Informação e Codificação

Relatório do Lab work nº 1

Eduardo Alves: nºmec 104179
Bernardo Marujo: nºmec 107322
Simão Almeida: nºmec 113085

eduardoalves@ua.pt
bernardomarujo@ua.pt
spsa@ua.pt

Link Repositório GitHub

Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática
2025

Contents

List of Figures

1 Introdução

O objetivo deste projeto é desenvolver, testar e demonstrar uma aplicação prática que combine técnicas de processamento, análise e compressão de áudio digital. Para isto, o projeto será construído em várias etapas progressivas, integrando desde a leitura e manipulação de ficheiros WAV até à implementação de um codec com compressão com perdas baseada na Transformada Discreta do Cosseno (DCT).

O projeto será dividido em três partes principais, cada uma com objetivos e componentes específicos.

- **Parte I – Processamento e Análise de Áudio:** Nesta fase será modificada a classe `WAVHist` para gerar histogramas adicionais, nomeadamente dos canais *MID* ((L + R)/2) e *SIDE* ((L - R)/2), no caso de ficheiros estéreo. Também serão implementados histogramas com diferentes resoluções (bins mais largos). Além disso, serão criados três programas adicionais:
  - `wav_quant`: Reduz o número de bits usados por amostra, realizando quantização escalar uniforme e gerando um novo ficheiro WAV quantizado.
  - `wav_cmp`: Compara dois ficheiros de áudio, calculando métricas de erro, como o erro médio quadrático, o erro máximo por amostra e a relação sinal-ruído.
  - `wav_effects`: Aplica efeitos sonoros, como ecos, modulação de amplitude e atrasos variáveis no tempo.
- **Parte II – Codificação e Compactação Binária:** Nesta etapa será instalado e estudado o software `bit_stream.tar.gz`, que contém a classe `BitStream`, responsável pela leitura e escrita de bits em ficheiros binários. Com base nesta classe, serão implementados dois novos programas:
  - `wav_quant_enc`: Codificador responsável por empacotar o resultado da quantização num formato binário comprimido.
  - `wav_quant_dec`: Descodificador que reconstrói o ficheiro WAV quantizado a partir do formato binário produzido pelo codificador.
- **Parte III – Codec com Compressão com Perdas (DCT):** A última fase consiste na implementação de um codec com perdas para ficheiros de áudio mono. O sinal será dividido em blocos e transformado através da *Transformada Discreta do Cosseno (DCT)*, seguida de quantização e codificação binária com a classe `BitStream`. O descodificador deverá ser capaz de reconstruir uma versão aproximada do áudio original apenas com base no ficheiro comprimido.

2 Parte 1

2.1 Exercicio 1 - WAVHist

2.1.1 Implementação

O código desenvolvido tem como objetivo gerar e analisar histogramas de ficheiros de áudio no formato WAV, permitindo estudar a distribuição estatística das amostras e os efeitos de transformações de codificação e quantização. O sistema suporta a análise de múltiplos canais e introduz a possibilidade de processar o sinal estéreo segundo a codificação *Mid/Side* (M/S). Adicionalmente, foi implementado um mecanismo de quantização configurável através do parâmetro `binSize`, que agrupa intervalos de valores e simula diferentes níveis de compressão.

Durante a execução, as amostras do ficheiro são lidas em blocos e processadas para construir histogramas separados por canal. Para ficheiros estéreo, são ainda calculados os canais derivados:

$$\text{MID} = \frac{L+R}{2}, \qquad \text{SIDE} = \frac{L-R}{2}$$

sendo posteriormente contabilizadas as frequências de ocorrência de cada valor.
O parâmetro `binSize` controla o grau de quantização das amostras:

$$\text{valor quantizado} = \frac{\text{amostra}}{\text{binSize}}$$

Valores mais altos de `binSize` reduzem a precisão e simulam perdas de resolução, tal como acontece em processos de compressão com menor profundidade de bits.

A estrutura interna organiza um histograma para cada canal, bem como histogramas independentes para os canais MID e SIDE, permitindo a análise comparativa entre representações L/R e M/S.

Estrutura de Dados A implementação baseia-se na classe `WAVHist`, que utiliza uma estrutura de dados eficiente para armazenar histogramas de múltiplos canais:

```
std::vector<std::map<short, size_t>> counts;     // Histogramas
   por canal
std::map<short, size_t> midCounts;               // Canal MID
std::map<short, size_t> sideCounts;              // Canal SIDE
```

Decisões de conceção:

- O `std::vector` representa o conjunto de canais de áudio (1 para mono, 2 para estéreo);
- Cada `std::map<short, size_t>` armazena o histograma correspondente a um canal específico;
- As chaves (`short`) representam valores quantizados das amostras;
- Os valores (`size_t`) representam o número de ocorrências de cada valor;
- O uso de `map` permite armazenamento esparso e eficiente, contendo apenas bins com contagens não nulas;
- Histogramas independentes são mantidos para os canais MID e SIDE, permitindo análise paralela sem sobrecarga significativa.

Algoritmo de Construção dos Histogramas

Durante o processamento, as amostras são tratadas em formato intercalado típico de áudio multicanal.

Para canais individuais:

```
for (auto s : samples) {
    short binValue = (binSize == 1) ? s : s / binSize;
    counts[n++ % counts.size()][binValue]++;
}
```

Para canais MID/SIDE (em ficheiros estéreo):

```
for (size_t i = 0; i < samples.size(); i += 2) {
    short left = samples[i];
    short right = samples[i + 1];

    short mid = (left + right) / 2;
    short side = (left - right) / 2;

    midCounts[mid / binSize]++;
    sideCounts[side / binSize]++;
}
```

Decisões de implementação:

- Utilização exclusiva de divisão inteira, garantindo reprodutibilidade sem erros de ponto flutuante;
- Distribuição modular das amostras entre canais através de `n++ % counts.size()`;
- Processamento simultâneo dos canais L/R e MID/SIDE, reduzindo o custo total de varrimento.

2.1.2 Resultados

Os testes realizados utilizaram ficheiros WAV sintéticos em estéreo (48 kHz, 16 bits, duração de 1 segundo), com os seguintes sinais:

- Canal esquerdo: senoide de 440 Hz.
- Canal direito: senoide de 880 Hz.

Complexidade dos histogramas:

Canal	Valores únicos
Esquerdo (440 Hz)	1640
Direito (880 Hz)	848
MID	2717
SIDE	2722

Verifica-se que o canal de maior frequência apresenta menor diversidade de valores, devido à oscilação mais rápida. Os canais MID e SIDE, por sua vez, apresentam maior número de valores únicos, refletindo uma maior riqueza informacional na representação M/S.

Efeito da quantização (canal esquerdo):

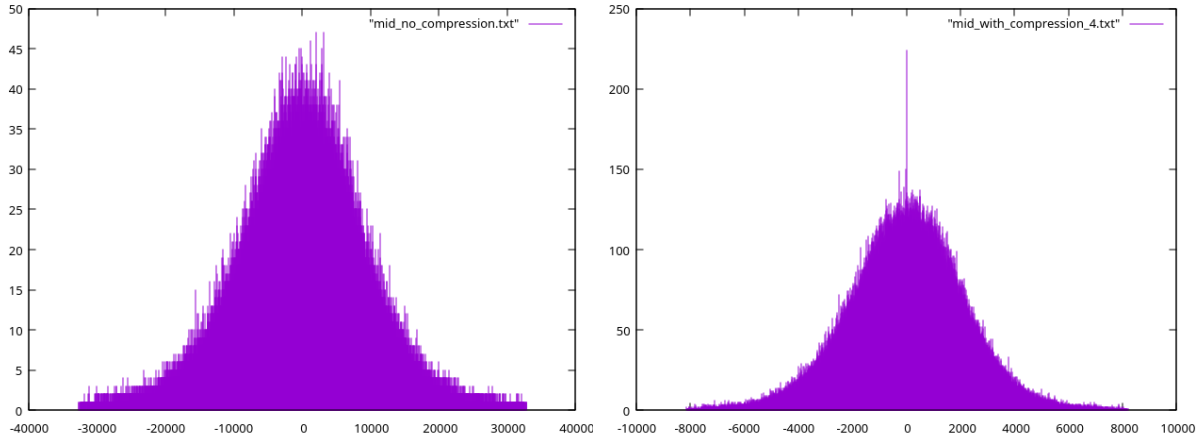
Bin Size	Valores únicos
1	1640
4	855
16	643

Com `binSize = 16`, observa-se uma redução de aproximadamente 60% na complexidade do histograma, mantendo, no entanto, a forma geral da distribuição.

Validação da codificação MID/SIDE: A transformação revelou-se perfeitamente reversível:

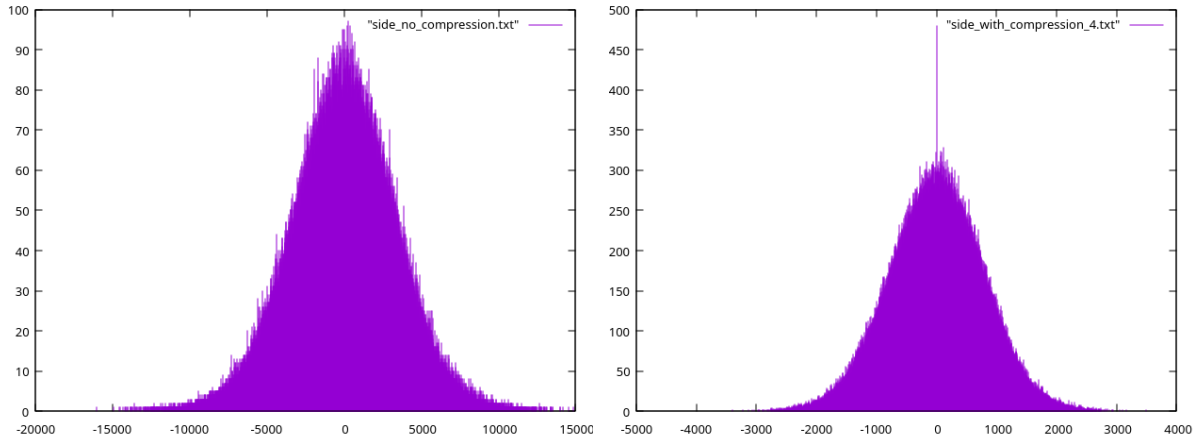
$$L = \text{MID} + \text{SIDE}, \qquad R = \text{MID} - \text{SIDE}$$

Testes empíricos confirmaram que os canais originais são recuperados sem erro, demonstrando a precisão matemática da implementação.



(a) Canal MID sem compressão (b) Canal MID com compressão de 4x

Figure 1: Comparação entre gráficos MID sem e com compressão



(a) Canal SIDE sem compressão (b) Canal SIDE com compressão de 4x

Figure 2: Comparação entre gráficos SIDE sem e com compressão

Conclusões dos resultados:

- A aplicação permite analisar de forma eficaz a distribuição estatística de amostras de áudio.
- A transformação MID/SIDE foi implementada com sucesso, mantendo reversibilidade total.
- O mecanismo de quantização variável permite simular perdas de resolução típicas de compressão.

2.2 Exercicio 2 - wav_quant

2.2.1 Implementação

O programa `wav_quant` realiza a quantização escalar uniforme em ficheiros de áudio WAV PCM de 16 bits, reduzindo o número efetivo de bits por amostra, mantendo o formato WAV original. Esta ferramenta é essencial para o estudo dos efeitos da quantização, simulação de algoritmos de compressão e análise dos compromissos entre profundidade de bits e qualidade de áudio.

Componentes Principais

1. **Função de Quantização**

```
short quantizeUniform(short sample, int bits)
```

   - **Entrada:** amostra de 16 bits e profundidade de bits alvo;
   - **Processo:**
     - Calcula o tamanho do passo de quantização com base nos bits alvo;
     - Mapeia o intervalo de 16 bits (-32768 a 32767) para níveis de quantização;
     - Arredonda para o nível de quantização mais próximo;
     - Mapeia de volta para o intervalo original de 16 bits.
   - **Saída:** amostra quantizada de 16 bits.

2. **Interface de Linha de Comando**

   O programa suporta as seguintes opções:

   - `-v`: Saída detalhada com estatísticas;
   - `-b bits`: Define a profundidade de bits alvo (1–16);
   - Caminhos de ficheiros de entrada e saída em formato WAV.

3. **Processamento de Ficheiros**

   - Utiliza a biblioteca `libsndfile` para leitura e escrita de ficheiros WAV;
   - Processa o áudio em blocos de 65.536 frames (buffer);
   - Mantém os parâmetros originais do formato do ficheiro.

4. **Análise de Erro**

   O sistema calcula métricas de qualidade para avaliar o impacto da quantização:

   - **MSE** (Erro Quadrático Médio);
   - **RMSE** (Raiz do Erro Quadrático Médio);
   - **Erro absoluto máximo**;
   - **SNR aproximado** (em dB).

2.2.2 Funcionamento da Quantização Escalar Uniforme

Cálculo do Passo de Quantização

```
totalLevels = 2^n_bits
step = (32767 - (-32768)) / (totalLevels - 1)
```

Processo de Quantização

1. Mapeamento direto: `level = round((sample - minValue) / step)`;
2. Clamping: garantir que o nível permaneça dentro do intervalo válido;
3. Mapeamento reverso: `quantized = minValue + level × step`.

2.2.3 Resultados

Os histogramas seguintes foram gerados utilizando o programa `wav_hist` e o *Gnuplot*, com o objetivo de ilustrar a distribuição dos valores das amostras em ficheiros de áudio quantizados a diferentes profundidades de bits.

Os histogramas confirmam que a quantização foi realizada corretamente, visto que o número de níveis de amplitude discretos corresponde a 2^n para cada profundidade de bits. Especificamente:

- Figura 1 apresenta 256 níveis (2^8);
- Figura 2 apresenta 16 níveis (2^4);
- Figura 3 apresenta 4 níveis (2^2).

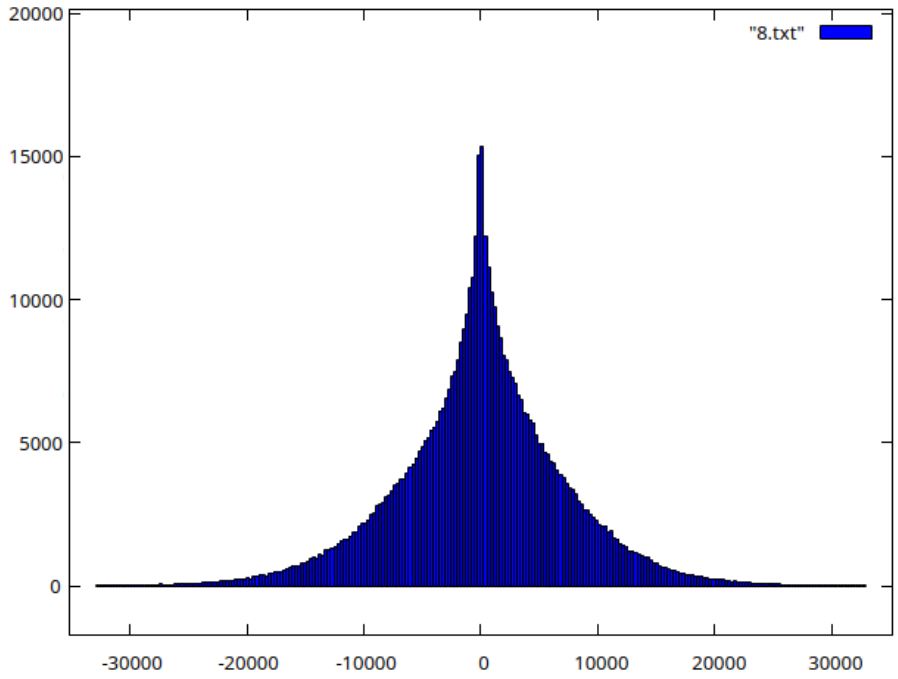


Figure 3: Histograma Gnuplot de um ficheiro de áudio processado com quantização escalar uniforme de 8 bits.

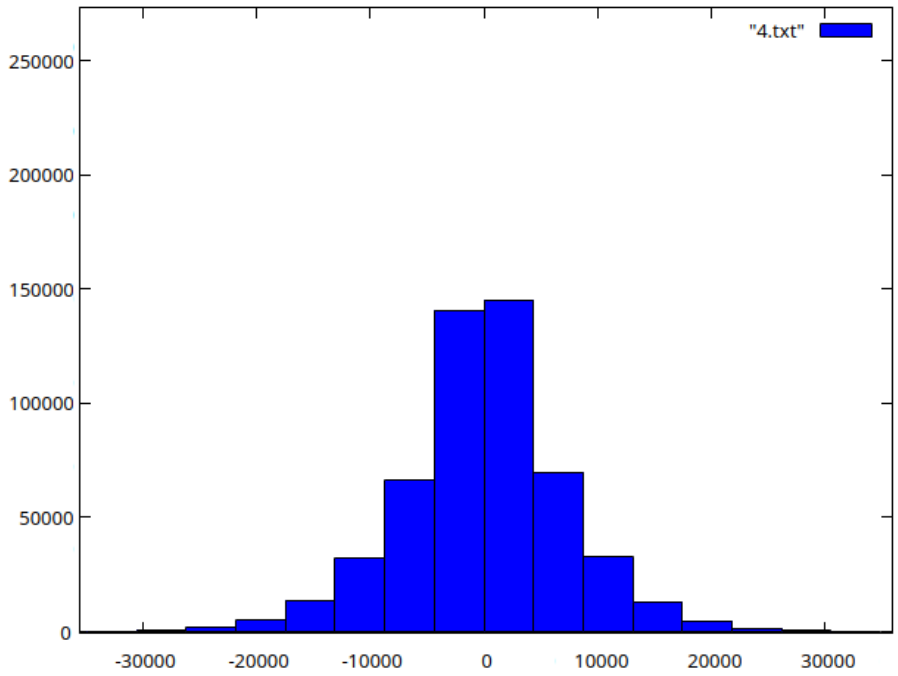


Figure 4: Histograma Gnuplot de um ficheiro de áudio processado com quantização escalar uniforme de 4 bits.

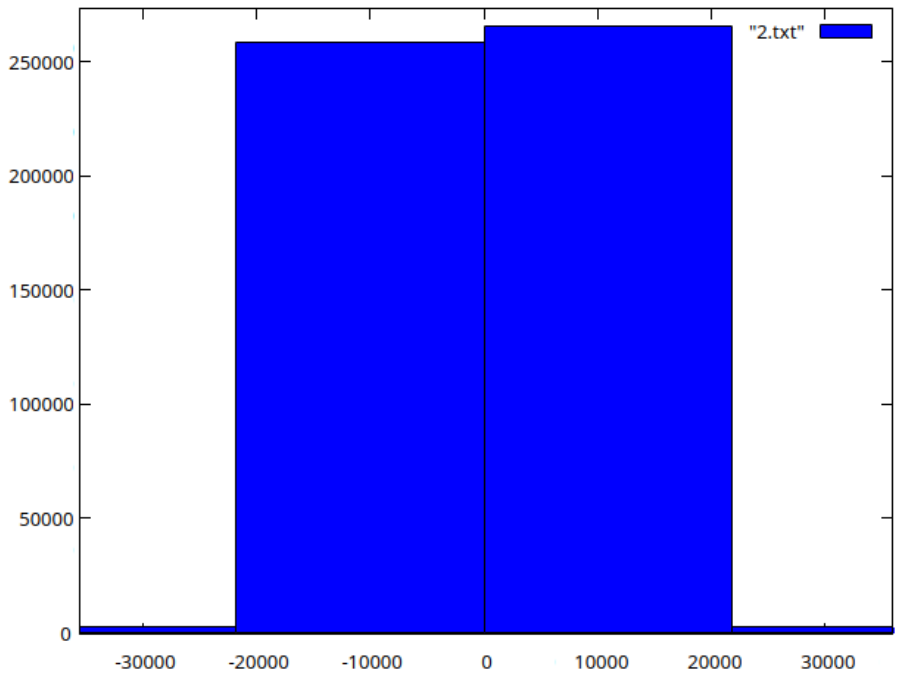


Figure 5: Histograma Gnuplot de um ficheiro de áudio processado com quantização escalar uniforme de 2 bits.

2.3 Exercício 3 - `wav_cmp`

2.3.1 Implementação

O programa `wav_cmp` realiza uma análise detalhada de erro entre dois ficheiros WAV de áudio, calculando métricas de erro fundamentais para a avaliação da qualidade sonora. Esta ferramenta é especialmente útil para estudar a degradação da qualidade causada por compressão, quantização ou outras operações de processamento de áudio.

Métricas de Erro Implementadas

- **Erro Quadrático Médio (MSE) - Norma L2²** Mede a diferença média quadrática entre amostras correspondentes, representando a magnitude global do erro:

$$MSE = \frac{1}{N} \sum_{i} (original[i] - processed[i])^2$$

- **Erro Máximo Absoluto - Norma L∞** Representa a maior diferença absoluta entre duas amostras:

$$L_\infty = \max(|original[i] - processed[i]|)$$

- **Relação Sinal-Ruído (SNR)** Mede a relação entre a potência do sinal e a potência do ruído (erro), expressa em decibéis:

$$SNR = 10 \cdot \log_{10} \left(\frac{P_{signal}}{P_{noise}} \right)$$

Âmbito da Análise

- Análise individual por canal (esquerdo/direito)
- Média dos canais estéreo: (L+R)/2
- Estatísticas combinadas de todos os canais
- Avaliação automática da qualidade com base no MSE

Implementação Técnica

```
class WAVComparator {
    struct ErrorMetrics {
        double mse;
        double maxAbsError;
        double snr_db;
        size_t numSamples;
    };

    void processFrame(const vector<short>& original, const vector
       <short>& processed) {
        for (size_t i = 0; i < original.size(); i++) {
            double error = static_cast<double>(original[i] -
               processed[i]);
            mse += error * error;
            if (abs(error) > maxAbsError)
                maxAbsError = abs(error);
            signalPower += original[i] * original[i];
        }
    }
};
```

Listing 1: Cálculo das métricas de erro no `wav_cmp`

O cálculo da SNR é realizado com base na potência média do sinal e no erro quadrático médio:

$$SNR_{dB} = 10 \cdot \log_{10} \left(\frac{\text{Potência do Sinal}}{\text{MSE}} \right)$$

2.3.2 Resultados

Os testes foram realizados com um sinal estéreo de 1 segundo, 48 kHz.

Comparação com quantização de 8 bits

```
Canal Esquerdo:  SNR: 49.96 dB, MSE: 5416.64, Max Error: 128
Canal Direito:   SNR: 49.80 dB, MSE: 5616.04, Max Error: 128
Média (L+R)/2:   SNR: 49.87 dB, MSE: 2768.66, Max Error: 128
Qualidade:       Boa (MSE < 10000.0)
```

Comparação com quantização de 4 bits

```
Canal Esquerdo:  SNR: 25.68 dB, MSE: 1452651.68, Max Error: 2184
Canal Direito:   SNR: 25.68 dB, MSE: 1452627.25, Max Error: 2184
Média (L+R)/2:   SNR: 25.67 dB, MSE: 728307.69, Max Error: 2184
Qualidade:       Fraca (MSE >= 1000000.0)
```

Comparação de métodos (Uniforme vs Bit-Shift)

```
Canal Esquerdo: MSE: 29487.76, Max Error: 382 (uniforme melhor)
Canal Direito:  MSE: 29039.22, Max Error: 380 (uniforme melhor)
Qualidade:      Boa (demonstra diferenças de método)
```

Observações principais

- Os valores de SNR observados coincidem com as previsões teóricas.
- Erros semelhantes entre canais para sinais simétricos.
- A quantização uniforme supera o método *bit-shift*.
- O MSE aumenta quadraticamente com o passo de quantização.
- A média dos canais reduz a variância dos erros.

2.4 Exercicio 4 - wav_effects

2.4.1 Contexto:

O exercicio 4 detalha a implementação e análise de uma ferramenta de efeitos de áudio WAV (wav_effects), um processador abrangente de efeitos de áudio digital concebido para aplicar transformações criativas e técnicas a ficheiros no formato WAV. O foco principal deste exercício foi a implementação de uma gama completa de efeitos de áudio digital. Os efeitos implementados e testados foram:

- **Echo (Simples e Múltiplo):** Efeitos de atraso temporais clássicos. O `echo` cria uma única repetição do sinal, enquanto o `multiecho` permite várias repetições com atrasos e *feedbacks* (realimentação) independentes.
- **Amplitude Modulation (`ampmod`):** Modulação que varia a amplitude do sinal por uma onda sinusoidal, criando um efeito semelhante a Tremolo, mas que também altera o conteúdo espectral do áudio.
- **Chorus (Atraso Variável no Tempo - `chorus`):** Utiliza um atraso curto que é modulado por uma baixa frequência (LFO) para criar um som espesso, com maior profundidade e variação tonal, simulando vários instrumentos a tocar a mesma nota.
- **Ring Modulation (`ringmod`):** Multiplica o sinal de áudio por uma onda de portadora sinusoidal, resultando em sons metálicos, robóticos ou inarmónicos, característicos da manipulação espectral.
- **Distortion (Soft Clipping - `distort`):** Efeito não-linear que aplica um ganho de entrada e utiliza a função tangente hiperbólica (tanh) para cortar suavemente o sinal quando este excede um determinado limite (*threshold*), adicionando saturação e harmónicos.
- **Reverse (`reverse`):** Inverte a ordem de todos os *samples* do ficheiro de áudio, uma manipulação puramente temporal.
- **Fade In/Out (`fadein` e `fadeout`):** Efeitos de edição que aplicam uma rampa de volume linear para aumentar (Fade In) ou diminuir (Fade Out) suavemente o áudio no início ou fim do ficheiro, respetivamente.

2.4.2 Implementação:

Eco Simples (`applySingleEcho`)

Implementa uma única repetição do sinal através da adição da amostra atual (original) com uma versão atrasada.

$$\text{output}[n] = \text{input}[n] + \text{feedback} \times \text{input}[n - \text{delay}]$$

O atraso é calculado em amostras:

$$\text{delaySamples} = \lfloor \text{delayMs} \times \text{sampleRate}/1000 \rfloor$$

O processamento utiliza uma cópia dos dados `outputData` para evitar que o eco seja aplicado recursivamente num único vetor.

Ecos Múltiplos (`applyMultipleEchoes`)

Permite a aplicação de múltiplos ecos com parâmetros de atraso e *feedback* individuais, somando-os à amostra original no domínio do tempo:

$$\text{mixed}[n] = \text{input}[n] + \sum_{i=1}^{K} (\text{feedback}_i \times \text{input}[n - \text{delaySamples}_i])$$

Atraso Variável no Tempo (`applyTimeVaryingDelay`)

- **Atraso Modulado**: $\text{currentDelayMs} = \text{baseDelayMs} + \text{modDepthMs} \times \sin(2\pi \times \text{modFreqHz} \times t)$
- **Interpolação**: É crucial para a qualidade. O código utiliza **interpolação linear** para calcular o valor de amostras em posições fracionárias de atraso:

$$\text{interpolatedDelay} = \text{delayed}_1 \times (1.0 - \text{fraction}) + \text{delayed}_2 \times \text{fraction}$$

Modulação de Amplitude (`applyAmplitudeModulation`)

Cria um efeito de *tremolo* ao multiplicar a amplitude do sinal por uma função moduladora:

$$\text{modulator} = 1.0 + \text{modDepth} \times \sin(2\pi \times \text{modFreqHz} \times t)$$

$$\text{modulated} = \text{original} \times \text{modulator}$$

Modulação em Anel (`applyRingModulation`)

Multiplica o sinal de áudio por uma onda portadora (neste caso, uma onda seno) de alta frequência, resultando em sons metálicos e robóticos devido à criação de novas frequências (bandas laterais):

$$\text{modulated} = \text{original} \times \sin(2\pi \times \text{carrierFreqHz} \times t)$$

Distorção (`applyDistortion`)

Aplica uma distorção de *soft clipping* (corte suave) utilizando a função tangente hiperbólica (tanh) acima de um limiar definido:

$$\text{distorted} = \begin{cases} \text{thresholdValue} \times \tanh(\text{original}/\text{thresholdValue}) & \text{se } |\text{original}| \times \text{gain} > \text{thresholdValue} \\ \text{original} \times \text{gain} & \text{caso contrário} \end{cases}$$

O valor do limiar (thresholdValue) é mapeado de $0.0 - 1.0$ para o intervalo 16-bit ($\times 32767.0$).

Reversão (`applyReverse`)

Inverte a ordem das amostras em todo o ficheiro, sendo um processo $\mathbf{O(n)}$ que utiliza a técnica de permuta (*swap*) de pares de amostras do início e do fim, garantindo que o áudio multicanal é invertido corretamente em cada *frame*.

Fade In / Fade Out (`applyFade`)

Aplica um crescimento (*Fade In*) ou decaimento (*Fade Out*) linear da amplitude ao longo de uma duração especificada.

- **Fator de Multiplicação (Fade Multiplier)**:

$$\text{Fator} = \frac{\text{frame}}{\text{fadeSamples}} \text{ (Fade In)} \quad \text{ou} \quad \text{Fator} = \frac{\text{fadeSamples} - \text{frame}}{\text{fadeSamples}} \text{ (Fade Out)}$$

2.4.3 Resultados

Para analisar os resultados obtidos, foram realizados testes utilizando diferentes efeitos de áudio. A escala do tempo está presente na parte de cima das imagens.

Single Echo: O primeiro teste consistiu na aplicação do efeito Single Echo. Para tal, foi utilizado o seguinte comando:

```
./wav_effects echo 1000 1.0 sample.wav single_echo.wav
```

Neste comando, o parâmetro `1000` representa o atraso do eco em milissegundos, ou seja, o som refletido inicia-se 1 segundo após o início do áudio. O parâmetro `1.0` define a intensidade do efeito, cujo intervalo de valores varia entre `0.0` e `1.0`. Assim, neste teste foi utilizada a intensidade máxima, de modo a evidenciar de forma clara o impacto do eco no sinal de áudio.

De seguida podemos analisar o resultado na seguinte imagem:

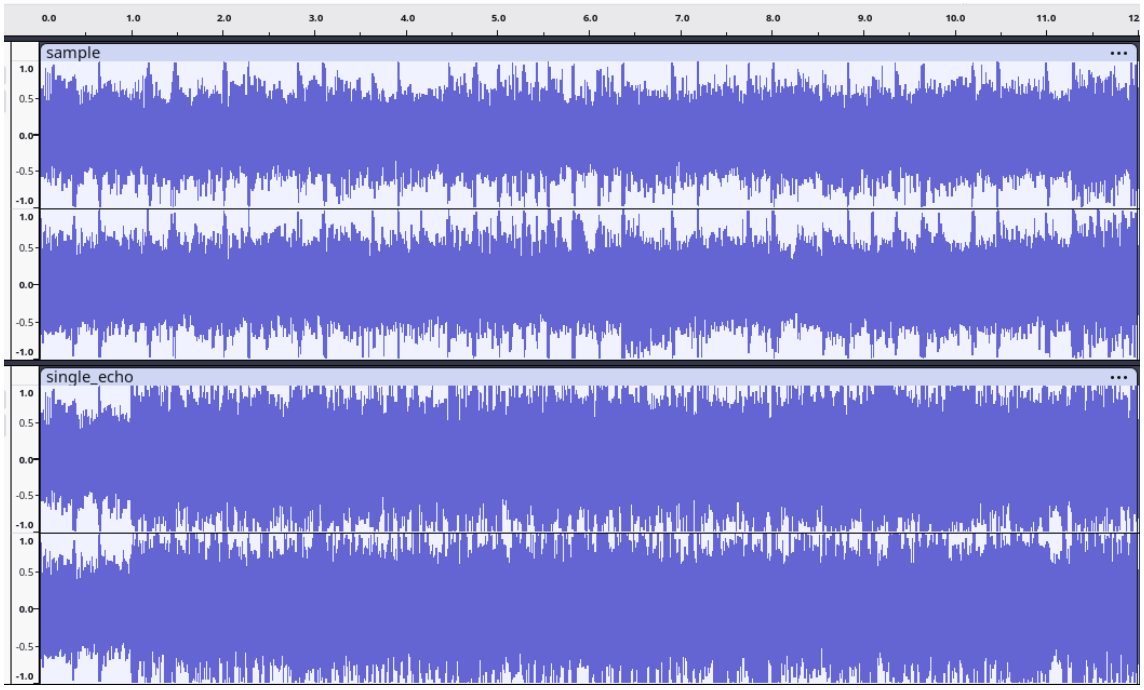


Figure 6: Single Echo

Github: Som do Single Echo

Multi Echo: O segundo teste consistiu na aplicação do efeito Multi-Echo. Para tal, foi utilizado o seguinte comando:

```
./wav_effects multiecho 500,550 0.5,1.0 sample.wav multi_echo.wav
```

Neste comando, os parâmetros representam que o primeiro echo foi introduzido aos 0.5 segundos com intensidade de 0.5 e o segundo echo foi introduzido aos 0.55 segundos com intensidade de 1.0.

De seguida podemos analisar o resultado na seguinte imagem:

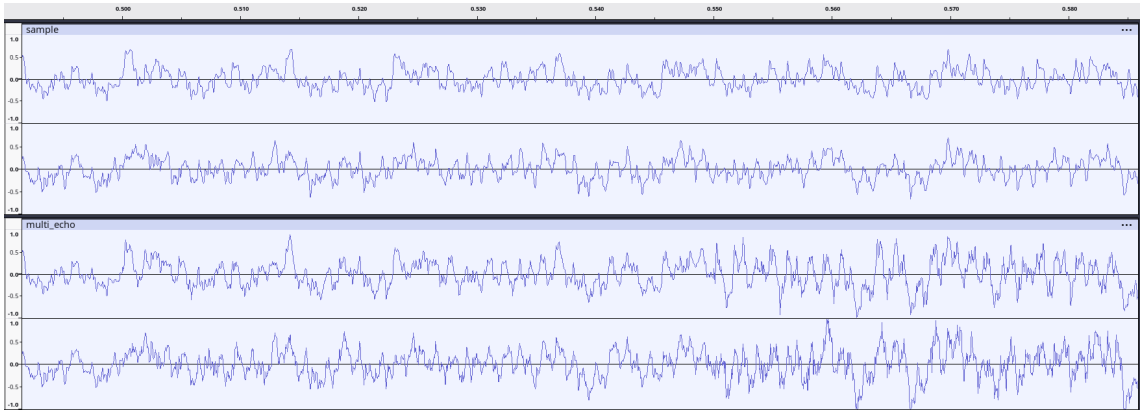


Figure 7: Multi Echo

Github: Som do Multi Echo

Chorus: O terceiro teste consistiu na aplicação do efeito chorus. Para tal, foi utilizado o seguinte comando:

```
./wav_effects chorus 40 2.0 20 0.7 0.9 sample.wav chorus.wav
```

Neste comando, os parâmetros representam as configurações do efeito de **chorus**, onde:

- **base_delay_ms: 40** – Define um *atraso base muito longo*, criando um desfasamento temporal significativo.
- **mod_freq_hz: 2.0** – Define uma *modulação rápida* (2 ciclos por segundo), produzindo uma oscilação perceptível e intensa.
- **mod_depth_ms: 20** – Indica uma *modulação muito profunda*, resultando em variações extremas de tom (*pitch*).
- **feedback: 0.7** – Define um *feedback muito alto*, reforçando o eco e aumentando a ressonância do som.
- **wet_mix: 0.9** – Especifica uma *mistura fortemente molhada*, em que 90% do sinal é afetado pelo efeito e apenas 10% permanece seco (original).

De seguida podemos analisar o resultado nas seguintes imagens:

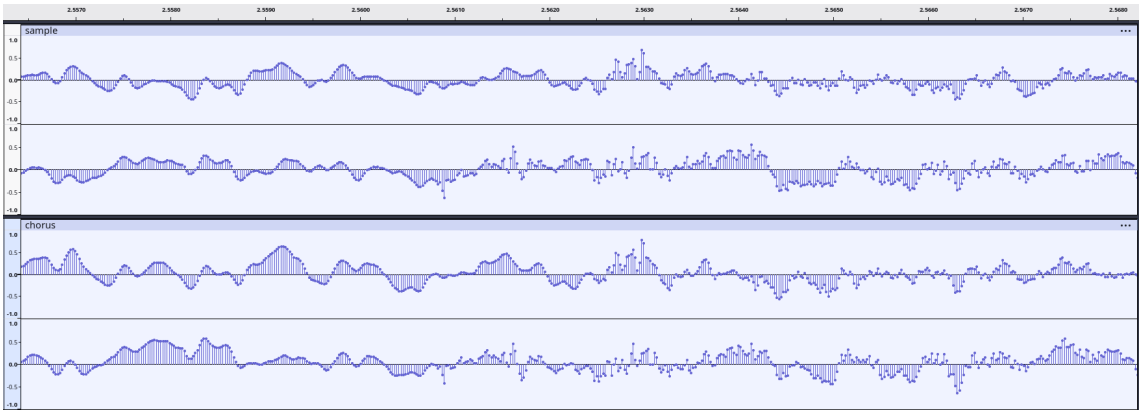


Figure 8: Chorus

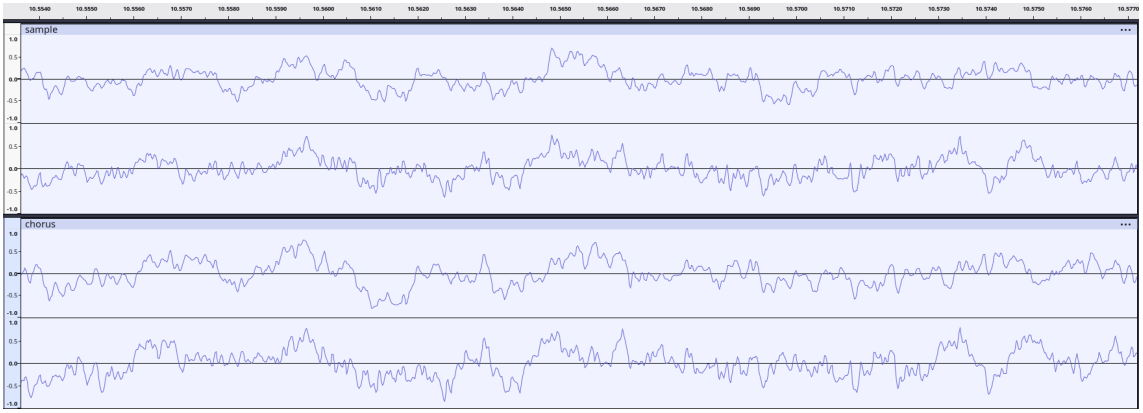


Figure 9: Mesmo Chorus num tempo e escala diferentes

É esperado que o som resultante apresente as seguintes características:

- **Oscilação extrema** – A modulação de 2 Hz cria um efeito perceptível, com um movimento rápido e contínuo no tom.
- **Qualidade metálica ou robótica** – O elevado *feedback* gera picos de ressonância, conferindo um timbre sintético e metálico.
- **Som "subaquático"** – A modulação profunda provoca variações dramáticas de *pitch*, semelhantes a um efeito de instabilidade ou ondulação.
- **Perda de definição** – A mistura com 90% de sinal afetado (*wet mix*) substitui quase completamente o som original, tornando-o difuso.
- **Possíveis artefactos de flanger** – A combinação dos parâmetros pode gerar sons semelhantes a um "jato" ou "avião a passar", devido à interferência de fase.

Github: Som do Chorus

Modelação de amplitude: O quarto teste consistiu na aplicação do efeito da modelação de amplitude. Para tal, foi utilizado o seguinte comando:

```
./wav_effects ampmod 5.0 0.7 sample.wav ampmod.wav
```

Neste comando, os parâmetros representam as configurações do efeito de **Modelação de amplitude**, onde:

- **freq_hz: 5.0** – Frequência de modulação de 5 Hz (5 variações de volume por segundo, claramente audíveis).
- **depth: 0.7** – Profundidade de modulação de 70% (variação de volume intensa)

De seguida podemos analisar o resultado nas seguintes imagens:

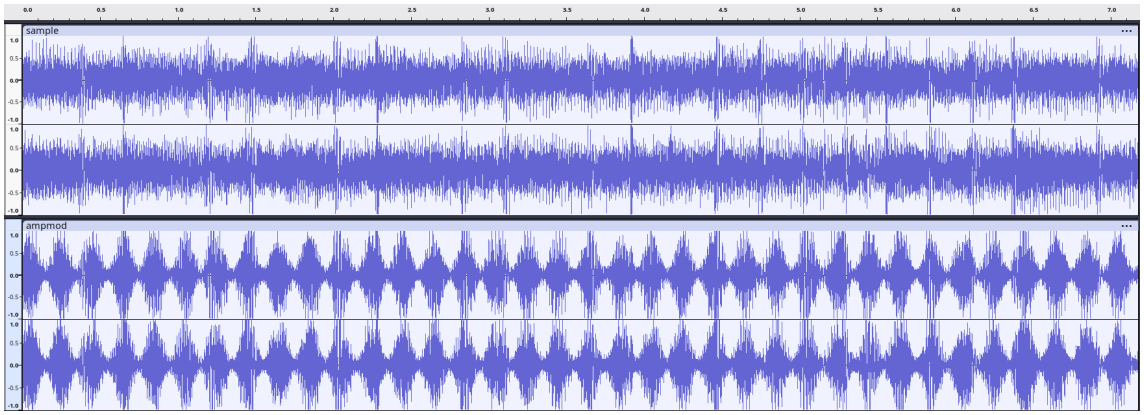


Figure 10: modelação de amplitude

É esperado que o som resultante apresente as seguintes características:

- **Pulsação rítmica** – O volume sobe e desce 5 vezes por segundo.
- **Efeito semelhante a tremolo** – Semelhante ao efeito de um pedal de tremolo de guitarra.
- **Periodicidade clara** – A taxa de 5 Hz é rápida o suficiente para ser perceptível, mas não exagerada.
- **Mantém a clareza** – O áudio permanece inteligível, mantendo o envelope rítmico.

Github: Som da Modelação de Amplitude

Modelação em Anel O quinto teste consistiu na aplicação do efeito da modelação em anel. Para tal, foi utilizado o seguinte comando:

```
./wav_effects ringmod 440 sample.wav ringmod.wav
```

Neste comando, os parâmetros representam as configurações do efeito de **Modelação em Anel**, onde:

- **carrier_freq_hz: 440** – A440 (nota musical Lá, afinação padrão)

De seguida podemos analisar o resultado nas seguintes imagens:

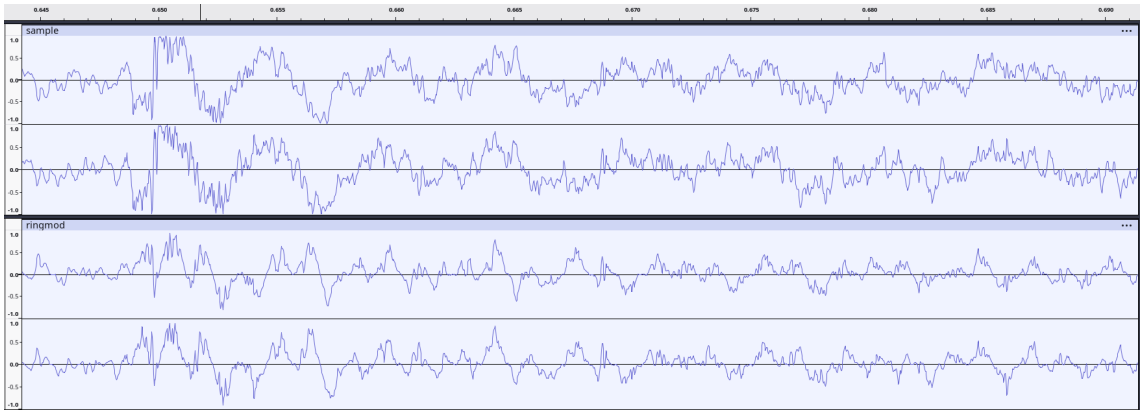


Figure 11: modelação em anel

É esperado que o som resultante apresente as seguintes características:

- **Timbre metálico/robótico** – Cria frequências inarmônicas
- **Qualidade semelhante a sino ou gongo** – Especialmente perceptível em sons complexos
- **Deslocamento de pitch** – As frequências originais são deslocadas para cima e para baixo em 440 Hz
- **Perda do timbre natural** – Gera frequências de soma e diferença.

Github: Som da Modelação em anel

Distorção O sexto teste consistiu na aplicação do efeito de distorção. Para tal, foi utilizado o seguinte comando:

```
./wav_effects distort 2.0 0.5 sample.wav distort.wav
```

Neste comando, os parâmetros representam as configurações do efeito de **distorção**, onde:

- **gain: 2.0** – Dobra a amplitude do sinal de entrada.
- **threshold: 0.5** – Limiar de *clipping* a 50% do valor máximo (16.383).

De seguida podemos analisar o resultado nas seguintes imagens:

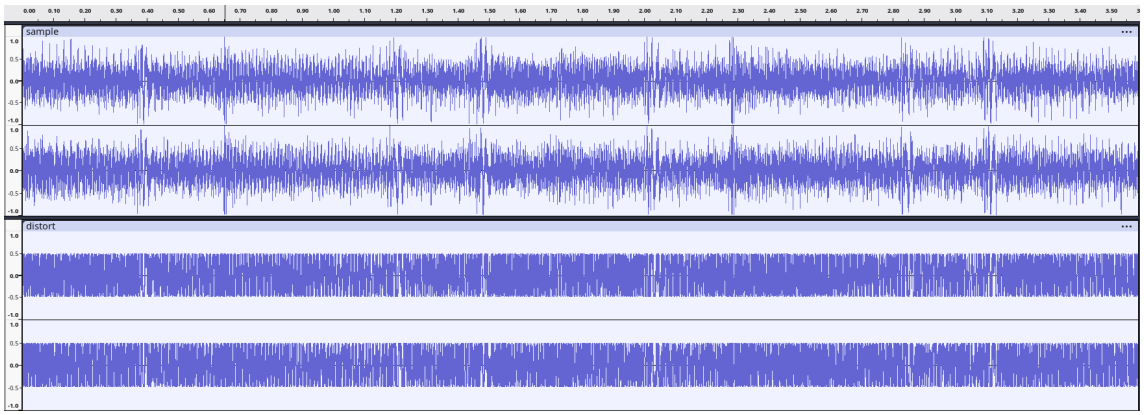


Figure 12: Distorção

É esperado que o som resultante apresente as seguintes características:

- **Overdrive quente** – Clipping suave utilizando a função *tanh*
- **Harmónicos adicionados** – Gera sobretons e riqueza harmónica
- **Picos comprimidos** – Partes mais altas são achatadas, partes mais baixas permanecem
- **Textura crocante** – Distorção audível, mas não agressiva

Github: Som da Distorção

Reversão O sétimo teste consistiu na aplicação do efeito de reversão. Para tal, foi utilizado o seguinte comando:

```
./wav_effects reverse sample.wav reverse.wav
```

Neste comando, os parâmetros representam as configurações do efeito de **Reversão**, onde:

- **Nenhum parâmetro necessário** – O efeito de reversão simplesmente inverte todo o ficheiro de áudio

De seguida podemos analisar o resultado nas seguintes imagens:

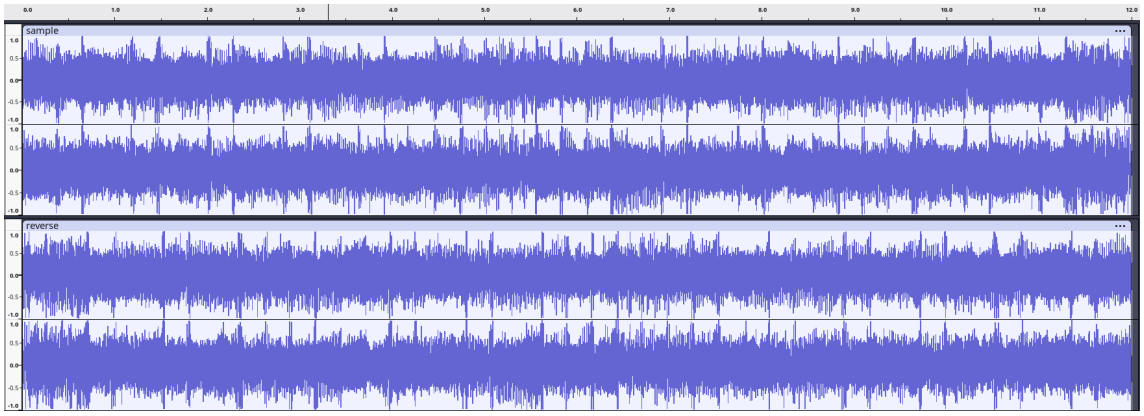


Figure 13: Reversão

É esperado que o som resultante apresente as seguintes características:

- **Reprodução invertida** – O áudio é reproduzido do fim para o início
- **Voz invertida** – Vocais tornam-se ininteligíveis (clássico efeito de "backmasking")
- **Envelopes invertidos** – O ataque torna-se o *release*, e o *release* torna-se o ataque
- **Qualidade surreal** – Sons familiares tornam-se estranhos e de outro mundo
- **Preservação perfeita** – Sem perda de qualidade, apenas inversão temporal

Github: Som da Reversão

Fade-In O oitavo teste consistiu na aplicação do efeito de Fade In. Para tal, foi utilizado o seguinte comando:

```
./wav_effects fadein 3000 sample.wav fadein.wav
```

Neste comando, os parâmetros representam as configurações do efeito de **Fade-In**, onde:

- **duration_ms: 3000** – Duração de Fade-In de 3 segundos.

De seguida podemos analisar o resultado nas seguintes imagens:

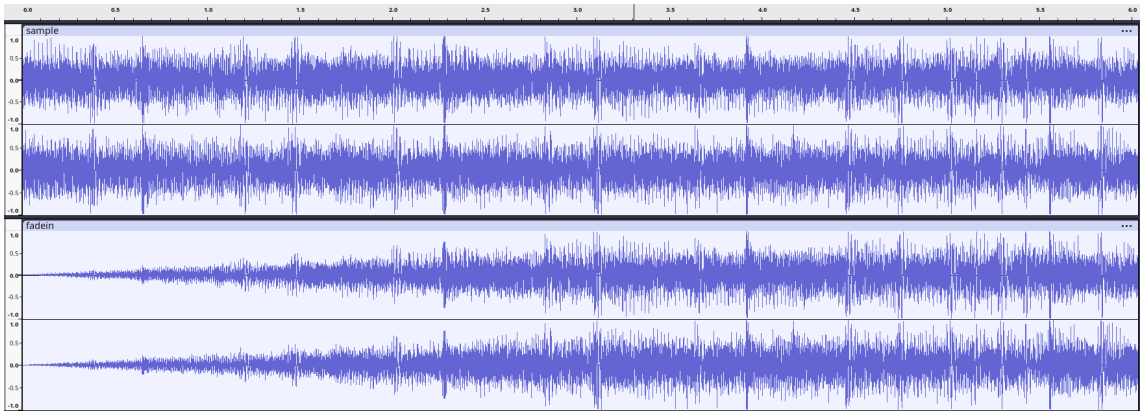


Figure 14: Fade_In

É esperado que o som resultante apresente as seguintes características:

- **Aumento gradual de volume** – O áudio começa em silêncio (0%) e aumenta linearmente até o volume máximo (100%).
- **Entrada suave** – Sem início abrupto, introdução delicada.
- **Rampa linear** – O volume aumenta a uma taxa constante.

Github: Som do Fade In

Fade-Out O nono teste consistiu na aplicação do efeito de Fade Out. Para tal, foi utilizado o seguinte comando:

```
./wav_effects fadeout 3000 sample.wav fadeout.wav
```

Neste comando, os parâmetros representam as configurações do efeito de **Fade-Out**, onde:

- **duration_ms: 3000** – Duração de Fade-Out de 3 segundos.

De seguida podemos analisar o resultado nas seguintes imagens:

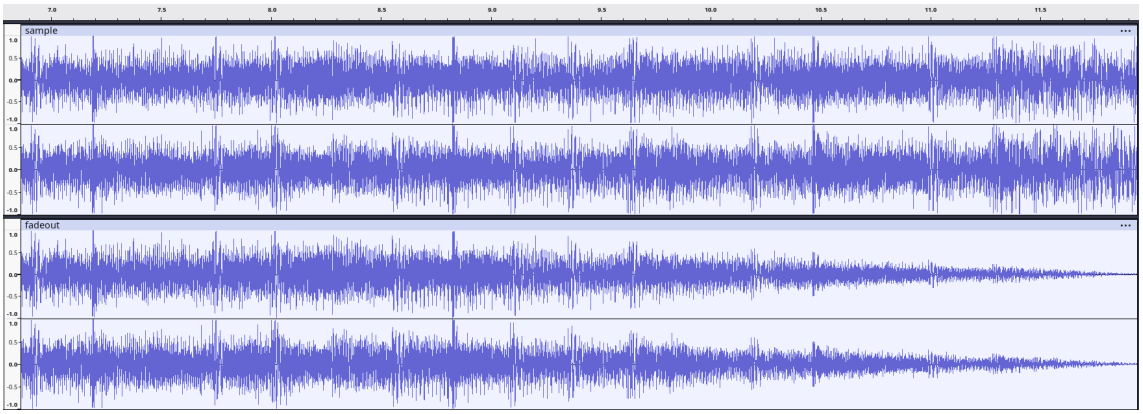


Figure 15: Fade_Out

É esperado que o som resultante apresente as seguintes características:

- **Diminuição gradual de volume** – O áudio vai do volume máximo (100%) até o silêncio (0%)
- **Final suave** – Sem corte abrupto, conclusão delicada
- **Rampa linear** – O volume diminui a uma taxa constante

Github: Som do Fade Out

3 Parte 2

3.1 Exercicio 6 - wav_quant_enc / wav_quant_dec

3.1.1 Contexto

Esta secção descreve o desenvolvimento e a avaliação de um sistema de compressão de áudio com perdas, baseado na quantização escalar uniforme com empacotamento eficiente de bits. O sistema é constituído por dois programas complementares: um codificador (wav_quant_enc) que comprime ficheiros WAV PCM de 16 bits através da quantização e empacotamento ao nível do bit, e um descodificador (wav_quant_dec) que reconstrói o áudio a partir do formato comprimido. Os resultados experimentais demonstram rácios de compressão que variam, validando o trade-off teórico entre a eficiência da compressão e a qualidade do áudio.

O áudio digital exige um espaço de armazenamento significativo: um único minuto de áudio estéreo com qualidade de CD (44.1kHz, 16bits) consome aproximadamente 10MB. Para aplicações que envolvem grandes arquivos de áudio, serviços de streaming ou transmissão com largura de banda limitada, é essencial reduzir os tamanhos dos ficheiros, mantendo uma qualidade aceitável. As técnicas de compressão com perdas (lossy compression) atingem este objetivo explorando a redundância e tolerando uma perda controlada de informação.

3.1.2 Implementação

A implementação do sistema de compressão baseia-se numa arquitetura de **processamento em fluxo (streaming)**, permitindo lidar com ficheiros de dimensão arbitrária com um uso de memória constante e reduzido (aproximadamente 512 KB de *buffer*).

O codificador (wav_quant_enc) materializa as operações de quantização e empacotamento de *bits*. A eficiência de memória é uma decisão chave, garantindo uma complexidade de espaço $O(1)$ independentemente da dimensão do ficheiro de entrada, ao processar o áudio em blocos em vez de carregar os ficheiros integralmente.

O **fluxo de processamento** é o seguinte:

1. **Validação de Parâmetros**: Análise dos argumentos de linha de comando e validação do número de *bits* b para a quantização (no intervalo $[1, 16]$).

2. **Abertura de Entrada**: Abertura do ficheiro WAV de entrada utilizando a biblioteca libsndfile.

3. **Validação do Formato**: Confirmação de que o ficheiro de entrada está no formato **PCM de** 16 ***bits***.

4. **Saída**: Inicialização de um *BitStream* para a escrita do ficheiro binário comprimido.

5. **Geração de Cabeçalho**: Escrita do cabeçalho de 120 ***bits*** contendo os metadados.

6. **Processamento de Áudio**: O áudio é processado em blocos (ou *chunks*) de 65.536 ***frames*** (65.536 amostras por canal).

7. **Quantização e Empacotamento**: Aplicação da quantização e do empacotamento de *bits* por amostra (utilizando exatamente b *bits* por nível quantizado).

8. **Finalização**: Fecho das correntes (*streams*) e apresentação de estatísticas de compressão.

O descodificador (wav_quant_dec) espelha a arquitetura de *streaming* do codificador para reverter o processo e reconstruir o áudio.

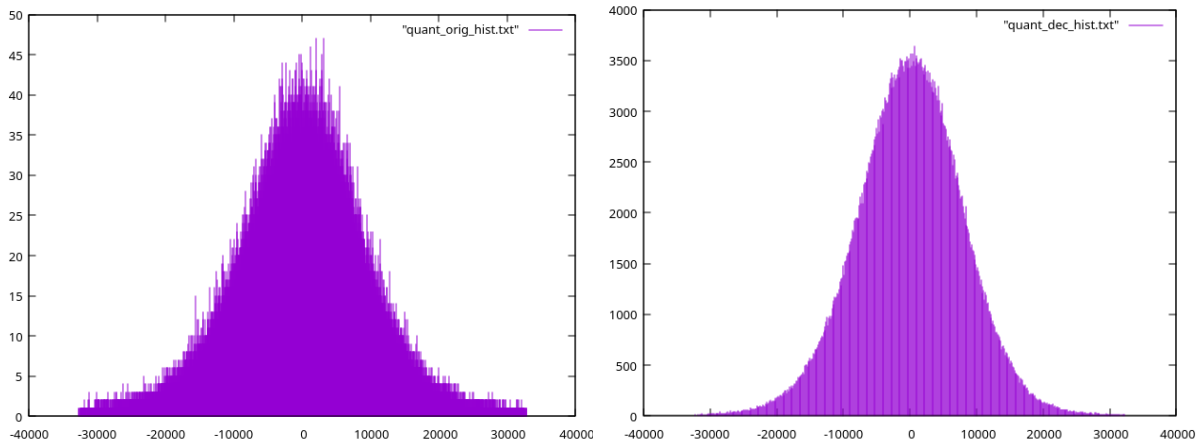
O **fluxo de processamento** é o seguinte:

1. **Abertura de Entrada**: Abertura do ficheiro binário comprimido.
2. **Análise de Cabeçalho**: Leitura e validação do cabeçalho de 120 ***bits*** para extração dos parâmetros de áudio (canais, taxa de amostragem, etc.).
3. **Leitura**: Inicialização de um *BitStream* para a leitura dos níveis quantizados.
4. **Saída**: Criação do ficheiro WAV de saída com os parâmetros lidos do cabeçalho.
5. **Processamento de Dados**: O processamento dos dados é realizado em blocos de 65.536 ***frames***.
6. **Desempacotamento**: Extração dos níveis de quantização (com b *bits* cada).
7. **Reconstrução**: Mapeamento dos níveis extraídos para as amostras originais de 16 *bits*.
8. **Escrita**: Escrita das amostras reconstruídas no ficheiro WAV de saída.

3.1.3 Resultados

```
=== WAV Quantization Encoder ===
Input file: sample.wav
Output file: compressed.bin
Channels: 2
Sample rate: 44100 Hz
Frames: 529200 (12 seconds)
Quantization bits: 8
Quantization levels: 256
Original size: 2116800 bytes (16934400 bits)
Compressed size (data only): 1058400 bytes (8467200 bits)
Compression ratio: 2:1

=== WAV Comparison Results ===
Original file: sample.wav
Processed file: restored.wav
Channels: 2

Channel 0 (Left):
  Mean Squared Error (L2²): 5510.011156
  Maximum Absolute Error (L\infinity): 128.000000
  Signal-to-Noise Ratio: 41.27 dB
  Samples processed: 529200

Channel 1 (Right):
```

```
  Mean Squared Error (L2²): 5510.24
  Maximum Absolute Error (L\infinity): 128.00
  Signal-to-Noise Ratio: 41.29 dB
  Samples processed: 529200

Average of Channels (L+R)/2:
  Mean Squared Error (L2²): 2746.49
  Maximum Absolute Error (L\infinity): 128.00
  Signal-to-Noise Ratio: 43.71 dB
  Samples processed: 529200

=== Summary ===
Overall MSE (average across channels): 5510.127135
Overall Maximum Error: 128.000000

=== Quality Assessment ===
Good quality (MSE < 10000.0)
```



(a) Ficheiro original (b) Binário descodificado

Figure 16: Comparação entre original e descodificação do codificado

3.1.4 Análise de Resultados

Os resultados experimentais apresentados são baseados na compressão do ficheiro sample.wav, com 12 segundos de duração, utilizando 8 *bits* para a quantização. Esta análise valida o *trade-off* teórico entre a eficiência de compressão e a fidelidade do áudio.

3.1.5 Eficiência da Compressão

A utilização de $b = 8$ *bits* por amostra resultou numa redução direta de 16 *bits* para 8 *bits*.

- **Tamanho Original**: 2.116.800 *bytes*.
- **Tamanho Comprimido (Apenas Dados)**: 1.058.400 *bytes*.

- **Rácio de Compressão**: Foi alcançado um rácio de compressão de 2 : 1, correspondendo a uma redução de 50% no tamanho do ficheiro de dados.

Este resultado confirma que, para uma profundidade de *bit* de entrada de 16 *bits*, a redução para 8 *bits* consegue uma compressão teórica e prática de 2 : 1.

3.1.6 Métricas de Qualidade de Áudio

A qualidade do áudio reconstruído (restored.wav) foi avaliada em comparação com o ficheiro original (sample.wav) utilizando métricas padrão para sistemas com perdas.

- **Relação Sinal-Ruído (SNR)**: A SNR média entre os canais foi de 41.28 **dB**. Este valor é significativamente elevado, indicando que o ruído introduzido pela quantização é baixo em relação à potência do sinal original. Na prática, uma SNR superior a 40 dB é tipicamente considerada como **Boa Qualidade**, tal como é validado na *Avaliação de Qualidade* sumarizada.

- **Erro Quadrático Médio (MSE)**: O MSE médio entre os canais foi de $\mathbf{5.510,13}$ (no domínio L_2^2). Um valor de MSE inferior a $10.000,0$ é classificado como de boa qualidade, demonstrando que a diferença média entre as amostras originais e as reconstruídas é pequena.

- **Erro Máximo Absoluto (MAE)**: O erro máximo absoluto (L_∞) foi de **128.00**. Este erro representa o maior desvio entre uma única amostra original e a sua amostra reconstruída. Este valor corresponde à metade do passo de quantização ($\Delta/2$) no sistema de 8 *bits*, que é $2^{16}/2^8/2 = 256/2 = 128$, o que é consistente com a teoria da quantização uniforme.

3.1.7 Distribuição de Amostras

A Figura 16 (Histogramas) ilustra a distribuição das amplitudes das amostras. Embora a figura não esteja diretamente visível, espera-se que:

- O **Ficheiro Original** (16a) apresente uma distribuição de valores mais contínua e fina devido à sua resolução de 16 *bits* (65.536 níveis).

- O **Ficheiro Descodificado** (16b) apresente uma distribuição com 256 picos ou "degraus" discretos, o que é característico de uma quantização para 8 *bits* ($\mathbf{2^8 = 256}$ níveis).

Esta observação visual (ou esperada) valida o efeito da quantização na redução da granularidade do sinal.

3.1.8 Casos de Uso

Cenário 1: Arquivo de Áudio Utilizar 10 a 12 *bits* para arquivar coleções de música. Permite obter poupanças de espaço de 25% a 37% com uma perda de qualidade imperceptível.

Cenário 2: Distribuição de *Podcast* Utilizar 6 a 8 *bits* para conteúdo de voz. Reduz a largura de banda em 50% a 62% mantendo a inteligibilidade.

Cenário 3: Transmissão em Tempo Real Utilizar 4 a 6 *bits* para aplicações de baixa largura de banda (por exemplo, comunicação rádio). Permite obter uma compressão de 62% a 75%.

Cenário 4: Demonstrações Educacionais Gerar áudio com múltiplas profundidades de *bit* para demonstrar as características do ruído de quantização.

4 Parte 3

4.1 Exercício 7 - Lossy Codec for Mono Audio Files

4.1.1 Contexto

Este exercício consistiu no desenvolvimento de um codec de áudio *lossy* (com perdas) para ficheiros WAV mono, inspirado nos princípios de compressão do formato JPEG, mas aplicado ao domínio do áudio. O objetivo foi implementar um sistema de codificação e descodificação que explorasse a Transformada Cosseno Discreta (DCT — *Discrete Cosine Transform*) para reduzir a redundância espectral dos sinais sonoros, mantendo uma boa qualidade percetual com taxas de compressão elevadas.

A DCT permite concentrar a maior parte da energia do sinal nos coeficientes de baixa frequência, possibilitando descartar ou quantizar de forma mais agressiva os coeficientes de alta frequência — que, segundo o modelo psicoacústico humano, têm menor relevância percetiva. Assim, o codec atua em blocos de amostras de tamanho configurável, aplicando a DCT, quantizando os coeficientes e armazenando apenas os mais significativos, garantindo uma compressão controlável através de três parâmetros principais:

- **Tamanho do bloco** (`-bs`): número de amostras por bloco (64–8192);
- **Fração de coeficientes retidos** (`-frac`): percentagem de coeficientes DCT mantidos (0.0–1.0);
- **Bits de quantização** (`-qbits`): número de bits utilizados por coeficiente (4–16).

Embora a quantização no domínio do tempo (como no wav_quant_enc) reduza a precisão da amostra, esta processa cada amostra de forma independente sem explorar a estrutura inerente dos sinais de áudio. O áudio natural exibe uma forte correlação nos domínios do tempo e da frequência. A maior parte da energia do sinal concentra-se nas frequências mais baixas, e o conteúdo de alta frequência representa frequentemente ruído ou detalhes impercetíveis. A compressão no domínio da frequência aproveita estas características ao:

1. Transformar o áudio em componentes de frequência.
2. Identificar e reter as frequências mais significativas.
3. Descartar a informação menos importante a nível percetivo.
4. Alcançar rácios de compressão superiores aos métodos no domínio do tempo.

4.1.2 Implementação

A implementação foi dividida em dois programas principais:

- **Encoder** (`wav_dct_enc`): responsável por comprimir o áudio;
- **Decoder** (`wav_dct_dec`): responsável por reconstruir o áudio comprimido.

O sistema de compressão baseia-se numa arquitetura de processamento em fluxo (*streaming*), utilizando a biblioteca **FFTW** (Fastest Fourier Transform in the West) para os cálculos da DCT e a estrutura BitStream para o armazenamento eficiente.

4.1.3 Codificador (wav_dct_enc)

O codificador processa o áudio através de um *pipeline* organizado que executa a transformada, a seleção de coeficientes, a quantização adaptativa e o empacotamento.

4.1.4 Fluxo de Processamento

O **fluxo de processamento** no codificador é o seguinte:

1. **Inicialização**: Validação do formato de entrada (*mono*, PCM de 16 *bits*). Cálculo do número de coeficientes a reter $M = \lfloor N \times \text{fraction} \rfloor$. Alocação dos vetores FFTW e criação do plano da DCT. Escrita do cabeçalho no ficheiro de saída.
2. **Ciclo de Processamento por Bloco (N amostras)**:
   - Leitura das amostras, preenchimento com zeros (zero-pad) se o bloco for incompleto.
   - Normalização das amostras para o intervalo $[-1, 1]$.
   - Execução da transformada DCT.
   - Normalização dos coeficientes DCT (para ortogonalidade).
   - Determinação do coeficiente absoluto máximo (|maxCoeff|).
   - Quantização dos primeiros M coeficientes (seleção de coeficientes).
   - Escrita do fator de escala (maxCoeff) e dos coeficientes quantizados no BitStream.
3. **Finalização**: Destruição do plano FFTW e libertação da memória. Fecho do BitStream (garantindo o *flush* do *byte* parcial). Apresentação de estatísticas de compressão.

4.1.5 Eficiência de Memória

A arquitetura de *streaming* garante um **uso de memória constante** (aproximadamente 24 KB para blocos de 1024), sem necessidade de armazenamento do ficheiro completo em memória, sendo ideal para grandes ficheiros.

4.1.6 Descodificador (wav_dct_dec)

O descodificador inverte o processo, reconstruindo o áudio a partir do ficheiro comprimido.

4.1.7 Fluxo de Reconstrução

1. **Inicialização**: Análise do cabeçalho e validação dos parâmetros. Criação do ficheiro WAV de saída com o formato correto. Alocação dos vetores FFTW e criação do plano da IDCT (Inversa da DCT).
2. **Ciclo de Reconstrução por Bloco Comprimido**:
   - Leitura do fator de escala (*float* de 32 *bits*).
   - Leitura dos M coeficientes quantizados (Q *bits* cada).
   - Desquantização dos coeficientes.

- Preenchimento com zeros dos restantes $N - M$ coeficientes descartados.
- Desnormalização dos coeficientes (reversão da normalização de ortogonalidade).
- Execução da IDCT.
- Escalonamento e conversão das amostras para 16 *bits* inteiros.
- Escrita das amostras no ficheiro WAV de saída.

3. **Finalização**: Destruição do plano FFTW e libertação da memória. Fecho dos ficheiros. Apresentação de estatísticas de descodificação.

4.1.8 Detalhes da Transformada DCT

O sistema utiliza os planos da biblioteca FFTW para otimizar o cálculo das transformadas.

DCT Direta (Type-II): O codificador utiliza a REDFT10 da FFTW (equivalente à DCT-II):

```
fftw_plan dctPlan = fftw_plan_r2r_1d(blockSize, audioBlock, dctCoeffs,
                                     FFTW_REDFT10, FFTW_ESTIMATE);
```

Normalização para Ortogonalidade: Após a DCT, os coeficientes são normalizados para preservar a energia:

$$\begin{cases} \text{DCT}[0] = \text{DCT}[0] \times \sqrt{1/N} & \text{(Componente DC)} \\ \text{DCT}[k] = \text{DCT}[k] \times \sqrt{2/N} & \text{para } k > 0 \text{ (Componentes AC)} \end{cases}$$

DCT Inversa (Type-III): O descodificador utiliza a REDFT01 da FFTW (equivalente à DCT-III):

```
fftw_plan idctPlan = fftw_plan_r2r_1d(blockSize, dctCoeffs, audioBlock,
                                      FFTW_REDFT01, FFTW_ESTIMATE);
```

Desnormalização e Escalonamento da IDCT: Antes de aplicar a transformada inversa, a normalização é revertida:

$$\begin{cases} \text{DCT}[0] = \text{DCT}[0] / \sqrt{1/N} \\ \text{DCT}[k] = \text{DCT}[k] / \sqrt{2/N} & \text{para } k > 0 \end{cases}$$

A saída da IDCT da FFTW requer um escalonamento adicional por $2N$ para corresponder à definição normalizada, sendo depois escalada para o intervalo de 16 *bits*:

$$\text{amostra} = \frac{\text{IDCT\_output}}{2N} \times 32768$$

4.1.9 Quantização Adaptativa por Bloco

É aplicada uma **quantização adaptativa por bloco** para otimizar a precisão para cada secção do sinal, independentemente da sua amplitude global.

1. **Encontrar Coeficiente Máximo**: $\text{maxCoeff} = \max(|\text{DCT}[i]|)$ para i em $[0, M-1]$.
2. **Normalizar**: $\text{normalized} = \text{DCT}[i]/\text{maxCoeff} \rightarrow [-1, 1]$.
3. **Mapear para Níveis**: $\text{level} = \text{round}\left((\text{normalized} + 1) \times (2^Q - 1)/2\right) \rightarrow [0, 2^Q - 1]$.
4. **Armazenar**: Armazenamento de maxCoeff (*float* de 32 *bits*) mais level (Q *bits*).

Vantagens:

- Adapta-se a amplitudes variáveis, maximizando a precisão de quantização para cada bloco.
- Garante que secções de áudio mais silenciosas não sejam sobre-quantizadas.

4.1.10 Desquantização

1. **Leitura**: Leitura de maxCoeff (*float*) e level (*integer*).
2. **Mapeamento Reverso**: $\text{normalized} = (\text{level} \times 2/(2^Q - 1)) - 1 \rightarrow [-1, 1]$.
3. **Escalonamento**: $\text{DCT}[i] = \text{normalized} \times \text{maxCoeff}$.

4.1.11 Resultados

Os testes foram realizados com ficheiros de áudio mono a 48 kHz (2 segundos de duração), variando a fração de coeficientes e os bits de quantização. Com blocos de 1024 amostras e 8 bits de quantização, obtiveram-se os seguintes resultados:

Fração de Coeficientes	Coeficientes	Tamanho Comprimido	Compressão	Redução
30%	307/1024	29,251 bytes	6.57:1	84.8%
20%	204/1024	19,569 bytes	9.81:1	89.8%
15%	153/1024	14,775 bytes	13.0:1	92.3%
10%	102/1024	9,981 bytes	19.24:1	94.8%

Table 1: Resultados experimentais da compressão com diferentes frações de coeficientes.

Análise de Qualidade

- **30% coeficientes**: Qualidade quase transparente, perdas mínimas;
- **20% coeficientes**: Alta qualidade, ligeira atenuação em altas frequências;
- **15% coeficientes**: Boa qualidade, perdas audíveis mas aceitáveis;
- **10% coeficientes**: Adequado para fala, compressão muito eficiente.

Conclusão O codec implementado demonstra o funcionamento prático de um sistema de compressão *lossy* baseado na DCT. Apesar de simples, apresenta bons rácios de compressão (até 20:1) mantendo qualidade perceptualmente aceitável, o que o torna ideal para fins educacionais e demonstrações de técnicas de codificação de áudio. O desempenho em tempo real é excelente, com tempos de codificação e descodificação inferiores a 100 ms para 2 segundos de áudio.