Laboratórios de Informática 2014/2015
DETI - Universidade de Aveiro

Docentes
João Paulo Barraca <jpbarraca@ua.pt>
Diogo Gomes <dgomes@ua.pt>
João Manuel Rodrigues <jmr@ua.pt>
Mário Antunes <mario.antunes@ua.pt>

TEMA 18

Representação de Som

Objetivos:
- Representação de informação sonora
- Operações sobre som

18.1 Representação de informação sonora

No mundo físico o som é transmitido através de ondas sonoras que são flutuações contínuas rápidas da pressão do ar. 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.

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 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 (sinusoides) de diferentes frequências e amplitudes (e diferentes desfasamentos). A combinação de diferentes frequências, com diferentes

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

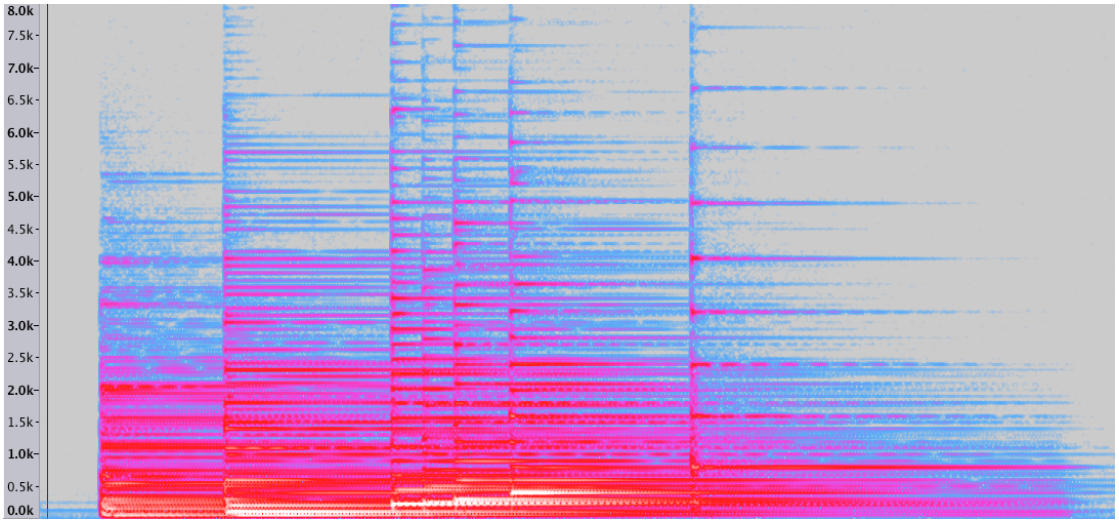

Figura 18.1: Várias notas de um piano capturadas num espetrograma.

amplitudes para cada frequência, produz toda a multitude de sons que somos capazes de reconhecer.

A Figura 18.1 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 pode ser visto, cada nota contém múltiplas componentes de frequências bem definidas (linhas horizontais). 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 esplahada por diversas frequências (linhas verticais), o que é característico de sons mais curtos e explosivos e aparece no som do piano pela ação de percussão dos martelos nas cordas.

Exercício 18.1

Utilize o programa *Audacity* e analise o ficheiro `piano-c5-c6.wav` que foi fornecido pelos docentes. Em particular pode experimentar 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 numa das notas e usar a função de *Analyze->Plot Spectrum* e

Um microfone converte as variações de pressão em variações de tensão (ou 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. Sistemas eletrónicos analógicos (formados por resistências, condensadores e outros componentes) permitem amplificar e processar estes sinais diretamente, mas um sistema digital não consegue processar ou armazenar a infinidade de detalhe que existe numa onda que varia continuamente ao longo do tempo. Por isso, recorre-se a conversores analógico-digitais (*analog-to-digital converters*, ou Analog do Digital Converter (ADC)), que são dispositivos que fazem duas operações:

1. tiram amostras instantâneas do sinal a intervalos regulares (amostragem) e
2. medem a tensão de cada amostra, convertendo-a num número binário de precisão finita (quantização).

O sinal analógico que é uma função real, contínua no tempo, é assim convertido numa sequência de números compostos de alguns dígitos binários. Este sinal digital resultante é portanto descontínuo no tempo e em amplitude, mas tem a vantagem de se poder armazenar e processar num computador.

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. Os sistemas actuais 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. De notar que, embora a resolução também seja importante, a frequência de amostragem é talvez o factor mais determinante. De acordo com um célebre teorema,[2] a frequência de amostragem deve ser, no mínimo, o dobro da máxima frequência do sinal registado para permitir a sua reconstrução exata. Portanto, considerando que o ouvido humano detecta frequências até perto dos 20Khz, será necessária uma frequência de amostragem mínima de 44100Hz para registar devidamente todas as frequências

[2]Teorema da amostragem de Nyquist (ou Nyquist-Shannon e outros).

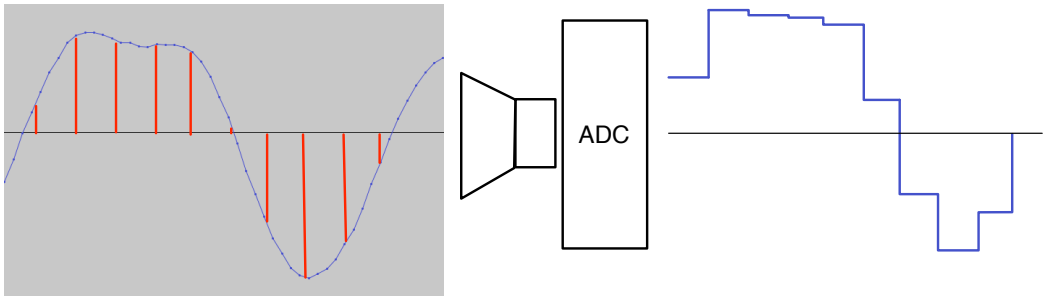

Figura 18.2: Processo de aquisição de um som por um sistema digital.

audíveis. Não é portanto surpresa que esta (ou 48000Hz) seja a frequência típica com que se registam músicas. Quando se pretende registar voz, visto que 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.

A Figura 18.2 demonstra um período de uma onda sonora com várias frequências. Se fosse uma frequência pura, ela não teria pequenas distorções como este caso apresenta. As linhas verticais vermelhas representam os instantes de amostragem. Como se pode ver, são necessárias no mínimo duas amostras da onda para poder registar a sua frequência fundamental, mas mais amostras resultam em valores mais precisos, essenciais caso o som tenha componentes de frequência mais alta, como neste caso.

A reprodução de sinais utiliza um sistema chamado Digital to Analog Converter (DAC), que realiza o processo inverso e converte sequências de valores numéricos em sinais analógicos contínuos.

18.1.1 Armazenamento de som

Num computador o som é armazenado através da sequência de valores medidos nos instante 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 *Mono*, com um canal apenas, e ficheiros *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, estes ficheiros comprimidos 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.

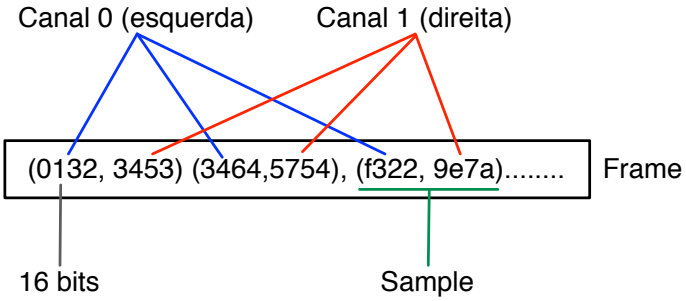

Figura 18.3: Estrutura de uma *Frame* num ficheiro WAVE

Os ficheiros WAVE 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. Este formato está apresentado na Figura 18.3.

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 18.1.2. 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

[3] ver `https://docs.python.org/2/library/wave.html`

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 18.2

Implemente um programa que obtenha a frequência 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.

```
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 18.3

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.

18.1.2 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.

De uma forma simplificada pode-se considerar que para gerar um som numa frequência f é necessário criar uma lista de valores através da função seguinte:

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

em que $v(i)$ representa um dado impulso, $freq$ 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',v)

    wv.writeframes(wvData)
    wv.close()

main(sys.argv)
```

Exercício 18.4

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

Um aspeto interessante é que podem ser geradas múltiplas frequências em simultâneo, bastando para isso somar os valores obtidos na fórmula anterior. Por exemplo, se se pretendesse criar um ton composto por duas frequências, uma a 440Hz e outra a 880Hz, poderia-se 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 18.5

Crie um novo programa baseado no anterior que gere um ficheiro 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 letras através de ligações analógicas. Cada símbolo é convertido para um par de frequências e enviado, sendo depois simples a descodificação do lado do receptor. A tabela utilizada 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

Codificando um número no sistema DTMF o resultado é o demonstrado no espectrograma representado na figura seguinte:

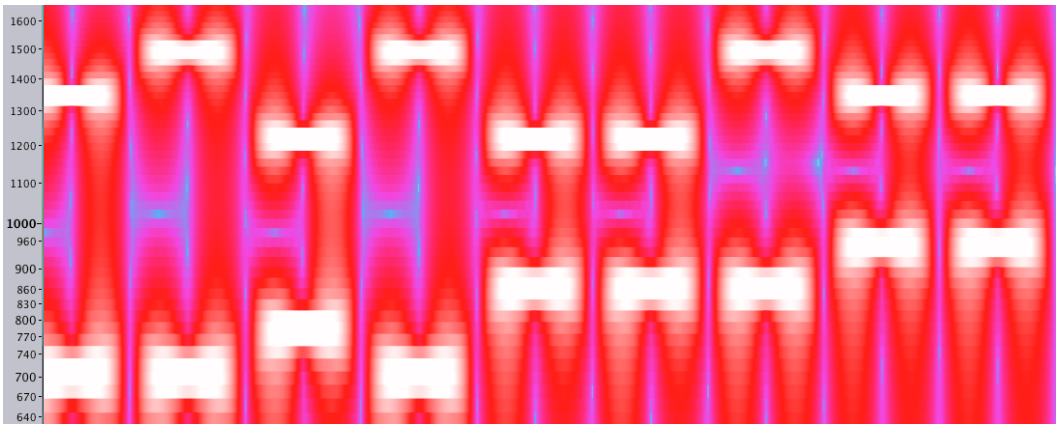

Figura 18.4: Número de telefone codificado em DTMF

Exercício 18.6

Identifique qual o número de telefone representado na Figura 18.4.

Em *Python* uma maneira simples de implementar esta tabela seria através de um dicionário, em que cada valor 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 demonsta 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 18.7

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

18.2 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.

De uma forma geral pode-se considerar o seguinte trecho de código *Python* para transformar um ficheiro WAVE numa lista de valores e voltar a codificar os valores para o formato esperado. 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 = 3
    while i < len(argv):
        if argv[i] == 'xpto':
            data = xpto(data)
        elif argv[i] == 'foo':
            param = int(argv[i+1])
            data = foo(data, param)
            i += 1
        elif... #Outros filtros

        i += 1


    wvData=''
    for v in output_data:
        wvData += pack('h',v)

    stream = wave.open('out-'+argv[1], 'wb')
    stream.setnchannels(1)
    stream.setsampwidth(2)
    stream.setframerate(sample_rate)
    stream.setnframes(len(wvData))
    stream.writeframes(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 18.8

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 18.9

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`**.

18.2.1 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 18.10

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*.

18.2.2 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 18.11

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*.

18.2.3 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 18.5 demonstra um som a que foi aplicado um *Fade In* e *Fade Out*.

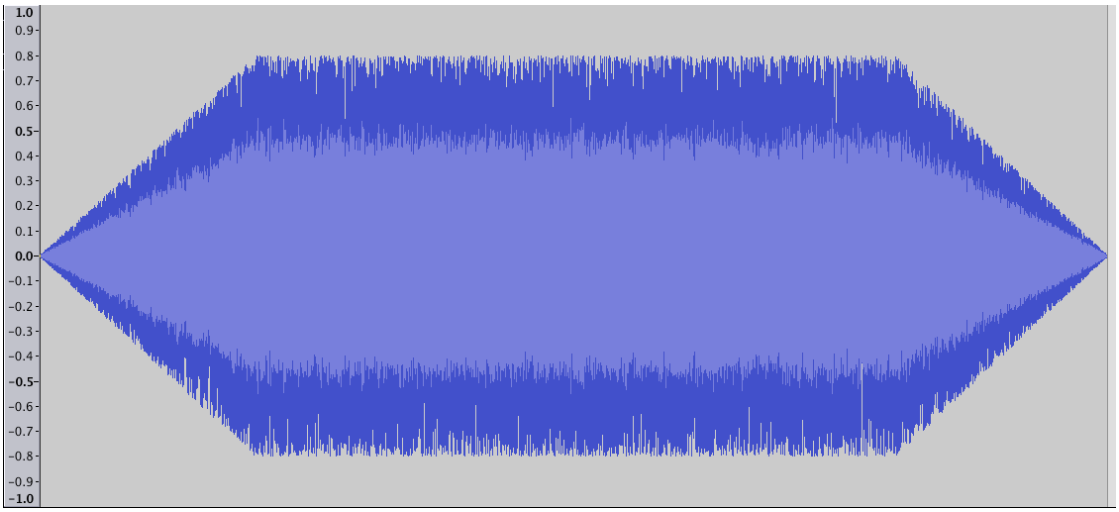

Figura 18.5: Exemplo de *Fade In* e *Fade Out*

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 fade-in(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 18.12

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 18.13

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`**

18.2.4 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 18.14

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*.

18.2.5 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 18.15

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*.

18.2.6 Atraso

O atraso é normalmente chamado de *Reverb* o *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)
    ...
    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 18.16

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 18.17

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

18.2.7 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 18.18

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 18.19

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.

18.3 Para aprofundar

Exercício 18.20

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 18.21

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 18.22

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 do Digital Converter

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