Magno Alessandro: 4478234

STUDIO E IMPLEMENTAZIONE ALGORITMO FFT

OBIETTIVO

Lo scopo di questo progetto è di mostrare come e perché conviene calcolare la *trasformata discreta di Fourier* (DFT) mediante un algoritmo veloce, chiamato *trasformata veloce di Fourier* (Fast Fourier Transform o FFT).

CENNO STORICO

L'algoritmo FFT venne sviluppato nel 1965 da James W. Cooley and John W. Tukey ma la sua formulazione avvenne 160 anni prima grazie a Gauss. Quest'ultimo lo utilizzò per approssimare le orbite degli asteroidi Pallade e Giunone dai dati di misurazione, in quanto era richiesto uno schema di interpolazione altamente accurato. Mentre Gauss eseguiva i calcoli a mente e su carta, si accorse che necessitava di un algoritmo veloce e sviluppò la FFT. Tuttavia, egli non la considerò come un grande passo avanti e la sua enunciazione apparve diversi anni dopo, nel 1866, nelle sue note compilate.

TRASFORMATA DISCRETA DI FOURIER (DFT)

La trasformata discreta di Fourier o DFT è data dalla seguente formula:

$$F_k = \sum_{n=0}^{N-1} f_n e^{-i2\pi nk/N}$$

con n, k indici che vanno da 0 a N-1

f_n array finito di valori campionati con valori di $f \in \mathbb{R}$

F_k coefficienti di Fourier con valori di $F \in \mathbb{C}$

La DFT è un operatore lineare (una matrice) che mappa i valori di f sul dominio della frequenza F:

$$\{f_0, f_1 \cdots, f_{N-1}\} \overset{DFT}{\Longrightarrow} \{F_0, F_1 \cdots, F_{N-1}\}$$

Per un dato numero di N valori campionati, la DFT rappresenta i valori utilizzando funzioni seno e coseno con la frequenza multipla fondamentale, $\omega_N = e^{-2\pi i/N}$. La DFT può essere calcolata per moltiplicazione di matrici:

$$\begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ \vdots \\ F_N \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix}.$$

Esempio calcolo DFT

caso con N=1: $F_0 = f_0 * e^{-i2\pi 0k/N}$

operazioni effettuate: 1

caso con N=2: $F_0 = f_0 * e^{-i2\pi 0k/N} + f_1 * e^{-i2\pi k/N}$

$F_1 = f_0 * e^{-i2\pi 0k/N} + f_1 * e^{-i2\pi k/N}$

operazioni effettuate: 4

caso con N=3: $F_0 = f_0 * e^{-i2\pi 0k/N} + f_1 * e^{-i2\pi k/N} + f_2 * e^{-i4\pi k/N}$

$F_1 = f_0 * e^{-i2\pi 0k/N} + f_1 * e^{-i2\pi k/N} + f_2 * e^{-i4\pi k/N}$

$F_2 = f_0 * e^{-i2\pi 0k/N} + f_1 * e^{-i2\pi k/N} + f_2 * e^{-i4\pi k/N}$

operazioni effettuate: 9

Dall'esempio emerge che per calcolare un coefficiente di Fourier occorrono N moltiplicazioni e N-1 addizioni. Se si esprime la complessità computazionale in termini di Big-O, è chiaro che il calcolo di un coefficiente di Fourier richiede $O(N)$ operazioni. Si devono calcolare N coefficienti di Fourier, ciò significa che il numero totale di operazioni necessarie per l'approccio è $O(N^2)$. Fortunatamente, con l'algoritmo FFT si riesce a ridurre la complessità in O($N \log N$) di uno stesso problema.

A prima vista, questo potrebbe non sembrare un grande vantaggio. Tuttavia, quando N è abbastanza grande, può fare la differenza. Questo si può vedere nella seguente tabella.

N	1000	10^6	10^9
N^2	10^6	10^{12}	10^{18}
$N \log_2 N$	10^4	20×10^6	30×10^9

Se si suppone che ci sia voluto 1 nanosecondo per eseguire un'operazione, l'algoritmo Fast Fourier Transform impiegherebbe circa 30 secondi per calcolare la trasformata discreta di Fourier di un problema di dimensioni N = 10^9. Al contrario, l'algoritmo regolare richiederebbe diversi decenni.

$$10^{18} ns \rightarrow 31.2\ anni$$

$$30 \times 10^9 ns \rightarrow 30\ secondi$$

FAST FOURIER TRANSFORM (FFT)

Per velocizzare la DFT, si suppone di separarla in sotto-sequenze di indici pari $x_0, x_2, x_4 \cdots$ e dispari $x_1, x_3, x_5 \cdots$; in tal modo si può scrivere l'indice $n = 2m\ se\ pari,\ n = 2m + 1\ se\ dispari$, dove $m = 0,1,2 \cdots, \frac{N}{2} - 1$. Sostituendo *n* nella formula della DFT:

$$F_k = \sum_{m=0}^{\frac{N}{2}-1} f_{2m} e^{\frac{-i2\pi k(2m)}{N}} + \sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} e^{\frac{-i2\pi k(2m+1)}{N}}$$

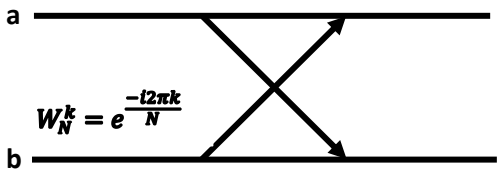
$$F_k = \sum_{m=0}^{\frac{N}{2}-1} f_{2m} e^{\frac{-i2\pi k(m)}{N/2}} + \sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} e^{\frac{-i2\pi k(m+\frac{1}{2})}{N/2}}$$

$$F_k = \sum_{m=0}^{\frac{N}{2}-1} f_{2m} e^{\frac{-i2\pi k(m)}{N/2}} + \sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} e^{\frac{-i2\pi km}{N/2} - \frac{i\pi k}{N/2}}$$

$$F_k = \sum_{m=0}^{\frac{N}{2}-1} f_{2m} e^{\frac{-i2\pi k(m)}{N/2}} + (e^{-\frac{i2\pi k}{N}}) \sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} e^{\frac{-i2\pi km}{N/2}}$$

Dopo aver eseguito i precedenti calcoli algebrici, si ottiene una DFT come somma di due sommatorie minori, ognuna con metà dimensione rispetto all'originale. Di per sé il risultato appena ottenuto non lascia emergere la sua reale potenza che invece, si nota subito, se si considera che si può iterare il procedimento appena effettuato per i primi due termini. Con l'intento di far trasparire il procedimento generale per N molto grandi si illustra, di seguito, un esempio esplicativo.

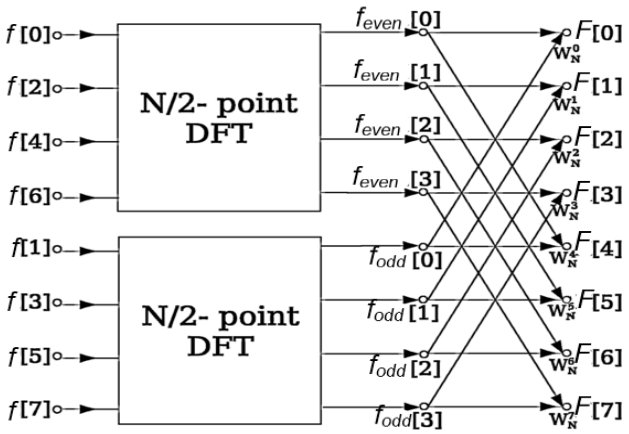
Prima di procedere al calcolo, per visualizzare il flusso di dati nel tempo, si introduce una rappresentazione grafica tramite i cosiddetti *butterfly diagrams* descrivendone il loro simbolismo:



La freccia verso l'alto indica un'addizione mentre quella verso il basso indica una sottrazione; il fattore esponenziale indica che, prima di eseguire l'operazione di somma o differenza, la variabile b viene moltiplicata per tale valore. Quindi gli output saranno la somma (in alto) e la differenza (in basso) tra a e $W_N^k b$.

Esempio calcolo FFT

Preso N = 8, si calcola la Trasformata di Fourier discreta per i termini pari e dispari, sfruttando il vantaggio che queste sotto-sequenze possono essere calcolate contemporaneamente. Dopo, si calcolano i coefficienti di Fourier utilizzando la formula descritta in precedenza.



Si esprime la complessità computazionale in termini di notazione Big-O come segue:

DFT su N/2 elementi

2 DFT

$$F[k] = f_{pari}[k] + W_N^k f_{dispari}[k]$$

$$2 \times \left(\frac{N}{2}\right)^2 + N$$

$$= 2 \times \frac{N^2}{4} + N$$

$$= \frac{N^2}{2} + N$$

$$O\left(\frac{N^2}{2} + N\right) \sim O(N^2)$$

Il primo termine deriva dal fatto che si calcola la Trasformata di Fourier discreta due volte. Questo si moltiplica per il tempo impiegato per calcolare la DFT su metà dell'input originale. Nel passaggio finale sono necessari N passi per sommare la trasformata di Fourier per un particolare k. Si tiene conto di ciò aggiungendo N al prodotto finale. A questo punto, si nota come si è stati in grado di ridurre il tempo impiegato per calcolare la trasformata di Fourier di un fattore 2. È possibile migliorare ulteriormente l'algoritmo, applicando l'approccio di divide-et-impera, cioè dimezzando ogni volta il costo computazionale. In altre parole, si può continuare a dividere la dimensione del problema fino a quando non restano gruppi di due e si possono, quindi, calcolare direttamente le trasformazioni discrete di Fourier per ciascuna di quelle coppie. L'algoritmo ottiene la sua velocità riutilizzando i risultati dei calcoli intermedi per calcolare più uscite DFT, ad esempio $f_{even}[0]$ è utilizzato per calcolare F_0 e F_4.

$$\frac{N}{2} \to 2\left(\frac{N}{2}\right)^2 + N = \frac{N^2}{2} + N$$

$$\frac{N}{4} \to 2\left(2\left(\frac{N}{4}\right)^2 + \frac{N}{2}\right) + N = \frac{N^2}{4} + 2N$$

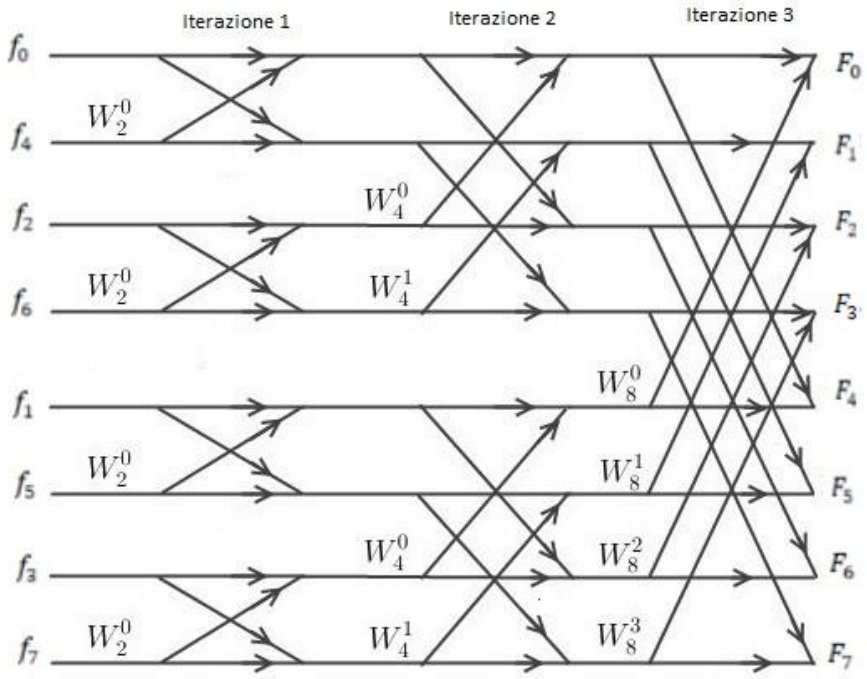
$$\frac{N}{8} \to 2\left(2\left(2\left(\frac{N}{8}\right)^2 + \frac{N}{4}\right) + \frac{N}{2}\right) + N = \frac{N^2}{8} + 3N$$

$$\vdots$$

$$\frac{N}{2^P} \to \frac{\mathrm{N}^2}{2^P} + \mathrm{PN} = \frac{\mathrm{N}^2}{\mathrm{N}} + (\log_2 N)\mathrm{N} = \mathrm{N} + (\log_2 N)\mathrm{N}$$

$$\sim O(N + N\log_2 N) \sim O(N\log_2 N)$$

Finché N è una potenza di 2, il numero massimo di volte che si può dividere in due metà uguali è dato da $P = \log_2 N$. Quindi, utilizzando l'algoritmo FFT per un problema di dimensione N = 8, P = 3 si ha il seguente diagramma:



Per eseguire il calcolo esplicito, si pone $W_N = e^{-i2\pi/N}$

$$F_k = \sum_{n=0}^{8-1} f_n e^{\frac{-i2\pi nk}{8}} = \sum_{m=0}^{4-1} f_{2m} e^{\frac{-i2\pi(2m)k}{8}} + \sum_{m=0}^{4-1} f_{2m+1} e^{\frac{-i2\pi(2m+1)k}{8}} =$$

$$= \sum_{m=0}^{4-1} f_{2m} e^{\frac{-i2\pi mk}{4}} + W_8^k \sum_{m=0}^{4-1} f_{2m+1} e^{\frac{-i2\pi mk}{4}}$$

si divide i due termini ulteriormente

$$F_k = \sum_{m=0}^{2-1} f_{4m} e^{\frac{-i2\pi(2m)k}{4}} + \sum_{m=0}^{2-1} f_{4m+2} e^{\frac{-i2\pi(2m+1)k}{4}} + W_8^k \sum_{m=0}^{2-1} f_{4m+1} e^{\frac{-i2\pi(2m)k}{4}} +$$

$$+W_8^k \sum_{m=0}^{2-1} f_{4m+3} e^{\frac{-i2\pi(2m+1)k}{4}} = \sum_{m=0}^{2-1} f_{4m} e^{\frac{-i2\pi mk}{2}} + W_4^k \sum_{m=0}^{2-1} f_{4m+2} e^{\frac{-i2\pi mk}{2}} +$$

$$+W_8^k \sum_{m=0}^{2-1} f_{4m+1} e^{\frac{-i2\pi mk}{2}} + W_8^k W_4^k \sum_{m=0}^{2-1} f_{4m+3} e^{\frac{-i2\pi mk}{2}}$$

e si termina con un'ultima iterazione

$$F_k = \sum_{m=0}^{1-1} f_{8m} e^{\frac{-i2\pi(2m)k}{2}} + \sum_{m=0}^{1-1} f_{8m+4} e^{\frac{-i2\pi(2m+1)k}{2}} + W_4^k \sum_{m=0}^{1-1} f_{8m+2} e^{\frac{-i2\pi(2m)k}{2}} +$$

$$+ W_4^k \sum_{m=0}^{1-1} f_{8m+6} e^{\frac{-i2\pi(2m+1)k}{2}} + W_8^k \sum_{m=0}^{1-1} f_{8m+1} e^{\frac{-i2\pi(2m)k}{2}} +$$

$$+ W_8^k \sum_{m=0}^{1-1} f_{8m+5} e^{\frac{-i2\pi(2m+1)k}{2}} + W_8^k W_4^k \sum_{m=0}^{1-1} f_{8m+3} e^{\frac{-i2\pi(2m)k}{2}} +$$

$$+ W_8^k W_4^k \sum_{m=0}^{1-1} f_{8m+7} e^{\frac{-i2\pi(2m+1)k}{2}} =$$

$$= f_0 + W_2^k f_4 + W_4^k f_2 + W_4^k W_2^k f_6 + W_8^k f_1 + W_8^k W_2^k f_5 + W_8^k W_4^k f_3 + W_8^k W_4^k W_2^k f_7$$

Il diagramma e il calcolo esplicito aiutano a capire il numero effettivo di calcoli da effettuare. Si nota che si hanno 3 iterazioni, ognuna composta da 8 somme; non stupisce, quindi, constatare che il numero di calcoli da effettuare effettivamente è $N \log_2 N = 8 \log_2 8 = 24$ più un numero di operazioni (in genere irrilevante per N grandi) per calcolare i vari W_N^k . Un'attenta lettura di quanto detto precedentemente fa intuire come questo algoritmo valga soltanto per un numero di campioni che sia potenza di 2. In effetti, con un numero che non è potenza di 2, non è possibile trovare un numero di iterazioni pari a $\log_2 N$ e quindi non varrebbe più il metodo sviluppato. Una soluzione pratica, nel caso di un numero di campioni che non è potenza di 2, è quella di riempire il pattern di dati con tanti termini nulli fino alla potenza di 2 successiva. Questa soluzione non introduce nessuna perturbazione al segnale perché, ovviamente, i termini nulli inseriti non contribuiscono in nessun modo al risultato finale della FFT. Questo è l'unico limite di applicazione dell'algoritmo.

PSEUDOCODICE E RISULTATI OTTENUTI DALL'IMPLEMENTAZIONE

```
#Precondizioni:
#     Input: f è un array a valori reali di lunghezza N, con N potenza di 2
#     Output: F è un array a valori complessi, il quale è la DFT dell'input
procedure FFT(f, N):
    if N = 1 then return f
    else
        W_N ← e^(-2πi/N)        #costante complessa
        #divisione degli indici
        f_e ← (f_0, f_2 ... f_(N-2))  #indici pari
        f_o ← (f_1, f_3 ... f_(N-1))  #indici dispari
        #calcolo dei valori intermedi
```

$f_{even} \leftarrow \mathrm{FFT}(f_e)$

$f_{odd} \leftarrow \mathrm{FFT}(f_o)$

F ← Array(N) #definito un array di output di lunghezza N

#si eseguono somme e differenze dei valori intermedi

#per ottenere i coefficienti di Fourier

for k←0 to N/2 – 1 **do**

F[k] ← $f_{even}[k] + W_N\text{^}k * f_{odd}[k]$

F[k+N/2] ← $f_{even}[k] - W_N\text{^}k * f_{odd}[k]$

return F

Posto N = 1024 si sono ottenuti i seguenti risultati:

DFT: 107 ms ± 5.68 ms per loop (mean ± std. dev. of 5 runs, 100 loops each)

FFT: 11.4 ms ± 246 µs per loop (mean ± std. dev. of 5 runs, 100 loops each)

FFT (libreria numpy): 6.91 µs ± 648 ns per loop (mean ± std. dev. of 5 runs, 100 loops each)

I valori ottenuti rappresentano la media e la deviazione standard del tempo di esecuzione del codice, calcolata su un numero di esecuzioni e un numero di loop all'interno di ogni esecuzione. Nei risultati sopra, la DFT è stata valutata per 5 esecuzioni ognuna con 100 loop. Ciò ha richiesto in media 107 millisecondi con una deviazione standard di 5.68 millisecondi. Confrontando i valori si nota che la FFT è significativamente più veloce della DFT. Tuttavia, la FFT della libreria numpy è più veloce della FFT implementata perché, durante gli anni, si sono effettuate numerose modifiche e ottimizzazioni.