

# Elaborazione Sequenziale di Segnali Audio con DFT e IDFT

## Contents

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Fase di test . . . . .	2
<b>2</b>	<b>Implementazione Sequenziale</b>	<b>2</b>
2.1	Lettura e Scrittura di File Audio . . . . .	2
2.2	Trasformata Discreta di Fourier (DFT) . . . . .	4
2.3	Filtro Passa-Basso . . . . .	4
2.4	Trasformata Inversa (IDFT) . . . . .	4
2.5	Misurazione delle Prestazioni . . . . .	5
<b>3</b>	<b>Esecuzione del Programma</b>	<b>6</b>
<b>4</b>	<b>Risultati</b>	<b>7</b>

## 1 Introduzione

Questo progetto si propone di elaborare segnali audio utilizzando un approccio sequenziale. L'obiettivo è implementare trasformazioni nel dominio delle frequenze per applicare un filtro passa-basso e produrre un segnale audio modificato. Il processo comprende le seguenti fasi:

- Lettura di un file audio in formato `.wav`;
- Calcolo della Trasformata Discreta di Fourier (DFT);
- Applicazione di un filtro passa-basso nel dominio delle frequenze;
- Calcolo della Trasformata Inversa (IDFT);
- Scrittura del segnale audio modificato su file;
- Generazione di un report sui tempi di esecuzione.

## 1.1 Fase di test

Tutti i codici sviluppati verranno testati su due macchine differenti per apprezzare la differenza tra architetture delle CPU e lo scarto generazionale delle GPU. Le macchine in questione hanno le seguenti specifiche:

	Macchina 1	Macchina 2
CPU	Intel Core i7-12650H	AMD Ryzen 7 8845HS
GPU	NVIDIA GeForce 3050 Mobile	NVIDIA GeForce RTX 4050 Mobile
O.S.	Ubuntu 22.04 on wsl	Fedora Linux 41

I programmi è sono stati testati su un audio mono con 44.1 kHz di frequenza di campionamento di due diverse durate `short_noise.wav` delle durata di 0.5 s e `long_noise.wav` della durata di 5 s contenente rumore bianco.

I risultati della applicazione del filtro si trovano nella directory `./output`, ogni file generato dal programma possiede nel nome il timestamp dell'esecuzione nel formato `Ymd_HMS` (anno, mese, giorno, ore, minuti, secondi).

I tempi di esecuzione sono riportati nei file all'interno della directory `./reports`. I risultati mostrano tempi di calcolo consistenti con le complessità delle operazioni.

## 2 Implementazione Sequenziale

Il codice è stato scritto in linguaggio C e utilizza le librerie standard per la manipolazione di file e la misurazione del tempo. La struttura principale del programma comprende:

### 2.1 Lettura e Scrittura di File Audio

I file audio sono letti e scritti in formato `.wav` standard, utilizzando un'intestazione di 44 byte per rappresentare i metadati del file. Le seguenti funzioni implementano la lettura e scrittura dei file audio:

Listing 1: Funzione per leggere file audio `.wav`.

```
1 // Funzione per leggere i campioni audio da un file .wav
2 void readWavFile(const char *filename, double *x, int N) {
3     FILE *file = fopen(filename, "rb");
4     if (file == NULL) {
5         printf("Errore nell'apertura del file %s\n", filename);
6         exit(1);
7     }
8
9     // Salta l'intestazione del file .wav (44 byte standard)
10    fseek(file, 44, SEEK_SET);
11
12    // Leggi i campioni audio come int16_t e convertili in double
13    int16_t *buffer = (int16_t *)malloc(N * sizeof(int16_t));
14    fread(buffer, sizeof(int16_t), N, file);
15    for (int i = 0; i < N; i++) {
16        x[i] = (double)buffer[i];
17    }
18 }
```

```

19     free(buffer);
20     fclose(file);
21 }

```

Listing 2: Funzione per scrivere file audio .wav.

```

1  // Funzione per scrivere un file .wav con l'intestazione
2  void writeWavFile(const char *filename, double *x, int N) {
3      FILE *file = fopen(filename, "wb");
4      if (file == NULL) {
5          printf("Errore nell'apertura del file %s\n", filename);
6          exit(1);
7      }
8
9      // Scrivi un'intestazione standard per un file .wav a 16 bit, mono,
10     44.1 kHz
11     uint8_t header[44] = {
12         'R', 'I', 'F', 'F',
13         0, 0, 0, 0, // Placeholder per la dimensione del file
14         'W', 'A', 'V', 'E',
15         'f', 'm', 't', ' ',
16         16, 0, 0, 0, // Dimensione del blocco fmt
17         1, 0, // PCM
18         1, 0, // Canali (mono)
19         0x44, 0xAC, 0x00, 0x00, // Frequenza di campionamento: 44100 Hz
20         0x88, 0x58, 0x01, 0x00, // Byte rate: 44100 * 2
21         2, 0, // Block align: 2 byte
22         16, 0, // Bit depth: 16 bit
23         'd', 'a', 't', 'a',
24         0, 0, 0, 0 // Placeholder per la dimensione dei dati
25     };
26
27     // Calcola la dimensione totale del file e dei dati
28     int dataSize = N * sizeof(int16_t);
29     int fileSize = 44 + dataSize - 8;
30
31     // Aggiorna i campi dell'intestazione
32     header[4] = (fileSize & 0xFF);
33     header[5] = ((fileSize >> 8) & 0xFF);
34     header[6] = ((fileSize >> 16) & 0xFF);
35     header[7] = ((fileSize >> 24) & 0xFF);
36
37     header[40] = (dataSize & 0xFF);
38     header[41] = ((dataSize >> 8) & 0xFF);
39     header[42] = ((dataSize >> 16) & 0xFF);
40     header[43] = ((dataSize >> 24) & 0xFF);
41
42     // Scrivi l'intestazione
43     fwrite(header, sizeof(uint8_t), 44, file);
44
45     // Scrivi i campioni audio convertiti in int16_t
46     int16_t *buffer = (int16_t *)malloc(N * sizeof(int16_t));
47     for (int i = 0; i < N; i++) {
48         buffer[i] = (int16_t)x[i];
49     }
50     fwrite(buffer, sizeof(int16_t), N, file);
51     free(buffer);

```

```

52     fclose(file);
53 }

```

## 2.2 Trasformata Discreta di Fourier (DFT)

La DFT converte un segnale audio dal dominio del tempo a quello delle frequenze. La funzione `dft()` implementa la seguente formula:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot \cos\left(\frac{2\pi kn}{N}\right) \quad (1)$$

Listing 3: Implementazione della DFT.

```

1 // Funzione che calcola la Discrete Fourier Transform di un segnale
  audio
2 void dft(double *x, double *X, int N) { // N = numero di campioni,
  complessit  O(N^2)
3     for (int k = 0; k < N; k++) {
4         X[k] = 0;
5         for (int n = 0; n < N; n++) {
6             X[k] += x[n] * cos(2 * PI * k * n / N);
7         }
8     }
9 }

```

## 2.3 Filtro Passa-Basso

Un filtro passa-basso   applicato nel dominio delle frequenze per rimuovere le componenti indesiderate. La funzione `filtro()` azzer  le frequenze al di fuori di una soglia specificata:

Listing 4: Applicazione del filtro passa-basso.

```

1 // Funzione che applica un filtro passa-basso al segnale audio
2 void filtro(double *X, int N, int fc) {
3     for (int k = 0; k < N; k++) {
4         if (k > fc && k < N - fc) {
5             X[k] = 0;
6         }
7     }
8 }

```

## 2.4 Trasformata Inversa (IDFT)

La IDFT riporta il segnale audio dal dominio delle frequenze al dominio del tempo. La funzione `idft()`   basata sulla formula:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot \cos\left(\frac{2\pi kn}{N}\right) \quad (2)$$

Listing 5: Implementazione della IDFT.

```

1 // Funzione che calcola l'Inverse Discrete Fourier Transform del
  // segnale audio filtrato
2 void idft(double *X, double *x, int N) { // N = numero di campioni,
  // complessit   $O(N^2)$ 
3     for (int n = 0; n < N; n++) {
4         x[n] = 0;
5         for (int k = 0; k < N; k++) {
6             x[n] += X[k] * cos(2 * PI * k * n / N);
7         }
8         x[n] /= N;
9     }
10 }

```

## 2.5 Misurazione delle Prestazioni

I tempi di esecuzione delle diverse fasi del programma vengono misurati e riportati in un file di testo:

Listing 6: Funzione per scrivere il report dei tempi.

```

1 // Funzione per scrivere un report dei tempi di esecuzione
2 void writeReport(const char *filename, double dftTime, double
  filterTime, double idftTime, double totalTime) {
3     FILE *file = fopen(filename, "w");
4     if (file == NULL) {
5         printf("Errore nell'apertura del file %s\n", filename);
6         exit(1);
7     }
8
9     fprintf(file, "Report tempi di esecuzione:\n");
10    fprintf(file, "-----\n");
11    fprintf(file, "DFT : %f secondi\n", dftTime);
12    fprintf(file, "Filtro: %f secondi\n", filterTime);
13    fprintf(file, "IDFT : %f secondi\n", idftTime);
14    fprintf(file, "Totale: %f secondi\n", totalTime);
15    fprintf(file, "-----\n");
16    fclose(file);
17 }

```

### 3 Esecuzione del Programma

La funzione `main()` coordina l'intero processo, dalla lettura del file audio alla scrittura dei risultati. Viene utilizzata la libreria `time.h` per misurare i tempi di esecuzione:

Listing 7: Funzione `main` del programma.

```
1 // Leggi il file audio
2 readWavFile(filename, x, N);
3
4 // Calcola la DFT
5 start = clock();
6 dft(x, X, N);
7 end = clock();
8 dftTime = (double)(end - start) / CLOCKS_PER_SEC;
9
10 // Applica il filtro passa-basso
11 start = clock();
12 filtro(X, N, 1000);
13 end = clock();
14 filterTime = (double)(end - start) / CLOCKS_PER_SEC;
15
16 // Calcola la IDFT
17 start = clock();
18 idft(X, y, N);
19 end = clock();
20 idftTime = (double)(end - start) / CLOCKS_PER_SEC;
21
22 // Scrivi il file output
23 writeWavFile(outputFile, y, N);
24
25 // Scrivi il report dei tempi
26 writeReport(reportFile, dftTime, filterTime, idftTime, dftTime +
    filterTime + idftTime);
```

## 4 Risultati

Le seguenti tabelle mostrano i risultati ottenuti su entrambe le macchine e entrambi i file di input.

Table 1: Macchina 1

File input	Funzione	Tempo
short_noise.wav	DFT	7.639986s
	Filtro	0.000030s
	IDFT	7.476071s
	<b>Totale</b>	15.116087s
long_noise.wav	DFT	484.563154s
	Filtro	0.000362s
	IDFT	483.476877s
	<b>Totale</b>	968.040393s

Table 2: Macchina 2

File input	Funzione	Tempo
short_noise.wav	DFT	5.122438s
	Filtro	0.000025s
	IDFT	4.305453s
	<b>Totale</b>	9.427916s
long_noise.wav	DFT	428.709111s
	Filtro	0.000262s
	IDFT	433.415536s
	<b>Totale</b>	862.124909s