Tema 8

Representação de Som

Objetivos:

- Representação de informação sonora
- Operações sobre som

8.1 Princípios de acústica

No mundo físico o som é transmitido através de ondas sonoras que são flutuações contínuas da pressão do ar ao longo do tempo e do espaço. Se a frequência de flutuação for alta, resulta um som agudo. Se a frequência for baixa, resulta um som grave. A amplitude da flutuação está relacionada com a *força* do som. Assim, um som muito fraco como um sussurro tem uma amplitude muito baixa, enquanto um som forte como um motor tem uma amplitude mais alta. A figura 8.1 mostra as flutuações de pressão de um som ao longo de um intervalo de tempo num certo ponto do espaço. O som representado é mais fraco e agudo no início, mais forte e grave no fim.

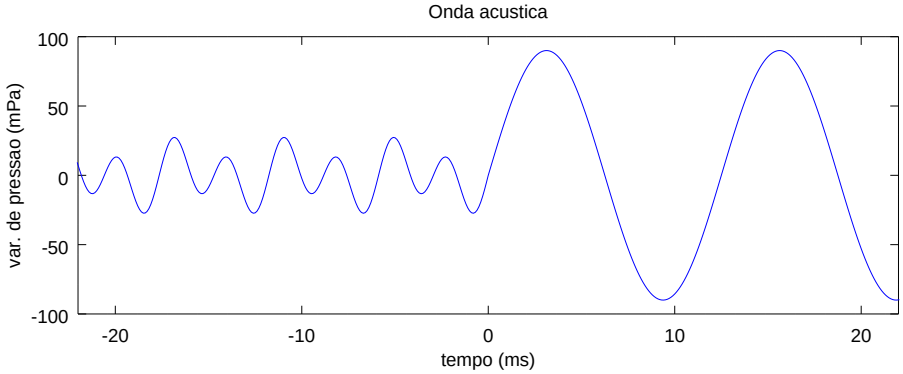

Figura 8.1: Forma de onda de um som medida ao longo do tempo num ponto do espaço.

Um tom dito puro corresponde a uma variação de pressão que é uma função sinusoidal do tempo: $\Delta p(t) = A \sin(2\pi f t)$. Aqui, $\Delta p(t)$ representa a variação de pressão do ar no instante t, A é a amplitude da variação e f é a frequência da variação, indicada em ciclos por unidade de tempo.[1]

A maioria dos sons, mesmo quando emitidos por instrumentos musicais, têm formas de onda mais complexas, mas que se podem sempre considerar como combinações de tons puros (sinusóides) de diferentes frequências e amplitudes (e diferentes desfasamentos). A combinação de diferentes frequências, com diferentes amplitudes para cada frequência, produz toda a multitude de sons que somos capazes de reconhecer.

A Figura 8.2 demonstra as frequências emitidas por sete notas diferentes de um piano. O eixo horizontal representa o tempo, enquanto o vertical representa a frequência das componentes sinusoidais do som. Uma cor quente (vermelho/branco) num ponto da figura indica uma componente forte (amplitude alta) nessa frequência e instante de tempo. Cores frias (azul, cinza) indicam componentes fracas ou mesmo ausentes. Este tipo de representação chama-se um espetrograma.

Como se pode ver, cada nota contém múltiplas componentes de frequências bem definidas (linhas horizontais), com amplitude suavemente decrescente ao longo da duração da nota. Isto é mais evidente na última nota, mais aguda, em que as componentes aparecem mais afastadas entre si, a primeira com $f \approx 800$Hz e as seguintes em frequências múltiplas dessa.

Também se percebe, no início de cada nota, uma maior intensidade, mas mais espalhada por diversas frequências (linhas verticais), o que é característico de sons mais curtos e explosivos, que neste caso correspondem a componentes transitórias do som causadas pela ação de percussão dos martelos nas cordas.

Exercício 8.1

Utilize o programa *Audacity* e analise o ficheiro `piano-c5-c6.wav` que foi fornecido pelos docentes. Em particular, experimente as diferentes formas de visualização do sinal, comutando entre forma de onda e espectrograma, por exemplo. Pode aceder a esta função se pressionar a seta que se encontra à direita do nome do ficheiro, do lado esquerdo da aplicação. Também pode selecionar um trecho curto de uma das notas e usar a função de *Analyze->Plot Spectrum* para ver o espetro desse segmento. Os picos no espectro indicam as frequências mais fortes presentes nesse trecho.

[1] A unidade de frequência é o Hertz (Hz) e corresponde a um ciclo por segundo: $1\text{Hz} = 1\text{s}^{-1}$.

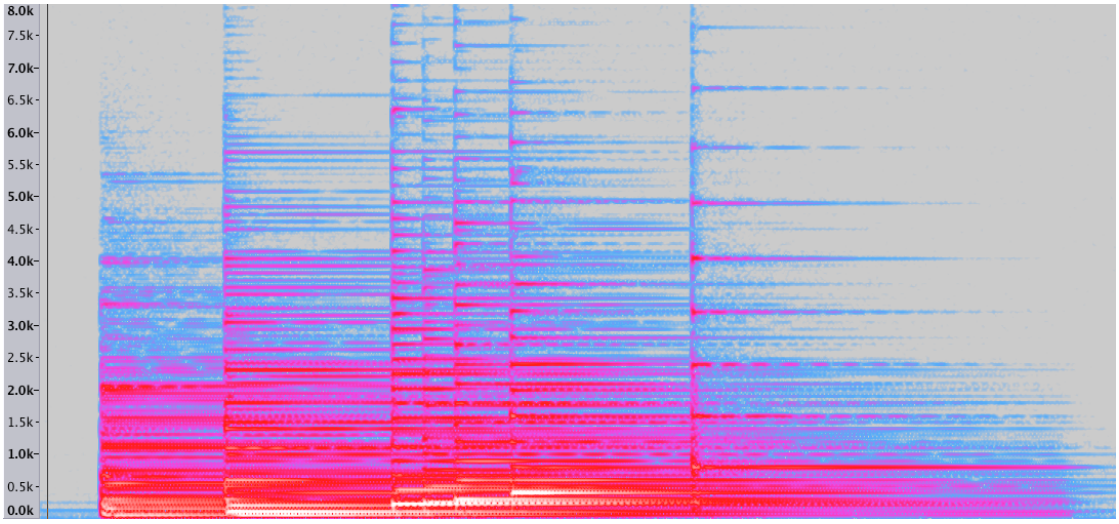

Figura 8.2: Várias notas de um piano capturadas num espetrograma.

8.2 Representação de informação sonora

Um microfone converte as variações de pressão em variações de tensão (ou de intensidade) de uma corrente elétrica. A tensão do sinal elétrico varia continuamente em função do tempo, de forma análoga à variação de pressão do sinal acústico. Por isso diz-se que é um *sinal analógico*.[2] Sistemas eletrónicos analógicos (formados por resistências, condensadores, transístores e outros componentes) permitem amplificar, processar e até armazenar sinais analógicos diretamente.

Os rádios AM ou FM, as televisões antigas (não TDT), os gravadores de fita magnética de áudio (cassetes) ou vídeo (VHS), os gira-discos de vinil são exemplos de sistemas de transmissão, processamento e armazenamento puramente analógicos.

O problema destes sistemas é que em cada passo de processamento, os sinais vão-se degradando com o acumular de pequenas distorções, interferências e ruído eletrónico. Os sistemas digitais resolvem esse problema, mas para os usar, é preciso transformar os sinais analógicos em sinais digitais. Para isso, recorre-se a um conversor analógico-digital, ou Analog to Digital Converter (ADC), que é um dispositivo que faz duas operações:

1. tira amostras instantâneas do sinal a intervalos regulares (amostragem) e
2. mede a tensão de cada amostra, e converte-a num número binário com um número fixo de dígitos (quantização).

[2] Deveria talvez designar-se *sinal análogo*, mas *sinal analógico* está legitimado pelo uso.

A figura 8.3 mostra estas duas operações aplicadas a um trecho do sinal analógico correspondente ao som da figura 8.1.

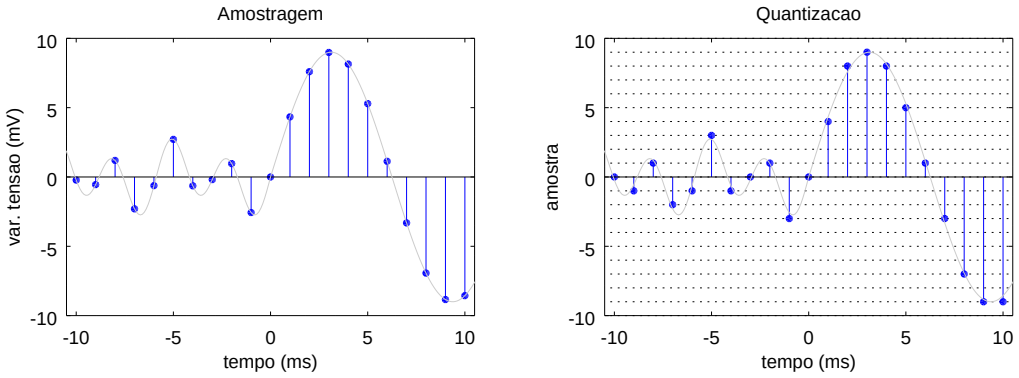

Figura 8.3: Conversão analógico-digital: amostragem e quantização. O sinal analógico original é representado em cinzento. As linhas verticais azuis representam os instantes de amostragem. A quantização aproxima as amplitudes por valores de um conjunto finito.

O sinal analógico que é uma função real, contínua no tempo, é assim convertido numa sequência de números inteiros. Este *sinal digital* resultante é portanto descontínuo no tempo e descontínuo em amplitude. Não consegue por isso representar a infinidade de detalhe que existe numa onda que varia continuamente ao longo do tempo, mas tem a vantagem de se poder armazenar, processar e transmitir virtualmente sem acumulação de degradação.

O número de amostras que uma ADC tira por segundo é chamada a sua *frequência de amostragem* e o número de bits usado em cada amostra é a sua *resolução*. As ADCs usadas atualmente para sinais áudio utilizam resoluções típicas de 8, 16 ou 24 bits e frequências de amostragem típicas de 8000, 11025, 22050, 44100, 48000 ou 96000Hz. Quanto maiores forem este valores, mais precisa será a representação digital feita do som original. No entanto, também será necessário possuir um sistema mais veloz e mais espaço de armazenamento para a informação. Os valores ideais dependem das limitações da sensibilidade auditiva e da aplicação.

De acordo com teorema da amostragem de Nyquist, a frequência de amostragem deve ser, no mínimo, o dobro da máxima frequência do sinal registado para permitir a sua reconstrução exata. Como o ouvido humano detecta frequências até perto dos 20Khz, será necessária uma frequência de amostragem superior a 40000Hz para registar devidamente todas as frequências audíveis. Não é portanto surpresa que a frequência de amostragem típica para registos musicais (CD, MP3) seja de 44100Hz (ou 48000Hz). Quando se pretende registar voz, como as componentes mais importantes da voz humana estão compreendidas entre 100 e 3000Hz, basta uma frequência de amostragem de 8000Hz, que é um valor popular nas comunicações móveis.

Para reproduzir um som a partir de um sinal digital, é preciso fazer o processo inverso: usar um Digital to Analog Converter (DAC) para converter a sequência de números num sinal elétrico analógico; amplificar esse sinal e convertê-lo em ondas de pressão acústica através de um altifalante.

8.2.1 Armazenamento de som

Num computador o som é armazenado através da sequência de valores medidos nos instantes de amostragem. Um ficheiro sonoro pode registar uma sequência obtida de um único microfone ou pode ter várias sequências obtidas em simultâneo de vários microfones. Chama-se canal a cada sequência registada em simultâneo no mesmo ficheiro. São vulgares os ficheiros *monofónicos* (ou *Mono*), com um canal apenas, e os ficheiros estereofónicos (*Stereo*), com dois canais (esquerdo/direito), mas é possível ter ficheiros com 7 ou mais canais como é o caso dos ficheiros para os sistemas *Surround*.

A informação em si pode estar armazenada sob a forma de texto, mas tal só é comum em aplicações científicas. De resto espera-se que esteja armazenada numa forma binária, eventualmente comprimida. A compressão destina-se a reduzir os requisitos para o armazenamento de informação.

Há métodos de compressão chamados *Lossless* (sem perdas), que permitem a recuperação exata da sequência de valores originais, e métodos de compressão *Lossy* (com perdas), que introduzem distorção nos sinais, mas fazem-no descartando ou alterando componentes menos perceptíveis pelo sistema auditivo humano.

Esta é a abordagem seguida quando se cria um ficheiro *MP3*, por exemplo. Portanto, os sons guardardos num fomato *Lossy* não são iguais ao original, mas usando taxas de compressão razoáveis, soam-nos igual ao original. Durante este guião o foco serão os ficheiros não comprimidos como é o caso do formato WAVEform audio file format (WAVE) pois facilitam a manipulação da informação contida.

Os ficheiros WAVE (ver Figura 8.4) são constituídos por um pequeno cabeçalho onde é possível indicar alguma meta-informação, sendo este cabeçalho seguido de blocos com a informação sonora, geralmente no formato Linear Pulse Code Modulation (LPCM). A cada bloco dá-se o nome de *Frame*, e contém os valores registados para cada canal num certo instante de amostragem. Cada um dos valores numa *Frame* é uma amostra (*Sample*) de um dos canais e é codificada num número de bytes suficiente para a resolução usada. Por exemplo, considerando um ficheiro com dois canais, com resolução de 16 bits e frequência de amostragem de 44100Hz, cada segundo de som teria 44100 frames com 2 samples de 2 bytes cada, ou seja, um ritmo de 176400 octetos por segundo. Nestas condições, uma música de 5 minutos gera um ficheiro com um pouco mais de 50MB.

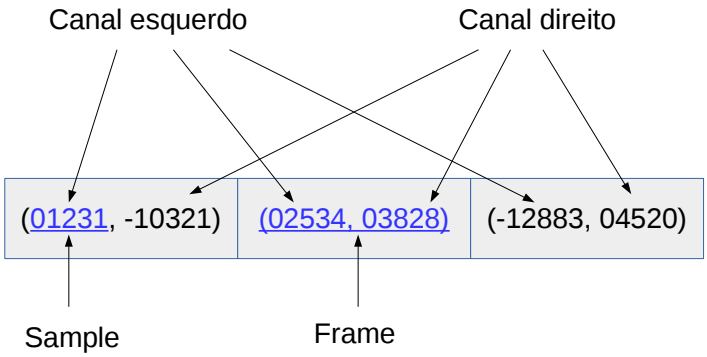

Figura 8.4: Estrutura dum ficheiro WAVE.

Em *Python* é possível inspeccionar os metadados dos ficheiros WAVE e mesmo obter a informação sonora. Também é possível criar novos ficheiros, o que será efectuado na Subseção 8.2.2. Para isto é necessário utilizar o módulo **`wave`**[3] que pode ser instalado das formas usuais em *Python*. O exemplo seguinte demonstra como pode ser aberto um ficheiro WAVE e obtida informação do seu cabeçalho.

```
import wave
import sys

def main(argv):
    wf = wave.open(argv[1], "rb")
    print(wf.getnchannels())
    ...
    wf.close()

main(sys.argv)
```

Exercício 8.2

Implemente um programa que obtenha a frequência de amostragem de um ficheiro, o tamanho de cada *Sample*, o número de canais e o número de *Frames* de som contidas no ficheiro.

Também é possível obter os dados sonoros e reproduzir o som directamente de forma programática. O módulo que permite isto possui o nome de **`PyAudio`** podendo ser instalado através de **`pip`** ou pelo gestor de pacotes da distribuição. O exemplo seguinte cria um **`player`** que pode ser utilizado para reproduzir um ficheiro WAVE.

[3] ver `https://docs.python.org/3/library/wave.html`

```
import pyaudio
player = pyaudio.PyAudio()

...

stream = player.open(format = player.get_format_from_width(sample_width),
                     channels = nchannels, rate = frame_rate, output = True)

while True:
    data = wf.readframes(1024)
    if not data:
        break

    stream.write(data)

stream.close()
player.terminate()
```

Exercício 8.3

Melhore o programa anterior de forma a que apresente informação de um dado ficheiro e o reproduza. Experimente modificar a variável **`frame_rate`** para um qualquer outro valor e volte a reproduzir o ficheiro.

8.2.2 Geração de tons

Além da leitura de ficheiros WAVE, ou a sua construção através da leitura do microfone, também é possível a criação de sons através da sintetização das diversas frequências de uma forma matemática. Isto porque sendo um som composto por uma onda a oscilar numa frequência específica e com uma determinada amplitude, esta onda pode ser recriada através da função **`sin`** (seno) multiplicada por um factor de amplitude.

Para gerar um tom com frequência f é necessário criar uma sequêmcia de valores através da expressão:

$$v(i) = amplitude * sin(\frac{2 * \pi * freq * i}{rate}) \quad (1)$$

em que $v(i)$ representa a amostra no instante i, $freq$ a frequência desejada e $rate$ a frequência de amostragem do som (p.ex 44100Hz).

Aplicando a fórmula à linguagem *Python*, podem ser gerados ficheiros WAVE com tons puros da seguinte forma:

```
from struct import pack
from math import sin, pi
import wave
import sys

def main(argv):
    rate=44100
    wv = wave.open(argv[1], "w")
    wv.setparams((1, 2, rate, 0, "NONE", "not compressed"))

    amplitude = 10000
    data = []
    freq = 440
    duration = 1 # Em segundos
    for i in range(0, rate * duration):
        data.append(amplitude*sin(2*pi*freq*i/rate))

    # Gerar (pack) a informação no formato correto (16bits)
    wvData = []
    for v in data:
        wvData += pack("h", int(v))

    wv.writeframes(bytearray(wvData))
    wv.close()

main(sys.argv)
```

Exercício 8.4

Implemente o exemplo anterior e gere ficheiros com vários tons. Analise os ficheiros criados através da aplicação *Audacity*.

Podemos criar sons compostos somando tons de múltiplas frequências gerados em simultâneo. Por exemplo, para criar um som com duas componentes, uma a 440Hz e outra a 880Hz, podemos fazer:

```
    ...
    freq_a = 440
    freq_b = 880
    for i in range(0, rate):
        data.append(
            amplitude*sin(2*math.pi*freq_a*i/rate) +
            amplitude*sin(2*math.pi*freq_b*i/rate)
            )
    ...
```

Exercício 8.5

Crie um novo programa baseado no anterior que gere um som composto por dois tons. Analise o resultado na aplicação *Audacity*.

A simplicidade deste método foi explorada em muitos sistemas, sendo que um dos mais famosos é o sistema Dual-Tone Multi-Frequency signaling (DTMF). Este sistema é utilizado para enviar algarismos e outros 6 símbolos através de ligações analógicas.

Cada símbolo é codificado como um par de tons com certas frequências e enviado para o recetor. O receptor separa e deteta o par de frequências para descodificar o símbolo. A tabela de codificação é a seguinte:

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

A figura 8.5 mostra o espectrograma de um número codificado no sistema DTMF.

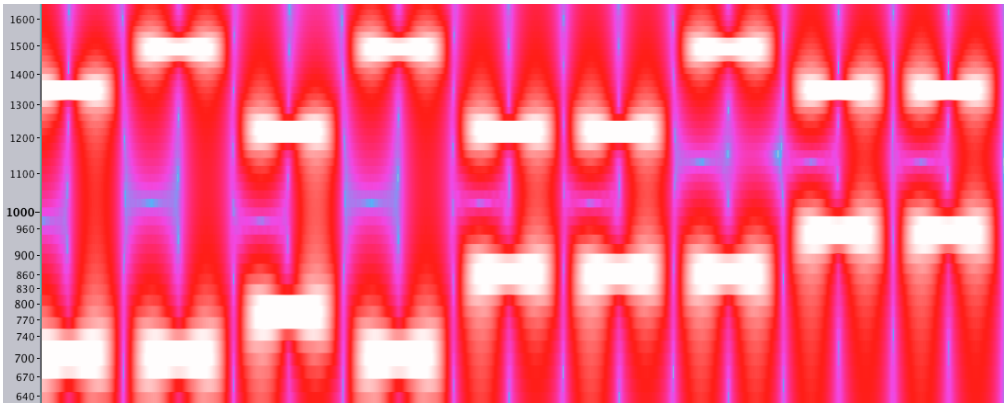

Figura 8.5: Número de telefone codificado em DTMF

Exercício 8.6

Identifique qual o número de telefone representado na Figura 8.5.

Em *Python* uma maneira simples de implementar esta tabela seria através de um dicionário, em que cada símbolo possui uma lista com as frequências a utilizar.

```
...
tones = {\
    "1": (697, 1029), \
    "2": (697, 1336), \
    ...}
...
```

Isto pode depois ser utilizado para gerar sons DTMF, com duração de 40ms, seguidos de uma pausa de outros 40ms, tal como um telefone ou telemóvel actual fazem. O exemplo seguinte demonstra a estrutura básica de um programa para codificar qualquer número em DTMF.

```
...
tones = {...}
number = "" # número a codificar
for n in number:
    # Códigos DTMF
    for i in range(0, int(rate*0.040)):
        data.append(
            # Valores dos tons
        )
    # Pausa (silêncio)
    for i in range(0, int(rate*0.040)):
        data.append(
            # Silêncio
        )
...
```

Exercício 8.7

Crie um programa que leia um número do teclado e gere um ficheiro com os códigos DTMF respectivos. Analise o resultado na aplicação *Audacity*.

8.3 Operações sobre som

São várias as operações que podem ser efectuadas sobre os ficheiros de som, além claro de os reproduzir. Nomeadamente é possível aplicar transformações, normalmente denominados de efeitos, que alterem as características sonoras da informação. Muitas das transformações são complexas, necessitando de conceitos mais complexos sobre o processamento de sinal. Alguns são bastante triviais ou relativamente simples de implementar, em particular os que operam sobre a informação numa perspectiva puramente temporal.

O seguinte trecho de código *Python* mostra uma forma bastante geral de aplicar uma qualquer transformação a um ficheiro WAVE, devolvendo o resultado noutro ficheiro do mesmo tipo. As secções seguintes deverão fazer uso deste programa, ou de um com estrutura semelhante para testar os efeitos desenvolvidos.

```
import wave
import struct
import sys
from struct import pack
import math

def copy(data):
    output = []
    for index,value in enumerate(data):
        output.append(value)
    return output

def main(argv):
    stream = wave.open(argv[1], "rb")
    sample_rate = stream.getframerate()
    num_frames = stream.getnframes()
    raw_data = stream.readframes( num_frames )
    stream.close()

    data = struct.unpack("%dh" % num_frames, raw_data) # "B" para ficheiros 8bits
    # Aplica efeito sobre data, para output_data
    i = 2
    output_data = []
    while i < len(argv):
        if argv[i] == "copy":
            output_data = copy(data)
        elif argv[i] == "foo":
            param = int(argv[i+1])
            output_data = foo(data, param)
            i += 1
        elif... #Outros filtros

        i += 1

    wvData = b""
    for v in output_data:
        wvData += pack("h", int(v))

    stream = wave.open("out-"+argv[1], "wb")
    stream.setnchannels(1)
    stream.setsampwidth(2)
    stream.setframerate(sample_rate)
    stream.setnframes(len(wvData))
    stream.writeframes(bytearray(wvData))
    stream.close()

if len(sys.argv) < 3:
    print("Usage: %s wave-file filter1 <params> filter2 <params> ..." % sys.argv[0])
else:
    main(sys.argv)
```

Exercício 8.8

Crie um programa com o código anterior e verifique que consegue processar um ficheiro WAVE, criando uma cópia semelhante ao original. Use para isso um filtro chamado `copy`. Invoque o programa criado com a sintaxe: `python process.py file.wav copy`.

Exercício 8.9

Adicione um filtro ao código anterior que devolva `reversed(data)` e verifique que consegue processar um ficheiro WAVE, criando uma cópia invertida do original. Use para isso um filtro chamado `reverse`. Invoque o programa criado com a sintaxe: `python process.py file.wav reverse`.

8.3.1 Controlo de Volume

O volume a que o som é reproduzido depende essencialmente da amplitude do sinal, o que no caso de um ficheiro WAVE é representado pelo valor em absoluto de cada impulso. Para controlar o volume basta multiplicar todos os valores de amplitude por um factor multiplicativo. Se este factor for 0.5 o volume deverá ser diminuído em metade. Se for 2.0 o volume deverá ser multiplicado por 2.

Exercício 8.10

Implemente um filtro chamado `volume` que aceite um factor multiplicativo como parâmetro. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav volume 0.5`. Verifique o resultado obtido através da aplicação *Audacity*.

8.3.2 Normalização

A normalização de um ficheiro diz respeito a controlar o volume de forma a que o valor máximo encontrado corresponda ao valor máximo possível. No caso de um ficheiro de 16bits, um ficheiro normalizado para o valor máximo deverá conter pelo menos um valor igual a -32768 ou a igual a 32767.

Este filtro necessita de dois passos de processamento. O primeiro determina qual o valor absoluto máximo dos valores. Daqui pode-se calcular um factor multiplicativo. O segundo passo aplica o factor multiplicativo tal como no caso do filtro de volume.

Exercício 8.11

Implemente um filtro chamado `normalize`. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav normalize`. Verifique o resultado obtido através da aplicação *Audacity*.

8.3.3 Fade In-Out

Um filtro de *Fade* aplica um envelope progressivo à amplitude do som de forma a que o seu volume aumente ou diminua de forma contínua. A Figura 8.6 demonstra um som a que foi aplicado um *Fade In* e *Fade Out*.

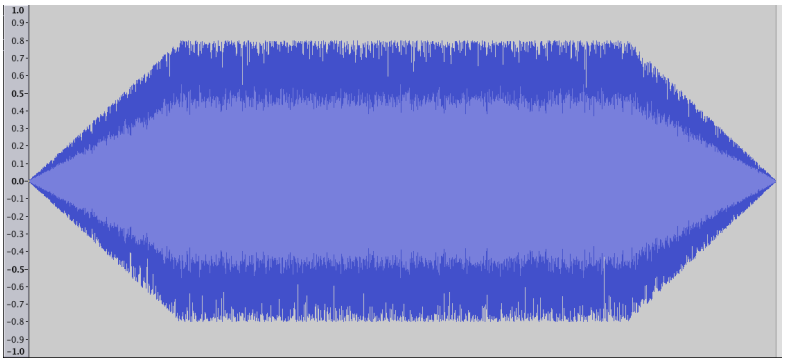

Figura 8.6: Exemplo de *Fade In* e *Fade Out*

A aplicação deste filtro é em tudo semelhante ao controlo de amplitude, com a diferença que o valor de amplitude é variável e só aplicado num intervalo temporal. Para ambos os casos é importante determinar onde iniciar e terminar a aplicação do efeito, que deve ter em consideração a frequência de amostragem do som.

Também se deve ter em consideração qual o declive do factor multiplicativo a aplicar. Desta forma, o valor final do sinal será $vf_i = vo_i * index * step$. Em que vf_i representa o valor final i, vo_i o valor original i, $index$ o número do *Sample* e $step$ o tamanho de cada incremento ao longo do processo de *Fade*.

O código *Python* seguinte demonstra o início de uma função aplicando este efeito:

```
def fadein(data, sample_rate, duration):
    time_start = 0
    time_stop = duration * sample_rate
    step = 1.0 / (sample_rate * duration)
    for index, value in enumerate(data):
        ...
```

Exercício 8.12

Implemente um filtro chamado **`fade-in`** que aceite uma duração em segundos como parâmetro. Deve conseguir invocar o programa desenvolvido através da sintaxe: **`python process.py file.wav fade-in 2`**. Verifique o resultado obtido através da aplicação *Audacity*.

Exercício 8.13

Implemente um filtro chamado `fade-out` que aceite uma duração em segundos como parâmetro. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav fade-out 2`. Verifique o resultado obtido através da aplicação *Audacity*.
Pode inclusive aplicar os dois efeitos usando: `python process.py file.wav fade-in 2 fade-out 2`

8.3.4 Máscaras

A aplicação de máscaras serve para omitir parte do conteúdo do som. É frequentemente utilizado para remover partes sensíveis, tal como palavras menos cuidadas. A operação deste filtro resume-se à substituição dos valores sonoros por outros no intervalo pretendido. Estes novos valores (vo_i)podem ser o resultado de:

- uma sinusóide com uma frequência específica (um tom): $vo_i = amplitude * sin(\frac{2*math.pi*freq*i}{rate})$
- silêncio: $vo_i = 0$
- valores aleatórios: $vo_i = random.randint(-32768, 32767)$

Este filtro pode ter como como parâmetro o tipo de máscara a aplicar, o instante de tempo inicial e o instante de tempo final para aplicação do efeito. Tal como no caso do filtro anterior é necessário calcular em que *Sample* iniciar a máscara e em que *Sample* terminar a máscara.

Exercício 8.14

Implemente um filtro chamado `mask` que aceite como parâmetro um tipo de máscara com os valores `silence`, `noise`, `tone`, um instante de início e uma duração em segundos. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav mask silence 2 2`. Verifique o resultado obtido através da aplicação *Audacity*.

8.3.5 Modulação

A modulação tem como princípio multiplicar um dado som por um outro tom. Considerando $vf(i)$ o valor final, $vo(i)$ o valor original, $freq$ uma frequência de modulação e $rate$ uma frequência de amostragem, o filtro pode ser concretizado com:

$$vf(i) = vo(i) * sin(\frac{2 * math.pi * freq * i}{rate}) \quad (2)$$

Uma variante deste efeito, mas considerando um tom variável, é o *Wah Wah* aplicado a guitarras e voz. O resultado de usar um tom fixo é um som com aspeto metálico, como se tivesse sido emitido por um robot num filme de televisão. Se a frequência for muito baixa o resultado é apenas uma variação cíclica no volume do som original.

Exercício 8.15

Implemente um filtro chamado **`modulate`** que aceite como parâmetro uma frequência em *Hertz*. Deve conseguir invocar o programa desenvolvido através da sintaxe: **`python process.py file.wav modulate 3000`**. Verifique o resultado obtido através da aplicação *Audacity*.

8.3.6 Atraso

O atraso é normalmente chamado de *Reverb* ou *Echo* e consiste na repetição de um sinal emitido num instante t_i, para um instante $t_i + x$ mas com uma amplitude ligeiramente inferior. Devidamente aplicado este efeito confere profundidade ao som, simulando as ondas refletidas naturalmente quando um som é reproduzido numa sala.

Considerando uma lista **`output`** que irá conter o resultado final, e uma lista **`data`** que contém o som original, pode-se construir este filtro através da seguinte estrutura:

```
    output = [0] * len(data)+tdelay
    ...
    for index,value in enumerate(data):
        output[index] += value
        output[index+tdelay] += value * amount
```

Neste caso o valor **`tdelay`** consiste no número de *Samples* que se pretende atrasar o som (consideram-se valores na ordem de 0.5-1.5 segundos), e **`amount`** consiste na força do efeito. Considera-se um valor inferior a 1. Um aspeto interessante deste efeito é que ele pode ser aplicado de forma recursiva, sendo que em cada nova iteração o efeito deve ser aplicado mais tarde e ter menos força.

Considerando que o efeito se encontra implementado numa função chamada **`delay`**, pode-se implementar este principio da seguinte forma:

```
def delay(data, sample_rate, amount, delay):
    if amount < 0.05:
        return data
    ...
    ...

    #Repetir com 80% da força e com 20% mais de atraso.
    return delay(output, sample_rate, amount * 0.8, delay * 1.2)
```

Exercício 8.16

Implemente um filtro chamado **`delay`** que aceite como parâmetro um atraso segundos e uma força. Deve conseguir invocar o programa desenvolvido através da sintaxe: **`python process.py file.wav delay 0.8 0.6`**. Verifique o resultado obtido através da aplicação *Audacity*.

Exercício 8.17

Implemente a versão recursiva deste filtro e avalie a complexidade do efeito resultante, em comparação com a versão mais simples.

8.3.7 Esteganografia

Os ficheiros de som, em particular no formato WAVE também permitem a aplicação de técnicas de esteganografia. Em particular é simples codificar mensagens através da manipulação do último bit de de cada *Sample*. Pode-se considerar que para codificar um valor 0, o valor do último bit deverá ser 0, sendo 1 para codificar um valor 1. Quando aplicado com a linguagem *Python*, é possível codificar um texto em *Samples* sonoros através da seguinte estrutura:

```
def steg_add(data, message):
    bitstream = "" # Irá conter uma string. ex: "011101010"
    for c in message:
        bitstream += format(ord(c),2)

    output = []
    encoded_bit = 0
    for index, value in enumerate(data):
        ....
```

A descodificação é muito semelhante sendo que um dado caráter **`c`** (neste caso o primeiro) pode ser obtido de uma *String* com os bits fazendo: **`c = chr(int(decoded_bits[0:8],2))`**.

Normalmente é útil adicionar um marcador para sinalizar a existência e/ou final de uma mensagem, separadores ou mesmo códigos de detecção de erros.

Exercício 8.18

Implemente um filtro chamado **`steg_add`** que aceite como parâmetro uma mensagem. Deve conseguir invocar o programa desenvolvido através da sintaxe: **`python process.py file.wav steg_add 'mensagem de teste'`**. Verifique o resultado obtido através da aplicação *Audacity*.

Exercício 8.19

Implemente um filtro chamado `steg_get`. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav steg_get` e este deverá imprimir a mensagem previamente escondida.

8.4 Para aprofundar

Exercício 8.20

Considere as notas musicais podem ser reconstruídas a partir de uma nota inicial segundo a seguinte fórmula:$freq_n = 2^{n/12} * 440$. Relembre que as notas são: La, La#, Si, Do, Do#, Re, Re#, Mi, Fa, Fa#, Sol, Sol#, La, La#, Si, Do..., correspondendo estas notas a valores de n entre 1 e 16.
Implemente um programa que leia uma sequência de notas e as reproduza (p.ex: 1 1 8 8 10 10 8)

Exercício 8.21

Melhore o programa anterior de forma a considerar duração das notas e pausas entre as notas. Pode faze-lo através de um caráter como o '-' para indicar a duração e 0 para indicar uma pausa. Uma nota Do# com um terço da duração ficaria 2-3 (Do-um_terço) e uma pausa com um quarto da duração normal de uma nota ficaria 0-4 (Pausa-um_quarto).

Exercício 8.22

Desenvolva outros efeitos a aplicar a informação sonora, nomeadamente:

- Expansor: Sons menores com uma amplitude menor que x são aumentados para o máximo de amplitude. Os restantes mantêm-se inalterados.
- Compressor: Possui dois intervalos x_{max} e x_{min}. Amplitudes superiores a x_{max} são iguais a x_{max}, enquanto amplitudes inferiores a x_{min} são iguais a x_{min}.
- Limitador: Limita a amplitude a um valor. Ou seja, se o valor for superior a x, este passa a x.

Glossário

ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
DTMF	Dual-Tone Multi-Frequency signaling
LPCM	Linear Pulse Code Modulation
WAVE	WAVEform audio file format