1 Calcolo approssimato della serie di Fourier

Come è noto, se $f(x)$ è una funzione periodica di periodo L di classe $\mathbf{C^1}$ e si definisce, per ogni intero k,

$$F_k = \frac{1}{L}\int_{-L/2}^{L/2} dx\ f(x)\ e^{i\frac{2\pi}{L}kx} \tag{1.1}$$

allora, se N è un intero pari,

$$f_N(x) \equiv \sum_{k=-N/2}^{N/2-1} F_k e^{-i\frac{2\pi}{L}kx} \xrightarrow[N\to\infty]{} f(x) \tag{1.2}$$

la convergenza essendo uniforme sui compatti. Nel caso in cui $f(x)$ sia di classe $\mathbf{C^1}$ a tratti, nei punti di discontinuità $f_N(x)$ è ancora convergente, ma

$$f_N(x) \xrightarrow[N\to\infty]{} \frac{f(x_+)+f(x_-)}{2} \tag{1.3}$$

Supponiamo ora di calcolare approssimativamente i coefficienti F_k, sostituendo nella (1.1) l'integrale con la somma di Riemann $F_k^{(N)}$, corrispondente ad una suddivisione dell'intervallo $[-L/2, L/2]$ in N parti uguali:

$$F_k^{(N)} = \frac{1}{L}\sum_{r=0}^{N-1} \frac{L}{N} f(x_r)\ e^{i\frac{2\pi}{L}kx_r} \quad , \quad x_r = -\frac{L}{2} + \frac{L}{N}r \tag{1.4}$$

Si vede subito che

$$F_k^{(N)} = \frac{(-1)^k}{N}\sum_{r=0}^{N-1} f(x_r)\ e^{i\frac{2\pi}{N}kr} \tag{1.5}$$

e che $F_k^{(N)}$ è periodica in k di periodo N. Quest'ultima proprietà implica anche che $F_k^{(N)}$ non è una buona approssimazione di F_k per k dell'ordine di N. Tuttavia, usando l'identità

$$\frac{1}{N}\sum_{k=0}^{N-1} e^{i\frac{2\pi}{N}k(r-s)} = \delta_{r,s} \tag{1.6}$$

è facile verificare che

$$f(x_r) = \sum_{k=-N/2}^{N/2-1} F_k^{(N)} e^{-i\frac{2\pi}{L}kx_r} \quad , \quad r = 0, \ldots, N-1 \tag{1.7}$$

cioè la sostituzione di F_k con $F_k^{(N)}$ nella (1.2) permette di ricostruire esattamente i valori della funzione $f(x)$ nei punti x_r, $r = 0, \ldots, N-1$.

2 Calcolo approssimato della trasformata di Fourier

Sia $f(t)$ una funzione continua definita su $\mathbb{R}$ e sommabile. Allora è ben definita la sua *trasformata di Fourier*

$$F(\nu) = \int_{-\infty}^{+\infty} dt\ f(t)\ e^{i2\pi\nu t} \tag{2.8}$$

e si può dimostrare che

$$f(t) = \int_{-\infty}^{+\infty} d\nu\ F(\nu)\ e^{-i2\pi\nu t} \tag{2.9}$$

Inoltre, $F(\nu)$ è tanto più regolare quanto più velocemente va a 0 $f(t)$ quando $t \to \infty$, così come $F(\nu)$ va tanto più velocemente a 0 per $\nu \to \infty$ quanto più regolare è $f(t)$.

$F(\nu)$ può essere ben approssimata, per T abbastanza grande, dall'integrale

$$F_T(\nu) = \int_{-T/2}^{+T/2} dt\ f(t)\ e^{i2\pi\nu t} \tag{2.10}$$

Si noti ora che, se $\nu = \nu_k \equiv k/T$, con k intero, $T^{-1}F_T(\nu_k)$ coincide con il coefficiente di Fourier di indice k della funzione periodica $f_T(t)$, ottenuta restringendo la funzione $f(t)$ all'intervallo $[-T/2, T/2]$ e poi prolungandola come funzione periodica di periodo T a tutto $\mathbb{R}$. Pertanto, per la (1.5),

$$T^{-1}F_T(\nu_k) \simeq F_k^{(T,N)} \equiv \frac{(-1)^k}{N} \sum_{r=0}^{N-1} f(-T/2 + rT/N)\ e^{i\frac{2\pi}{N}kr} \tag{2.11}$$

Ci si aspetta (e lo si può dimostrare sotto opportune ipotesi di regolarità) che, per ogni $\nu \in \mathbb{R}$,

$$F(\nu) = \lim_{T\to\infty} \left(T \lim_{N\to\infty} F_{[T\nu]}^{(T,N)} \right) \tag{2.12}$$

dove $[\cdot]$ indica la parte intera.

Si noti che, se $|k| \leq N/2$, $|\nu_k| \leq \nu^* = N/(2T)$; pertanto, essendo $F_k^{(T,N)}$ periodica in k di periodo N, la (2.11) permette di calcolare approssimativamente la trasformata di Fourier solo nell'intervallo $[-\nu^*, \nu^*]$ (ν^* è detta frequenza di Nyquist) e non ci si deve aspettare che l'approssimazione sia buona vicino agli estremi di questo intervallo.

3 Trasformata di Fourier veloce

Nei paragrafi 1 e 2 abbiamo mostrato che il calcolo approssimato dei coefficienti di Fourier e della trasformata di Fourier possono ambedue ridursi al calcolo della cosiddetta *Trasformata di Fourier discreta di ordine N*, definita come la trasformazione che associa ad ogni N-pla di numeri complessi $f_0, \ldots, f_{N-1}$ la N-pla

$$F_k = \frac{1}{N} \sum_{r=0}^{N-1} f_r \; e^{i\frac{2\pi}{N}kr} \tag{3.13}$$

Inoltre, per quanto abbiamo detto nel par. 1, questa trasformazione è invertibile e si ha

$$f_r = \sum_{k=0}^{N-1} F_k \; e^{-i\frac{2\pi}{N}kr} \tag{3.14}$$

Vogliamo ora illustrare, nel caso in cui N è una potenza di 2, un algoritmo di calcolo dei coefficienti F_k (*metodo di Cooley-Tukey*), che è notevolmente più efficiente rispetto a quello banale, basato sulla definizione (3.13) (la quale richiede un numero di operazioni dell'ordine di N^2). Supponiamo pertanto che $N = 2^M$, con M intero, e notiamo che la (3.13) può riscriversi nella forma

$$\hat{f}_k \equiv NF_k = \sum_{j=0}^{N/2-1} f_{2j} W_N^{2jk} + W_N^k \sum_{j=0}^{N/2-1} f_{2j+1} W_N^{2jk} \quad , \quad W_N = e^{i\frac{2\pi}{N}} \tag{3.15}$$

D'altra parte $W_N^{2jk} = W_{N/2}^{jk}$, per cui

$$\hat{f}_k = \hat{f}_k^0 + W_N^k \hat{f}_k^1 \quad , \quad \hat{f}_k^{a_0} \equiv \sum_{j=0}^{N/2-1} f_{2j+a_0} W_{N/2}^{jk} \quad , \quad a_0 \in \{0, 1\} \tag{3.16}$$

L'osservazione cruciale è che $\hat{f}_k^{a_0}$ coincide con la TFD (trasformata di Fourier discreta) di ordine $N/2$ dei numeri f_{2j+a_0}, $j = 0, \ldots, N/2-1$ (è pertanto periodica in k di periodo $N/2$), per cui la (3.16) permette di definire un algoritmo iterativo, la cui utilità è basata sulle seguente ulteriori osservazioni. Si noti che

$$W_N^{k+N/2} = -W_N^k \quad , \quad k = 0, \ldots, N/2-1 \tag{3.17}$$

Pertanto, se si pone

$$A_k = \begin{cases} \hat{f}_k^0 & k = 0, \ldots, N/2-1 \\ \hat{f}_{k-N/2}^1 & k = N/2, \ldots, N-1 \end{cases} \tag{3.18}$$

si ha

$$\hat{f}_k = \begin{cases} A_k + W_N^k A_{k+N/2} & k = 0, \ldots, N/2-1 \\ A_{k-N/2} - W_N^{k-N/2} A_k & k = N/2, \ldots, N-1 \end{cases} \tag{3.19}$$

Ne segue in particolare che, per ogni intero $k \in [0, N/2-1]$, il calcolo di $\hat{f}_k$ e $\hat{f}_{k+N/2}$ coinvolge solo i valori di A_k e $A_{k+N/2}$ ed il calcolo di W_N^k. Pertanto è possibile calcolare il vettore $\hat{\mathbf{f}} = (\hat{f}_0, \ldots, \hat{f}_{N-1})$, noto il vettore $\mathbf{A} = (A_0, \ldots, A_{N-1})$, senza che sia necessario allocare altra memoria oltre quella necessaria per memorizzare $\mathbf{A}$; basta aggiornare consecutivamente le posizioni di memoria corrispondenti ad A_k e $A_{k+N/2}$, $k \in [0, N/2-1]$, utilizzando una singola variabile di appoggio.

Iteriamo ora la decomposizione (3.16); si trova

$$\begin{aligned} \hat{f}_k^{a_0} &= \hat{f}_k^{a_0,0} + W_{N/2}^k \hat{f}_k^{a_0,1} \quad , \quad k \in [0, N/4-1] \\ \hat{f}_k^{a_0,a_1} &= \sum_{j=0}^{N/4-1} f_{4j+2a_1+a_0} W_{N/4}^{jk} \end{aligned} \tag{3.20}$$

$\hat{f}_k^{a_0,a_1}$, $a_0, a_1 \in \{0,1\}$, essendo una TFD di ordine $N/4$ che coinvolge solo i numeri $f_{4j+2a_1+a_0}$, $j \in [0, n/4-1]$. Questo procedimento può essere ripetuto per M volte (comprese le prime due); dopo s iterazioni, $1 < s < M$, si trova

$$\begin{aligned} \hat{f}_k^{a_0,\ldots,a_{s-2}} &= \hat{f}_k^{a_0,\ldots,a_{s-2},0} + W_{N/2^{s-1}}^k \hat{f}_k^{a_0,\ldots,a_{s-2},1} \quad , \quad k \in [0, N/2^s-1] \\ \hat{f}_k^{a_0,\ldots,a_{s-1}} &= \sum_{j=0}^{N/2^s-1} f_{2^s j+2^{s-1}a_{s-1}+\ldots+2a_1+a_0} W_{N/2^s}^{jk} \end{aligned} \tag{3.21}$$

L'iterazione si arresta al passo M-esimo, cioè quando si arriva a definire le TFD di ordine 2

$$\begin{aligned} \hat{f}_k^{a_0,\ldots,a_{M-2}} &= \hat{f}_k^{a_0,\ldots,a_{M-2},0} + W_2^k \hat{f}_k^{a_0,\ldots,a_{M-2},1} = \\ &= \hat{f}_k^{a_0,\ldots,a_{M-2},0} + (-1)^k \hat{f}_k^{a_0,\ldots,a_{M-2},1} \quad , \quad k = 0,1 \\ \hat{f}_k^{a_0,\ldots,a_{M-2},a_{M-1}} &= f_{2^{M-1}a_{M-1}+\ldots+2a_1+a_0} \end{aligned} \tag{3.22}$$

Le considerazioni precedenti mostrano che è possibile costruire un algoritmo che, a partire da un vettore $\mathbf{A}$ con N componenti, contenente i numeri f_j, $j \in [0, N-1]$, riordinati coerentemente con la definizione contenuta nell'ultima riga della (3.22), permette di costruire il vettore $\hat{\mathbf{f}}$, aggiornando M volte il vettore $\mathbf{A}$. Basta fare iterativamente i calcoli indicati nella prima riga della (3.21), partendo da $s = M-2$ e finendo con $s = -1$, interpretando la (3.21) come la (3.16), se $s = -1$. È facile convincersi che, per ogni s, il calcolo coinvolge tutti i blocchi di 2^{M-s-1} elementi consecutivi del vettore $\mathbf{A}$. Poiché ad ogni passo bisogna compiere un numero di operazioni proporzionale ad N, il numero totale di operazioni è proporzionale a $NM = N \log_2 N$.

L'unico problema che rimane è individuare un algoritmo che produce, con un numero di operazioni ancora di ordine NM, l'ordinamento corretto dei dati di partenza, cioè quello tale che, se si pone

$$k = 2^{M-1}a_0 + 2^{M-2}a_1 + \ldots + 2a_{M-2} + a_{M-1} \quad , \quad a_i \in \{0,1\} \tag{3.23}$$

allora

$$A_k = f_{j(k)} \quad , \quad j(k) = 2^{M-1}a_{M-1} + \ldots + 2a_1 + a_0 \tag{3.24}$$

Si noti che $j(k)$ si ottiene da k invertendone la rappresentazione in base 2; ne segue in particolare che $j(0) = 0$ e che $j(j(k)) = k$. Possiamo pertanto procedere nel modo seguente. Supponiamo di aver calcolato $j(k)$ per un certo $k < N - 1$; allora, per calcolare $j(k + 1)$, osserviamo che, se definiamo

$$k + 1 = 2^{M-1}\tilde{a}_0 + 2^{M-2}\tilde{a}_1 + \ldots + 2\tilde{a}_{M-2} + \tilde{a}_{M-1} \tag{3.25}$$

e poniamo

$$\bar{r} = \max\{r \leq M - 1 : a_r = 0\} \tag{3.26}$$

($\bar{r}$ è sempre definito se $k < 2^M - 1 = N - 1$) allora

$$\tilde{a}_r = \begin{cases} 0 & r > \bar{r} \\ 1 & r = \bar{r} \\ a_r & r < \bar{r} \end{cases} \tag{3.27}$$

Ne segue facilmente che $j(k + 1) = j(k) + 2^{M-1}$, se $\bar{r} = M - 1$, altrimenti

$$j(k + 1) = j(k) + 2^{\bar{r}} - \sum_{r=\bar{r}+1}^{M-1} 2^r \tag{3.28}$$

Basta pertanto inizializzare un vettore J ad N componenti così che $J = (0, 1, \ldots, N - 1)$ ed elaborare il calcolo appena descritto per valori crescenti di k, partendo da $k = 1$; ad ogni passo dell'iterazione, se $j(k) > k$, si scambia la coppia $(k, j(k))$. Il fatto che $j(j(k)) = k$, insieme alla condizione $j(k) > k$, garantisce che ogni valore di k può essere coinvolto al più una volta in uno scambio; pertanto, alla fine dell'iterazione, J contiene gli indici riordinati nel modo corretto.

4 Programma Scilab per il calcolo della FFT

Il programma seguente, scritto in Scilab, realizza l'algoritmo descritto nel par. 3, in termini di una funzione con due argomenti. Più precisamente, se il primo argomento (**data**) è un vettore complesso a n componenti (con n potenza di 2) $(f_0, f_1, \ldots, f_{n-1})$ ed il secondo (**segno**) è eguale a $+1$, il programma calcola il vettore $(\hat{f}_0, \ldots, \hat{f}_{n-1})$, definito dalla (3.15); se invece la variabile **segno** è eguale a -1 ed il vettore **data** contiene i numeri $(F_0, \ldots, F_{n-1})$, il programma calcola i numeri $(f_0, f_1, \ldots, f_{n-1})$, usando la (3.14).

Se la funzione viene chiamata nella forma **x=FFT(x,segno)**, il risultato del calcolo è memorizzato nello stesso vettore **x**, contenente i dati di partenza. Se si vuole conservare il vettore **x** dopo il calcolo della FFT, si deve usare una chiamata del tipo **y=FFT(x,segno)**; **y** sarà un vettore di dimensione

n contenente il risultato del calcolo. Ciò è dovuto al fatto che in Scilab le variabili sono passate alle funzioni come valore, per cui l'unico modo di modificare una variabile tramite una funzione è di metterla sia fra le variabili di ingresso che fra quelle di uscita.

```
function data=FFT(data,segno)
n=size(data,2);
// Realizza l'algoritmo di riordinamento descritto nel
// paragrafo precedente. Si noti che, rispetto alla
// notazione di quel paragrafo, i=k+1, j=j(k)+1
nhalf=n/2;
j=1;
for i=1:n
// Al primo passo di questo ciclo i=j=1=j(0), per cui l'unico
// effetto delle istruzioni seguenti e' di porre j=n/2+1=j(1)+1,
// il valore corrispondente a k=1, quindi a i=2. Al passo
// successivo viene effettuato lo scambio fra data(2) e data(j)
// e viene calcolato il valore di j corrispondente a i=3.
// Al passo i-esimo, viene effettuato lo scambio fra data(i) e
// data(j), ma solo se j>i, quindi viene calcolato il valore
// di j corrispondente a i+1.
    if(j>i) then
        temp=data(i);
        data(i)=data(j);
        data(j)=temp;
    end
m=nhalf;
while m>=2 & j>m
    j=j-m;
    m=m/2;
end
j=j+m;
end
// Calcola la TFD tramite l'algoritmo iterativo descritto
// nel paragrafo precedente. Si comincia con il calcolare
// le trasformate delle coppie di dati consecutivi, quindi
// si calcolano in successione quelle di ordine crescente,
// tramite un ciclo "while" controllato dalla variabile kmax,
// che assume i valori 1,2,4,8,...,n/2.
// Al passo k del ciclo "for" principale si calcolano le
// componenti di indice k e k+kmax delle n/(2*kmax) trasformate
// di ordine step=2*kmax, tramite un ciclo secondario.
```

```
kmax=1;
while n>kmax;
    step=2*kmax;
    w0=exp(-2*segno*\%i*\%pi/step);w=1;
    for k=1:kmax
        for i=k:step:n
            j=i+kmax;
            temp=data(j)*w;
            data(j)=data(i)-temp;
            data(i)=data(i)+temp;
        end
    w=w*w0;
    end
kmax=step;
end
endfunction
```