Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Report di Laboratorio
Sistemi Elettronici e Sensori per l'Informatica

Studenti

Dedi Vittorio
Palladino Danilo

Anno Accademico 2024-2025

Sommario

Introduzione.....**2**
Capitolo 1: Generazione e Analisi dei Segnali.....**3**
1.1 Prima Parte.....7
1.2 Seconda Parte.....8
1.3 Terza Parte.....9
1.4 Quarta Parte.....9
Capitolo 2: Implementazione socket.....**10**
2.1 Server (Transmitter).....10
2.2 Client (Receiver).....12
Conclusioni.....**13**
Bibliografia.....**13**

Introduzione

Questa relazione illustra lo svolgimento di un'esercitazione finalizzata all'analisi della trasformata e antitrasformata di Fourier. Il lavoro si suddivide in due parti principali: una focalizzata sulla generazione, analisi e sintesi di segnali attraverso la serie di Fourier e la trasformata di Fourier, e l'altra dedicata alla trasmissione e ricezione di segnali via socket TCP/IP. Abbiamo sviluppato tre programmi distinti, il primo che risponde alle richieste fondamentali, che vedremo successivamente, mentre il secondo ed il terzo riguardano la parte opzionale (uno si occupa dell'invio e l'altro della ricezione dei dati). La relazione è strutturata in 2 capitoli, questa suddivisione è stata pensata per seguire un percorso logico che rispecchia l'evoluzione delle richieste pratiche.
Il primo capitolo si concentra sulla generazione e analisi dei segnali, facendo una piccola introduzione alla teoria dei segnali e alla trasformata di Fourier, poi proseguirà con la spiegazione del primo codice da noi implementato.
Il secondo capitolo esplorerà la trasmissione del segnale tramite socket TCP/IP, focalizzandoci sulla ricezione e visualizzazione del segnale.

Capitolo 1: Generazione e Analisi dei Segnali

La teoria dei segnali si occupa dello studio delle informazioni rappresentate sotto forma di funzioni del tempo. In molte applicazioni pratiche, i segnali sono analizzati e manipolati sia nel dominio del tempo, che ne descrive la variazione temporale, sia nel dominio della frequenza, che ne rappresenta la composizione armonica.
La trasformata di Fourier è lo strumento matematico che consente di passare tra questi due domini, scomponendo un segnale in una serie di componenti sinusoidali con ampiezze, frequenze e fasi ben definite.

In questa sezione ci concentreremo sull'applicazione pratica dei concetti appena citati, l'obiettivo iniziale è la generazione di un segnale di treno di onda quadra, a partire dai suoi coefficienti di Fourier. Questo primo passo ci consentirà di osservare come i segnali complessi possano essere approssimati sommando una serie di componenti armoniche e di valutare l'influenza del numero di coefficienti utilizzati sulla qualità del segnale ottenuto. Proseguiremo analizzando segnali di onde sinusoidali, triangolari e quadre a frequenze specifiche (100 Hz, 200 Hz, 440 Hz) nel tempo. Di queste ne genereremo il plot e realizzeremo per ciascuna di esse lo studio in frequenza, applicando la trasformata di Fourier e generando i plot relativi ai spettri di potenza. Infine, sposteremo l'attenzione sulla rappresentazione in frequenza di un segnale composito formato dalla somma delle tre onde sinusoidali precedenti. Per facilitare la comprensione del testo che segue divideremo il codice in quattro parti segnalate dai commenti all'interno di esso

Partiamo mostrando il codice implementato:

```
#Parte 1
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq

T = 1
f = 1 / T
t = np.linspace(0, 2 * T, 1000)
n_coefficients = 1000

def square_wave_fourier(t, T, n_coefficients):
    signal = np.zeros_like(t)
```

```
    for n in range(1, n_coefficients + 1, 2):
        signal += (4 / (np.pi * n)) * np.sin(2 * np.pi * n * t / T)
    return signal
signal = square_wave_fourier(t, T, n_coefficients)

plt.figure(figsize=(10, 6))
plt.plot(t, signal, label=f"Onda quadra con {n_coefficients}
coefficienti di Fourier")
plt.title("Sintesi di un'onda quadra usando i coefficienti di Fourier")
plt.xlabel("Tempo (s)")
plt.ylabel("Ampiezza")
plt.grid(True)
plt.legend()
```

#Parte 2

```
fs = 10000
duration = 0.02
t = np.linspace(0, duration, int(fs * duration), endpoint=False)

def sinusoidal(frequency, t):
    return np.sin(2 * np.pi * frequency * t)

def triangular(frequency, t):
    return 2 * np.abs(2 * ((t * frequency) % 1) - 1) - 1

def square(frequency, t):
    return np.sign(np.sin(2 * np.pi * frequency * t))

frequencies = [100, 200, 440]

plt.figure(figsize=(12, 10))
for i, freq in enumerate(frequencies):
    plt.subplot(3, 3, i + 1)
    plt.plot(t, sinusoidal(freq, t))
    plt.title(f"Onda Sinusoidale {freq} Hz")
    plt.xlabel("Tempo (s)")
    plt.ylabel("Ampiezza")
    plt.grid(True)

    plt.subplot(3, 3, i + 4)
    plt.plot(t, triangular(freq, t))
    plt.title(f"Onda Triangolare {freq} Hz")
    plt.xlabel("Tempo (s)")
    plt.ylabel("Ampiezza")
```

```
    plt.grid(True)

    plt.subplot(3, 3, i + 7)
    plt.plot(t, square(freq, t))
    plt.title(f"Onda Quadra {freq} Hz")
    plt.xlabel("Tempo (s)")
    plt.ylabel("Ampiezza")
    plt.grid(True)
plt.tight_layout()
```

#Parte 3

```
def frequencies_fft(fun):
    return fftfreq(len(fun), 1 / fs)

def power_spectrum(fun):
    signal_fft = fft(fun, norm="forward")
    return np.abs(signal_fft) ** 2

plt.figure(figsize=(12, 10))
for i, freq in enumerate(frequencies):
    plt.subplot(3, 3, i + 1)
    plt.plot(frequencies_fft(sinusoidal(freq, t)),
power_spectrum(sinusoidal(freq, t)))
    plt.title(f"Trasformata Onda Sinusoidale {freq} Hz")
    plt.xlabel("Frequenza fft")
    plt.ylabel("Ampiezza")
    plt.xlim(-2000, 2000)
    plt.grid(True)

    plt.subplot(3, 3, i + 4)
    plt.plot(frequencies_fft(triangular(freq, t)),
power_spectrum(triangular(freq, t)))
    plt.title(f"Trasformata Onda Triangolare {freq} Hz")
    plt.xlabel("Frequenza fft")
    plt.ylabel("Ampiezza")
    plt.xlim(-2000, 2000)
    plt.grid(True)

    plt.subplot(3, 3, i + 7)
    plt.plot(frequencies_fft(square(freq, t)),
power_spectrum(square(freq, t)))
    plt.title(f"Trasformata Onda Quadra {freq} Hz")
    plt.xlabel("Frequenza fft")
    plt.ylabel("Ampiezza")
```

```
    plt.xlim(-2000, 2000)
    plt.grid(True)
plt.tight_layout()
```

```
#Parte 4
```

```
plt.figure(figsize=(12, 10))
plt.plot(t, sinusoidal(frequencies[0], t) + sinusoidal(frequencies[1],
t) + sinusoidal(frequencies[2], t), color='b')
plt.title("somma di tre onde sinusoidali")
plt.xlabel("Tempo (s)")
plt.ylabel("Ampiezza")
plt.grid(True)
plt.tight_layout()
```

```
plt.figure(figsize=(12, 10))
plt.plot(frequencies_fft(sinusoidal(frequencies[0] + frequencies[1] +
frequencies[2], t)), power_spectrum(sinusoidal(frequencies[0], t) +
sinusoidal(frequencies[1], t) + sinusoidal(frequencies[2], t)),
color='b')
plt.title("somma di tre onde sinusoidali")
plt.xlabel("Frequenza fft")
plt.xlim(-2000, 2000)
plt.ylabel("Ampiezza")
plt.grid(True)
plt.tight_layout()

plt.show()
```

1.1 Prima Parte

Il primo approccio con il codice ha riguardato la generazione di un onda quadra e per fare questo abbiamo bisogno di definire alcuni parametri fondamentali. La variabile **`T`** rappresenta il periodo dell'onda e viene impostata a 1. Questo significa che l'onda completa un ciclo ogni 1 secondo. Di conseguenza, la frequenza **`f`**, calcolata come l'inverso del periodo (f = 1 / T), risulta essere 1 Hz, indicando che l'onda si ripete una volta al secondo.
Successivamente, creiamo un array di valori di tempo **`t`** utilizzando la funzione **`np.linspace(0, 2 * T, 1000)`**. Questa funzione genera una sequenza di 1000 valori equidistanti tra 0 e il doppio del periodo **`T`**, ovvero 2 secondi. L'array di tempo ci consente di osservare l'andamento del segnale su un periodo completo e un secondo periodo, in modo da avere una visione più chiara della sua periodicità.

Il numero di coefficienti di Fourier da utilizzare per sintetizzare l'onda quadra è definito dalla variabile **`n_coefficients`**, a cui abbiamo assegnato il valore di 1000. Questo parametro è importante poiché, maggiore è il numero di coefficienti usati, più precisa sarà la rappresentazione dell'onda quadra. La funzione principale che genera l'onda quadra è **`square_wave_fourier(t, T, n_coefficients)`**. All'interno di questa funzione, iniziamo creando un array di zeri della stessa lunghezza di **`t`**, che conterrà il segnale finale. Poi, attraverso un ciclo for, sommiamo le singole componenti sinusoidali per ottenere l'onda quadra. Ogni componente è determinata da un termine della serie di Fourier e contribuisce alla formazione dell'onda. In particolare, per ogni n dispari (1, 3, 5, 7,...), sommiamo un seno con un'ampiezza inversamente proporzionale alla frequenza.
Ogni seno ha una frequenza multipla del periodo T, e la somma di tutti questi seni produce l'onda quadra approssimata. Una volta che la funzione ha completato la somma di tutte le componenti sinusoidali, otteniamo il segnale finale, che viene assegnato alla variabile **`signal`**. Questo segnale viene poi plottato utilizzando la libreria **`matplotlib`**.
La funzione **`plt.plot(t, signal)`** traccia il segnale nel dominio del tempo, con l'asse orizzontale che rappresenta il tempo e quello verticale che mostra l'ampiezza del segnale. Il risultato finale è un grafico che mostra come la somma delle componenti sinusoidali generi un'onda quadra.

1.2 Seconda Parte

Successivamente l'obiettivo era quello di visualizzare tre tipi di onde nel dominio del tempo, ovvero l'onda sinusoidale, triangolare e quadra, per tre frequenze differenti. Innanzitutto, vengono definiti alcuni parametri. La variabile **`fs`** è impostata a 10.000, che rappresenta la frequenza di campionamento, cioè il numero di campioni per secondo che verranno presi per rappresentare il segnale. Con questa impostazione, ogni secondo di tempo sarà rappresentato da 10.000 punti, che ci consente di avere una risoluzione temporale adeguata per catturare i dettagli dei segnali. La durata del segnale viene fissata a 0.02 secondi, utilizzando la variabile **`duration`**. Questa durata rappresenta il periodo di osservazione in cui il segnale viene generato. Utilizzando la funzione **`np.linspace`**, viene creato un array **`t`** che rappresenta il tempo, con un intervallo che va da 0 a 0.02 secondi. L'array **`t`** è composto da un numero di campioni pari a **`int(fs * duration)`**, ovvero 200 punti, così da coprire l'intera durata del segnale con una risoluzione temporale adeguata.
In seguito, vengono definite tre funzioni per generare le diverse forme d'onda. La funzione **`sinusoidal(frequency, t)`** genera un'onda sinusoidale per una data frequenza. Utilizzando la formula standard per una sinusoide, la funzione calcola il valore dell'onda in funzione del tempo **`t`**, moltiplicato per la frequenza **`frequency`**.
Questo produce un'onda sinusoidale con la frequenza desiderata. La seconda funzione, **`triangular(frequency, t)`**, genera un'onda triangolare, che viene ottenuta tramite un'operazione che sfrutta il modulo e la manipolazione del tempo. Il risultato è un'onda che cresce e decresce in modo lineare, producendo una forma triangolare. Infine, la funzione **`square(frequency, t)`** genera un'onda quadra. Questa è realizzata utilizzando la

funzione **np.sign()**, che restituisce 1 o -1 in base al segno dell'argomento. Nel caso dell'onda quadra, l'argomento è una sinusoide, per cui l'onda quadra avrà una forma caratterizzata da salti improvvisi tra 1 e -1.
Le frequenze per cui poi verranno generati i segnali sono 100 Hz, 200 Hz e 440 Hz, e vengono memorizzate nella lista **frequencies**. Il codice poi crea una figura composta da più sottogruppi di grafici, ognuno dei quali mostra una delle tre onde (sinusoidale, triangolare e quadra) per ciascuna delle frequenze specificate. In totale, otterremo un grafico di 9 sotto-grafici, disposti in una griglia 3x3. Ogni sottogruppo mostrerà un tipo di onda per ciascuna delle frequenze 100 Hz, 200 Hz e 440 Hz. Ogni grafico viene etichettato con il tipo di onda e la frequenza in Hz, e l'asse x rappresenta il tempo in secondi, mentre l'asse y rappresenta l'ampiezza del segnale.
Infine, la funzione **plt.tight_layout()** viene chiamata per ottimizzare la disposizione dei sottogruppi all'interno della figura, evitando sovrapposizioni tra i grafici.

1.3 Terza Parte

In questa parte di codice che segue, viene definita la funzione **frequencies_fft(fun)** che calcola le frequenze corrispondenti agli assi orizzontali di uno spettro di Fourier, utilizzando la funzione **fftfreq** del modulo **Scipy**. Questa funzione prende come input la lunghezza del segnale **(len(fun))** e la frequenza di campionamento **fs** per determinare l'intervallo delle frequenze. Il risultato è un array di frequenze che corrisponde ai punti nel dominio della frequenza in cui il segnale viene campionato.
La funzione **power_spectrum(fun)** invece calcola lo spettro di potenza del segnale, utilizzando la trasformata di Fourier. La funzione **fft** di **Scipy** viene applicata al segnale per ottenere la sua rappresentazione nel dominio della frequenza. La funzione **np.abs(signal_fft) ** 2** restituisce il quadrato della magnitudine della trasformata di Fourier, che corrisponde alla densità spettrale di potenza del segnale. Questo valore fornisce una misura dell'intensità del segnale a ciascuna frequenza, evidenziando le componenti dominanti in termini di ampiezza.
Poi viene creato un grafico con una disposizione di 9 sottogruppi, in cui ogni sottogruppo visualizza lo spettro di potenza di una delle onde (sinusoidale, triangolare e quadra) per le tre frequenze specificate (100 Hz, 200 Hz e 440 Hz). Ogni grafico mostra come si distribuisce la potenza del segnale nelle frequenze, con l'asse x che rappresenta le frequenze in Hz, mentre l'asse y mostra l'ampiezza, che rappresenta la potenza associata a ciascuna frequenza. L'intervallo delle frequenze visualizzate è limitato da -2000 a 2000 Hz per garantire che le componenti fondamentali e armoniche siano visibili.
In ogni sottogruppo, viene tracciato il grafico della trasformata di Fourier per un tipo di onda e frequenza. Ad esempio, la trasformata di Fourier di un'onda sinusoidale a 100 Hz viene tracciata nel primo sottogruppo della figura, mentre quella di un'onda triangolare a 200 Hz viene tracciata nel secondo sottogruppo, e così via. I grafici sono etichettati con il tipo di onda e la frequenza in Hz, mentre le etichette sugli assi specificano la frequenza e l'ampiezza.

1.4 Quarta Parte

Infine abbiamo l'ultima parte di codice dove vengono prese in causa le ultime due richieste fondamentali: la somma di tre onde sinusoidali nel dominio del tempo e la trasformata di Fourier della somma di queste onde nel dominio della frequenza.
Nella prima parte, viene creata una figura con dimensioni 12x10 pollici. Qui, viene tracciato il grafico della somma di tre onde sinusoidali, ognuna con una frequenza diversa (100 Hz, 200 Hz e 440 Hz). La somma di queste onde viene calcolata combinando i valori restituiti dalla funzione **`sinusoidal`** per ciascuna frequenza, e il risultato viene plottato nel dominio del tempo, con l'asse x che rappresenta il tempo (in secondi) e l'asse y l'ampiezza del segnale. L'onda risultante mostra come si comporta la combinazione di queste tre onde sinusoidali. Il grafico viene etichettato con il titolo "Somma di tre onde sinusoidali" e le etichette degli assi sono rispettivamente "Tempo (s)" e "Ampiezza". Inoltre, viene aggiunta la griglia per rendere il grafico più leggibile.
Nella seconda parte, viene analizzata la trasformata di Fourier della somma delle tre onde sinusoidali. Il grafico rappresentato in questo blocco mostra lo spettro di frequenza del segnale risultante dalla somma delle onde, ottenuto tramite la trasformata di Fourier. La funzione **`frequencies_fft`** calcola la frequenza in funzione della lunghezza del segnale, mentre la funzione **`power_spectrum`** calcola lo spettro di potenza del segnale, che rappresenta l'intensità delle diverse componenti di frequenza. Nel grafico risultante, l'asse x rappresenta la frequenza (in Hz) e l'asse y rappresenta l'ampiezza dello spettro di potenza. Come in precedenza il range di frequenze è limitato tra -2000 e 2000 Hz per focalizzarci sulle frequenze più significative. Anche in questo caso, vengono applicate etichette agli assi e viene utilizzato **`plt.tight_layout()`** per migliorare la disposizione visiva della figura.

Capitolo 2: Implementazione socket

Gli ultimi due programmi si concentrano sulla trasmissione di segnali attraverso una rete locale utilizzando socket TCP/IP. Un server (Transmitter) crea e invia un'onda quadra a un client (Receiver) tramite una connessione socket.

2.1 Server (Transmitter)

Questo primo codice mostra come un segnale generato possa essere trasmesso in modo affidabile, con il protocollo TCP che garantisce che i dati vengano inviati correttamente e in modo sicuro.

```
import socket
import numpy as np
```

```
import matplotlib.pyplot as plt

def square(frequency, t):
    return np.sign(np.sin(2 * np.pi * frequency * t))

fs = 512
t = np.linspace(0, 1, fs)
freq = 5

waveform_sq = square(freq, t)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('localhost', 65432))
sock.listen(1)

print("Waiting for a connection...")
connection, client_address = sock.accept()

try:
    print("Connection from", client_address)
    connection.sendall(waveform_sq.tobytes())
finally:
    connection.close()
    sock.close()
```

In questo codice viene realizzato un server che invia un'onda quadra a un client tramite una connessione socket. La prima parte del codice definisce la funzione **`square`**, che genera un'onda quadra in base alla frequenza richiesta. L'onda quadra viene calcolata utilizzando la funzione **`np.sign`**, che restituisce +1 o -1 a seconda del segno del seno del segnale generato.

Subito dopo, viene definita la frequenza di campionamento **`fs`** e viene creato un array di tempo **`t`** utilizzando **`np.linspace`**, che crea un intervallo di tempo uniformemente distribuito tra 0 e 1 secondo. La variabile **`freq`** è impostata su 5 Hz, il che significa che l'onda quadra avrà una frequenza di 5 cicli al secondo. Utilizzando la funzione square, viene quindi generata l'onda quadra corrispondente e memorizzata nella variabile **`waveform_sq`**. A seguire ci si occupa della configurazione della connessione socket. Viene creato un socket con il protocollo AF_INET (IPv4) e il tipo SOCK_STREAM (TCP). Il socket è poi legato all'indirizzo localhost sulla porta 65432, e viene messo in ascolto per accettare una connessione in entrata. Il server poi stampa un messaggio sul terminale che indica che sta aspettando una connessione da parte di un client.

Una volta che il client si connette, il server stabilisce una connessione con il client e stampa l'indirizzo del client a cui è stato connesso, infine il server invia i dati dell'onda quadra generata **`(waveform_sq.tobytes())`**, che vengono convertiti in un formato binario prima di essere inviati attraverso la connessione. Come ultima cosa la connessione viene chiusa, così come il socket stesso.

2.2 Client (Receiver)

Il codice seguente implementa il client che si connette a un server per ricevere un'onda quadra e la visualizza graficamente.

```
import socket
import numpy as np
import matplotlib.pyplot as plt

fs = 512
t = np.linspace(0, 1, fs)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 65432))

data_sq = sock.recv(4096)
waveform_sq = np.frombuffer(data_sq, dtype=np.float64)

plt.figure(figsize=(10, 6))
plt.plot(t, waveform_sq)
plt.title("Received Square Waveform")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid(True)

plt.show()

sock.close()
```

Inizialmente viene definita la frequenza di campionamento **`fs`** a 512 Hz e viene creato un array di tempo **`t`** utilizzando **`np.linspace`**, che crea un intervallo di valori da 0 a 1 secondo, suddiviso in **`fs`** punti. Questo intervallo temporale viene utilizzato per tracciare il grafico dell'onda quadra che verrà ricevuta dal server.

Successivamente, viene creato un socket TCP (SOCK_STREAM) con il protocollo IPv4 (AF_INET). Il client si connette a un server in ascolto all'indirizzo 'localhost' sulla porta 65432, utilizzando il metodo connect. Una volta stabilita la connessione, il client è pronto a ricevere dati.
Il client riceve i dati dall'indirizzo e porta specificati tramite il metodo **`recv(4096)`**, che legge fino a 4096 byte di dati. Questi dati sono presumibilmente l'onda quadra inviata dal server. Poiché l'onda quadra è rappresentata come una sequenza di numeri in virgola mobile (tipo **`float64`**), il codice utilizza **`np.frombuffer(data_sq, dtype=np.float64)`** per convertire i dati ricevuti in un array **`NumPy`** con valori numerici in formato **`float64`**.

A questo punto, i dati dell'onda quadra ricevuti sono memorizzati nella variabile **`waveform_sq`**. Viene poi creato un grafico con **`matplotlib`**, in cui viene tracciata l'onda quadra nel dominio del tempo, utilizzando l'array **`t`** per l'asse orizzontale (tempo) e **`waveform_sq`** per l'asse verticale (ampiezza). Infine, il client visualizza il grafico con **`plt.show()`** e chiude la connessione con il server utilizzando **`sock.close()`**. Questo segna la fine della sessione di comunicazione tra il client e il server.

Conclusioni

Questa serie di programmi rappresenta un esempio pratico delle potenzialità offerte dalla matematica e dall'informatica per la manipolazione e l'analisi di segnali. La prima parte del lavoro ha dimostrato come i segnali possano essere generati, approssimati e analizzati nel dominio del tempo e delle frequenze. Le parti successive hanno mostrato come tali segnali possano essere trasmessi e ricevuti nel nostro caso utilizzando socket TCP/IP, che garantisce affidabilità e coerenza tra il segnale trasmesso e quello ricevuto.

Bibliografia

1. NumPy - NumPy User Guide: https://numpy.org/doc/stable/user/index.html#user Documentazione ufficiale della libreria NumPy. (Accesso: Dicembre 2024)
2. Matplotlib: https://matplotlib.org/stable/users/index.html# Documentazione ufficiale di Matplotlib (Accesso: Dicembre 2024)
3. Scipy - Scipy Documentation User Guide. https://docs.scipy.org/doc/scipy/tutorial/index.html#user-guide La documentazione ufficiale di SciPy (Accesso: Dicembre 2024)
4. Ambrosi, G. (2024). Dispense sulle Trasformata di Fourier. Università degli Studi di Perugia.