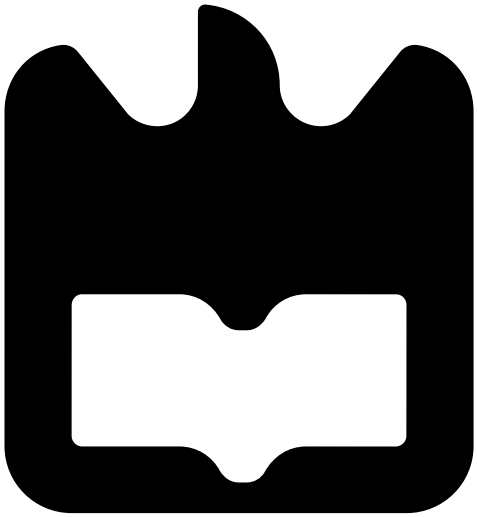

Relatório do Trabalho Laboratorial nº 1

Informação e Codificação (2025/26)

Pedro Miguel Miranda de Melo (114208)
Departamento de Eletrónica, Telecomunicações e Informática (DETI)
Universidade de Aveiro

Outubro de 2025

Conteúdo

1 Introdução

Este relatório descreve os passos e as decisões tomadas no desenvolvimento do software para o Trabalho Laboratorial nº 1 da unidade curricular de Informação e Codificação (2025/26), do Departamento de Eletrónica, Telecomunicações e Informática (DETI) da Universidade de Aveiro. O trabalho foi implementado em C++, utilizando a biblioteca `libsndfile` e o *wrapper* `sndfile.hh` para manipulação de ficheiros áudio WAV. O código-fonte completo do projeto está disponível publicamente no repositório GitHub: `https://github.com/pedromelo1316/IC-Trabalho1`.

2 Parte I: Ferramentas de Análise e Quantização

2.1 Programa `wav_hist`

A classe `WAVHist` foi modificada para calcular e apresentar o histograma dos canais MID e SIDE, bem como suportar *bins* mais largos (2^k valores).

2.1.1 Compilação, Execução e Visualização

A compilação do executável `wav_hist` é realizada utilizando o `g++` e ligando à biblioteca `libsndfile`.

A utilização do programa em linha de comando segue o seguinte formato, onde os argumentos são opcionais:

```
./wav_hist <input file> [channel] [k] [--save]
```

Listing 1: Sintaxe de Uso do wav_hist (Argumento Channel Opcional)

O parâmetro `k` define o expoente para o tamanho do bin (2^k). Se o argumento `channel` for omitido, o programa analisa todos os canais.

Exemplos de Teste Abaixo estão exemplos de comandos usados para testar as funcionalidades com um ficheiro estéreo (`sample.wav`), demonstrando as diversas opções de bin size e exportação.

1. **Modo Dump (Output Terminal - Todos os Canais):** Executa o programa sem especificar canal ou o expoente k. Os histogramas de **L, R, MID e SIDE** são impressos no terminal na resolução máxima (bin size 1, $k = 0$ por omissão).

   ```
   ./wav_hist sample.wav
   ```

2. **Exportação da Linha Base (Bin size 1):** Utiliza o flag `-save` para exportar os histogramas de **L, R, MID e SIDE** para ficheiros `.txt` na resolução máxima.

   ```
   ./wav_hist sample.wav --save
   ```

   Os histogramas gerados podem ser visualizados na Figura 5, que apresenta os gráficos para cada canal com bin size 1.

2.1.2 Histogramas com Bin Size 1 (Resolução Máxima)

3. **Exportação com Bins mais largos (k = 6):** Exporta os histogramas (L, R, MID, SIDE) para ficheiros `.txt` com o expoente $k = 6$, resultando num *bin size* de $2^6 = 64$.

   ```
   ./wav_hist sample.wav 6 --save
   ```

   Os histogramas gerados podem ser visualizados na Figura 10, que apresenta os gráficos para cada canal com bin size 64.

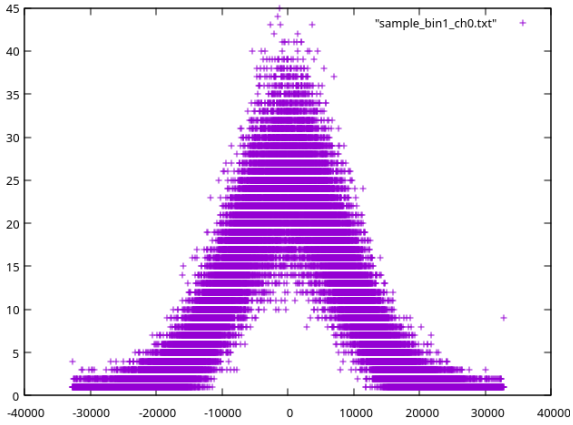

Figura 1: Canal Esquerdo (L)

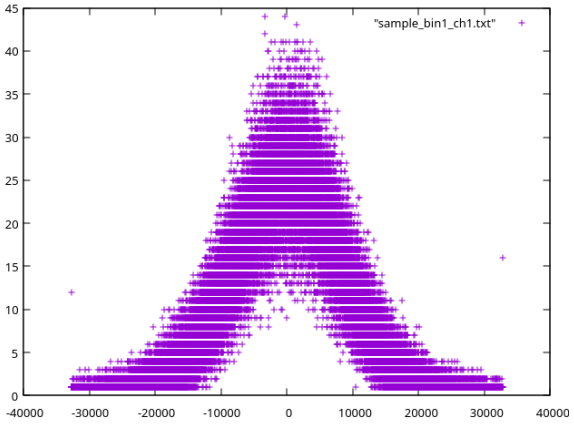

Figura 2: Canal Direito (R)

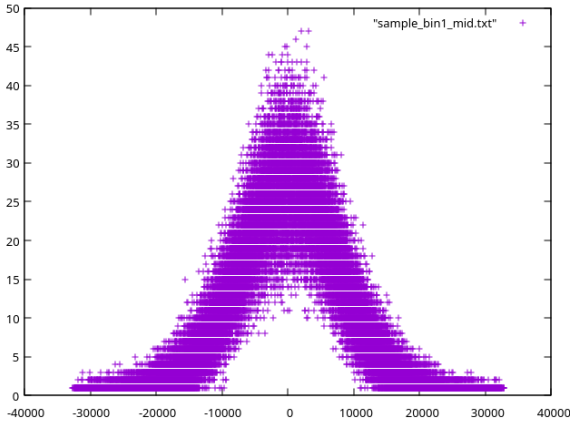

Figura 3: Canal MID

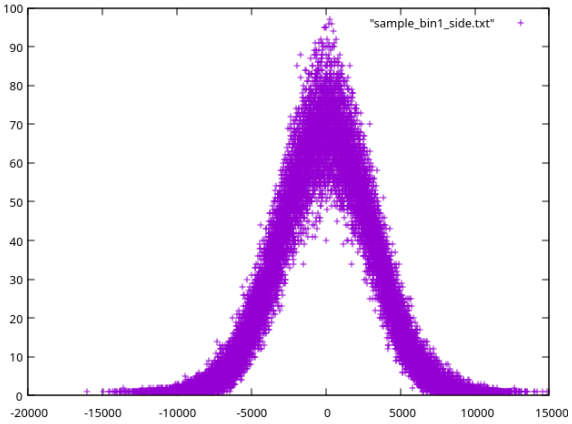

Figura 4: Canal SIDE

Figura 5: Histogramas dos canais L, R, MID e SIDE com bin size 1 (k=0)

2.1.3 Histogramas com Bin Size 64 (k=6)

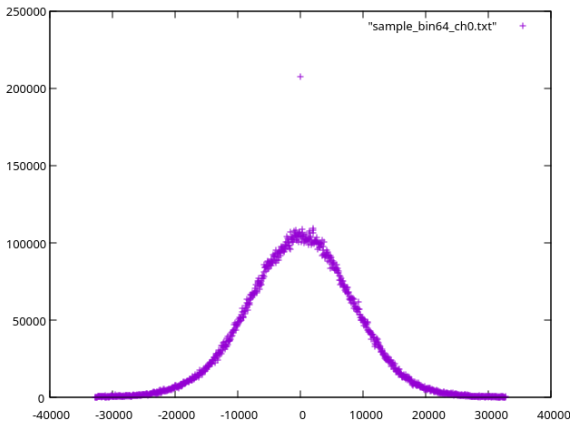

Figura 6: Canal Esquerdo (L)

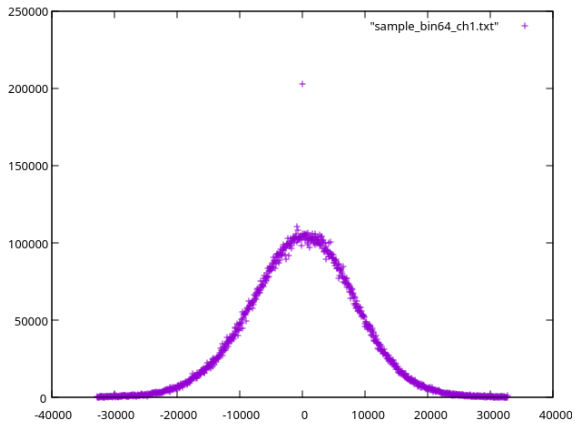

Figura 7: Canal Direito (R)

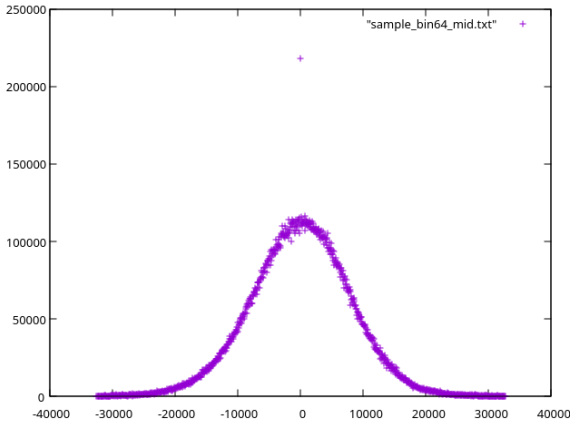

Figura 8: Canal MID

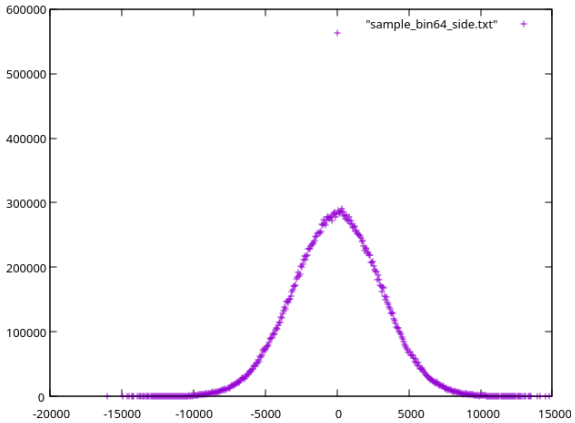

Figura 9: Canal SIDE

Figura 10: Histogramas dos canais L, R, MID e SIDE com bin size 64 (k=6)

4. **Dump Terminal (Canal Específico + MID/SIDE):** Exibe no terminal o histograma do canal Direito (1), do canal MID e do canal SIDE, com o expoente $k = 4$ (bin size 16).

```
./wav_hist sample.wav 1 4
```

Visualização de Histograma Os dados exportados via flag `-save` são representados graficamente, por exemplo, usando o `gnuplot`. Os ficheiros de saída terão o formato `<filename>_bin<size>_ch<channel>.txt`.

```
gnuplot
# Exemplo para comparar o canal Esquerdo (L) na resolucao maxima e bin size 64:
plot "sample_bin1_ch0.txt" using 1:2 with points title "L_bin_1", \
     "sample_bin64_ch0.txt" using 1:2 with points title "L_bin_64"
```

Listing 2: Comando de visualização Gnuplot para comparação de Bins mais largos

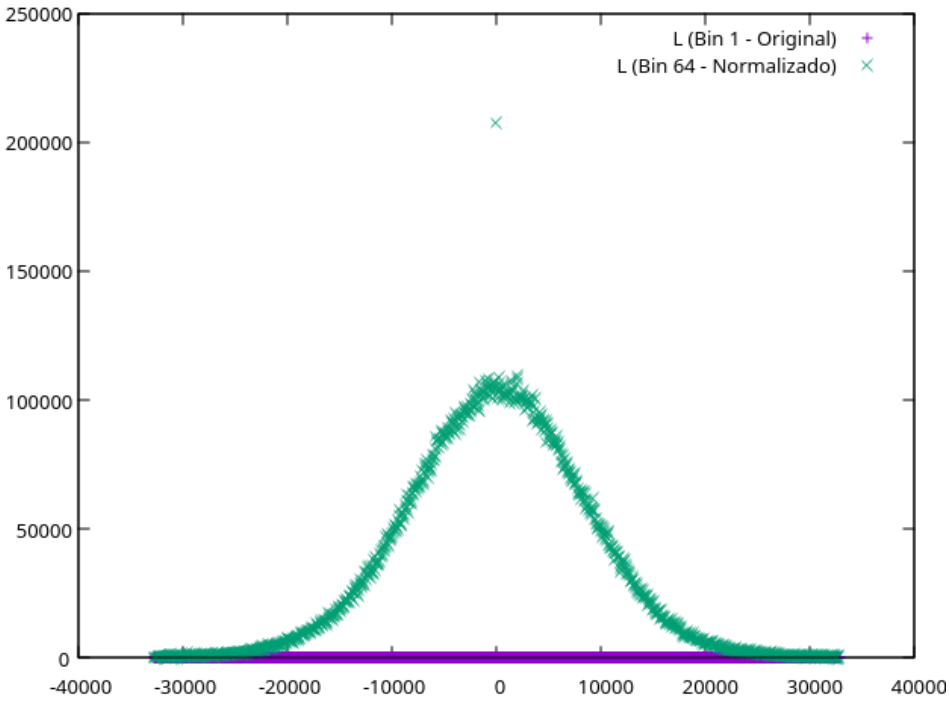

Figura 11: Comparação dos histogramas do canal esquerdo (L) com bin size 1 e bin size 64

Análise da Comparação de Bins A Figura 11 compara diretamente o histograma do canal esquerdo obtido com a resolução máxima (`bin size = 1`) e com *bins* mais largos (`bin size = 64`). A análise desta comparação permite retirar duas conclusões importantes:

- **Preservação da Forma Geral:** Mesmo com a aglutinação de 64 valores de amostra num único *bin*, a forma geral da distribuição de probabilidade é claramente preservada. A maior concentração de amostras em torno do valor zero e a decadência simétrica para valores de maior amplitude são visíveis em ambos os gráficos.
- **Perda de Detalhe:** A utilização de *bins* mais largos resulta numa perda de detalhe de grão fino. O histograma com `bin size=1` revela picos de ocorrência para valores de amostra específicos, enquanto o de `bin size=64` suaviza estas variações, mostrando apenas a tendência geral da distribuição.

Esta técnica é útil para analisar a distribuição macroscópica da energia do sinal sem ruído visual, onde se reduz a precisão da informação em troca de uma representação mais simples dos dados.

2.1.4 Canais MID e SIDE e Recuperação Exata

Os canais MID (*mono*) e SIDE (*diferença*) são calculados através da **divisão inteira**:

$$\text{MID} = \left\lfloor \frac{L+R}{2} \right\rfloor \tag{1}$$

$$\text{SIDE} = \left\lfloor \frac{L-R}{2} \right\rfloor \tag{2}$$

onde L e R são amostras inteiras.

A recuperação exata de L e R é garantida pelas seguintes equações, que incorporam o bit de resto ($\rho \in \{0, 1\}$), representando a paridade:

$$\mathbf{L} = \mathbf{MID} + \mathbf{SIDE} + \rho$$

$$\mathbf{R} = \mathbf{MID} - \mathbf{SIDE}$$

Esta relação prova que o esquema MID/SIDE é **lossless** (sem perdas) se o bit $\rho = (L + R) \pmod 2$ for também transmitido.

Verificação Numérica A Tabela 1 demonstra numericamente a exatidão do processo.

Tabela 1: Verificação Numérica da Transformação MID/SIDE Lossless.

Originais		**Transformação**				**Recuperados**	
L	R	$L+R$	ρ	**MID**	**SIDE**	L'	R'
100	50	150	0	75	25	100	50
101	50	151	1	75	25	101	50
-80	20	-60	0	-30	-50	-80	20
-81	20	-61	1	-31	-51	-81	20

Como se pode observar na tabela, em todos os casos os valores recuperados (L', R') são iguais aos originais (L, R), confirmando que o método é *lossless*. O bit de paridade ρ é crucial para corrigir o erro de arredondamento da divisão inteira nos casos em que a soma $L + R$ é ímpar.

2.1.5 Bins de Histograma mais largos

A implementação de *bins* mais largos (de tamanho 2^k) foi realizada através da função `getBin` na classe `WAVHist`. O programa aceita o valor k como argumento, e o `binSize` é calculado como 2^k. Para um `binSize` dado, o valor de um bin é determinado pelo seu ponto de partida:

$$\text{Bin}_{\text{value}} = \left\lfloor \frac{\text{Amostra}}{2^k} \right\rfloor \times 2^k$$

O resultado final no `dump` representa a soma das ocorrências na gama $[\text{Bin}_{\text{value}}, \text{Bin}_{\text{value}} + 2^k - 1]$.

2.2 Programa `wav_quant` (Quantização Uniforme)

O programa `wav_quant` implementa a **quantização uniforme escalar** de um ficheiro WAV de 16 bits, reduzindo o número de bits de precisão para B (onde $1 \leq B \leq 16$). O ficheiro de saída é também um ficheiro WAV (PCM 16-bit).

2.2.1 Sintaxe e Lógica

A utilização do programa segue o formato:

```
./wav_quant <wavFileIn> <wavFileOut> [bits]
```

Listing 3: Sintaxe de Uso do wav_quant

Onde `[bits]` é o número de bits de precisão final desejado (B), sendo o valor por omissão $B = 8$.

A quantização é realizada utilizando o método de **arredondamento** ao múltiplo mais próximo do passo de quantização (Δ). O passo é calculado pela relação:

$$\Delta = 2^{16-B}$$

A amostra quantizada x_q é calculada por:

$$x_q = \left\lfloor \frac{x}{\Delta} + 0.5 \right\rfloor \cdot \Delta$$

Esta abordagem minimiza o erro médio quadrático, fornecendo a melhor representação possível para o número de bits escolhido.

Exemplos de Teste Abaixo estão exemplos de comandos para quantizar um ficheiro `input.wav`:

1. **Quantização por Omissão (8 bits):** Reduz a precisão de 16 para 8 bits. Este teste é fundamental para a análise de erro na próxima secção.

```
./wav_quant sample.wav sample_8bit.wav
```

2. **Quantização Agressiva (4 bits):** Demonstra a aplicação da quantização com um baixo número de bits, maximizando a distorção.

```
./wav_quant sample.wav sample_4bit.wav 4
```

2.3 Programa `wav_cmp` (Métricas de Erro)

O programa `wav_cmp` tem como objetivo calcular e reportar as métricas de erro introduzidas no áudio por um processo de transformação com perdas (como a quantização realizada em `wav_quant`). O programa compara um ficheiro original (`original_file`) com a sua versão modificada (`processed_file`).

2.3.1 Sintaxe e Métricas de Erro

A sintaxe de utilização é a seguinte:

```
./wav_cmp <original_file> <processed_file>
```

Listing 4: Sintaxe de Uso do wav_cmp

As métricas são calculadas para cada canal (L, R) e para o **Canal MID** (a média dos canais), que é tratado como um canal distinto.

1. Erro Quadrático Médio (MSE ou L^2 norm) O MSE (Mean Squared Error) representa a potência média do ruído de erro (diferença entre a amostra original $x[n]$ e a amostra processada $x_q[n]$):

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(x[n] - x_q[n])^2$$

2. Erro Absoluto Máximo (L^∞ norm) O erro absoluto máximo ($E_{\max}$) é o valor absoluto do maior erro encontrado, representando o pior caso de distorção introduzida:

$$E_{\max} = \max_n |x[n] - x_q[n]|$$

3. Relação Sinal-Ruído (SNR) O SNR (Signal-to-Noise Ratio) é a métrica mais comum para avaliar a qualidade de um sinal degradado. É dado pela razão da potência do sinal original (P_{sinal}) pela potência do ruído (MSE), expressa em decibéis (dB):

$$\text{SNR} = 10\log_{10}\left(\frac{P_{\text{sinal}}}{P_{\text{ruído}}}\right) \quad \text{onde } P_{\text{ruído}} = \text{MSE}$$

A potência média do sinal é calculada por:

$$P_{\text{sinal}} = \frac{1}{N}\sum_{n=1}^{N} x[n]^2$$

Exemplo de Teste Para demonstrar o impacto da quantização de 16 para 8 bits (utilizando o ficheiro `output_8bit.wav` gerado pelo `wav_quant`):

```
./wav_cmp sample.wav sample_8bit.wav
```

2.3.2 Resultados Experimentais

A seguir são apresentados os resultados obtidos na comparação entre o ficheiro original (`sample.wav`) e a sua versão quantizada para 8 bits (`sample_8bit.wav`):

```
Audio Comparison Results:
========================

Channel 0:
  Mean Squared Error (L2):     5453.610956
  Max Absolute Error (LLinf.):  128.000000
  Signal-to-Noise Ratio (SNR): 41.317584 dB

Channel 1:
  Mean Squared Error (L2):     5455.157326
  Max Absolute Error (Linf.):  128.000000
  Signal-to-Noise Ratio (SNR): 41.336981 dB

MID Channel (Average of Channels):
  Mean Squared Error (L2):     2720.671028
  Max Absolute Error (Linf.):  128.000000
  Signal-to-Noise Ratio (SNR): 43.752264 dB

Total samples processed per channel: 529200
Number of channels: 2
MID channel samples processed: 529200
```

Listing 5: Resultados das métricas de erro para quantização de 8 bits (16 para 8 bits)

Análise dos Resultados Os resultados mostram que:

- **MSE similar entre canais:** Os canais esquerdo (L) e direito (R) apresentam valores de MSE muito próximos (≈5454), indicando que a quantização afeta ambos os canais de forma equivalente.
- **Erro máximo constante:** O erro absoluto máximo de 128 corresponde exatamente à metade do passo de quantização ($\Delta = 2^{16-8} = 256$), confirmando o comportamento esperado da quantização uniforme.
- **SNR superior no canal MID:** O canal MID apresenta um SNR de 43.75 dB, superior aos canais individuais (≈41.33 dB). Este resultado é esperado devido ao efeito de *averaging* que reduz o ruído não correlacionado entre canais.
- **MSE reduzido no canal MID:** O MSE do canal MID (≈2721) é aproximadamente metade do MSE dos canais individuais, demonstrando que a conversão MID/SIDE pode oferecer melhor eficiência de codificação.

2.4 Programa `wav_effects` (Efeitos de Áudio)

O programa `wav_effects` foi implementado para introduzir diversos efeitos de áudio, servindo como uma demonstração prática do processamento de sinal em domínio do tempo. O programa utiliza uma abordagem de linha de comando para selecionar o efeito e passar os seus parâmetros.

2.4.1 Sintaxe e Efeitos Implementados

A sintaxe geral de utilização é a seguinte:

```
./wav_effects <input.wav> <output.wav> <effect> [parameters]
```

Listing 6: Sintaxe de Uso do wav_effects

Abaixo estão detalhados os efeitos implementados e as suas fórmulas fundamentais.

1. Eco Simples (`echo`) Amostra de saída $y[k]$ é a soma da amostra original $x[k]$ com uma versão atrasada e atenuada da mesma.

$$y[k] = x[k] + g \cdot x[k - D]$$

```
./wav_effects input.wav output_echo.wav echo 500 0.5
```

(Exemplo: $D = 500$ ms de atraso, $g = 0.5$ de ganho)

2. Eco Múltiplo com Feedback (`multi_echo`) Este efeito implementa um filtro de feedback, onde o eco é alimentado de volta na entrada, criando uma sucessão de ecos decadentes.

$$y[k] = x[k] + g \cdot y[k - D]$$

```
./wav_effects input.wav output_multiecho.wav multi_echo 100 0.7
```

(Exemplo: $D = 100$ ms de atraso, $g = 0.7$ de ganho)

3. Modulação de Amplitude (`amp_mod`) Este efeito multiplica a amostra de áudio $x(t)$ por um sinal modulador de frequência f_m e profundidade α:

$$y(t) = x(t) \cdot (1 + \alpha \cdot \cos(2\pi f_m t))$$

```
./wav_effects input.wav output_am.wav amp_mod 4.0 0.8
```

(Exemplo: $f_m = 4.0$ Hz, $\alpha = 0.8$ de profundidade)

4. Delay Variável no Tempo (`time_var_delay`) Este efeito utiliza a fórmula do eco simples, mas o atraso D é alterado aleatoriamente a cada segundo entre 0.5 e 2.0 segundos.

$$y[k] = x[k] + g \cdot x[k - D_{\text{aleatório}}]$$

```
./wav_effects input.wav output_tvd.wav time_var_delay 0.6
```

(Exemplo: $g = 0.6$ de ganho)

5. Tremolo (`tremolo`) O efeito de tremolo modula a amplitude do sinal com um LFO (Low-Frequency Oscillator), criando uma variação de volume.

$$y(t) = x(t) \cdot \left((1 - \alpha) + \alpha \cdot \frac{1 + \sin(2\pi f_m t)}{2} \right)$$

```
./wav_effects input.wav output_tremolo.wav tremolo 5.0 0.7
```

(Exemplo: $f_m = 5.0$ Hz, $\alpha = 0.7$ de profundidade)

6. Vibrato (`vibrato`) O vibrato modula a frequência do sinal através de um delay variável no tempo, controlado por um LFO.

$$y(t) = x(t - d(t)) \quad \text{onde } d(t) = \frac{\alpha}{1000} \cdot \sin(2\pi f_m t)$$

```
./wav_effects input.wav output_vibrato.wav vibrato 5.0 2.0
```

(Exemplo: $f_m = 5.0$ Hz, $\alpha = 2.0$ ms de profundidade)

7. Filtro Passa-Baixo (`low_pass`) Implementa um filtro IIR (Infinite Impulse Response) passa-baixo de primeira ordem.

$$y[k] = \alpha \cdot x[k] + (1 - \alpha) \cdot y[k - 1]$$

```
./wav_effects input.wav output_lp.wav low_pass 0.3
```

(Exemplo: $\alpha = 0.3$)

8. Filtro Passa-Alto (`high_pass`) Implementa um filtro IIR passa-alto de primeira ordem.

$$y[k] = \alpha \cdot (y[k - 1] + x[k] - x[k - 1])$$

```
./wav_effects input.wav output_hp.wav high_pass 0.8
```

(Exemplo: $\alpha = 0.8$)

9. Distorção (`distortion`) Aplica um ganho ao sinal e depois corta-o (hard clipping) num determinado limiar, introduzindo harmónicos.

$$y[k] = \text{clip}(x[k] \cdot g, -T, T)$$

```
./wav_effects input.wav output_dist.wav distortion 10.0 15000
```

(Exemplo: $g = 10.0$ de ganho, $T = 15000$ de limiar)

3 Parte II: Codec de Quantização Escalar com Empacotamento de Bits

Nesta segunda parte, foi desenvolvido um codec completo, composto por um codificador (`wav_quant_enc`) e um descodificador (`wav_quant_dec`). Este sistema utiliza a quantização uniforme da Parte I e as classes `BitStream` e `ByteStream` para criar uma representação compacta (comprimida) do áudio.

3.1 Codificador `wav_quant_enc`

O codificador é responsável por ler um ficheiro WAV, aplicar a quantização e escrever o resultado num formato binário comprimido.

3.1.1 Algoritmo de Codificação

O processo de codificação segue quatro passos fundamentais:

1. **Passo 1: Quantização Uniforme:** As amostras do ficheiro WAV (16 bits) são lidas e quantizadas para um número de bits de precisão B, usando a mesma lógica do programa `wav_quant`. A amostra quantizada x_q é obtida a partir da amostra original x através de:

$$x_q = \left\lfloor \frac{x}{\Delta} + 0.5 \right\rfloor \cdot \Delta \quad , \quad \Delta = 2^{16-B}$$

2. **Passo 2: Mapeamento para Intervalo Não-Negativo:** As amostras quantizadas, que são valores com sinal, são mapeadas para um índice de quantização i_q (um inteiro não-negativo no intervalo $[0, 2^B - 1]$). Este mapeamento é feito através da seguinte transformação:

$$i_q = \frac{x_q}{\Delta} + 2^{B-1}$$

3. **Passo 3: Escrita do Header:** Para que o descodificador possa reconstruir o ficheiro WAV, é necessário guardar metadados essenciais. O codificador escreve um cabeçalho no início do ficheiro de saída contendo: o número de bits (B), o número de canais, a taxa de amostragem e o número total de frames.

4. **Passo 4: Empacotamento de Bits:** Cada amostra mapeada (agora o índice i_q de B bits) é escrita no ficheiro de saída usando a função `write_n_bits(índice, B)` da classe `BitStream`. Isto garante que não há desperdício de espaço (ex: uma amostra de 4 bits ocupa exatamente 4 bits no ficheiro, e não um byte inteiro).

3.1.2 Sintaxe e Exemplos

A sintaxe do codificador é análoga à do `wav_quant`:

```
./wav_quant_enc <wavFileIn> <encodedFileOut> [bits]
```

Listing 7: Sintaxe de Uso do wav_quant_enc

Exemplos de Teste:

```
# Codificar para 8 bits (valor por omissao)
./bin/wav_quant_enc sample.wav sample_8bit.enc

# Codificar para 4 bits
./bin/wav_quant_enc sample.wav sample_4bit.enc 4
```

Listing 8: Codificação para 8 e 4 bits

3.2 Descodificador `wav_quant_dec`

O descodificador realiza o processo inverso: lê o ficheiro binário comprimido e reconstrói o ficheiro WAV quantizado.

3.2.1 Algoritmo de Descodificação

1. **Passo 1: Leitura do Header:** O descodificador começa por ler os metadados (bits B, canais, sample rate, frames) do cabeçalho do ficheiro codificado usando o `BitStream`.

2. **Passo 2: Leitura e Desmapeamento:** O programa lê o fluxo de bits em blocos de B bits usando `read_n_bits(B)` para obter o índice i_q. Cada índice é desmapeado de volta para o seu valor de amostra quantizada x_q, invertendo o processo do Passo 2 da codificação:

$$x_q = (i_q - 2^{B-1}) \cdot \Delta$$

3. **Passo 3: Reconstrução do Ficheiro WAV:** Com os metadados do cabeçalho e as amostras reconstruídas x_q, é criado um novo ficheiro WAV com as mesmas propriedades do original (antes da compressão), mas com os dados de áudio quantizados.

3.2.2 Sintaxe e Exemplos

```
./wav_quant_dec <encodedFileIn> <wavFileOut>
```

Listing 9: Sintaxe de Uso do wav_quant_dec

Exemplos de Teste:

```
# Descodificar o ficheiro de 8 bits
./bin/wav_quant_dec sample_8bit.enc sample_8bit_dec.wav

# Descodificar o ficheiro de 4 bits
./bin/wav_quant_dec sample_4bit.enc sample_4bit_dec.wav
```

Listing 10: Descodificação dos ficheiros de 8 e 4 bits

3.3 Análise de Desempenho e Compressão

3.3.1 Métricas de Compressão

A taxa de compressão é a razão entre o tamanho do ficheiro original e o tamanho do ficheiro comprimido.

Tabela 2: Análise de Compressão do Codec BitStream

Ficheiro	Tamanho Original	Tamanho Comprimido	Taxa de Compressão
sample_8bit.enc	2,116,844 bytes	1,058,415 bytes	2 : 1
sample_4bit.enc	2,116,844 bytes	529,215 bytes	4 : 1

Análise dos Resultados: A taxa de compressão obtida é exatamente a esperada: ao reduzir a precisão de 16 para 8 bits (metade), o tamanho do ficheiro é reduzido para metade (2:1). Ao reduzir para 4 bits (um quarto), a taxa de compressão é de 4:1. Isto confirma a eficiência do empacotamento de bits.

3.3.2 Métricas de Qualidade de Áudio

Para avaliar a perda de qualidade, comparamos o ficheiro original com as versões descodificadas usando o `wav_cmp`.

Tabela 3: Métricas de Qualidade para Diferentes Valores de B

Métrica	**Canal**	**B = 8 bits**	**B = 4 bits**
MSE	Canal 0	5455.03	1,403,654.17
	Canal 1	5457.54	1,400,161.70
	MID	2721.89	702,538.78
Max Abs Error	Todos	255.00	4095.00
SNR (dB)	Canal 0	41.32 dB	17.21 dB
	Canal 1	41.34 dB	17.24 dB
	MID	43.75 dB	19.63 dB

Os resultados de erro (MSE, Max Abs Error, SNR), medidos com o programa `wav_cmp`, são praticamente idênticos aos obtidos na análise do `wav_quant`. Isto prova que o ciclo de codificação/descodificação com `BitStream` é **lossless** em relação aos dados quantizados. A única perda de informação ocorre na etapa de quantização, como esperado.

3.3.3 Análise de Tempo de Processamento

Os tempos de execução para codificar e descodificar o ficheiro `sample.wav` (duração de 12 segundos) foram medidos.

Tabela 4: Tempos de Execução do Codec (ms)

Processo	**B = 8 bits**	**B = 4 bits**
Codificação (`wav_quant_enc`)	22.20 ms	18.60 ms
Descodificação (`wav_quant_dec`)	32.56 ms	16.28 ms

Conclusões sobre Desempenho: O codec é extremamente rápido, processando um ficheiro de 12 segundos em dezenas de milissegundos. O tempo de processamento é dominado pela leitura/escrita de ficheiros, sendo a lógica de quantização e empacotamento de bits computacionalmente muito leve.

3.4 Comparação com Codec `wav_quant`

Ambos os sistemas (`wav_quant` e o codec `wav_quant_enc`/`dec`) aplicam a mesma quantização, mas produzem saídas diferentes.

Tabela 5: Comparação: `wav_quant` vs. Codec BitStream (para B=8)

Característica	`wav_quant`	**Codec BitStream**
Ficheiro de Saída	WAV (PCM 16-bit)	Binário customizado (.enc)
Tamanho (8 bits)	2,116,844 bytes	1,058,421 bytes
Compressão Real	Nenhuma (1:1)	Sim (2:1)
Compatibilidade	Alta (qualquer player)	Nenhuma (requer decoder)

Vantagens do Codec BitStream: A principal vantagem é a **compressão real**. Enquanto o `wav_quant` reduz a precisão mas mantém a estrutura de 16 bits por amostra no ficheiro WAV, o codec utiliza o `BitStream` para eliminar os bits redundantes, resultando numa poupança de espaço significativa e diretamente proporcional à redução de bits.

4 Parte III: Codec de Áudio com Transformada DCT

Nesta secção, é apresentado um codec com perdas para ficheiros de áudio mono, baseado na Transformada Discreta de Cosseno (DCT). A estratégia de compressão adotada baseia-se na **preservação dos coeficientes de baixa frequência mais significativos e no descarte dos de alta frequência**, que, por norma, contribuem menos para a perceção auditiva da energia do sinal.

O sistema é composto por um codificador (`mono_dct_enc`) e um descodificador (`mono_dct_dec`), que processam o áudio em blocos para alcançar a compressão.

4.1 Codificador `mono_dct_enc`

O codificador transforma o sinal de áudio do domínio do tempo para o domínio da frequência, descarta os coeficientes menos relevantes e escreve os restantes de forma compacta num ficheiro binário.

4.1.1 Algoritmo de Codificação

1. **Leitura e Preparação:** O ficheiro WAV mono é lido e dividido em blocos de amostras $x[n]$ de tamanho N (`blockSize`).

2. **Cálculo de Coeficientes a Manter:** O utilizador especifica uma taxa de bits alvo por amostra (parâmetro `bits`, e.g., 4 bits/amostra). Com base nisto, e sabendo que cada coeficiente guardado ocupa 32 bits, o programa calcula o número de coeficientes a preservar (N_c) por bloco:
$$N_c = \left\lfloor \frac{\texttt{bits} \times \texttt{blockSize}}{32} \right\rfloor$$
Esta abordagem liga diretamente o parâmetro de entrada à taxa de compressão final.

3. **Escrita do Cabeçalho:** Um cabeçalho é escrito no ficheiro de saída, contendo os metadados essenciais: taxa de amostragem, número de *frames*, `blockSize` (N) e o número de coeficientes a manter (N_c).

4. **Transformada DCT:** Para cada bloco $x[n]$, é aplicada a DCT (Tipo-II) para obter os coeficientes de frequência $C[k]$.

5. **Normalização e Truncação:** Os N_c primeiros coeficientes $C[k]$ (baixas frequências) são normalizados por um fator de $1/(2N)$ para cancelar o ganho da transformada. Os restantes coeficientes são descartados.
$$C'_{\text{norm}}[k] = \frac{C[k]}{2N}, \quad \text{para } k \in [0, N_c - 1]$$

6. **Empacotamento de Bits:** Cada coeficiente normalizado é arredondado para o inteiro de 32 bits mais próximo e escrito no ficheiro de saída usando a classe `BitStream`. Apenas os N_c coeficientes são escritos.

4.1.2 Sintaxe e Exemplos

```
./mono_dct_enc <input.wav> <output.bin> <blockSize> [bits]
```

Listing 11: Sintaxe de Uso do mono_dct_enc

Exemplos de Teste:

```
# Codificar para uma média de 8 bits/amostra
./bin/mono_dct_enc sample_mono.wav sample_mono_8bit.enc 1024 8

# Codificar para uma média de 4 bits/amostra
./bin/mono_dct_enc sample_mono.wav sample_mono_4bit.enc 1024 4
```

Listing 12: Codificação com blockSize

4.2 Descodificador `mono_dct_dec`

O descodificador reconstrói o sinal de áudio a partir do ficheiro binário comprimido.

4.2.1 Algoritmo de Descodificação

1. **Leitura do Cabeçalho:** Os metadados (taxa de amostragem, *frames*, `blockSize` N, e o número de coeficientes a ler N_c) são lidos do cabeçalho.
2. **Reconstrução dos Coeficientes:** Para cada bloco, é criado um vetor de coeficientes de tamanho N, inicializado a zero. O programa lê os N_c coeficientes do `BitStream` e preenche com eles o início do vetor. Os restantes $N - N_c$ coeficientes permanecem a zero.
3. **Transformada Inversa DCT:** É aplicada a DCT Inversa (IDCT) a cada bloco de coeficientes reconstruído. Como a normalização foi feita no codificador, o resultado da IDCT já está na escala correta.
4. **Reconstrução do Ficheiro WAV:** As amostras de áudio reconstruídas são arredondadas para o inteiro de 16 bits mais próximo e escritas num novo ficheiro WAV.

4.3 Análise de Desempenho e Compressão

Foi utilizado um ficheiro mono (`sample_mono.wav`) com um tamanho de 1.1 MB para os testes, com um `blockSize` de 1024.

4.3.1 Métricas de Compressão

A taxa de compressão teórica para B bits por amostra é de $16/B : 1$.

Tabela 6: Análise de Compressão do Codec DCT (blockSize=1024)

Bits Alvo	**Tamanho Original**	**Tamanho Comprimido**	**Taxa de Compressão**
8 bits ($16 \rightarrow 8$)	1,058,478 bytes	529,424 bytes	2 : 1
4 bits ($16 \rightarrow 4$)	1,058,478 bytes	264,720 bytes	4 : 1

4.3.2 Métricas de Qualidade de Áudio

A qualidade foi avaliada comparando o ficheiro original com as versões descodificadas para diferentes taxas de bits.

Tabela 7: Métricas de Qualidade para o Codec DCT (blockSize=1024)

Métrica	**bits = 8**	**bits = 4**
MSE	573,319.39	3,319,399.79
Max Abs Error	15,582.00	28,885.00
SNR (dB)	**20.52 dB**	**12.89 dB**

Análise dos Resultados: Os resultados obtidos demonstram a eficácia da compressão no domínio da frequência através da truncação de coeficientes.

- **Qualidade vs. Compressão:** Confirma-se o *trade-off* esperado entre a qualidade do áudio e a taxa de compressão. Ao duplicar a compressão (de 8 para 4 bits), o MSE aumenta significativamente e o SNR diminui cerca de 7.6 dB, indicando uma maior perda de informação.
- **Resultados de Qualidade:** Com uma compressão de 2:1 (8 bits/amostra), obtém-se um SNR de 20.52 dB. Este valor é considerado aceitável para aplicações com perdas, indicando que a maior parte da energia do sinal original foi preservada nos coeficientes mantidos.
- **Degradação Graciosa:** Mesmo com uma compressão mais agressiva de 4:1 (4 bits/amostra), o SNR mantém-se em 12.89 dB. Embora a qualidade seja visivelmente inferior, a estrutura fundamental do sinal de áudio é preservada.
- **Eficiência da DCT:** Estes resultados provam o princípio da compactação de energia da DCT: a maior parte da energia de um sinal de áudio está concentrada num pequeno número de coeficientes de baixa frequência. Ao preservar apenas estes, é possível obter uma representação compacta e razoavelmente fiel do sinal original.

4.3.3 Análise de Tempo de Processamento

Os tempos de execução foram medidos para os testes de codificação e descodificação.

Tabela 8: Tempos de Execução do Codec DCT (ms)

Processo	**bits = 8**	**bits = 4**
Codificação (`mono_dct_enc`)	15.9 ms	14.1 ms
Descodificação (`mono_dct_dec`)	21.2 ms	16.3 ms

Conclusões sobre Desempenho: O codec é computacionalmente eficiente, com tempos de codificação na ordem dos 15 milissegundos para um ficheiro de áudio de 12 segundos. A codificação para 4 bits é marginalmente mais rápida devido à menor quantidade de dados escritos em disco. A utilização de um plano pré-calculado pela biblioteca FFTW contribui significativamente para este bom desempenho.

5 Análise Extra: Estudo do Sinal de Erro

Para uma análise mais aprofundada da natureza da distorção introduzida pelos codecs, foi desenvolvido um programa adicional, o `wav_error`. Este programa permite isolar e analisar a distribuição estatística do sinal de erro, fornecendo uma visão qualitativa que complementa as métricas numéricas como o SNR.

5.1 Programa `wav_error`

5.1.1 Funcionalidade e Utilização

O programa `wav_error` compara dois ficheiros de áudio (um original e um processado) e gera um terceiro ficheiro WAV contendo o sinal de erro, calculado como a diferença amostra a amostra:

$$\text{erro}[n] = \text{original}[n] - \text{processado}[n]$$

A sintaxe de utilização é a seguinte:

```
./wav_error <ficheiro_original.wav> <ficheiro_processado.wav> <
    ficheiro_erro_saida.wav>
```

Listing 13: Sintaxe de Uso do wav_error

O ficheiro de erro gerado pode depois ser analisado com o programa `wav_hist` para se obter o histograma da distribuição do erro.

5.2 Análise Comparativa dos Histogramas de Erro

5.2.1 Erro de Quantização Uniforme (4 bits vs. 8 bits)

A Figura 14 compara os histogramas do erro de quantização para 8 e 4 bits.

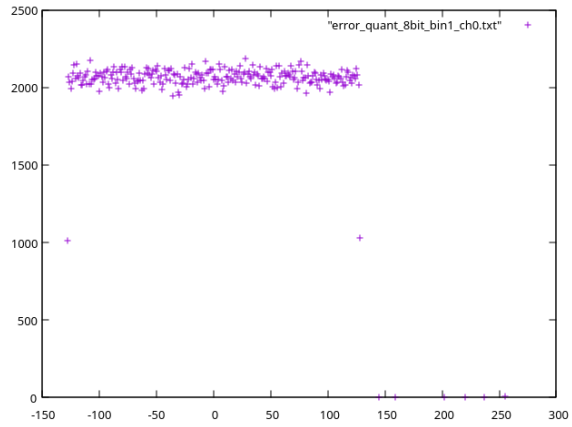

Figura 12: Erro de Quantização (8 bits)

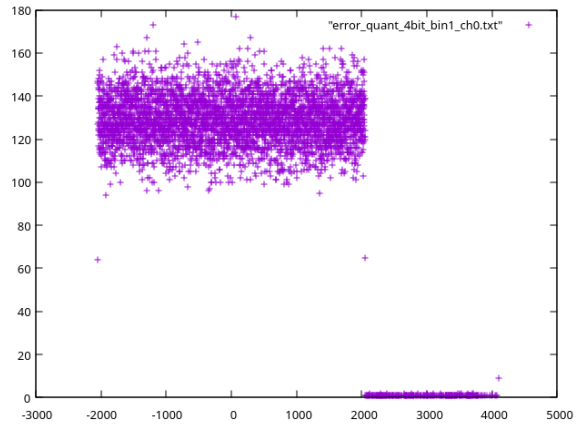

Figura 13: Erro de Quantização (4 bits)

Figura 14: Comparação do histograma do erro de quantização uniforme para 8 e 4 bits.

Análise: Ambos os histogramas apresentam uma **distribuição retangular (uniforme)**, o que valida visualmente o modelo teórico do ruído de quantização. A principal diferença é a largura da distribuição: para 8 bits ($\Delta = 256$), o erro está confinado ao intervalo $[-128, 128]$, enquanto para 4 bits ($\Delta = 4096$), o erro espalha-se pelo intervalo $[-2048, 2048]$. Isto demonstra graficamente o aumento drástico da magnitude do erro quando a precisão é reduzida.

5.2.2 Erro de Quantização no Canal MID (4 bits vs. 8 bits)

A Figura 17 mostra os histogramas de erro para o canal MID.

Análise: Confirma-se que o erro no canal MID exibe uma **distribuição triangular**. Este resultado é esperado, pois o erro do MID é a média dos erros (uniformemente distribuídos) dos canais L e R. Tal como no caso anterior, a base do triângulo para 4 bits é muito mais larga, indicando erros de maior amplitude.

5.2.3 Erro de Compressão DCT (4 bits vs. 8 bits)

A Figura 20 compara os histogramas do erro do codec DCT.

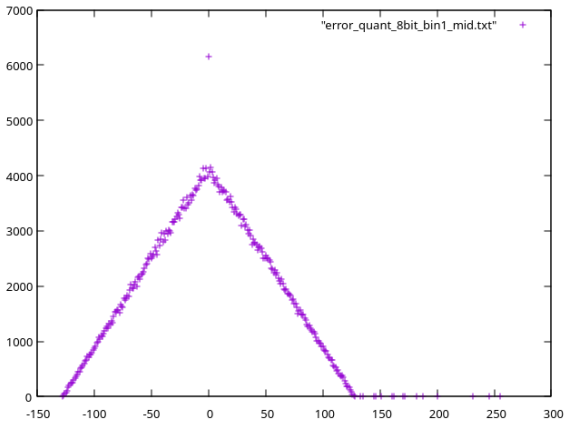

Figura 15: Erro do Canal MID (8 bits)

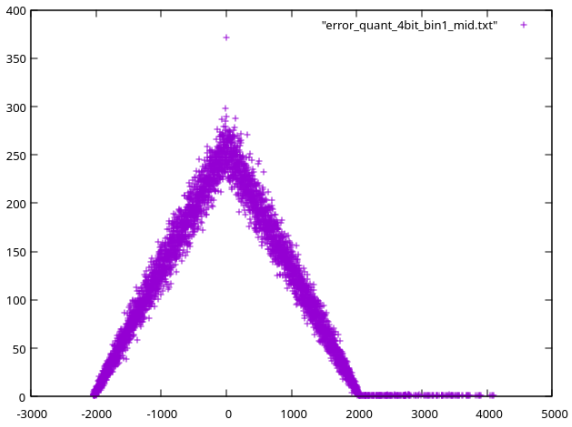

Figura 16: Erro do Canal MID (4 bits)

Figura 17: Comparação do histograma do erro no canal MID para 8 e 4 bits.

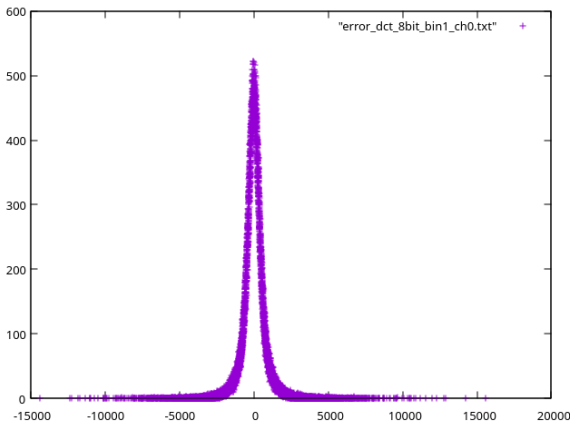

Figura 18: Erro do Codec DCT (8 bits)

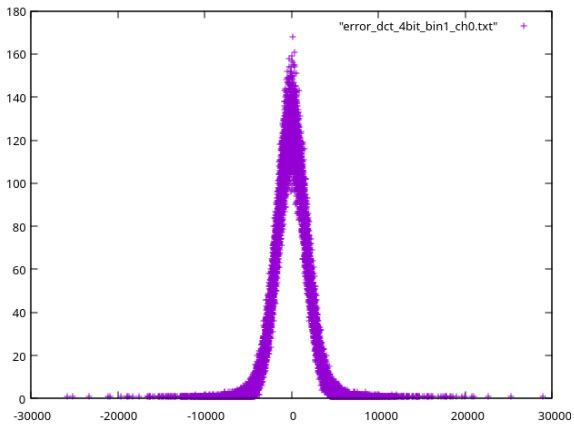

Figura 19: Erro do Codec DCT (4 bits)

Figura 20: Comparação do histograma do erro do codec DCT para 8 e 4 bits.

Análise: A distribuição do erro da compressão DCT é fundamentalmente diferente da quantização uniforme. Apresenta uma **distribuição em forma de sino com um pico muito acentuado em zero** (semelhante a uma distribuição de Laplace). Isto indica que a maioria dos erros são nulos ou muito próximos de zero, com a probabilidade de ocorrência a diminuir drasticamente para erros de maior amplitude. A distribuição para 4 bits é visivelmente mais "larga"e menos "pontiguda", o que significa que, embora a maioria dos erros ainda sejam pequenos, a probabilidade de erros maiores ocorrer é mais elevada, justificando a menor qualidade (SNR).

5.2.4 Comparação Direta: Quantização Uniforme vs. Compressão DCT

A Figura 23 coloca lado a lado a natureza do erro dos dois codecs para uma taxa de compressão equivalente (8 bits/amostra).

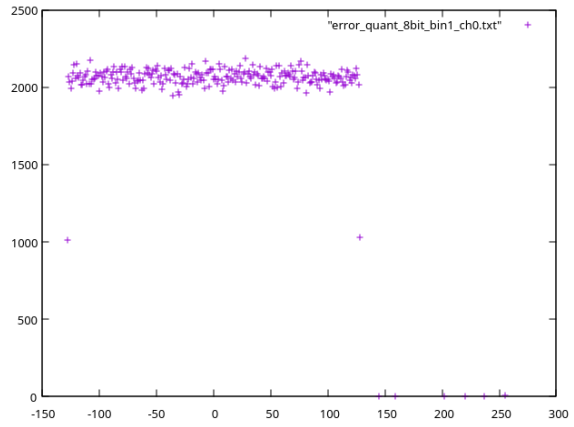

Figura 21: Erro da Quantização Uniforme (8 bits)

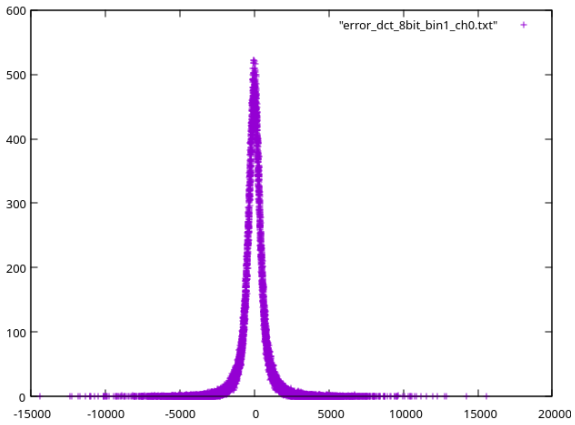

Figura 22: Erro do Codec DCT (8 bits)

Figura 23: Comparação da distribuição do erro entre os dois codecs.

Análise Final: Esta comparação visual demonstra a natureza fundamentalmente distinta do erro introduzido por cada codec. O erro da quantização uniforme (Figura 23, esquerda) é "não-estruturado"e equiprovável dentro dos seus limites, validando o modelo teórico. Em contraste, o erro do codec DCT (Figura 23, direita) é altamente "estruturado", com uma distribuição acentuadamente concentrada em zero, o que significa que o codec tenta ativamente minimizar a magnitude do erro.

É crucial notar que embora a distribuição do erro do DCT seja teoricamente mais vantajosa do ponto de vista percetual, o SNR medido para o codec de quantização uniforme foi significativamente superior (41.3 dB vs. 20.5 dB para 8 bits). Esta discrepância ocorre porque o codec DCT desenvolvido é um modelo simplificado que não implementa modelos percetuais avançadas.

A conclusão principal a retirar é que, embora a quantização uniforme tenha tido um melhor desempenho numérico neste teste específico, o codec DCT demonstra uma estratégia de gestão de erro muito mais sofisticada. É esta estratégia de concentrar o erro em valores pequenos que, quando combinada com modelos percetuais, permite aos codecs modernos alcançar uma qualidade de áudio superior a taxas de compressão muito mais elevadas.

6 Conclusões

Este trabalho permitiu uma exploração prática e aprofundada dos conceitos fundamentais de codificação de áudio, desde a análise de sinal até à implementação de codecs com e sem perdas.

Na **Parte I**, foram desenvolvidas ferramentas essenciais em C++ que serviram de base para as restantes secções. A análise de histogramas com `wav_hist`, incluindo a transformação para canais MID/SIDE, demonstrou como a representação do sinal pode ser otimizada, observando-se que o canal MID concentra a maior parte da energia e apresenta um SNR superior após a quantização. O programa `wav_cmp` forneceu as métricas quantitativas (MSE, SNR) cruciais para avaliar a degradação da qualidade de forma objetiva.

A **Parte II** focou-se na implementação de um codec de quantização escalar com empacotamento de bits. O sistema `wav_quant_enc`/`dec` provou a eficácia da classe `BitStream`, alcançando as taxas de compressão teóricas (e.g., 2:1 para 8 bits) ao eliminar a redundância de dados. Confirmou-se que o ciclo de codificação/descodificação é *lossless* em relação aos dados quantizados, isolando a perda de qualidade exclusivamente na etapa de quantização.

Finalmente, na **Parte III**, foi desenvolvido um codec baseado na DCT, que demonstrou a eficácia da compressão no domínio da frequência ao preservar seletivamente os coeficientes mais significativos. A abordagem de truncação de coeficientes provou ser eficaz, resultando num codec com um bom balanço entre a taxa de compressão e a qualidade do áudio.

Em suma, o projeto proporcionou uma base sólida sobre os desafios e as técnicas de compressão de áudio, desde a manipulação de bits até às transformadas de frequência.