f(2j)
        fo(j)=f(2j+1)
    fim do para
    fftrec(even,fe,n/2,dir)
    fftrec(odd ,fo,n/2,dir)
    para j=0,n-1
        se (dir) então
            eij = exp(- i * j * pi / n)
            senão
            eij = exp(i * j * pi / n)
        fim do se
        c(j) = even(j)+eij * odd(j)
        c(j+n) = even(j)-eij * odd(j)
    fim do para
fim do se
```

Observações: As variáveis no código acima são complexas. O mesmo programa serve tanto para a transformada direta como para a transformada inversa de Fourier, sendo cada caso controlado pela variável lógica dir. Se dir é verdadeira, a transformada direta é executada e caso contrário, executa-se a transformada inversa. No caso da transformada direta, ao final da execução ainda deve-se dividir os coeficientes c_k pelo fator de normalização $2N$.

Análise de Fourier e filtragem

Dados os valores de uma função F em pontos uniformemente espaçados usamos as transformadas de Fourier para fazer sua decomposição em harmônicos, correpondentes às diversas frequências. Em nossa descrição sobre a transformada de Fourier, supusemos um período 2π. Sendo o período igual a T, temos apenas que fazer uma mudança de variáveis. Assim a k-ésima frequência associada a um número de onda k é dada por $\omega_k = k/T$. Note que a amplitude do harmônico de ordem k é dada por $A_k = 2|a_k| = 2|c_k| = 2|c_{2N-k}|, k = 1, .., N-1$, quando temos $2N$ pontos amostrados. A N-ésima frequência $\omega_N = N/T$ é a máxima frequência que pode ser resolvida com esse número de pontos, e é chamada de frequência de Nyquist (com amplitude do harmônico igual a $|a_N| = |c_N|$). Calculando a transformada de Fourier de F, podemos então verificar, por exemplo, quais são suas frequências dominantes. Podemos também aplicar uma filtragem seletiva das frequências.

Um filtro consiste de um corte (ou redução) de algumas frequências da função em questão (um sinal ou série temporal, por exemplo). O filtro é do tipo "passa baixa" quando ele deixa passar frequências baixas (ou

seja, as mantém sem alterações) e elimina as frequências altas. A aplicação de um tal filtro zera os coeficientes de frequências mais altas, a partir de uma dada frequência de corte K (ou seja, zeramos os coeficientes dos harmônicos de ordem $k > K$, que são $c_k,\ k = K+1, .., 2N-K-1$). A seguir, através da transformada inversa de Fourier, reconstitui-se a função (filtrada) com as frequências altas eliminadas.

Analogamente, temos filtros "passa alta", em que frequências abaixo de um valor de corte K são zeradas, mantendo as frequências altas inalteradas. O coeficiente c_0 referente ao termo constante não é zerado, caso se deseje a preservação da amplitude média do sinal.

Temos ainda filtros "passa banda", que mantêm apenas as frequências dentro de uma certa faixa $K_1 < k < K_2$, zerando as restantes.

Note que a frequência ω_k (associada ao coeficiente c_k) tem k ciclos completos de oscilação no período completo T. Portanto o período P_k de uma oscilação relativa à frequência ω_k será dado por $P_k = 1/\omega_k = T/k$.

Tarefas iniciais e verificação

Você deverá implementar as transformadas diretas e inversas de Fourier, tanto em sua forma mais direta (e menos eficiente), como descrito nas equações (5) e (6), como através da FFT recursiva. Através dos testes e da aplicação das transformadas posteriormente, você irá comprovar a significativa diferença de eficiência computacional entre as duas versões. Você deverá confrontar ainda a eficência de sua implementação com a de rotinas de um pacote computacional (FFTPACK4: disponível em `http://people.sc.fsu.edu/~jburkardt/c_src/fftpack4/fftpack4.html`. Você irá necessitar as rotinas: ezffti, ezfftf e ezfftb - para inicialização e transformadas diretas e inversas. Estas são rotinas específicas para dados reais. O resultado das transformadas não é diretamente comparável ao das rotinas que você irá implementar, uma vez que ezfftf tem como saída os coeficientes da expansão em $\cos kx$ e $\sin kx$, que devem ser convertidos em c_k (veja as documentações das rotinas e também a equação (2) para entender as relações). Devido à anti-simetria dos coeficientes neste caso, pode-se economizar memória e tempo computacional. Mais ainda, estas implementações não fazem uso de funções recursivas. Apesar de por vezes facilitar a implementação de algoritmos, o uso de funções recursivas costuma ser pouco eficiente em comparação com códigos que acabam por resolver a recursão. Para entender bem isto você pode calcular $n!$ através de uma função recursiva (se $n = 1$, $fat(n) = 1$, senão $fat(n) = n * fat(n-1)$) ou através de um loop ($fatn = 1$, para i de 2 a n faça $fatn = fatn * i$) e comparar. Além disso essas rotinas admitem dados com tamanhos mais gerais que potências de 2. Você deve comparar os tempos computacionais das 3 versões das transformadas que irá testar.

Veja a documentação das rotinas do FFTPACK4 e tome cuidado com a normalização (a divisão pelo comprimento da transformada - $2N$ em nossa descrição), que nem sempre é feita nas rotinas. Você irá notar que os resultados da transformação direta não são normalizados ao final (falta a divisão por $2N$). Note este fato na hora de comparar resultados.

Execute os seguintes testes para validar suas implementações das transformadas.

a) considere a função com os seguintes valores em $0, \pi/2, \pi$ e $3\pi/2$: (5,-1,3,1). O resultado da transformada direta deve ser: $c = (2, 1/2 + i/2, 2, 1/2 - i/2)$. Aplicando a transformada inversa deve-se recuperar os valores da função original. (Use este exemplo para entender bem a saída de ezfft !)

b) considere a função com os seguintes valores $f = (6, 2, 5, 2, 11, 2, 8, 8)$ e compare seus códigos com as rotinas do FFTPACK4.

c) Seja $F(x) = 10sin(x) + 7cos(30x) + 11sin(352x) - 8cos(711x)$. Construa o vetor com os valores $F(x_j)$, com $x_j = j\pi/512, j = 0, 1023$ e calcule a transformada direta. Interprete os valores obtidos. Aplique a transformada inversa para a recuperação de F.

Tratamento de dados incompletos

Nas aplicações você deverá transformar funções que foram medidas em instantes (ou posições) uniformemente espaçados. No entanto, você pode ter que lidar com duas situações:

a) É possível que os dados sejam incompletos, com lacunas no meio. Por exemplo, se são dados diários, pode ser que faltem informações referentes a alguns dias. Neste caso, a tabela deverá ser completada através de interpolação cúbica (use por exemplo o método de Lagrange). Tome dois valores anteriores e dois posteriores mais próximos ao ponto desejado e interpole para a posição desejada. Caso exista apenas um ponto de um dos lados, você pode trocar para interpolação linear neste caso.

b) Caso o número de valores tabelados (após as interpolações para as lacunas) não seja uma potência de 2, use o valor médio destes para completar a tabela até a próxima potência de 2. (se o número de elementos for ligeiramente superior a uma potência de 2, você poderia também descartar os pontos que estão a mais).

3 Decomposição de sons

Neste exercício programa trabalharemos com o conceito de decomposição de sinais, no nosso caso sinais sonoros, em seus vários harmônicos.

Qualquer movimento vibratório de ar na entrada do ouvido corresponde a um tom musical que pode ser exibido como uma soma de movimentos vibratórios simples, correspondendo aos sons parciais desse som. Em uma música, cada frequência apresenta uma amplitude, e a composição destas leva à formação do som.

O objetivo deste EP é decompor sons em suas frequências, analisar as frequências dominantes e aplicar filtros e compressões.

Compressão

Em geral os sons podem ser muito bem representado com poucas frequências, pelo menos para a nossa capacidade auditiva. Por exemplo, um som composto por 10.000 valores amostrados em um segundo, dependendo de sua complexidade, pode ser bem representado talvez com menos que 100 harmônicos. Métodos de compressão de música usam esse pressuposto para armazenamento. Ao invés de armazenar os 10.000 valores, armazenam-se os 100 coeficientes de Fourier, e quando for necessário recompor o som para ouvir a música, aplica-se a transformada inversa de Fourier.

Neste EP vamos analisar taxas de compressão e o erro associado ao processo. Para a taxa de compressão, dado um limiar ϵ, determinamos o número (n_ϵ) de harmônicos com amplitude abaixo deste limiar e definimos a taxa de compressão como $tc = \frac{n_\epsilon}{2N}100\%$, onde $2N$ é o número total de valores amostrados. A verificação da qualidade sonora desta compressão é feita através da reconstrução do som representado apenas pelos harmônicos com amplitude acima do limiar definido, sendo zerados os coeficientes dos harmônicos restantes. Com isso é possível verificar o que foi perdido do som ouvindo-o. Uma métrica numérica para avaliar o efeito da compressão é o erro relativo, definido a seguir.

Sendo f_i, $i = 1, ..., 2N$, o sinal original e $\tilde{f}_i$, o sinal reconstruído, definimos o erro relativo:

$$ER = \max_{i=1,\ldots,2N} |g_i| \tag{11}$$

$$g_i = \begin{cases} \frac{|\tilde{f}_i - f_i|}{|f_i|}, & \text{se } f_i \neq 0; \\ 0, & \text{se } \tilde{f}_i = f_i; \\ 1, & f_i = 0 \neq \tilde{f}_i \end{cases} \tag{12}$$

Naturalmente ao jogarmos fora muitos coeficientes de Fourier reduzimos a complexidade do som, mas esperamos fazer isso sem descartar o essencial. Por exemplo, quando um músico toca a nota *dó* em um piano, enquanto estiver tocando somente esta nota, estará emitindo um som com apenas uma frequência, a de 261.6 Hz. Isso significa que ao fazermos a decomposição do som em série de Fourier haverá apenas um coeficiente, o que refere-se a frequência da nota *dó*.

Dependendo do som, esse tipo de técnica também pode ser usada para filtrar ruídos. Além disso, para comprimir músicas sem perda de qualidade, pode-se eliminar as frequências fora da faixa audível aos seres

humanos inaudíveis dos seres humanos, que fica entre 20 Hz e 20 KHz.

Para converter frequências ϕ em Hertz para coeficientes de Fourier de ordem k, e vice versa, basta tomar $\phi = \frac{k}{T}$, onde T é o tempo máximo de amostragem dado em segundos.

4 Tarefa

Estudaremos sons, ou uma música simples, através da análise de Fourier. A partir da decomposição do som em harmônicos podemos filtrar algumas frequências, removendo graves ou agudos, ou testar a compressão (armazenar o som utilizando menos espaço). Ao final reconstruiremos o som utilizando as técnicas de transformada inversa de Fourier. Ao ouvir o som filtrado com ou sem compressão podemos verificar se o resultado foi satisfatório.

Para fazer a análise de Fourier de sinais sonoros é preciso primeiro convertê-los em uma série de dados. Para isso sugerimos o programa SOX, cujos detalhes seguem em anexo no final do enunciado. Outros softwares podem ser usados, mas há a necessidade se se seguir o padrão do arquivo de dados que será descrito para o SOX. Iremos fornecer os sons tanto em formato ".wav" quanto em formato ".dat" (já convertido para dados), caso você venha a ter problemas para fazer as conversões. Uma vez analisado (aplicando filtros/compressão), o som é recontruído via transformada inversa em dados numéricos. Podemos então usar o SOX, ou outro software de conversão, para podermor ouuvir o som. Essa etapa irá servir para avaliarmos a qualidade do som obtido.

A tarefa consiste nas seguintes etapas:

1. Preparo do arquivo de som
   (a) Obter sons do tipo ".wav". Serão fornecidos alguns sons, mas você pode testar com outros se desejar, incluindo músicas.
   (b) Converter o som para o formato de dados ".dat", seguindo o padrão do software SOX, descrito no anexo.
   (c) Serão fornecidos arquivos de sons já convertidos em dados
2. Entrada do EP:
   (a) Nome do arquivo .dat a ser lido
   (b) Opção da transformada a ser realizada
   (c) Parâmetros de filtro/compresão
3. Transformada:
   (a) Calcular os coeficientes de Fourier da série de dados via transformada discreta de Fourier usual (a "lenta" que você deve implementar).
   (b) Calcular os coeficientes de Fourier via FFT recursiva que você irá implementar.
   (c) Calcular os coeficientes de Fourier usando a FFT do FFTPACK4.
4. Análise do Espectro:
   (a) Faça gráficos dos espectro de Fourier colocando no eixo x a frequência relativo ao coeficiente de Fourier e no eixo y a amplitude do modo, isto é, o módulo do coeficiente.
   (b) Destaque quais são as frequências dominantes para o som analisado.
5. Filtros e Compressão:
   (a) Aplicar filtros de passa baixa e passa alta para percebermos os efeitos de eliminar as frequências altas e baixas.
   (b) Aplicar métodos de compressão nos coeficientes.
6. Transformada Inversa:
   (a) Obter uma nova série de dados (após filtragem e/ou compressão) usando transformada inversa discreta de Fourier, sempre usando o mesmo algorítmo usado na transformada direta.

7. Saída: A série de dados deve ser impressa em um arquivo no padrão que pode ser lido pelo SOX. Você pode então verificar os efeitos das suas modificações no som convertendo os dados em .wav. Você também pode imprimir informações com as frequências dominantes e informações sobre a compressão, como a taxa de compressão e o erro relativo. Imprimir também o tempo de execução do código, para avaliarmos a efiência das diferentes transformadas.

5 Testes

1. Análise:

   Analise as frequências dominantes dos sons fornecidos.

   Analise um dos dois canais do som "PianoSol.wav", ou dos dados de uma coluna de "PianoSol.dat" e verifique quais as frequências são capturadas com a nota Sol dada em um piano.

2. Filtro passa baixa: o que acontece com o som?

   Utilize o arquivo de som "plane.wav", ou os dados "plane.dat" com parâmetro de corte $K = 25000$.

   Utilize o arquivo de som "dog.wav", ou os dados "dog.dat" com parâmetro de corte $K = 8000$.

3. Filtro passa alta: o que acontece com o som?

   Utilize o arquivo de som "plane.wav", ou os dados "plane.dat" com parâmetro de corte $K = 4000$.

   Utilize o arquivo de som "dog.wav", ou os dados "dog.dat" com parâmetro de corte $K = 4000$.

4. Compressão: Seu programa deve imprimir a taxa de compressão para cada caso.

   Utilize o arquivo de som "plane.wav", ou os dados "plane.dat" com parâmetro $\epsilon = 10^{-3}$.

   Utilize o arquivo de som "dog.wav", ou os dados "dog.dat" com parâmetro $\epsilon = 10^{-2}$.

5. Outros desafios:

   Tente eliminar o ruído por trás da voz do exterminador do futuro, contida no arquivo "terminator.wav".

   Investigue o tempo necessário para cada algoritmo transformar e filtrar o som fornecido em "theforce.wav". Será que o seu programa consegue transformar e filtrar a música clássica de Stefano Demicheli e Francesco Geminiani fornecida no arquivo demicheligeminiani.wav?

São fornecidos alguns outros sons que você pode analisar, filtrar e comprimir. Você também pode analisar outros sons que achar interessante. Discuta no relatório os resultados obtidos.

6 Regras do Jogo

As analises e resultados obtidos devem ser organizados em um relatório que deve minimamente discutir os problemas estudados e os resultados obtidos.

O exercício deve ser feito em linguagem C.

O exercício pode ser feito em duplas, mas sempre com alguém da mesma área - Elétrica (não necessariamente da mesma turma).

Apenas **um** aluno deve entregar o exercício, destacando no relatório e código o nome de ambos os alunos.

A entrega deve conter o relatório (em .pdf), contendo a análise do problema estudado, e o código usado para as simulações computacionais (arquivos .c). A entrega deve ser feita em um arquivo compactado único.

O seu código deve estar bem comentado e estruturado. A entrada e saída devem ser feitas de forma a ajudar o usuário a executar o programa e deve facilitar a análise dos resultados. Inclua qualquer arquivo adicional necessário para o seu programa no arquivo compactado a ser entregue. Como o seu programa terá que ler arquivos de entrada, considere que os mesmos encontram-se na mesma pasta do executável, ou solicite o caminho/nome do arquivo ao usuário.

Você deve resolver tanto os exercícios relativos à aplicação, quanto os testes iniciais descritos na seção de Trasformada de Fourier.

A Apêndice: Convertendo Sons em Dados

Sons podem estar em diversos formatos, alguns que armazenam a música já comprimida e outros não. Para podermos fazer a análise harmônica de um som precisamos convertê-lo em uma série de dados. Existem diversas formas de se fazer isso, mostraremos algumas formas neste documento.

SOX

O programa SOX - SOund eXchange é um software livre que roda em Windows, Linux e MacOS X, utilizado para converter diversos tipos de arquivos de músicas e sons para outros tipos, e ainda possibilita a aplicação de efeitos na música. Usaremos a conversão de arquivo do tipo WAV para arquivos de texto.

O site do software é o `sox.sourceforge.net/`. Ele vem pronto para ser usado com arquivos do tipo WAV, se você quiser usá-lo com MP3 é preciso ou converter o MP3 para WAV ou instalar uma biblioteca conforme descrito no site do programa.

Usuários de Windows

Baixe e instale o SOX no seu computador seguindo as instruções do site do SOX. O SOX é um programa para ser usado via linha de comando (`https://pt.wikihow.com/Abrir-o-Prompt-de-Comando-no-Windows`).

Para transformar um arquivo .wav em série de dados (.dat), ou o contrário, faça o seguinte. Primeiro identifique aonde o SOX foi instalado, exemplo `C:\ProgramFiles\sox-14-4-2\sox.exe`. Acesse o diretório que contém os seus dados .wav a serem convertidos via linha de comando (exemplo
`https://medium.com/@adsonrocha/como-abrir-e-navegar-entre-pastas-com-o-prompt-de-comandos-do-windows-1`

De .wav para .dat, digite:

`C:\ProgramFiles\sox-14-4-2\sox.exe` arquivo.wav arquivo.dat

De .dat para .wav, digite:

`C:\ProgramFiles\sox-14-4-2\sox.exe` arquivo.dat arquivo.wav

Usuários de Linux e MacOS

Instale o SOX usando as instruções do site (é simples em sistemas Debian: apt-get install sox). Depois use o comando “sox nome.wav nome.dat” para transformar arquivos de som em dados (texto). E use “sox nome.dat nome.wav” para fazer o inverso.

O arquivo de dados

O arquivo gerado pelo SOX consiste no seguinte:

Na primeira linha deve aparecer "; Sample Rate N", onde N é a taxa de amostragem do som por segundo. Isso é o que vai dar uma idéia de qualidade do som. Quanto maior o número de informações por segundo, mais detalhado é o som.

Na segunda linha deve aparecer "; Channels M ", onde M é o número de canais no som. Os canais referem-se quantas sequências de som compões o som como um todo. Por exemplo, sons estereos possuem dois canais: direita e esquerda, já home theaters possuem 6 canais (5 caixas de som + 1 subwoofer). Os canais permitem efeitos sonoros. No nosso EP vamos trabalhar sempre em MONO, 1 canal, ou ESTEREO, dois canais. Desta forma vamos lidar sempre com uma ou duas séries de dados somente.

Nas próximas linhas aparecem 2 ou 3 colunas: A primeiro indica o instante de tempo em segundos, e a segundo o valor da função que define o som. É essa coluna que usaremos na análise de Fourier no caso MONO. No caso ESTEREO terá uma terceira coluna.

Exemplo de arquivo de dados:

```
; Sample Rate 1000
; Channels 1
0.000000 3.000000
0.001000 3.022149
0.002000 3.042591
0.003000 3.061323
0.004000 3.078342
0.005000 3.093650
0.006000 3.107250
0.007000 3.119148
0.008000 3.129353
0.009000 3.137876
0.010000 3.144730
0.011000 3.149931
0.012000 3.153498
0.013000 3.155452
0.014000 3.155814
0.015000 3.154612
...          ...
```

Audacity

Outra possibilidade é usar o software Audacity (`https://www.audacityteam.org/download/`) para converter os dados de/para .wav.

Para converter um .wav em arquivo de texto primeiro você deve abrir o arquivo .wav no software e depois usar o Analisar − > Sample Data Export para criar o arquivo de texto (`https://manual.audacityteam.org/man/sample_data_export.html`). Use o formato "linear"de saída e escolha o tamanho da amostra a ser exportado. O arquivo criado será um .txt e conterá um cabeçalho e os dados que definem o som (o Audacity não imprime o tempo como Sox).

Exemplo de saída:

```
test.txt   1 channel (mono)
Sample Rate: 11127 Hz. Sample values on linear scale.
Length processed: 10000 samples 0.89871 seconds.
```

```
3.000000
3.022149
3.042591
3.061323
3.078342
3.093650
3.107250
3.119148
...
```

Para ler um arquivo de dados no Audacity você deve ter uma série sem cabeçalho dada em um arquivo .txt. Para ler esse arquivo no Audacity use Gerar $->$ Sample Data Import (`https://manual.audacityteam.org/man/sample_data_import.html`). Ao ter o som carregado você pode salvá-lo como .wav.