La trasformata discreta di Fourier e la FFT

Dario A. Bini, Università di Pisa

7 febbraio 2020

Sommario

Questo modulo didattico contiene risultati relativi alla trasformata discreta di Fourier e agli algoritmi per il suo calcolo, in particolare gli algoritmi FFT di Cooley-Tukey e di Sande-Tukey. Si dà un cenno ad alcune applicazioni.

La trasformata discreta di Fourier è una operazione che permette di rappresentare vettori nel campo complesso in una base speciale, la base di Fourier. Da questa particolare rappresentazione si possono ricavare informazioni interessanti del vettore originario che sono di grande utilità in molte applicazioni.

Una caratteristica di fondamentale importanza di questa trasformazione è che il suo costo computazionale è estremamente basso. Infatti per effettuare il cambio di base di un vettore di n componenti bastano circa $\frac{3}{2}n\log_2 n$ operazioni purché n sia una potenza intera di 2. Inoltre la trasformazione è ben condizionata e gli algoritmi veloci per il suo calcolo, chiamati Fast Fourier Transform o FFT, sono numericamente stabili. Per queste caratteristiche particolarmente favorevoli, la trasformata discreta di Fourier trova numerose impieghi in diversi campi della matematica e delle sue applicazioni.

In questo articolo introduciamo questa trasformazione nel contesto dell'interpolazione, descriviamo due algoritmi FFT per il suo calcolo e mostriamo alcune applicazioni.

1 Interpolazione alle radici n-esime dell'unità

Dato un intero positivo n, consideriamo il polinomio $x^n - 1$. Le radici di questo polinomio vengono chiamate le *radici n-esime dell'unità*. Queste radici possono essere caratterizzate in termini di una radice speciale:

$$\omega_n = \cos\frac{2\pi}{n} + \mathtt{i}\sin\frac{2\pi}{n}$$

dove con $\mathtt{i}$ abbiamo denotato l'unità immaginaria tale che $\mathtt{i}^2 = -1$. Infatti, i numeri complessi

$$\omega_n^j = \cos\frac{2\pi j}{n} + \mathtt{i}\sin\frac{2\pi j}{n}, \quad j = 0, 1, \ldots, n-1,$$

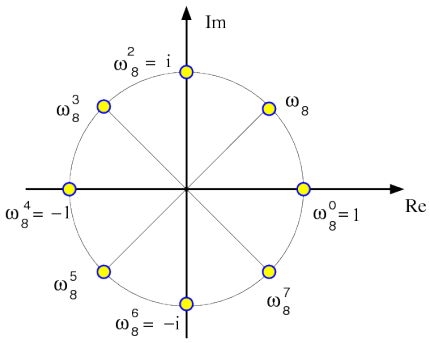

Figura 1: Radici ottave dell'unià

sono tutte e sole le radici n-esime dell'unità: esse sono n, essendo valori tutti distinti, e la loro potenza n-esima fa 1 dato che

$$(\omega_n^j)^n = (\omega_n^n)^j = 1^j = 1.$$

La figura 1 mostra graficamente nel piano complesso le radici ottave dell'unità

Ogni radice n-esima dell'unità ω tale che l'insieme $\{\omega^j : \quad j = 0, 1, \ldots, n-1\}$ costituisce l'insieme di tutte le radici n-esime viene detta *primitiva*. Si può verificare che se ζ è una radice n-esima primitiva tale che k e n siano primi tra loro, allora anche ζ^k è radice primitiva. In particolare, se n è un numero primo, ogni potenza non nulla di ω_n è radice primitiva. La radice ω_n come pure la sua coniugata sono radici primitive.

L'insieme delle radici n-esime forma un gruppo moltiplicativo essendo

$$\omega_n^p \omega_n^q = \omega_n^{p+q \bmod n}.$$

Consideriamo adesso il problema dell'interpolazione nel campo complesso e scegliamo come nodi le radici n-esime dell'unità che chiameremo *nodi di Fourier*. Poniamo cioè

$$x_j = \omega_n^j, \quad j = 0, 1, \ldots, n-1.$$

Dato allora il vettore $y = (y_0, y_1, \ldots, y_{n-1})$ vogliamo calcolare i coefficienti $(z_0, z_1, \ldots, z_{n-1})$ del polinomio $p(t) = \sum_{j=0}^{n-1} z_j t^j$ tale che $p(x_i) = y_i$ per $i = 0, 1, \ldots, n-1$. La condizione di interpolazione appena scritta conduce al sistema con matrice di Vandermonde

$$Vz = y, \quad V = (v_{i,j}), \quad v_{i,j} = x_i^j, \quad i, j = 0, \ldots, n-1,$$

dove, per la specificità dei nodi, la matrice V prende la forma $V = (\omega_n^{ij})_{i,j=0,\ldots,n-1}$. Questa matrice speciale di Vandermonde la indichiamo con Ω_n e la chiamiamo *matrice di Fourier*. Vale quindi

$$\Omega_n = (\omega_n^{ij \bmod n}).$$

Ad esempio, per $n = 7$ vale

$$\Omega_7 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_7 & \omega_7^2 & \omega_7^3 & \omega_7^4 & \omega_7^5 & \omega_7^6 \\ 1 & \omega_7^2 & \omega_7^4 & \omega_7^6 & \omega_7 & \omega_7^3 & \omega_7^5 \\ 1 & \omega_7^3 & \omega_7^6 & \omega_7^2 & \omega_7^5 & \omega_7 & \omega_7^4 \\ 1 & \omega_7^4 & \omega_7 & \omega_7^5 & \omega_7^2 & \omega_7^6 & \omega_7^3 \\ 1 & \omega_7^5 & \omega_7^3 & \omega_7 & \omega_7^6 & \omega_7^4 & \omega_7^2 \\ 1 & \omega_7^6 & \omega_7^5 & \omega_7^4 & \omega_7^3 & \omega_7^2 & \omega_7 \end{bmatrix}.$$

La matrice di Fourier gode di proprietà particolarmente interessanti. Premettiamo il seguente risultato di ortogonalità delle radici n-esime dell'unità:

Lemma 1 *Le radici n-esime dell'unità sono tali che*

$$\sum_{i=0}^{n-1} \omega_n^{ki} = \begin{cases} n & \text{se } k = 0 \bmod n \\ 0 & \text{se } k \neq 0 \bmod n \end{cases}$$

Dim.
Dalla identità

$$1 - x^n = (1 - x)(1 + x + x^2 + \cdots + x^{n-1})$$

ponendo $x = \omega_n^r$, deduciamo che, se $r \neq 0 \bmod n$ allora $1-x \neq 0$ mentre $1-x^n = 0$ e quindi, poiché siamo su un campo, ne segue $1+\omega_n^r+\omega_n^{2r}+\cdots+\omega_n^{(n-1)r} = 0$. D'altro canto, se $r = 0 \bmod n$ allora l'espressione $\sum_{k=0}^{n-1} \omega_n^{rk}$ si trasforma in una somma di n addendi uguali a 1 che dà n. □

Teorema 1 *La matrice Ω_n è tale che*

- $\Omega_n = \Omega_n^T$
- $\Omega_n^H \Omega_n = nI$, *dove* Ω_n^H *è la trasposta coniugata di* Ω_n
- $\Omega_n^2 = n\Pi_n$, *dove* Π_n *è la matrice di permutazione corrispondente alla permutazione* $\pi_0 = 0$, $\pi_j = n - j$, $j = 1, \ldots, n-1$.

Dim. La matrice Ω_n è chiaramente simmetrica essendo $\omega_n^{ij} = \omega_n^{ji}$. Per dimostrare il secondo punto sull'ortogonalità delle colonne di Ω_n dobbiamo fare vedere che il prodotto tra l'i-esima riga di Ω_n^H e la j-esima colonna di Ω_n vale zero se $i \neq j$ e vale n se $i = j$, cioè

$$\sum_{k=0}^{n-1} \bar{\omega}_n^{ik} \omega_n^{jk} = \begin{cases} n & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases}$$

La sommatoria nella precedente espressione si riduce a $\sum_{k=0}^{n-1} \omega_n^{rk}$ dove $r = j-i$. Per cui la tesi segue dal Lemma 1.

Per quanto riguarda la terza espressione ci si comporta in modo analogo. Infatti l'elemento di posto (i,j) di Ω_n^2 è dato da

$$\sum_{k=0}^{n-1} \omega_n^{ik} \omega_n^{jk} = \sum_{k=0}^{n-1} \omega_n^{k(i+j)}$$

e, per il lemma 1, vale zero se $i+j \neq 0 \bmod n$, vale n se $i+j = 0 \bmod n$. □

Si osservi che la permutazione individuata dalla matrice Π se applicata alla n-upla delle radici n-esime, listate in senso antiorario a partire da 1, ci fornisce la lista delle radici n-esime listate in senso orario a partire sempre da 1.

Il risultato precedente ci permette di scrivere l'inversa di Ω_n in forme computazionalmente convenienti. Vale infatti il seguente

Corollario 1 *Per la matrice inversa di Ω_n valgono le seguenti espressioni*

$$\Omega_n^{-1} = \frac{1}{n}\Omega_n^H, \qquad \Omega_n^{-1} = \frac{1}{n}\Omega_n\Pi_n, \quad \Omega_n^{-1} = \frac{1}{n}\Pi_n\Omega_n.$$

Si osserva in particolare che, per risolvere il problema dell'interpolazione sui nodi di Fourier, non c'è bisogno di risolvere nessun sistema lineare poiché l'inversa della matrice di Vandermonde è disponibile gratuitamente. Per cui, dato il vettore y, è possibile calcolare il vettore dei coefficienti z eseguendo un prodotto matrice vettore, infatti vale

$$z = \frac{1}{n}\Omega_n^H y, \quad z = \frac{1}{n}\Omega_n\Pi_n y, \quad z = \frac{1}{n}\Pi_n\Omega_n y.$$

Assumendo di avere a disposizione le radici n-esime dell'unità, il costo del calcolo dei coefficienti del polinomio di interpolazione è di al più n^2 moltiplicazioni e $n^2 - n$ addizioni. Vediamo tra poco che si può fare molto meglio con gli algoritmi FFT.

Un'altra conseguenza molto importante delle proprietà di ortogonalità delle radici n-esime è che la matrice $F_n = \frac{1}{\sqrt{n}}\Omega_n$ è unitaria, cioè $F_n^H F_n = I$, e quindi $\|F_n\|_2 = \|F_n^H\|_2 = 1$. Conseguentemente il suo numero di condizionamento in norma 2, cioè $\mu_2 = \|F_n\|_2\|F_n^H\|_2 = 1$. Ciò implica che anche la matrice Ω_n ha condizionamento in norma 2 uguale a 1, infatti essa differisce da F_n per una costante moltiplicativa. Quindi, diversamente dal caso dei nodi reali, il problema dell'interpolazione ai nodi di Fourier è ben condizionato.

La trasformazione lineare che associa il vettore y al vettore z è detta *trasformata discreta di Fourier* e si denota $z = \mathrm{DFT}_n(y)$, la trasformazione lineare che associa z a y viene detta *trasformata discreta inversa di Fourier* e si denota $y = \mathrm{IDFT}_n(z)$. Quindi in sintesi:

$$\begin{array}{ll} \mathrm{DFT}_n(y) := z = \frac{1}{n}\Omega_n^H y & Trasformata\ discreta\ di\ Fourier \\ \mathrm{IDFT}_n(z) := y = \Omega_n z & Trasformata\ discreta\ inversa\ di\ Fourier \end{array}$$

La trasformata discreta inversa di Fourier associa ai coefficienti del polinomio i valori che esso assume nelle radici n-esime dell'unità. Quindi risolve un *problema di valutazione*. La trasformata discreta di Fourier associa ai valori che un polinomio assume nei nodi di Fourier i suoi coefficienti. Quindi risolve un *problema di interpolazione*.

Un'altra osservazione interessante è che, poiché $y = \Omega_n z$, le componenti di z sono i coefficienti di rappresentazione del vettore y nella base di $\mathbb{C}^n$ data dalle colonne di Ω_n. Questa base la chiamiamo *base di Fourier*. Possiamo quindi interpretare la DFT e la IDFT come cambiamenti di base: dalla base di Fourier alla base canonica e dalla base canonica alla base di Fourier.

In letteratura ci sono diverse definizioni di DFT e IDFT a seconda del contesto in cui viene usata. Ad esempio, in alcuni casi si preferisce definire le due trasformazioni in modo unitario normalizzando entrambe col fattore $1/\sqrt{n}$. In Octave il comando `x = fft(y);` fornisce il valore di $n\mathrm{DFT}_n(y) = \Omega_n^H y$, mentre il comando `y = ifft(x);` dà il valore di $\frac{1}{n}\mathrm{IDFT}_n(x) = \frac{1}{n}\Omega_n x$.

2 Gli algoritmi FFT

Così come è stata definita la DFT e la IDFT si possono calcolare eseguendo un prodotto tra la matrice Ω_n e un vettore. Il costo computazionale è quindi di n^2 moltiplicazioni e n^2-n addizioni. È possibile però utilizzare la speciale struttura di Ω_n per poter calcolare la DFT e la IDFT con un costo sostanzialmente più basso.

Consideriamo il caso della IDFT. Dati i valori di $z_0, z_1, \ldots, z_{n-1}$ e date le radici n-esime dell'unità vogliamo calcolare i valori di

$$y_i = \sum_{j=0}^{n-1} \omega_n^{ij} z_j, \quad i = 0, 1, \ldots, n-1\,. \tag{1}$$

Consideriamo il caso particolare in cui n è una potenza intera di 2, cioè $n = 2^q$.

L'idea che si usa per costruire un algoritmo efficiente per il calcolo di (1) si basa sulla strategia del *divide et impera*. Cioè il calcolo di una IDFT su n nodi lo riconduciamo al calcolo di due IDFT su $\frac{n}{2}$ nodi. Per far questo riscriviamo la (1) separando gli addendi che hanno indice pari da quelli che hanno indice dispari. Otteniamo allora

$$y_i = \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{2ij} z_{2j} + \sum_{j=0}^{\frac{n}{2}-1} \omega_n^{i(2j+1)} z_{2j+1}, \quad i = 0, \ldots, n-1.$$

Adesso osserviamo che $\omega_n^2 = \omega_{\frac{n}{2}}$ per cui $\omega_n^{2ij} = \omega_{\frac{n}{2}}^{ij}$ e $\omega_n^{i(2j+1)} = \omega_n^i \omega_{\frac{n}{2}}^{ij}$. Possiamo allora riscrivere la precedente espressione come

$$y_i = \sum_{j=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{ij} z_{2j} + \omega_n^i \sum_{j=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{ij} z_{2j+1}, \quad i = 0, \ldots, n-1\,. \tag{2}$$

Se limitiamo il calcolo alle prime $\frac{n}{2}$ componenti la (2) ci dice che

$$(y_0, \ldots, y_{\frac{n}{2}-1})^T = \mathrm{IDFT}_{\frac{n}{2}}(z_{\mathrm{pari}}) + \mathrm{Diag}(1, \omega_n, \ldots, \omega_n^{\frac{n}{2}-1})\mathrm{IDFT}_{\frac{n}{2}}(z_{\mathrm{dispari}}), \quad (3)$$

dove abbiamo indicato rispettivamente con z_{pari} e z_{dispari} la parte del vettore z costituita dalle componenti pari e la parte costituita dalle componenti dispari. Abbiamo cioè espresso la prima metà di y in termini di due trasformate discrete di ordine la metà.

La seconda metà del vettore y, cioè quella costituita dalle componenti $y_{\frac{n}{2}}, \ldots y_{n-1}$, si ottiene mettendo al posto dell'indice i nella (2) il valore $\frac{n}{2} + i$. Poiché $\omega_{\frac{n}{2}}^{\frac{n}{2}+i} = \omega_{\frac{n}{2}}^i$, e $\omega_n^{\frac{n}{2}+i} = -\omega_n^i$, si ottiene quindi

$$y_{\frac{n}{2}+i} = \sum_{j=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{ij} z_{2j} - \omega_n^i \sum_{j=0}^{\frac{n}{2}-1} \omega_{\frac{n}{2}}^{ij} z_{2j+1}, \quad i = 0, \ldots, \frac{n}{2} - 1.$$

Cioè in termini di $\mathrm{IDFT}_{\frac{n}{2}}(z_{\mathrm{pari}})$ e $\mathrm{IDFT}_{\frac{n}{2}}(z_{\mathrm{dispari}})$ vale

$$(y_{\frac{n}{2}}, \ldots, y_{n-1})^T = \mathrm{IDFT}_{\frac{n}{2}}(z_{\mathrm{pari}}) - \mathrm{Diag}(1, \omega_n, \ldots, \omega_n^{\frac{n}{2}-1})\mathrm{IDFT}_{\frac{n}{2}}(z_{\mathrm{dispari}}). \quad (4)$$

Mettendo insieme la (3) e la (4) possiamo rappresentare il vettore y in termini delle IDFT della parte pari e della parte dispari di z. Più precisamente si procede come segue

- calcolare $w^{(1)} = \mathrm{IDFT}_{\frac{n}{2}}(z_{\mathrm{pari}})$ e $w^{(2)} = \mathrm{IDFT}_{\frac{n}{2}}(z_{\mathrm{dispari}})$;
- moltiplicare $w^{(2)}$ per i valori $1, \omega_n, \ldots, \omega_n^{\frac{n}{2}-1}$, cioè calcolare
$$w^{(3)} = \mathrm{Diag}(1, \omega_n, \ldots, \omega_n^{\frac{n}{2}-1})w^{(2)};$$
- infine calcolare i vettori $y^{(1)} = w^{(1)} + w^{(3)}$, $y^{(2)} = w^{(1)} - w^{(3)}$, e porre $y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix}$.

Questo procedimento comporta il calcolo di due trasformate discrete inverse di ordine la metà, $\frac{n}{2}$ moltiplicazioni, $\frac{n}{2}$ addizioni e $\frac{n}{2}$ sottrazioni.

Poiché $n = 2^q$ è potenza di 2, possiamo allora ripetere questa strategia per calcolare le due trasformate di ordine $\frac{n}{2}$ mediante quattro trasformate di ordine $n/4$ e procedere ricorsivamente in questo modo finché non si arriva a trasformate di ordine 1 che non richiedono alcuna operazione. Indicando con $c(n)$ il costo computazionale per il calcolo di una IDFT di ordine n con questo metodo abbiamo la relazione

$$c(n) = 2c(\frac{n}{2}) + \frac{n}{2}M + nA$$

dove M indica "moltiplicazioni" e A indica "addizioni/sottrazioni". Poiché $c(1) = 0$ si può dimostrare induttivamente che

$$c(n) = (\frac{n}{2}M + nA)q = (\frac{n}{2}M + nA)\log_2 n.$$

L'algoritmo per il calcolo della IDFT che si ottiene in questo modo è noto come algoritmo di Cooley e Tukey.

Il calcolo della DFT si realizza utilizzando il teorema 1 e ha il costo aggiuntivo del calcolo di n divisioni per n.

2.1 Note computazionali

È possibile dare una interpretazione in termini di matrici dell'algoritmo per il calcolo della IDFT descritto sopra. Per questo introduciamo la matrice di permutazione pari-dispari P_n tale che

$$P_n z = \begin{bmatrix} z_{\text{pari}} \\ z_{\text{dispari}} \end{bmatrix}$$

e denotiamo $m = \frac{n}{2}$.

Dalla descrizione data nell'algoritmo 1 deduciamo che

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix} = \begin{bmatrix} I_m & I_m \\ I_m & -I_m \end{bmatrix} \begin{bmatrix} I_m & 0 \\ 0 & D_m \end{bmatrix} \begin{bmatrix} \Omega_m & 0 \\ 0 & \Omega_m \end{bmatrix} P_n z$$

dove

$$D_m = \text{Diag}(1, \omega_n, \ldots, \omega_n^{m-1}).$$

Questo ci permette di dire che la matrice di Fourier Ω_n si lascia fattorizzare nel modo seguente

$$\Omega_n = \begin{bmatrix} I_m & I_m \\ I_m & -I_m \end{bmatrix} \begin{bmatrix} I_m & 0 \\ 0 & D_m \end{bmatrix} \begin{bmatrix} \Omega_m & 0 \\ 0 & \Omega_m \end{bmatrix} P_n. \tag{5}$$

Questa fattorizzazione assume una forma più interessante se la riscriviamo in termini del prodotto di Kronecker $\otimes$ che definiamo nel modo seguente. Siano A e B matrici di dimensioni rispettivamente $k \times k$ e $h \times h$. Definiamo $C = A \otimes B$ la matrice $hk \times hk$ tale che.

$$C = \begin{bmatrix} a_{1,1}B & a_{1,2}B & \ldots & a_{1,k}B \\ a_{2,1}B & a_{2,2}B & \ldots & a_{2,k}B \\ \vdots & \ldots & \ldots & \vdots \\ a_{k,1}B & a_{k,2}B & \ldots & a_{k,k}B \end{bmatrix}$$

Poiché $\Omega_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, la (5) prende la forma

$$\Omega_n = (\Omega_2 \otimes I_m)\text{Diag}(I, D_m)(I_2 \otimes \Omega_m)P_n, \quad D_m = \text{Diag}(1, \omega_n, \ldots, \omega_n^{m-1}). \tag{6}$$

Questa rappresentazione matriciale permette di dedurre altre interessanti proprietà. Ad esempio, poichè $\Omega_n = \Omega_n^T$ la (6) implica che

$$\Omega_n = P_n^T(I_2 \otimes \Omega_m)\mathrm{Diag}(I, D_m)(\Omega_2 \otimes I_m). \tag{7}$$

L'applicazione ricorsiva della fattorizzazione (7) conduce ad un altro algoritmo per il calcolo della IDFT noto come algoritmo di Sande e Tukey, che ha la stessa complessità dell'algoritmo di Cooley e Tukey ma svolge le operazioni aritmetiche in modo diverso. Ad esempio, mentre nell'algoritmo di Cooley e Tukey la permutazione viene fatta all'inizio, nel metodo di Sande e Tukey la permutazione viene fatta alla fine. Per questa caratteristica il primo metodo viene detto algoritmo in base 2 di decimazione in tempo, e il secondo come algoritmo in base 2 di decimazione in frequenza. Questa terminologia proviene dalle applicazioni ingegneristiche della DFT.

Non è complicato dimostrare che se $n = rs$, con r, s interi positivi, allora vale

$$\Omega_n = (\Omega_r \otimes I_s)\mathrm{Diag}(I, D_s, D_s^2, \ldots, D_s^{r-1})(I_r \otimes \Omega_s)P_n^{(r)},$$

dove $D_s = \mathrm{Diag}(1, \omega_n, \ldots, \omega_n^{s-1})$, e dove $P_n^{(r)}$ è la matrice di permutazione associata alla permutazione che mette in testa gli indici congrui a 0 modulo r seguiti dagli indici congrui a 1 modulo r, e così via fino agli indici congrui a $r-1$ modulo r.

Questa fattorizzazione permette di costruire algoritmi efficienti per il calcolo della IDFT nel caso in cui n non sia una potenza di 2 ma sia altamente fattorizzabile.

L'implementazione degli algoritmi di Cooley e Tukey e di Sande e Tukey fatta con un linguaggio di programmazione che permette la ricorsività è molto semplice. È però possibile dare una implementazione più efficiente "in place" che evita di usare esplicitamente la ricorsività. Infatti, applicando ricorsivamente la (6) si ottiene che nella parte destra della fattorizzazione finale si accumulano tutte le permutazioni di tipo pari-dispari fatte ai vari livelli dell'algoritmo. Cooley e Tukey hanno dimostrato che la composizione di tali permutazioni è data dalla permutazione *bit reversal* cioè la permutazione che associa i a j se la rappresentazione binaria di i su q bit, dove $n = 2^q$, coincide con la rappresentazione di j con i bit listati in ordine inverso. Cioè se $i = \sum_{k=0}^{q-1} 2^k i_k$ e $j = \sum_{k=0}^{q-1} 2^k j_k$, con $i_k, j_k \in \{0, 1\}$, sono le rappresentazioni binarie di i e di j allora $j_k = i_{q-k}$ per $k = 0, \ldots, q-1$. Una volta eseguita questa permutazione il resto del calcolo consiste nello svolgere semplici operazioni aritmetiche tra componenti contigue senza la necessità di dover applicare nessuna permutazione.

Maggiori informazioni a riguardo si possono trovare in questo link di Wikipedia.

Esistono algoritmi per il calcolo della DFT che non richiedono nessuna proprietà particolare dell'intero n ed hanno un costo computazionale limitato da $\alpha n \log_2 n$. Però il valore di α è molto maggiore di $3/2$ per cui il loro interesse è principalmente teorico.

Gli algoritmi di trasformata veloce sono numericamente stabili. Ad esempio, per quanto riguarda l'algoritmo di Cooley e Tukey vale il seguente risultato la cui dimostrazione è riportata in [8].

Teorema 2 *Per $n = 2^q$, q intero positivo, sia $\hat{y}$ il valore effettivamente calcolato al posto di $y = IDFT_n(z)$ con l'algoritmo di Cooley e Tukey in aritmetica floating point con precisione u dove i valori $\hat{\omega}_n^i$ effettivamente usati al posto delle radici n-esime ω_n^i sono tali che $|\hat{\omega}_n^i - \omega_n^i| \leq u$. Allora*

$$\frac{\|y - \hat{y}\|_2}{\|y\|_2} \leq \frac{\gamma u \log_2 n}{1 - \gamma u \log_2 n}$$

dove $\gamma = 1 + (\sqrt{2} + u)\frac{4}{1-4u} = 1 + 4\sqrt{2} + O(u)$.

Ne segue che l'errore algoritmico relativo, misurato in norma 2, cresce col logaritmo dell'ordine della trasformata.

2.2 FFT in Matlab

Gli algoritmi FFT per vettori di lunghezza arbitraria sono implementati in Matlab nelle function `fft` e `ifft` con una piccola differenza notazionale. Il vettore ottenuto col comando `v = fft(u)` è tale che $v_i = \sum_{j=0}^{n-1} \omega_n^{-ij} u_j$ mentre il vettore ottenuto con `u = ifft(v)` è tale che $u_i = \frac{1}{n}\sum_{j=0}^{n-1} \omega_n^{ij} v_j$. Cioè la normalizzazione $\frac{1}{n}$ è utilizzata nella definizione della trasformata inversa, e, come radice primitiva, è considerata ω_n^{-1}, cioè l'ordinamento delle radici n-esime scelto da Matlab è quello orario.

3 DFT su altri campi

Se il vettore z di cui calcolare la IDFT è reale, allora si verifica che il vettore $y = \mathrm{IDFT}_n(z)$ ha le componenti y_0 e $y_{\frac{n}{2}}$ (se n è pari) reali, mentre le rimanenti componenti sono tali che $y_j = \bar{y}_{n-j}$, $j = 1, \ldots, \frac{n}{2}$. Questa proprietà può essere usata per ridurre leggermente il costo computazionale degli algoritmi veloci descritti nel caso complesso.

La definizione di DFT e IDFT e gli algoritmi di trasformata veloce possono essere dati anche su campi finiti o, sotto condizioni aggiuntive, su anelli in cui esiste una radice n-esima dell'unità. Ad esempio l'insieme $\mathbb{Z}_{17}$ costituisce un campo e l'elemento 2 è una radice ottava primitiva dell'unità. Infatti le sue potenze (modulo 17) sono date nell'ordine da

$$2, 4, 8, 16, 15, 13, 9, 1.$$

Mentre il numero 3 è una radice 16-ma primitiva; le sue potenze (modulo 17) sono date nell'ordine da

$$3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6, 1$$

È possibile costruire la matrice di Fourier Ω_{16} su $\mathbb{Z}_{17}$ e definire la DFT e la IDFT. Infatti tutte le elaborazioni che abbiamo fatto finora si applicano poiché abbiamo usato solamente le proprietà di campo, l'esistenza di una radice n-esima e la sua primitività. L'operazione di coniugazione di una radice n-esima va vista come calcolo del reciproco.

La situazione diventa più delicata sugli anelli. Ad esempio su $\mathbb{Z}_{16}$ il numero 3 è una radice quarta dell'unità, le sue potenze sono nell'ordine

$$3, 9, 11, 1$$

e le "coniugate" cioè i reciproci sono

$$11, 9, 3, 1$$

Però la matrice di Fourier Ω_4 e la sua "coniugata", date da

$$\Omega_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 3 & 9 & 11 \\ 1 & 9 & 1 & 9 \\ 1 & 11 & 9 & 3 \end{bmatrix}, \quad \Omega_4^H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 11 & 9 & 3 \\ 1 & 9 & 1 & 9 \\ 1 & 3 & 9 & 11 \end{bmatrix}$$

sono tali che

$$\Omega_4^H \Omega_4 = \begin{bmatrix} 4 & 8 & 4 & 8 \\ 8 & 4 & 8 & 4 \\ 4 & 8 & 4 & 8 \\ 8 & 4 & 8 & 4 \end{bmatrix} \neq 4I \bmod 16.$$

Infatti il lemma 1 non è più valido. La dimostrazione del lemma non vale più visto che si richiedeva che non esistessero divisori dello zero. Inoltre, nel nostro caso il valore $n = 4$ non ha reciproco modulo 16. Per cui, anche se il prodotto fosse nI, non si potrebbe scrivere l'inversa di Ω_n come $\frac{1}{n}\Omega_n^H$.

Per potere definire la DFT e la IDFT su un anello dobbiamo garantire l'esistenza di una radice n-esima ω_n che sia *principale* cioè sia tale che $\sum_{j=0}^{n-1} \omega_n^{ij} = 0$ per $i = 1, \ldots, n$, e che n sia coprimo con la caratteristica dell'anello.

4 Applicazioni

La trasformata discreta di Fourier ha applicazioni in numerosi campi. Diamo un breve cenno su alcuni esempi di applicazioni.

4.1 Aritmetica di polinomi

Supponiamo di aver assegnato i coefficienti di due polinomi $a(t)$ e $b(t)$ di grado p e definiamo il polinomio prodotto $c(t) = a(t)b(t)$. I coefficienti di $c(t)$ sono

legati ai coefficienti di $a(t)$ e $b(t)$ dalle relazioni

$$\begin{aligned}
&c_0 = a_0 b_0 \\
&c_1 = a_0 b_1 + a_1 b_0 \\
&\cdots \\
&c_i = \textstyle\sum_{r+s=i} a_r b_s \\
&\cdots \\
&c_{2p-1} = a_{p-1} b_p + a_p b_{p-1} \\
&c_p = a_p b_p
\end{aligned}$$

Il calcolo dei coefficienti di $c(t)$ svolto con le formule precedenti richiede $O(p^2)$ operazioni aritmetiche.

È possibile calcolare i coefficienti di $c(t)$ usando la FFT con un costo asintoticamente più basso nel grado dei fattori. Procediamo nel seguente modo

Algoritmo 1.

- sia n la più piccola potenza intera di 2 maggiore del grado di $c(t)$
- si calcola $y^{(a)} = (y_i^{(a)})$, $y_i^{(a)} = a(\omega_n^i)$, $i = 0, \ldots, n-1$, con una IDFT_n;
- si calcola $y^{(b)} = (y_i^{(b)})$, $y_i^{(b)} = b(\omega_n^i)$, $i = 0, \ldots, n-1$, con una IDFT_n;
- si calcola $c(\omega_n^i) = y_i^{(a)} y_i^{(b)}$, $i = 0, \ldots, n-1$ con n moltiplicazioni
- si interpolano i valori di $c(\omega_n^i)$ ottenuti al passo precedente e si ottengono i coefficienti di $c(t)$ mediante una DFT_n

L'algoritmo richiede il calcolo di 3 trasformate veloci di Fourier n moltiplicazioni per valutare $c(\omega_n^i)$ più n moltiplicazioni per $1/n$ nel calcolo della DFT. Il costo complessivo è dominato da $\frac{9}{2} n \log_2 n$ operazioni aritmetiche.

Gli algoritmi FFT possono essere usati anche per calcolare il reciproco modulo t^n di un polinomio $p(t)$, tale che $p_0 = p(0) \neq 0$, con costo $O(n \log n)$. Infatti si può dimostrare che la successione di polinomi $x^{(k)}(t)$ definiti da

$$\begin{aligned}
&x^{(k+1)}(t) = 2x^{(k)}(t) - (x^{(k)}(t))^2 p(t) \bmod t^{2^{k+1}}, \quad k = 0, 1, \ldots, \lceil \log_2 n \rceil \\
&x^{(0)}(t) = 1/p_0
\end{aligned}$$

è tale che $x^{(k)}(t)p(t) = 1 \bmod t^{2^k}$. Infatti gli elementi di questa successione, si ottengono applicando formalmente il metodo di Newton all'equazione $x(t)^{-1} - p(t) = 0$ e la proprietà descritta sopra segue dalla convergenza quadratica del metodo di Newton. Si può verificare che utilizzando il metodo FFT per il calcolo dei prodotti di polinomi nell'iterazione di Newton, il costo complessivo per il calcolo del reciproco modulo t^n rimane dell'ordine di $n \log_2 n$. Per maggiori dettagli si veda [2].

4.2 Aritmetica degli interi

Le proprietà della DFT possono essere usate per costruire algoritmi veloci per la moltiplicazione di interi. Si considerino due interi positivi a e b ciascuno rappresentato in base B con al più p cifre. Per comodità possiamo assumere che B sia la familiare base 10; nelle implementazioni su computer B è la più realistica base 2. Per calcolare le cifre del prodotto $c = ab$ generalmente andiamo a calcolare tutti i prodotti possibili tra le cifre di a e quelle di b svolgendo p^2 moltiplicazioni più circa altrettante addizioni, inclusi i riporti, per ottenere il risultato. Ad esempio per moltiplicare 123 con 257 si procede generalmente così

```
    123 x
    257 =
---------
    861
   615
  246
---------
  31611
```

Siamo quindi abituati a svolgere più di p^2 operazioni elementari tra cifre per calcolare il prodotto di interi di p cifre. Questo ci creerebbe problemi se gli interi da moltiplicare avessero decine di cifre. Anche in un computer questo metodo può richiedere tempi di calcolo elevati se i numeri da moltiplicare hanno milioni di cifre come può capitare in certe applicazioni legate alla crittoanalisi.

L'uso della DFT permette di accelerare significativamente questo calcolo. Per vedere questo scriviamo gli interi a, b e c nella loro rappresentazione in base

$$a = \sum_{i=0}^{p-1} a_i B^i, \quad b = \sum_{i=0}^{p-1} b_i B^i, \quad c = \sum_{i=0}^{2p-1} c_i B^i$$

e associamo ad essi i polinomi

$$a(t) = \sum_{i=0}^{p-1} a_i t^i, \quad b(t) = \sum_{i=0}^{p-1} b_i t^i, \quad c(t) = \sum_{i=0}^{2p-1} c_i t^i.$$

In questo modo vale $a = a(B)$, $b = b(B)$, $c = c(B)$.

Osserviamo ora che il polinomio prodotto $\hat{c}(t) = b(t)a(t)$ assume anch'esso il valore dell'intero c nel punto B, cioè vale $\hat{c}(B) = c$. Però i suoi coefficienti $\hat{c}_i$ sono in generale diversi da quelli di $c(t)$. Infatti, i coefficienti c_i di $c(t)$ sono singole cifre mentre i coefficienti $\hat{c}_i$ di $\hat{c}(t)$ sono somma di p termini, ciascuno è il prodotto di due cifre. Il loro valore non può superare quindi $(B-1)^2 p$ per cui in base B saranno rappresentati da non più di $2 + \log_B p$ cifre.

I coefficienti di $\hat{c}(t)$ possono essere calcolati con l'algoritmo 1 eseguendo $O(p \log p)$ operazioni aritmetiche. Poiché i valori che vanno calcolati sono rappresentabili con $O(\log p)$ cifre, e la trasformata veloce di Fourier è numericamente stabile, per calcolare i coefficienti di $\hat{c}(t)$ è sufficiente eseguire i calcoli

con una aritmetica floating point dotata di $O(\log p)$ cifre. Se questa aritmetica viene implementata col metodo standard per moltiplicare numeri floating point (o interi) il costo computazionale di ogni moltiplicazione diventa $O(\log^2 p)$ operazioni elementari tra cifre. In questo modo il costo totale, espresso in termini di operazioni elementari tra cifre, per il calcolo dei coefficienti $\hat{c}_i$, dato dal numero di operazioni aritmetiche per il costo di ciascuna operazione aritmetica, diventa $O((p \log p)(\log^2 p)) = O(p \log^3 p)$.

Una volta che i coefficienti $\hat{c}_i$ sono stati calcolati possiamo recuperare le cifre c_i dalle cifre dei numeri $\hat{c}_i$ con $O(p \log p)$ addizioni. Lasciamo i dettagli di questo calcolo al lettore.

In questo modo si riesce a calcolare i coefficienti del prodotto di due interi con un numero di operazioni elementari tra cifre dell'ordine di $p \log^3 p$, quindi inferiore al costo p^2 dell'algoritmo standard di moltiplicazione. È possibile ridurre ulteriormente il costo se la moltiplicazione di numeri di $O(\log p)$ cifre, anziché calcolarla con l'algoritmo standard di costo quadratico, viene calcolata ricorsivamente con l'algoritmo che stiamo descrivendo.

Un algoritmo più efficiente, basato sulla DFT ambientata in anelli finiti, è *l'algoritmo di Schoenhage-Strassen.* La complessità di questo algoritmo è di $O(p \log p \log \log p)$ operazioni elementari tra bit. L'algoritmo è stato migliorato nel 2007 da Martin Fürer. L'algoritmo di Fürer ha una complessità di $O(p \log p \, 2^{\log^* p})$ operazioni elementari dove $\log^* p$ è uguale al numero di volte in cui $\log_2$ compare nell'espressione $\log_2 \log_2 \cdots \log_2 p$ affinché il risultato sia compreso tra 0 e 1.

Sebbene l'algoritmo di Fürer sia asintoticamente più veloce dell'algoritmo di Schönhage - Strassen, in pratica diventa più efficiente solo per valori di p estremamente elevati. Per cui nella pratica non viene utilizzato nelle implementazione dei sistemi di calcolo in multiprecisione.

Maggiori informazioni sul metodo di Fürer si trovano nell'articolo "Faster integer multiplication". Informazioni sui metodi per la moltiplicazione di interi si trovano nella pagina di Wikipedia "Multiplication Algorithm"

Recentemente è uscito un articolo di David Harvey, Joris van der Hoeven [7] in cui si dimostra che bastano $O(n \log n)$ operazioni tra bit per calcolare il prodotto di interi dotati di n bit.

4.3 Interpolazione trigonometrica

Definiamo polinomio trigonometrico una espressione del tipo

$$F(x) = \begin{cases} \frac{\alpha_0}{2} + \sum_{j=1}^{m-1} (\alpha_j \cos jx + \beta_j \sin jx) & \text{se } n = 2m-1 \\ \frac{\alpha_0}{2} + \sum_{j=1}^{m-1} (\alpha_j \cos jx + \beta_j \sin jx) + \frac{\alpha_m}{2} \cos mx & \text{se } n = 2m \end{cases}$$

dove $\alpha_j, \beta_j \in \mathbb{R}$.

Il seguente risultato ci fornisce l'espressione del polinomio trigonometrico che interpola i valori $(x_i, y_i) \in \mathbb{R}^2$, $x_i = 2i\pi/n$ per $i = 0, \ldots, n-1$.

Teorema 3 *Sia* $z = (z_0, \ldots . z_{n-1}) = \mathrm{DFT}_n(y)$*, dove* $y = (y_0, \ldots, y_{n-1})$*. Il polinomio trigonometrico* $F(x)$ *definito dai coefficienti*

$$\alpha_i = 2\mathrm{re}(z_i) = \frac{2}{n}\sum_{j=0}^{n-1} y_j \cos ix_j, \quad \beta_i = -2\mathrm{im}(z_i) = \frac{2}{n}\sum_{j=0}^{n-1} y_j \sin ix_j$$

è tale che $F(x_i) = y_i$*,* $x_i = 2i\pi/n$*,* $i = 0, \ldots, n-1$*. Inoltre tale polinomio trigonometrico è unico.*

La dimostrazione di questo risultato è una semplice applicazione di note formule trigonometriche e non viene riportata.

Dal punto di vista computazionale i coefficienti del polinomio trigonometrico che interpola i valori (x_i, y_i) si calcolano con circa $\frac{2}{3}n\log_2 n$ operazioni aritmetiche se n è potenza di 2. Infatti, dati i valori di $y = (y_i)$ basta calcolare $z = \mathrm{DFT}_n(y)$ e ricavarne parte reale e immaginaria. Viceversa, i valori che un polinomio trigonometrico $F(x)$ assegnato in termini dei coefficienti α_i e β_i assume nei punti x_i si possono calcolare ancora con circa $\frac{3}{2}n\log_2 n$ operazioni aritmetiche. Infatti, dati i coefficienti α_i e β_i, basta calcolare i valori $z_i = 2(\alpha_i - \mathtt{i}\beta_i)$ e porre $y = \mathrm{IDFT}_n(z)$.

Questo fatto permette di effettuare operazioni di "filtraggio" di segnali e immagini digitali in modo efficiente.

Riferimenti bibliografici

[1] R. Bevilacqua, D.A. Bini, M. Capovani, O. Menchi. *Metodi Numerici*. Zanichelli, Bologna 1992

[2] D.A. Bini, V. Pan *Polynomial and Matrix Computations*, Birkhäuser, 1994.

[3] CooleyTukey FFT algorithm, Wikipedia `http://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm`

[4] Fast Fourier transform, Wikipedia `http://en.wikipedia.org/wiki/Fast_Fourier_transform`

[5] Fürer's Algorithm, Wikipedia `http://en.wikipedia.org/wiki/F%C3%BCrer%27s_algorithm`

[6] Martin Fürer, Fast Integer Multiplication, SIAM J. Comput., 39(3), 2009, 9791005.

[7] David Harvey, Joris van der Hoeven, Integer multiplication in time O(n log n), 2019. hal-02070778. `https://hal.archives-ouvertes.fr/hal-02070778/document`

[8] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia 2002.

[9] Multiplication Algorithm, Wikipedia `http://en.wikipedia.org/wiki/Multiplication_algorithm`

[10] Schönhage-Strassen Algorithm, Wikipedia http://en.wikipedia.org/wiki/Sch%C3%B6nhage%E2%80%93Strassen_algorithm