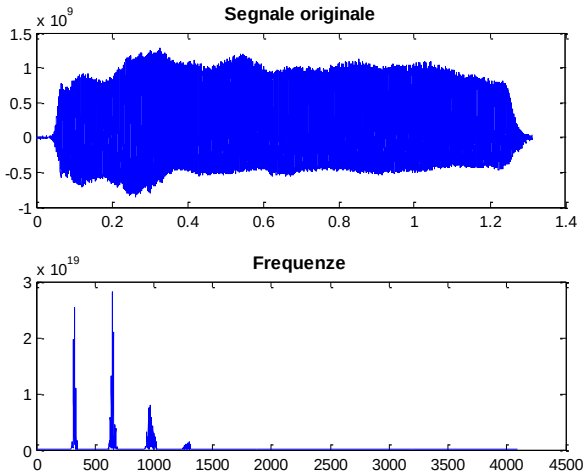
DFT -Algoritmo FFT-Applicazioni

La teoria che riguarda l'analisi di Fourier è studiata in altri corsi, in questo paragrafo daremo un breve cenno del problema che essa risolve e dove esso ha origine nelle applicazioni tecnico-scientifiche.
L'analisi di Fourier in sintesi consiste nella possibilità di rappresentare complicate strutture periodiche mediante combinazioni di semplici funzioni periodiche come seno e coseno, ossia nella trasformazione della funzione originale in un'altra, di tipo più semplice, così da ottenere informazioni non ovvie sulle proprietà della funzione originale.
Si ha che una funzione periodica f(t) si può esprimere come una combinazione lineare di funzioni del tipo $\cos(2\pi\nu x)$ e $\text{sen}(2\pi\nu x)$, dette **armoniche elementari**. I coefficienti $F(\omega)$ di tale combinazione hanno un particolare interesse : $F(\omega)$ = **ampiezza** della sinusoide corrispondente alla **frequenza** $\omega=2\pi\nu$.
Si ha cioè che una funzione può essere descritta dal suo valore f(t) al tempo t o dalla sua ampiezza $F(\omega)$ alla frequenza ω,il che consente di "vedere" le sue componenti ondulatorie.
Tale processo si basa sulla Trasformata di Fourier (FT) che converte f(t) in $F(\omega)$ e viceversa.
La FT è un *ponte* tra il dominio del tempo e quello delle frequenze di un fenomeno fisico, ossia è una lente che consente di "vedere" le componenti ondulatorie di un segnale. E' l'equivalente matematico di un prisma di cristallo che scompone un raggio di luce nelle sue frequenze pure (colori) che in esso non sono visibili.
Nella figura seguente vi è il grafico di un segnale sonoro nel tempo ed il grafico nel dominio delle frequenze, in cui si vedono le ampiezze delle differenti armoniche che sommate insieme creano il suono.



L'importanza dell'analisi di Fourier deriva dal fatto che molti fenomeni in natura hanno un carattere ciclico (dalla scala planetaria ai cristalli) e, grazie all'applicazione della serie e della trasformata di Fourier, si possono analizzare e studiare, ottenendo informazioni essenziali sulle loro caratteristiche, non deducibili immediatamente dal modello canonico. Inoltre la risoluzione di molti problemi matematici richiede il calcolo di somme di seni e coseni. Alcuni tipici campi di applicazione sono:

- interpolazione e approssimazione trigonometrica

- risoluzione di sistemi lineari
- problemi differenziali alle derivate parziali
- antenne
- ottica
- fisica dei quanti
- acustica
- elaborazione di segnali.

Trasformata Discreta di Fourier (DFT)

Poiché nelle applicazioni i dati sono discreti si usa la DFT (Trasformata Discreta di Fourier), che trasforma un vettore di reali o complessi in un altro vettore le cui componenti sono una combinazione di funzioni periodiche (seno e coseno).
Nella maggior parte delle applicazioni l'utilizzo della DFT deriva dal suo "legame" con la FT e con la serie di Fourier. In generale si applica nell'elaborazione di forme di informazione costituite da un numero discreto di dati. Una applicazione tra le più "originali" è l'algoritmo di Strassen, che è il più veloce algoritmo per il prodotto di due interi con molte cifre e consiste nel calcolare la DFT dei vettori delle cifre ed elaborare il risultato ottenuto così da ottenere il prodotto.
Dato un vettore(sequenza) *f* ad N componenti (reale o complesso), si definisce la trasformata discreta di Fourier (DFT) del vettore f il vettore (complesso):

$$F_k = \sum_{j=0}^{N-1} f_j w_N^{-jk} \quad , \text{k=0,…N-1}$$

dove $w_N = e^{2\pi i/N} = \cos(2\pi/N)+i\,\mathrm{sen}(2\pi/N)$, i=unità immaginaria.
La DFT è una trasformazione lineare tra vettori. La IDFT è la trasformazione inversa, che consente di riottenere, a partire dal vettore F, il vettore originario,IDFT del vettore F :

$$f_j = \frac{1}{N}\sum_{k=0}^{N-1} F_k w_N^{jk} \quad , \text{j=0,…N-1}$$

Si ottiene quindi una trasformazione di tipo circolare:

$$\underset{f}{\overset{DFT}{\rightarrow}}\ \underset{F}{\overset{IDFT}{\rightarrow}}\ f$$

Si ha:

$$F_k = \sum_{j=0}^{N-1} f_j \cos(\frac{2k\pi j}{N}) - i\sum_{j=0}^{N-1} f_j sen(\frac{2k\pi j}{N}) = a_k - ib_k \qquad \text{k=0,…N-1}$$

a_k=real(F_k),b_k=imag(F_k)
Se f è reale :

$$f_j = \frac{1}{N}\sum_{k=0}^{N-1}(a_k - ib_k)(\cos(\frac{2k\pi j}{N}) + isen(\frac{2k\pi j}{N})) = \frac{1}{N}\sum_{k=0}^{N-1} a_k \cos(\frac{2k\pi j}{N}) + b_k sen(\frac{2k\pi j}{N})$$

,j=0,…,N-1

Inoltre, posto per semplicità N pari, si ha che F_0 e $F_{N/2}$ sono reali e:

$$F_0 = \sum_{j=0}^{N-1} f_j = \text{DC component}$$ (componente continua o costante)

Per cui f_j in definitiva è la somma della DC component e di termini con frequenze diverse.

La DFT si può anche esprimere come prodotto di matrice per vettore:

$$F=Af$$

dove gli elementi della matrice di Fourier sono $a_{k,j}=w_N^{-jk}$.
In Matlab tale matrice può essere generata da:

```
>> n=4;
>> omega=exp(-2*pi*i/n);
>> j=0:n-1;
>> k=j';
>> A=omega.^(k*j)   %outer product dei due vettori
```

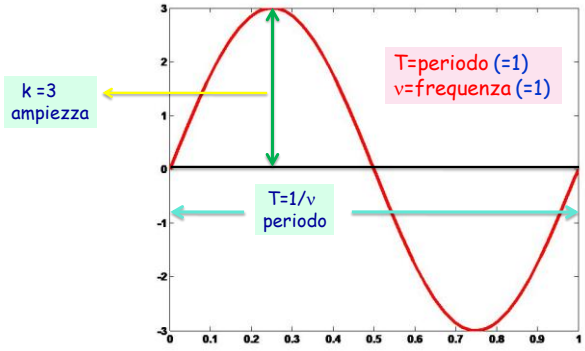
Analizziamo il significato e alcune proprietà della DFT.
Una funzione è periodica di periodo T (periodo fondamentale) se:

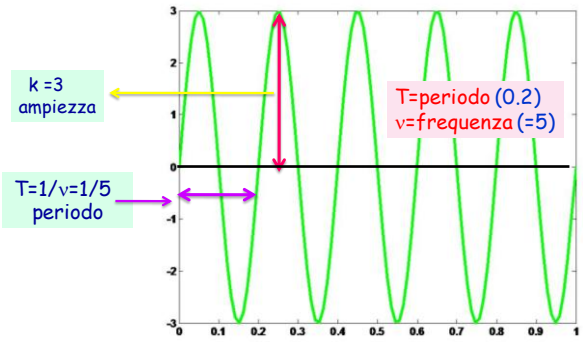
$$g(x+T)=g(x)$$ (è anche periodica di periodo 2T,3T,..)

osserviamo che: $h(x)=g(\nu x)$ è periodica di periodo T/ν.
Le funzioni $k\,sen(2\pi\nu t)$ e $k\cos(2\pi\nu t)$ sono dette armoniche elementari.
Si ha che:

- ν =frequenza (numero di oscillazioni per periodo)
- $T=1/\nu$ = periodo
- k=ampiezza
- $2\pi\nu$ =frequenza radiale o angolare

Se t è in secondi la frequenza si misura in cicli per secondo(hertz), se t è in unità di spazio il periodo è detto lunghezza d'onda e la frequenza numero d'onda.
Nelle figure seguenti vi sono i grafici di $3sen(2\pi t)$ e $3sen(2\pi 5t)$.





Data la funzione periodica f(t) in [0,T], per applicare la DFT bisogna campionare la funzione (spesso si hanno a disposizione solo i dati discreti), ossia fissare:

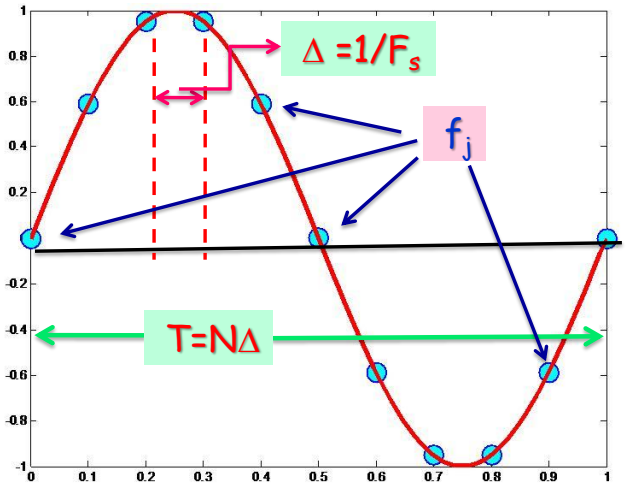
- Δ = unità di campionamento (passo di discretizzazione)
- $N = T/\Delta$ = numero di campioni

e generare:

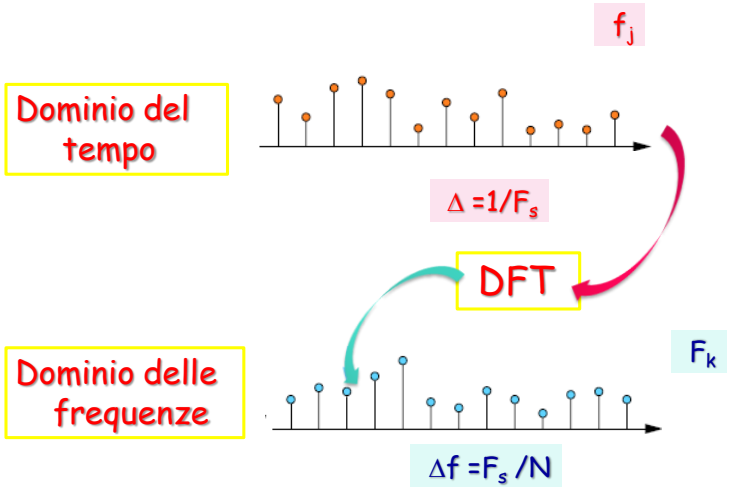
- $t_j = j\,\Delta$ j=0,…N-1 discretizzazione di [0,T]
- $f_j = f(t_j)$ j=0,…N-1

f_j è il vettore di campioni periodico di periodo N, ossia $f_{N+m}=f_m=f_{m-N}$, per ogni m.
Si ha:

- $T=N/\Delta$ = periodo fondamentale
- $F_S = 1/\Delta = N/T$ =frequenza di campionamento(numero campioni per unità di tempo)in Hz

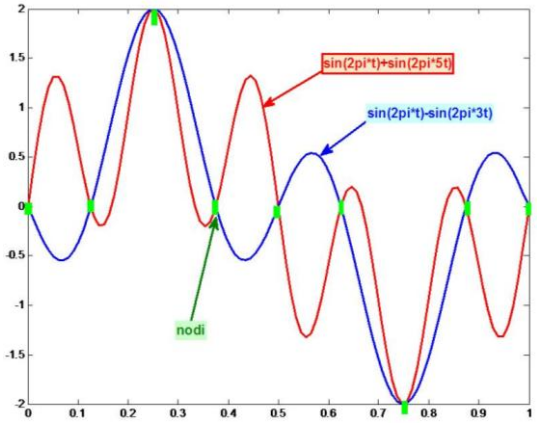


L'applicazione della DFT produce il vettore periodico, di periodo N, F_k, k=0,..,N-1; e l'applicazione della IDFT a tale vettore consente di rappresentare f_j come somma sulle frequenze angolari $2\pi jk/N = 2\pi j\Delta k/T = 2\pi t_j k/T$, k=0,…,N-1,da cui la frequenza fondamentale è $1/T = F_s/N$(è la frequenza più bassa # 0, cioè la minima distanza alla quale si devono trovare due frequenze per distinguerle ed è inversamente proporzionale ad N) e tutte le frequenze sono multipli interi di 1/T.



La frequenza maggiore dipende da Δ e si dimostra che è FN=1/(2Δ)=N/(2T)= frequenza di Nyquist (limite massimo delle frequenze ottenibili).
Ciò significa che, se f(t) ha frequenze in modulo $\leq$ FN, è completamente determinata dal campionamento scelto, e per avere frequenze maggiori bisogna diminuire il passo di discretizzazione. Se si campiona in [0,T], ho informazioni in (0, 1/(2Δ)).
Una spiegazione intuitiva è la seguente: si hanno bisogno di due campioni, uno nel picco in alto e uno in basso, per catturare un'oscillazione di un fenomeno periodico, ossia con N campioni la più piccola componente periodica che si può descrivere ha periodo 2T/N e quindi la frequenza più grande è N/(2T).
Se il campionamento non è sufficientemente fitto da catturare le frequenze più alte si verifica il fenomeno dell'aliasing.
Consideriamo ad esempio f(x)=sin(2πx)+sin(5πx) discretizzata con N=9 campioni in [0,1].

Nei nodi scelti sin(5πx) non si distingue dalla funzione con frequenza minore –sin(3πx), infatti FN=4. Solo aumentando il numero di nodi si ottiene correttamente la funzione a frequenza maggiore, ossia, fino a che la discretizzazione non è sufficientemente fitta per ottenere le frequenze più alte, queste andranno confuse quelle più basse .



Le frequenze che si possono considerare sono quindi tale che |freq|≤FN, cioè per k=-(N/2)+1,..., -1,0,1, ..., (N/2), ma k=0,…,N-1. Bisogna ricordare che F è periodico cioè $F_{N+m}=F_m=F_{m-N}$, ossia:

$F_{-N/2}=F_{N-N/2}= F_{N/2} =FN$
$F_{-N/2+1}=F_{N+(-N/2+1)}= F_{N/2+1}$
…..
$F_{N-1}=F_{-1}$

Da cui si ottiene:

- freq > 0 $1\leq k \leq N/2-1$
- freq < 0 $N/2+1\leq k \leq N-1$
- freq = 0 $k=0$ *spesso non si visualizza*
- Nyquist $k=N/2$

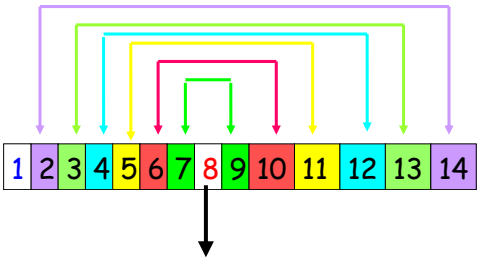
Riassumendo si ha:

- $T=N\Delta$ = periodo
- $F_S=1/\Delta =N/T$ = frequenza di campionamento
- $1/T =1/(N\Delta)=F_S/N$ = frequenza fondamentale
- $FN=1/(2\Delta)=N/(2T)$ = frequenza di Nyquist

Un'altra proprietà fondamentale è la simmetria. Se la sequenza è reale, la DFT ha modulo con simmetria pari intorno alla frequenza di Nyquist:

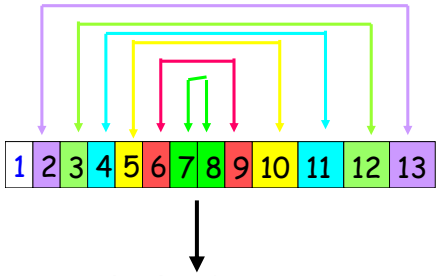
$real(F_k)=real(F_{N-k})$ k=1,..,N/2
$imag(F_k)=-imag(F_{N-k})$ k=1,..,N/2

Ossia F_{N-k} sono i complessi coniugati di F_k , non danno informazioni aggiuntive e spesso non è necessario visualizzarli. Osserviamo che non c'è simmetria per F_0. Se individuiamo con k=0 la frequenza di Nyquist si ha $F_k=F^*_{-k}$ (complesso coniugato). Negli schemi seguenti si mettono in evidenza i campioni che godono della simmetria nel caso di N pari e dispari (gli indici sono shiftati ad 1 poiché in Matlab i vettori cominciano da 1, la DC component ha indice 1,quella di Nyquist N/2+1).



N=14 pari

Nyquist (è al campione 14/2+1=8)



N=13 dispari

Nyquist (13/2+1=7.5 non esiste)

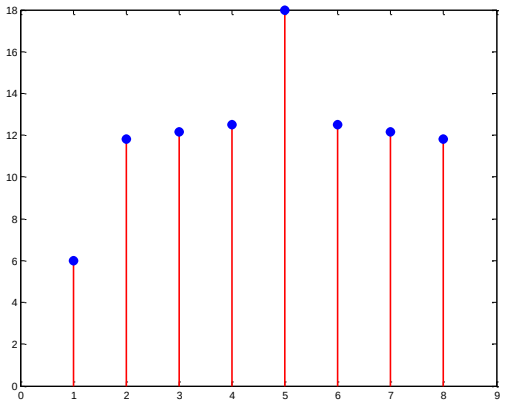
Verifichiamo le proprietà esposte con alcuni esempi in Matlab:

```
>> f=[4 3 7 -9 1 0 0 0]';      %N=8
>> y=fft(f)        %  function Matlab che calcola la DFT
y =
   6.0000                    DC component
  11.4853 - 2.7574i
  -2.0000 -12.0000i
  -5.4853 +11.2426i
  18.0000                   Nyquist
  -5.4853 -11.2426i
  -2.0000 +12.0000i
  11.4853 + 2.7574i
```

La prima componente è la DC component e la quinta è quella corrispondente alla frequenza di Nyquist. Facciamo la IDFT e verifichiamo che si riottene il vettore di partenza:

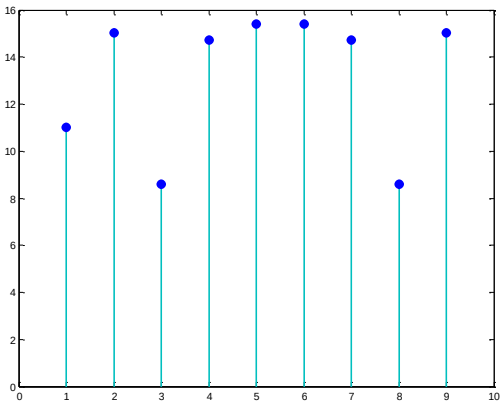
```
>> iy=ifft(y)              %  function Matlab che calcola la IDFT
iy =
    4.0000
    3.0000
    7.0000
   -9.0000
    1.0000
   -0.0000
         0
    0.0000
```

Poiché F_k è complesso si preferisce visualizzarne il **periodogramma**, che è il grafico dell'**ampiezza** (modulo)= $|F_k|$ o della **potenza** $P_k = |F_k|^2$ (o $|F_k|^2/N$) , k=0,..,N-1, plottate rispetto alle frequenze k/T in [0, 1/(2Δ)] (cicli/periodo). In molte applicazioni il termine per k=0 è omesso. Si definisce inoltre energia del fenomeno fisico la somma delle potenze.
Il grafico del modulo è:



Si osserva la simmetria intorno alla frequenza di Nyquist, tranne che per il primo punto.

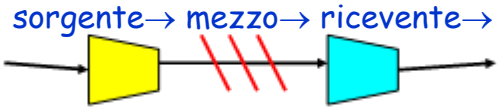
```
>> f=[4 3 7 -9 1 0 0 0 5]';    %N=9
>> y=fft(f)
y =
  11.0000
  14.9042 + 1.8441i
   4.0774 - 7.5760i
 -13.0000 + 6.9282i
   6.5184 +13.9626i
   6.5184 -13.9626i
 -13.0000 - 6.9282i
   4.0774 + 7.5760i
  14.9042 - 1.8441i
>> m=abs(y);
```



Si osserva la simmetria tranne che per il primo punto.

Filtraggio di un segnale (signal processing)

Uno dei problemi applicativi per il quale è fondamentale la DFT è il trattamento dei segnali (acustici, ottici, elettrici,etc.). Daremo qui un breve cenno di tale problema. Consideriamo un segnale acustico:



Uno degli obiettivi fondamentali consiste nel ridurre l'influenza di segnali estranei o del rumore (noise) e risalire al segnale originale : filtraggio .

Alcune fonti di segnali estranei o rumori sono:

- dispositivo di emissione o ricezione (radar, satellite, telefono, radio,…)
- mezzo di propagazione (aria,acqua,…)
- interferenze esterne (campo magnetico, elettrico,…)

Un segnale in genere rappresenta un fenomeno evolutivo in funzione del tempo di tipo ondulatorio :

onda = sovrapposizione di più onde elementari(armoniche)

Le armoniche elementari che costituiscono l'onda non si "vedono" nella sua espressione in funzione del tempo, ma si ottengono "scomponendo" l'onda con la trasformata di Fourier,che ne dà l'espressione in funzione di oscillazioni pure(seni e coseni).In tal modo si può effettuare il filtraggio di un segnale:individuare nel segnale eventuali armoniche spurie, eliminarle e ricostruire il segnale.

La prima fase del filtraggio consiste nel campionare il segnale (trasformarlo da continuo a discreto).

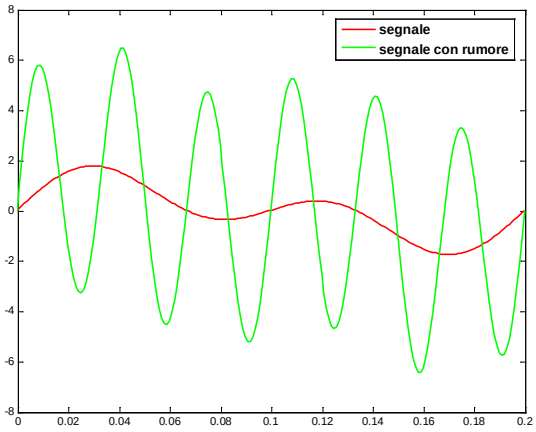
Dato il segnale continuo f(t) si ottengono i valori campionati f_j , j=0,…,N-1. Applicando la DFT al vettore così ottenuto si ha il vettore:F_k = ampiezze delle armoniche elementari .

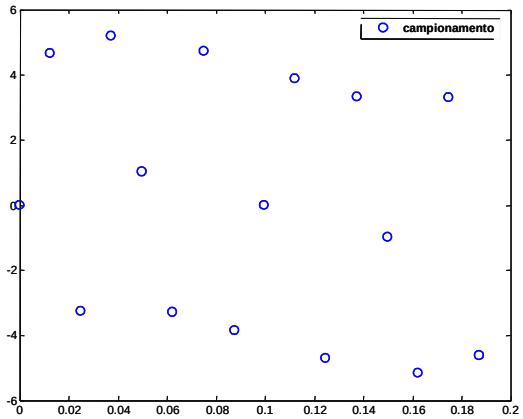
Ottenuto tale vettore il segnale originario si può esprimere, tramite la IDFT, come combinazione lineare delle armoniche che lo compongono:

$$f_j = \frac{1}{N}\sum_{k=0}^{N-1} F_k e^{\frac{2\pi i}{N}jk}$$ = segnale in funzione delle frequenze (discrete e distanti tra loro 1/T)

Il vettore F fornisce quindi le ampiezze delle frequenze che compongono il segnale ed è detto spettro del segnale.

Esempio. Supponiamo che il segnale ricevuto è il segnale originale f(t)=sin(2π5t)+sin(2π10t) a cui si è sovrapposto il ''rumore 5sin (2π30t). Sia Δ=0.0125, T=0.2, N=16, si ha :1/T=5, 1/ Δ=80, 1/(2Δ)=40.





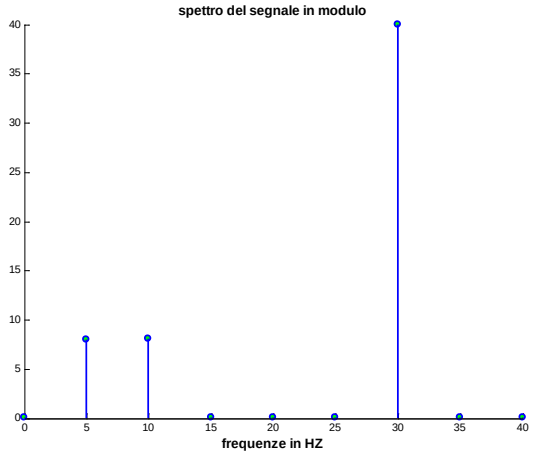
Applichiamo la DFT:

fy =

3.472833387713614e-014
2.975657543793466e-014 +7.999999999999966e+000i
-9.967995541639870e-015 +8.000000000000004e+000i
2.479866865842304e-014 -2.031708135064037e-014i
4.530267607331881e-015 -1.776356839400251e-014i
1.858141972052217e-014 -6.472600233564663e-014i
-8.533243355846108e-014 +4.000000000000000e+001i
1.487958690795757e-014 +8.437694987151190e-015i
-2.922051234127287e-014
1.487958690795757e-014 -8.437694987151190e-015i
-8.533243355846108e-014 -4.000000000000000e+001i
1.858141972052217e-014 +6.472600233564663e-014i
4.530267607331881e-015 +1.776356839400251e-014i
2.479866865842304e-014 +2.031708135064037e-014i
-9.967995541639870e-015 -8.000000000000004e+000i
2.975657543793466e-014 -7.999999999999966e+000i

Applicando la DFT si ottengono i valori delle ampiezze delle armoniche relative alle frequenze 0,5,10,15,…40. Ricordiamo che ,se il segnale è reale, la DFT ha modulo con simmetria pari intorno alla frequenza di Nyquist:$F_k=F_{-k}$, k=0=Nyquist. Di seguito vi è il programma matlab che produce il grafico del modulo (la prima metà per la simmetria).

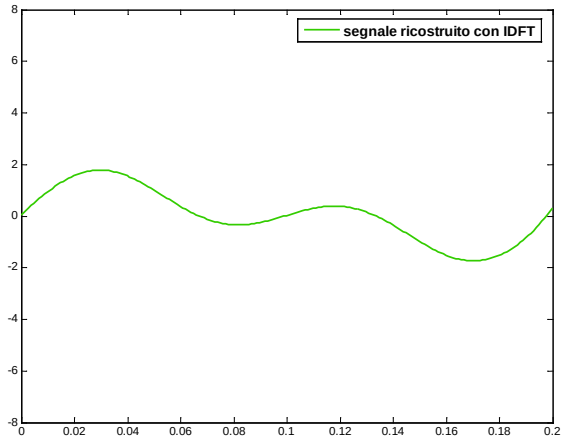
```
t=0:0.0125:0.2-0.0125;n=length(t);
y=f(t) ;                    %  function Matlab che rappresenta il segnale
dt=0.0125;
FS=1/dt;
freq=(0:n/2)*FS/n;
fy=fft(y,n);
stem(freq,abs(fy(1:n/2+1)))
```



A questo punto, come avviene nella realtà,l'individuazione del rumore dipende dalle informazioni sul segnale e dagli scopi del filtraggio. Supponiamo di sapere che il segnale originale non può avere frequenze superiori a 10 Hz. Eliminiamo quindi le armoniche con frequenza > 10 (ricordando la proprietà di simmetria) , si ottiene:

```
 3.472833387713614e-014
 2.975657543793466e-014 +7.999999999999966e+000i
-9.967995541639870e-015 +8.000000000000004e+000i
                       0
                       0
                       0
                       0
                       0
                       0
                       0
                       0
                       0
                       0
                       0
-9.967995541639870e-015 -8.000000000000004e+000i
 2.975657543793466e-014 -7.999999999999966e+000i
```

Applichiamo al vettore così ottenuto la IDFT ed otteniamo il segnale originale(si può effettuare un'interpolazione , ad esempio con spline cubiche).



Ovviamente i segnali con i quali si ha a che fare nella pratica sono molto più complessi, per cui quello che si ottiene è una buona approssimazione del segnale originale. In definitiva il filtraggio di un segnale consiste nella sintesi di Fourier:

Scomposizione(DFT)→eliminazione frequenze spurie→ricomposizione(IDFT)

Il processo computazionale effettuato nell'esempio precedente è un esempio di filtro digitale. In generale un filtro digitale è un algoritmo di tipo numerico che trasforma un segnale campionato(o una sequenza finita di valori) in un'altra:

segnale di input→ FILTRO→segnale di output

Molti filtri digitali sono basati sul calcolo di una o più DFT.
Alcuni esempi di campi di applicazione di filtri digitali basati su DFT sono:

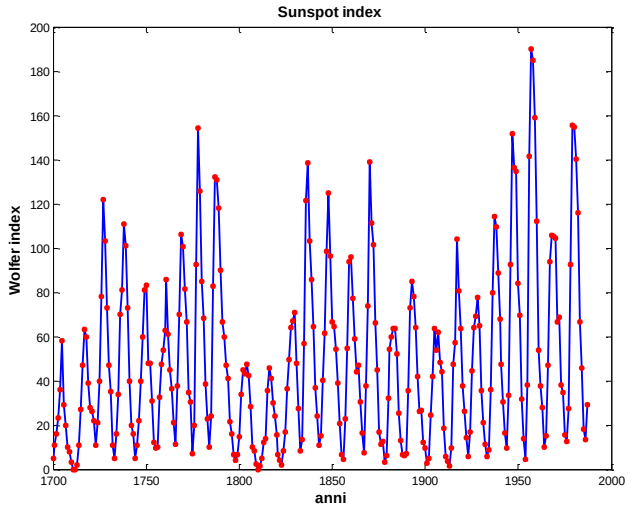
- ricostruzione di segnali provenienti da satelliti, da una stella, da un radar,…
- tecniche di medicina diagnostica quali tomografia assiale computerizzata(TAC), risonanza magnetica computerizzata(NRM)
- riconoscimento di forme
- compressione di immagini
- acustica

Analisi di dati che presentano caratteristiche di periodicità

Un esempio è lo studio del fenomeno delle macchie solari (*sunspot*). La faccia del sole non è uniforme ma vi appaiono alcune zone più scure in posizioni casuali in maniera ciclica. Questa attività è correlata al clima e ad altri fenomeni terrestri significativi anche da un punto di vista economico. Nel 1848 Wolfer propose una formula che accoppia il numero e l'ampiezza delle macchie in un singolo indice, cosicché si è determinata l'attività solare dal 1700. Nel Matlab demos c'è il file sunspot.dat costituito da due colonne di numeri: la prima contiene gli anni dal 1700 al 1987 e la seconda la media dell'indice di Wolfer per anno:

```
load sunspot.dat          %carica i dati
 t = sunspot(:,1)';
 wolfer = sunspot(:,2)';
 n = length(wolfer);
```

Il grafico del fenomeno è:



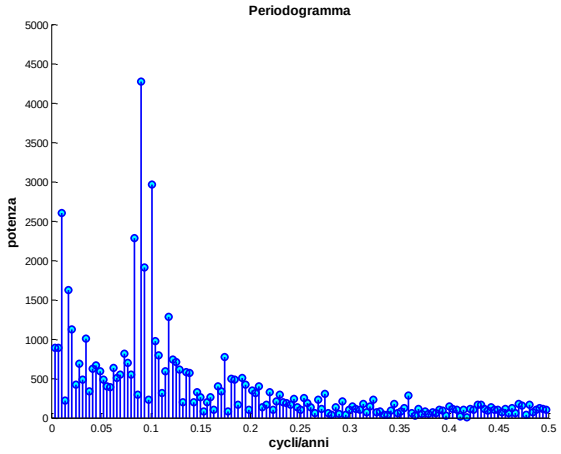
E' importante studiare in maniera precisa le caratteristiche di periodicità del fenomeno. Calcoliamo la DFT:

```
Y = fft(wolfer);
```

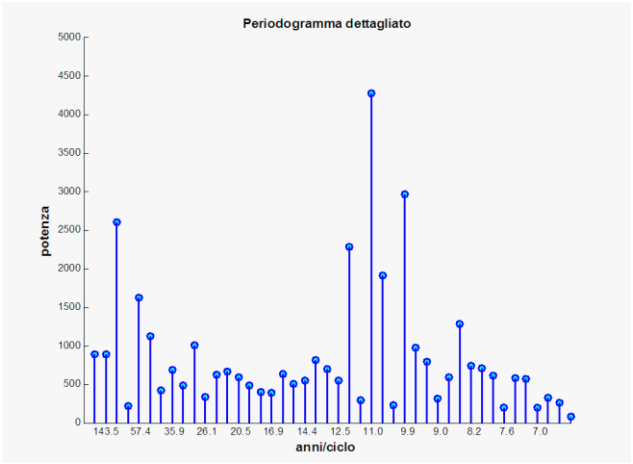
Eliminiamo la DC component, che è semplicemente la somma dei dati (si analizza il segnale che rappresenta le oscillazioni intorno al valor medio):

```
Y(1)=[];
n=length(Y);
```

Calcoliamo lo spettro tramite la DFT, ricordando che Fs=1 e che le frequenze vanno calcolate fino a quella di Nyquist, e visualizziamo il periodogramma (del modulo piuttosto che della potenza per una questione di scala del grafico).



E' evidente che alcune componenti sono più importanti di altre ed alcune frequenze caratterizzano meglio il fenomeno. La potenza massima si ha per la frequenza di circa 0.09 ed è circa 427, e si vuole il periodo corrispondente, che è il reciproco della frequenza, ed è 11.03 anni. Per fare ciò si può anche effettuare un plot ristretto intorno a tale frequenza usando come label dell'asse x al posto delle frequenze il periodo corrispondente (reciproco della frequenza):



Come ci aspettavamo il periodo dominante è di circa 11 anni , ossia questo è il ciclo in cui il fenomeno ha la massima potenza.

Dual Tone Multi-Frequency System (DTMF)

Il DTMF è un sistema di codifica usato per codificare codici numerici sotto forma di segnali sonori (telefonia, sistemi di integrazione computer-telefono, codici di carte di credito,…). Il telefono a tastiera (touch-tone) è un esempio dell'uso quotidiano della DFT, usata per la codifica e decodifica di segnali che contengono le informazioni sul numero di telefono digitato. Quando si digita un numero sulla tastiera di un telefono, ad ogni tasto premuto è trasmesso un segnale sonoro (tono) della durata di 50msec., con una frequenza Fs=8000Hz, somma di una coppia di sinusoidi, alla compagnia telefonica la quale, identificando la coppia di frequenze che compongono il segnale, riconosce le cifre che compongono il numero.

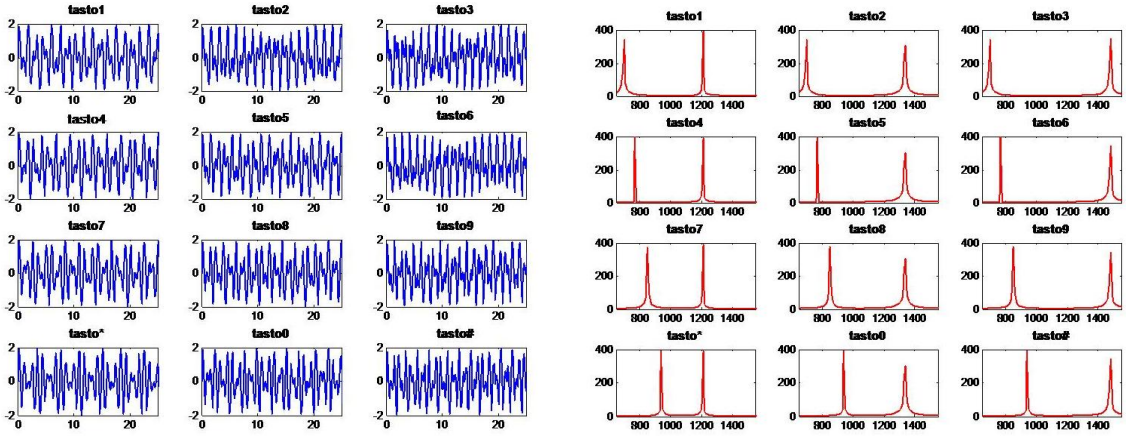
Il suono prodotto dal telefono quando è premuto un tasto è quindi un codice da decodificare per smistare la telefonata. Tramite la DFT sono individuate le frequenze del segnale e quindi il numero a cui sono univocamente associate.
La tastiera del telefono si può vedere come una matrice 4×3, ad ogni riga e colonna è associata una frequenza.

Hz	1209	1336	1477	1633
697	1	2	3	A
770	4	5	6	B
852	7	8	9	C
941	*	0	#	D

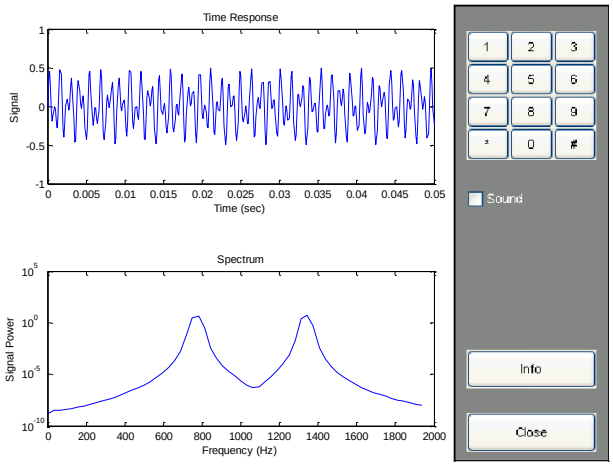
Premendo il tasto 4 è inviato un tono somma di due sinusoidi di frequenza 770 e 1209Hz.

$$\text{tasto 4} = \sin(2\pi 770t)+\sin(2\pi 1209t)$$

Nelle figure seguenti vi sono i grafici dei toni relativi ai tasti nel tempo ed in frequenza. Nel grafico in frequenza si distinguono perfettamente i diversi tasti.



La function Matlab *phone* è un esempio di come funziona un telefono a tastiera. L'output è visualizzato nella figura seguente. Cliccando su un numero del tastierino si visualizza il grafico del tono nel dominio del tempo e in quello delle frequenze, e si può ascoltare il suono corrispondente.



Il riconoscimento del numero si può ottenere applicando la DFT al tono, ed individuando le due frequenze corrispondenti ai picchi di massima potenza, si individua poi, tra le frequenze assegnate ai tasti, la coppia più"vicina" a quelle trovate.

Il suono in Matlab

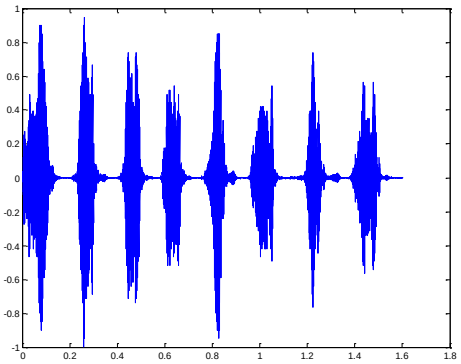
Ogni sorgente di segnale sonoro produce suoni in termini di fluttuazioni di pressione nel mezzo di trasmissione(aria,acqua,..).Il suono quindi consiste in vibrazioni che si propagano attraverso un mezzo e percepibili dall'orecchio umano (tra 20 e 20000Hz).
L'orecchio distingue un suono da un altro in base a tre caratteristiche:
l'altezza, individuata dalla frequenza : maggiore è la frequenza della vibrazione (ossia minore il tempo impiegato a compiere un'oscillazione) più acuto è il suono, viceversa ad una frequenza bassa corrispondono suoni bassi.
L'Intensità, che dipende dall'ampiezza delle frequenze, ed è il volume del suono (misurato in decibel=20log10(amp)).
Il Timbro, che dipende dal numero di armoniche presenti, e permette di distinguere un suono dall'altro.
La qualità del suono è determinata dal recordering rate (frequenza di registrazione),ossia da come rapidamente si registra il suono. Ad esempio per un CD la frequenza di campionamento è 44,1kHz, mentre per il telefono è limitata a 4kHz.
Un altro parametro che caratterizza un suono è la risoluzione dei dati registrati, ossia il numero di bit per campione,che non ha un effetto rimarchevole all'orecchio. Usualmente è 8bit(-128 a127) o 16bit (-32768 a 32767). In teoria 8bit è usato per la voce, ma dà buoni risultati anche per la musica, 16 bit riproduce musica ad alta qualità(CD).

Esempio. In questo esempio utilizziamo un file audio Matlab (help audiovideo dà un elenco dei file audio disponibili). Carico il file:

```
>> load chirp
>> whos
  Name      Size                Bytes  Class
  Fs        1x1                     8  double  array
  y     13129x1                105032  double  array
```

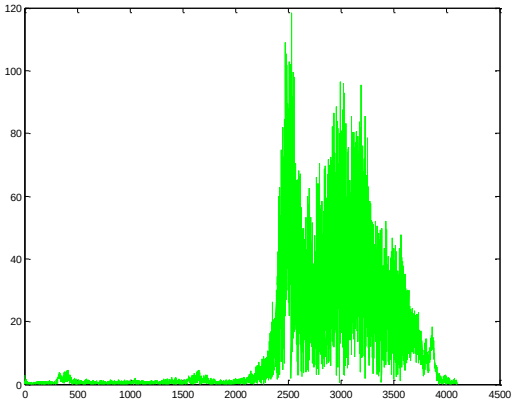
Il file contiene un vettore y (segnale campionato) e la relativa frequenza di campionamento.

```
y1=y;
>> n1=length(y1)
n1 =
      13129
>> t=0:1/Fs:(n1-1)/Fs;
>> plot(t,y1)
```



Effettuiamo l'analisi dello spettro del segnale facendo il grafico dell'ampiezza (la metà per simmetria intorno alla frequenza di Nyquist) rispetto alle frequenze:

```
>> Y=fft(y1);
>>amp=abs(Y(1:floor(n1/2)+1));
>>f=(0:n1/2)*Fs/n1;
plot(f,amp)
```



Il comando *soundsc(y,Fs)* converte un vettore in suono:

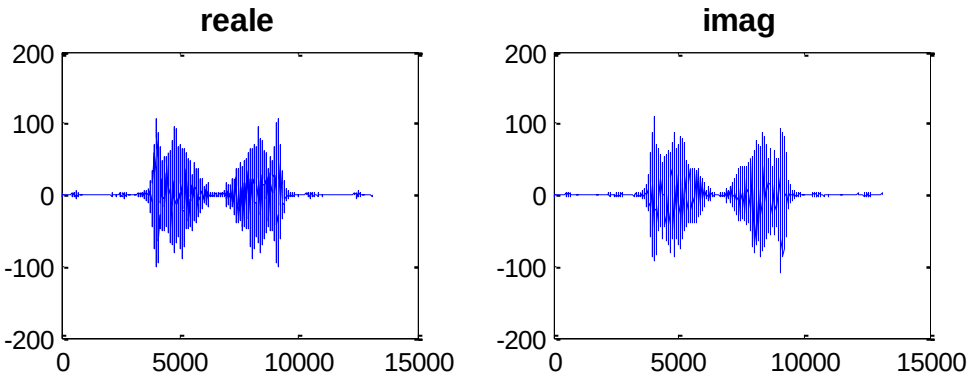
```
>> soundsc(y,Fs)
```

Calcoliamo la IDFT:

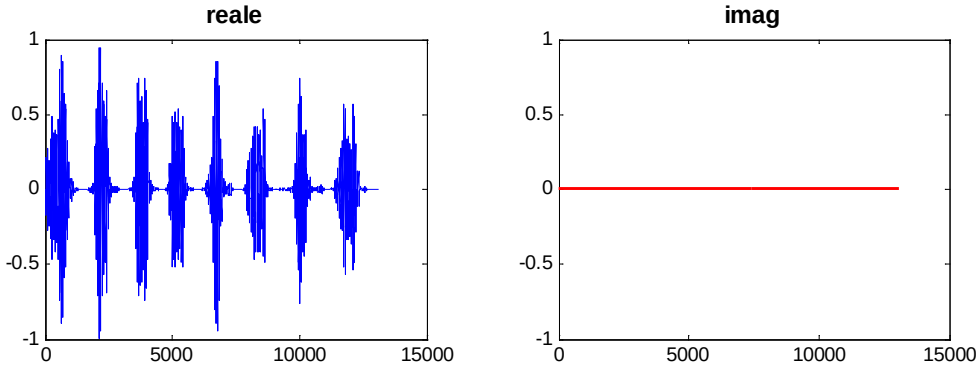
```
>>iiy=ifft(Y);
>> whos
  Name      Size                Bytes  Class
  Fs        1x1                     8  double array
  iiy   13129x1                105032  double array
  Y     13129x1                210064  double array (complex)
  y1    13129x1                105032  double array
```

Osserviamo che Y(DFT) è un vettore complesso, mentre iiy, come è da aspettarsi, è un vettore reale:

```
>> subplot(2,2,1),plot(real(Y))
>> subplot(2,2,2),plot(imag(Y))
```




```
>> subplot(2,2,1),plot(real(iy))
>> subplot(2,2,2),plot(imag(iy))
```



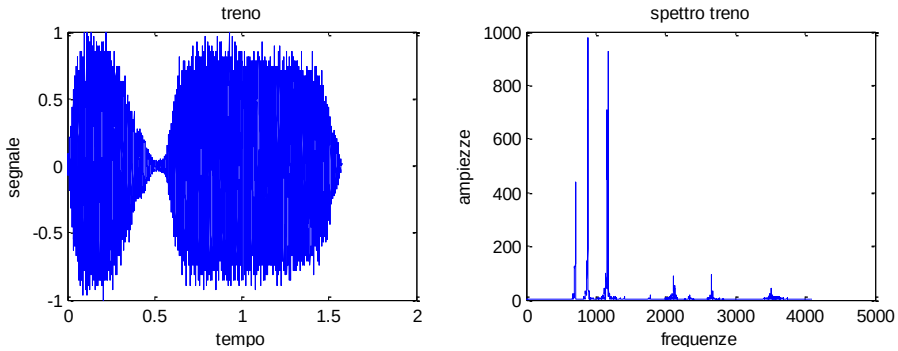
Digitando:

```
>> soundsc(iiy)
```

si risente il suono iniziale.

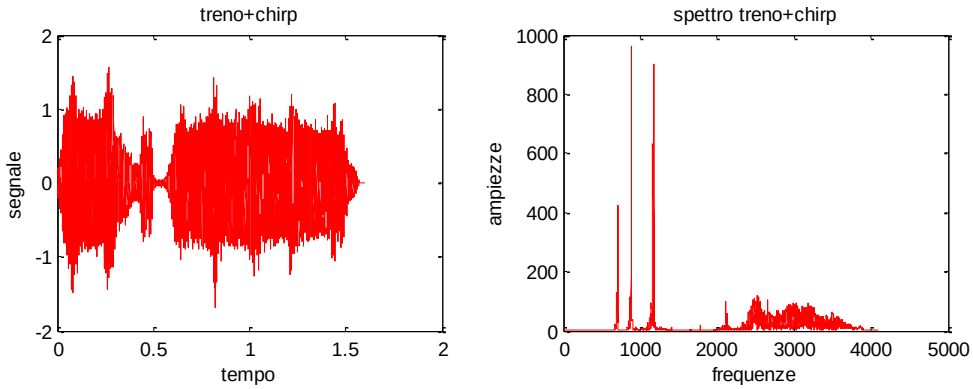
Esempio. Consideriamo il suono chirp usato precedentemente e prendiamo dalla libreria Matlab un altro suono, train, abbastanza separato in frequenza:

```
>>load train
>>n2=length(y);
 n2 =
      12880
>>y2=y;
>>YT=fft(y2);
>>amptreno=abs(YT(1:floor(n2/2)+1));   %ampiezze
>>ftreno=(0:n2/2)*Fs/n2;    %frequenze
```



Sovrapponiamo i suoni ottenendo un nuovo segnale:

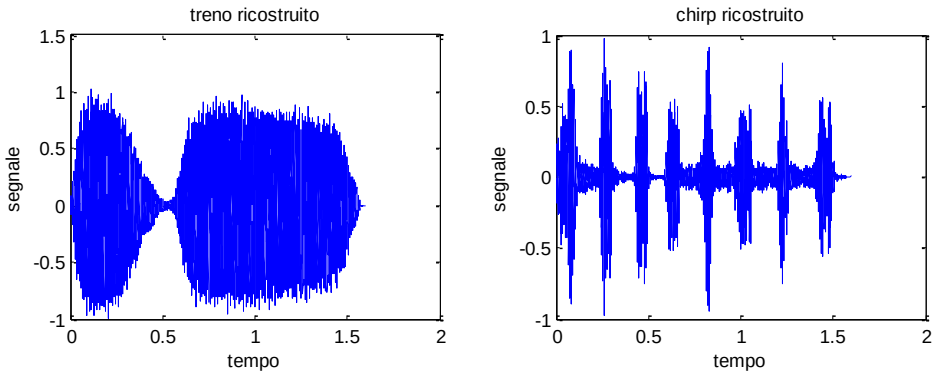
```
>>d=n1-n2;          %uguaglio la lunghezza dei campioni
>>y2=[y2;zeros(d,1)];
>>y=y1+y2;
>>n=length(y);
>>Y=fft(y);
>>amp=abs(Y(1:floor(n/2)+1));
>>f=(0:n/2)*Fs/n;        %frequenze
```




```
>>soundsc(y)  % ascolto il suono ottenuto
```

I due suoni che compongono il segnale sono indistinguibili nel tempo, mentre nel dominio delle frequenze risultano separati, circa intorno a 2200Hz. Separiamo i suoni e ricostruiamoli:

```
>>suono1=Y;
 >>ind=find(f>=2200); %determino gli indici delle frequenze>2200
% annullo i corrispondenti valori
 >>suono1(ind)=0;
 >>suono1(n+2-ind)=0;  %simmetria intorno alla frequenza di Nyquist
 >>treno=ifft(suono1); %ricostruisco
>>suono2=Y;
 >>suono2(1)=0;
 >>indc=find(f<2200);  %determino gli indici delle frequenze<2200
% annullo i corrispondenti valori
>> indc(1)=[];
 >>suono2(indc)=0;
 >>suono2(n+2-indc)=0; %la simmetria vale tranne che per il primo punto
 >>cip=ifft(suono2);  %ricostruisco
```



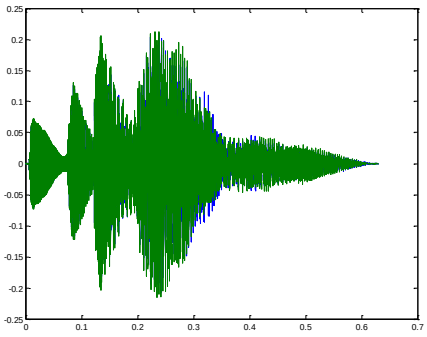
%ascolto i due suoni separati
>>soundsc(real(treno))
>>pause
>> soundsc(real(cip))

Importare ed esportare file audio(WAVE)

In Matlab è possibile importare ed esportare file audio nel formato .wav per elaborare suoni o altro. Inoltre è possibile registrare i suoni e scrivere i dati in un file .wav da un'unità di input audio di un PC. La function che legge un file.wav è:

[y,Fs,bits]= wavread('nomefile.wav')

y = campioni contenuti nel file
Fs = frequenza di campionamento
bits = numero di bit per campione usato per codificare i dati
Molti computer hanno file WAVE che riproducono semplici suoni, ad esempio, per caricare ed eseguire il file chimes.wav, che si trova in C:\Windows\Media :
>> [s,fs]=wavread('C:\windows\media\chimes.wav');
>> soundsc((s,fs)
Si può effettuare il grafico del segnale:
>> n=length(s);
>> dT=1/fs;
>> t=(0:n-1)*dT;
>> plot(t,s)



Se ad esempio si vuole creare un file sonoro che ripete tre volte il suono caricato:
>> suono=[s; s; s];
>> soundsc(suono,fs)
Se poi si vuole esportare un file in formato .wav si usa il comando:

wavwrite(y,Fs,nomefile.wav')

che esporta un file di campioni tagliando i valori non compresi tra -1 ed 1.
>> wavwrite(song,fs,'song.wav')
Il file si trova nella directory Matlab usata e si può copiare o riprodurre con un qualunque applicativo che supporta il formato .wav.
Usando un microfono si converte l'onda sonora in un segnale elettrico che può essere analizzato dal computer.
Se la scheda audio lo permette si possono registrare i suoni da un'unità di input audio di un PC con la function:

song=wavrecord(n,Fs)

n= numero di campioni
Fs = frequenza di campionamento, di default Fs=11025 Hz
Se si vuole registrare la propria voce per 5 secondi dal canale 1, si parla al microfono del computer mentre si esegue il programma:
Fs=11025;
voce=wavrecord(5*Fs,Fs)
Per ascoltare il file: wavplay(y,Fs).

Musica in Matlab

La musica è un suono particolare costituito da una successione di note, cioè di piccoli suoni dalle oscillazioni regolari. Un *rumore* è una vibrazione non periodica, ma caotica, il cui grafico ha una forma *brutta*, priva di regolarità.
Una nota è un'oscillazione che può essere rappresentata da :

$K\sin(2\pi ft)$, $K\cos(2\pi ft)$

e ciò che identifica una nota è la frequenza dell'armonica.
Vi sono 120 frequenze udibili dall'orecchio umano,da f_{-57} a f_{62}, divise in 10 ottave, ciascuna di 12 elementi, detti semitoni.
Ossia vi sono 12 note per ottava, in relazione matematica tra loro e calcolabili a partire da una nota fondamentale di frequenza stabilita, il La4 (440Hz), che è la nota centrale della tastiera di un pianoforte. Ogni nota è separata dal La4 da un numero finito di semitoni, ogni 12 semitoni si ha un raddoppio di frequenza, per cui si ha la relazione che lega tra loro le note: la frequenza di una nota che dista n semitoni dalla fondamentale è

$$freq=440x2^{n/12}Hz$$

Due note di frequenza doppia(intervallo 1 ottava) rispetto all'altra sembrano molto simili e sono chiamate con lo stesso nome.

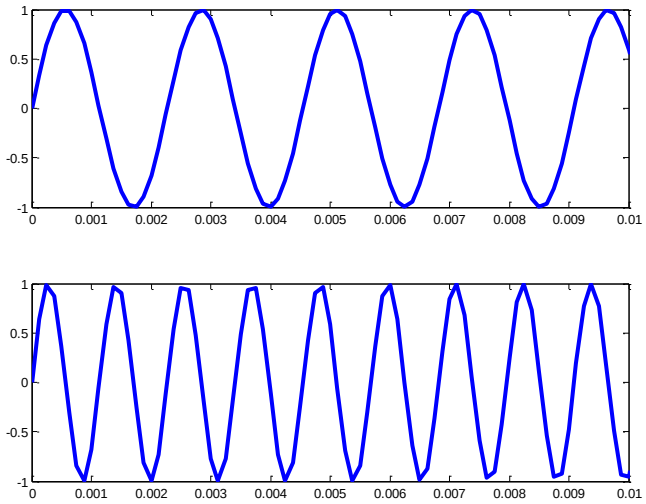
Ottave	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
do	16.35	32.7	65.41	130.81	261.63	523.25	1046.5	2093
$do^{\#}/re^{b}$	17.32	34.65	69.3	138.59	277.18	554.37	1108.73	2217.46
re	18.35	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32
$re^{\#}/mi^{b}$	19.45	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02
mi	20.6	41.2	82.41	164.81	329.63	659.26	1318.51	2637.02
fa	21.83	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83
$fa^{\#}/sol^{b}$	23.12	46.25	92.5	185	369.99	739.99	1479.98	2959.96
sol	24.5	49	98	196	392	783.99	1567.98	3135.96
$sol^{\#}/la^{b}$	25.96	51.91	103.83	207.65	415.3	830.61	1661.22	3322.44
la	27.5	55	110	220	440	880	1760	3520
$la^{\#}/si^{b}$	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31
si	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07

Ascoltiamo una sequenza di note per 2 sec.:

```
>>t=linspace(0,2,16000);
>> sol=sin(2*pi*392*t);
>> si=sin(2*pi*493.88*t);
>> la=sin(2*pi*440*t);
>> soundsc([si; la; sol; la; si; si; si],8000)
```

Nello script seguente è generato il grafico del La4 ed il La4 ad un'ottava seguente:

```
%test musica
%La4 e La4  ad un'ottava seguente
t=linspace(0,2,16000);
sound1=sin(2*pi*440*t);
sound2=sin(2*pi*880*t);
%grafico del primo 1/100 di secondo
subplot(2,1,1);plot(t,sound1);axis([0 .01 -1 1])
subplot(2,1,2);plot(t,sound2);axis([0 .01 -1 1])
```

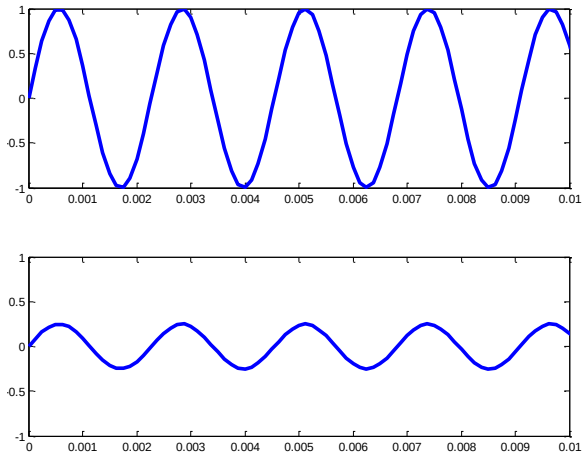


Ascoltiamo le due note:

```
>> soundsc(sound1)
>> soundsc(sound2)
```

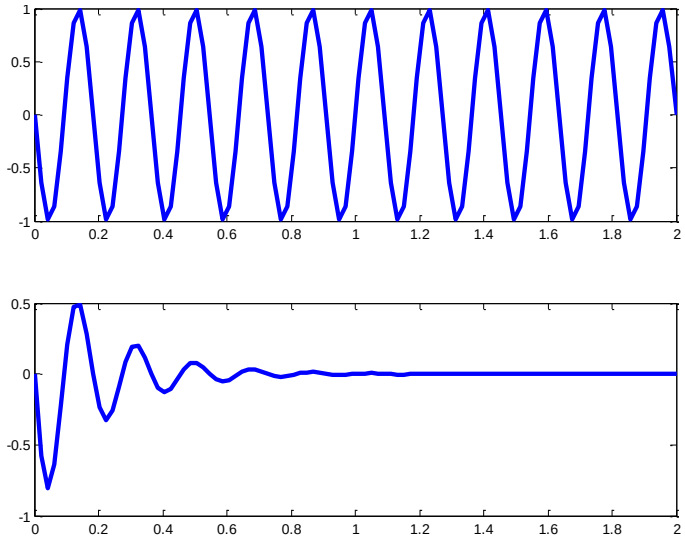
Generiamo il La4 ad un volume più basso:

```
r=0.25*sound1;
sound(r,8000)
%grafico del primo 1/100 di secondo
subplot(2,1,1);plot(t,sound1);axis([0 .01 -1 1])
subplot(2,1,2);plot(t,r);axis([0 .01 -1 1])
```



Il suono di uno strumento musicale non va direttamente alla massima intensità e si ferma improvvisamente . C'è un tempo di attacco ed uno di decadimento . Si simula la variazione di intensità (volume) moltiplicando l'ampiezza per un fattore di scala a(t).

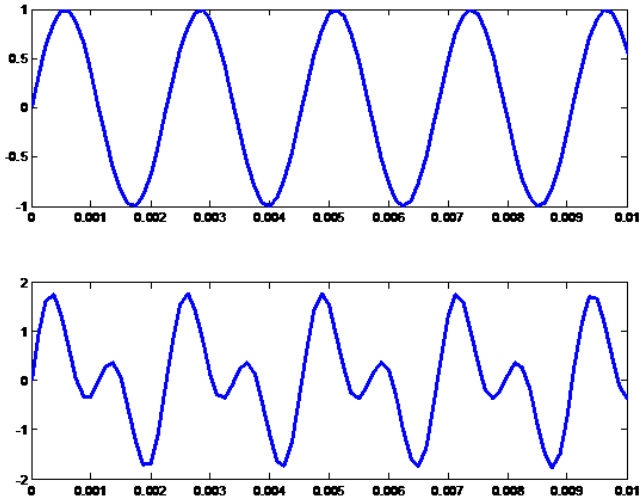
```
%semplice fattore di scala
>> s=exp(-5*t).*sound1;
>> sound(s,8000)
>> soundsc(s,8000)
>> tt=linspace(0,2);
>> subplot(2,1,1);plot(tt,sin(2*pi*440*tt))
>> subplot(2,1,2);plot(tt,exp(-5*tt).*sin(2*pi*440*tt))
```



Se un piano ed un flauto suonano entrambi il La4 hanno suoni diversi. Questo accade perché il suono generato non è un tono puro, che avrebbe un suono metallico, ma è generato anche da armoniche con frequenze multiple di 440, ossia è generata la stessa nota a differenti ottave. Nel linguaggio musicale 440 è detta la nota fondamentale, 880 la prima armonica, 1320 la seconda e così via. La differenza tra gli strumenti è relativa alle ampiezze delle armoniche usate ed è il timbro dello strumento. Il piano ad esempio ha la fondamentale più alta delle altre, mentre il flauto ha la prima armonica più alta della nota

fondamentale. La sintesi musicale imita gli strumenti sommando armoniche a diverse ampiezze.
Nello script seguente sono generate il La4 ed il La4 ad un'ottava seguente e poi sommate:

```
t=linspace(0,2,16000);
sound1=sin(2*pi*440*t);
sound2=sin(2*pi*880*t);
%sommiamo
>> sounds=sound1+sound2;
>> soundsc(sounds,8000)
subplot(2,1,1);plot(t,sound1)
subplot(2,1,2);plot(t,sounds,)
```



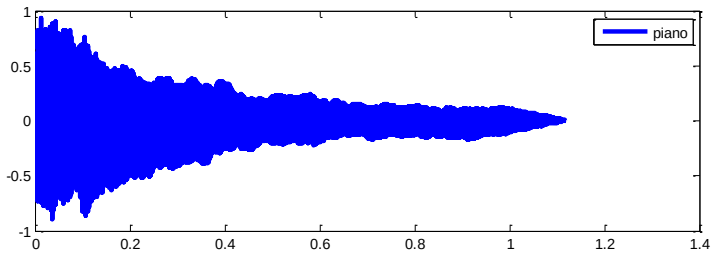
Nello script seguente sono caricati da un file .wav il La4 suonato da un piano e da un flauto:

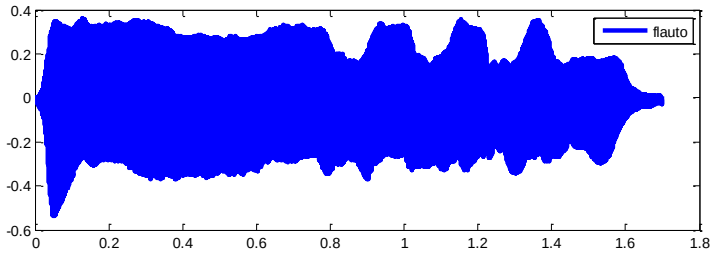
```
%test piano e flauto
[piano,Fspiano]=wavread('piano');
[flauto,Fsflauto]=wavread('flute');
% ascoltiamo le due note
soundsc(piano)
soundsc(flauto)
%campionamento nel tempo
np=length(piano);
tp=(0:np-1)/Fspiano;
nf=length(flauto);
tf=(0:nf-1)/Fsflauto;
subplot(2,1,1);plot(tp,piano);
subplot(2,1,2);plot(tf,flauto);
legend('flauto');
```

Visualizziamo la lunghezza dei campioni, la frequenza Fs e la durata dei due suoni:

```
>> np
np =
      24646
>> nf
```

```
nf =
      37485
>> Fspiano
Fspiano =
      22050
>> Fsflauto
Fsflauto =
      22050
>> tempop=np/Fspiano
tempop =
   1.1177
>> tempof=nf/Fsflauto
tempof =
   1.7000
```

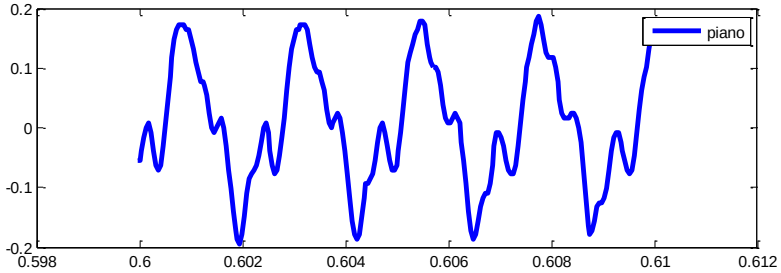


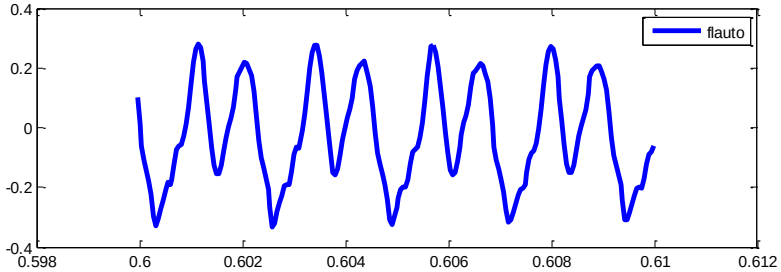


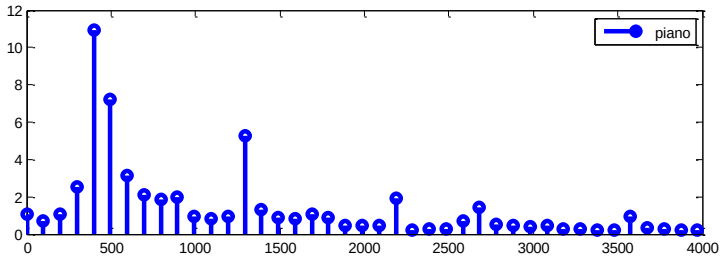
Facciamo il grafico in un intervallino più piccolo, ad esempio tra t=.6 a t=.61 sec. Poiché per t=0 a t=1 i campioni vanno da k=1 a k=Fs la relazione è lineare cioè k=(Fs-1)t+1 si ha
Facciamo anche il grafico dello spettro:

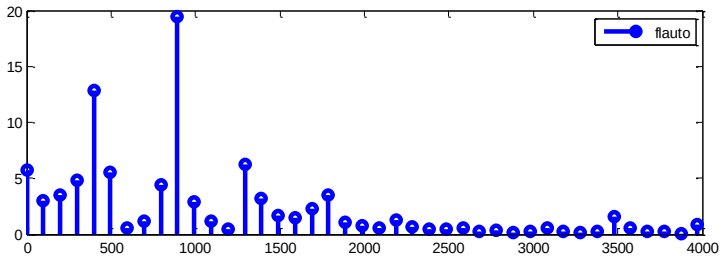
```
startp=round((Fspiano-1)*.6+1);
endp=round((Fspiano-1)*.61+1);
samplet=tp(startp:endp);
samplep=piano(startp:endp);
samplef=flauto(startp:endp);
subplot(2,1,1);plot(samplet,samplep);
subplot(2,1,2);plot(samplet,samplef);
N=length(samplep);
yp=abs(fft(samplep));
yf=abs(fft(samplef));
ampp=yp(1:floor(N/2)+1);
ampf=yf(1:floor(N/2)+1);
fp=(0:floor(N/2))*Fspiano/N;
ff=(0:floor(N/2))*Fsflauto/N;
```

subplot(2,1,1);stem(fp,ampp);set(gca,'Xlim',[0 4000])
subplot(2,1,2);stem(ff,ampf);set(gca,'Xlim',[0 4000])









%suoniamo insieme piano e flauto
>> d=nf-np;
>> piano1=[piano;zeros(d,1)];
>> mus=piano1+flauto;
>> soundsc(mus,Fsflauto)
%scaliamo i dati per evitare che vengano tagliati e memorizziamoli in un file .wav
>>y=0.99*mus;
>> wavwrite(y,Fspiano,'pianoflauto.wav');

Algoritmo FAST FOURIER TRANSFORM (FFT)

L'algoritmo FFT(Fast Fourier Transform) si può considerare il più importante algoritmo della matematica computazionale per il suo forte impatto nei settori applicativi più disparati, per l'impulso dato alla tecnologia e per gli ulteriori sviluppi applicativi che ha ancora oggi .
L'algoritmo FFT è un algoritmo per il calcolo veloce della DFT (Trasformata Discreta di Fourier).
Complessità computazionale della valutazione diretta della DFT *(o IDFT)*

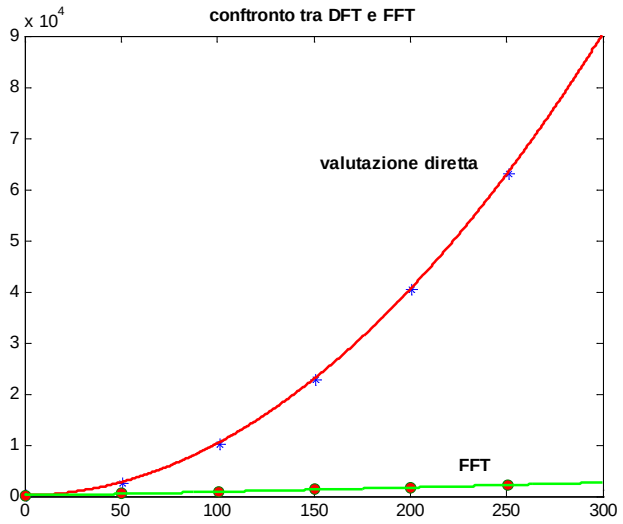
Il calcolo di: $F_k = \sum_{j=0}^{N-1} f_j w_N^{-jk}$, k=0,…N-1 richiede, per ciascun elemento del vettore:

- N moltiplicazioni complesse
- N-1 addizioni complesse
- La valutazione di N esponenziali complessi.

Supponendo noti gli esponenziali complessi, la complessità computazionale, in termini di moltiplicazioni complesse, è:$T_{DFT}(N)=O(N^2)$ (lo stesso valore si ottiene per la IDFT).
Nelle applicazioni è richiesto, in genere, il calcolo in *tempo reale* di DFT (IDFT) di vettori di lunghezza N con N *molto grande* (N>>1000) oppure il calcolo di m DFT di vettori di lunghezza N (m≈N); ad esempio per la ricostruzione di un'immagine di una NMR il vettore da trasformare può avere ordine di grandezza di circa 5 milioni. Per tale motivo la valutazione diretta della DFT, anche su calcolatori con elevata velocità di calcolo, ha un tempo di calcolo proibitivo e ciò "limitava" l'uso della DFT e lo sviluppo delle possibili applicazioni tecnologiche ad essa connesse.
Nel 1960 alcuni studi sull'elio richiesero il calcolo di più DFT ed il matematico Tukey ebbe l'incarico di sviluppare un algoritmo efficiente, a partire dal quale Cooley (ingegnere dell'IBM) sviluppò un programma. Nel 1965 fu pubblicato il primo articolo sull'algoritmo FFT(Fast Fourier Trasform) il cui risultato fondamentale è la riduzione drastica della complessità di calcolo di una DFT. Si ha infatti che:

$$T_{FFT}(N)=O(N\log_2 N)$$

Un confronto tra la complessità della valutazione diretta e quella dell'algoritmo FFT fa capire l'importanza ,da un punto di vista applicativo, di tale algoritmo, che ha reso possibile l'evoluzione tecnologica legata alle molteplici applicazioni della DFT e rappresenta, ad esempio, la nascita della moderna elaborazione numerica dei segnali.



ALGORITMO FFT radix-2(Cooley-Tukey)

La strategia su cui si basano gli algoritmi FFT è quella di ricondurre il calcolo di una DFT di lunghezza N al calcolo di più DFT di lunghezza inferiore (algoritmi tipo *divide and conquer*). In particolare negli algoritmi FFT radix-2 il calcolo di una DFT di lunghezza $N=2^m$ è ricondotto al calcolo di più DFT di lunghezza 2, la DFT più semplice da calcolare.
Una DFT di lunghezza 2 infatti è:

$$F_k = \sum_{j=0}^{1} f_j w_2^{-jk} \quad , k=0,1$$

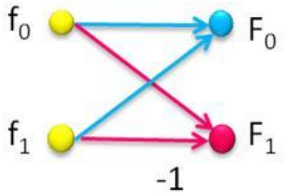
cioè , ricordando che $e^{-\pi i} = \cos\pi - i\sin\pi = -1$, si ha :

$$F_k = f_0 w_2^{0k} + f_1 w_2^{-1k} \quad k=0,1$$

Cioè:

$$F_0 = f_0 + f_1 \ , \ F_1 = f_0 - f_1$$

Una DFT di ordine 2 si descrive con uno schema a farfalla (butterfly) che rappresenta l'operazione fondamentale della FFT:



Il cerchio con frecce in ingresso indica che il valore corrispondente è la somma dei nodi da cui provengono le frecce, il numero vicino alla freccia è il peso del dato(se omesso vale 1).
Indicheremo d'ora in poi, per comodità di notazione, con w(jk/N) la quantità $w_N^{-jk} = e^{-2\pi ijk/N}$.
Si ha che tale quantità gode di alcune proprietà fondamentali per lo svolgimento dell'algoritmo:

1) ricordando che $w_N = e^{2\pi i/N}$. $=\cos(2\pi/N)+i\sin(2\pi/N)$ = radice N-ma dell'unità, si ha che

$$w_N^N = 1 \Rightarrow w_N^{Nr} = 1 \Rightarrow w(r) = w(rN/N) = 1 \quad \text{qualunque sia } r \varepsilon Z.$$

2) $w(r+s)=w(r)w(s)$
3) $w(jk/N)$ è periodica di periodo N cioè $w(jk/N)=w((j+N)k/N)=w(j(k+N)/N$
4) $w((x+N/2)/N)=w(x/N+1/2)=w(x/N)w(1/2)=w(x/N)(\cos\pi - i\sin\pi)=-w(x/N)$

Per comprendere il procedimento dell'algoritmo consideriamo prima un esempio con $N=2^3 = 8$, ossia bisogna calcolare:

$$F_k = \sum_{j=0}^{7} f_j w_8^{-jk} \quad , k=0,\ldots,7$$

L'idea di base dell'algoritmo è quella di rappresentare gli indici j e k in binario, ossia:

$$j = 2^2 \times p_0 + 2 \times p_1 + p_2 = (p_0, p_1, p_2)_2$$
$$k = 2^2 \times q_0 + 2 \times q_1 + q_2 = (q_0, q_1, q_2)_2 \qquad p_i, q_i = 0,1$$

ad esempio:

j	p_0	p_1	p_2
0	0	0	0
1	0	0	1
2	0	1	0
...	...	...	...

Ricordiamo che $w(jk/8)=w_8^{-jk}$. Si ha che:

$$F_k = \sum_{j=0}^{7} f_j w(jk/8) \Leftrightarrow F_k = F(q_0,q_1,q_2) = \sum_{p_2=0}^{1}\sum_{p_1=0}^{1}\sum_{p_0=0}^{1} f(p_0,p_1,p_2) w(\frac{kp_0 2^2 + kp_1 2 + kp_2}{8})$$

Si ha che:

$$w((kp_0 2^2 + kp_1 2 + kp_2)/8) = w(4kp_0/8)w(2kp_1/8)w(kp_2/8)$$

da cui:

$$F_k = \sum_{p_2=0}^{1} w(\frac{kp_2}{8}) \sum_{p_1=0}^{1} w(\frac{kp_1}{4}) \sum_{p_0=0}^{1} f(p_0,p_1,p_2) w(\frac{kp_0}{2})$$

L'algoritmo procede per passi, ad ogni passo calcola , per tutte le componenti di F,una somma di due addendi, a partire da quello più interno. In totale ad ogni passo calcola una somma di due addendi per ciascuna delle componenti del vettore F. Analizziamo l'algoritmo passo per passo.

Passo 1:

bisogna calcolare:

$$\sum_{p_0=0}^{1} f(p_0,p_1,p_2) w(\frac{kp_0}{2})$$

Ma:

$$w(kp_0/2) = w(p_0(2^2 q_0 + 2q_1 + q_2)/2) = w(2p_0q_0)w(p_0q_1)w(p_0q_2/2)$$

Ricordando la proprietà 1) di w si ha : $w(2p_0q_0)=w(p_0q_1)=1$, al primo passo si calcola il vettore:

$$c_1(q_2,p_1,p_2) = \sum_{p_0=0}^{1} f(p_0,p_1,p_2) w(\frac{p_0 q_2}{2}) = f(0,p_1,p_2)w(0) + f(1,p_1,p_2)w(q_2/2)$$

di 8 componenti che si ottengono al variare di $q_2,p_1,p_2=0,1$. Più in dettaglio, ricordando che, per la proprietà 4) $w(1/2)=-w(0)=-1$, si ha:

$c_1(0)=c_1(0,0,0)=f(0,0,0)+f(1,0,0)w(0)= f_0+f_4$
$c_1(1)=c_1(0,0,1)=f(0,0,1)+f(1,0,1)w(0)= f_1+f_5$
$c_1(2)=c_1(0,1,0)=f(0,1,0)+f(1,1,0)w(0)= f_2+f_6$
$c_1(3)=c_1(0,1,1)=f(0,1,1)+f(1,1,1)w(0)= f_3+f_7$
$c_1(4)=c_1(1,0,0)=f(0,0,0)+f(1,0,0)w(1/2)= f_0-f_4$
$c_1(5)=c_1(1,0,1)=f(0,0,1)+f(1,0,1)w(1/2)= f_1-f_5$
$c_1(6)=c_1(1,1,0)=f(0,1,0)+f(1,1,0)w(1/2)= f_2-f_6$
$c_1(7)=c_1(1,1,1)=f(0,1,1)+f(1,1,1)w(1/2)= f_3-f_7$

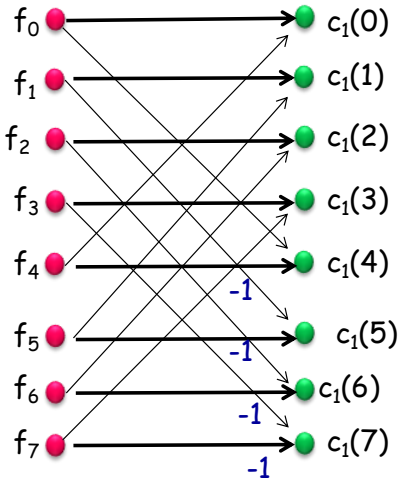
Osserviamo che:

$$c_1(0)=f_0+f_4 \text{ , } c_1(4)=f_0-f_4$$

Da cui *la coppia* $(c_1(0),c_1(4))$ *è la* DFT di ordine 2 *del vettore* (f(0),f(4)). In generale si ha:

$(c_1(0),c_1(4))$=DFT del vettore (f(0),f(4))
$(c_1(1),c_1(5))$ =DFT del vettore (f(1),f(5))
$(c_1(2),c_1(6))$ =DFT del vettore (f(2),f(6))
$(c_1(3),c_1(7))$ =DFT del vettore (f(3),f(7))

Si ha quindi che per ottenere il vettore c_1 è necessario calcolare *4 DFT di lunghezza 2*.



Dopo il primo passo , si ottiene:

$$F_k = F(q_0,q_1,q_2) = \sum_{p_2=0}^{1} w(\frac{kp_2}{8}) \sum_{p_1=0}^{1} w(\frac{kp_1}{4}) c_1(q_2,p_1,p_2)$$

Analogamente a come si è proceduto precedentemente:

$w(kp_1/4)= w(p_1(2^2q_0 +2\ q_1 +q_2)/4)=w(p_1q_0)w((2p_1q_1+ p_1q_2/4)$

Ricordando la proprietà 1) di w si ha : $w(p_1q_0)=1$, da cui:

$$F_k = F(q_0,q_1,q_2) = \sum_{p_2=0}^{1} w(\frac{kp_2}{8}) \sum_{p_1=0}^{1} w(\frac{2q_1p_1+q_2p_1}{4}) c_1(q_2,p_1,p_2)$$

Passo 2:

si calcola il vettore: $c_2(q_2,q_1,p_2) = \sum_{p_1=0}^{1} c_1(q_2,p_1,p_2) w(\frac{2q_1p_1+q_2p_1}{4}) =$

$= c_1(q_2,0,p_2)w(0)+c_1(q_2,1,p_2)w((2q_1+q_2)/4)$

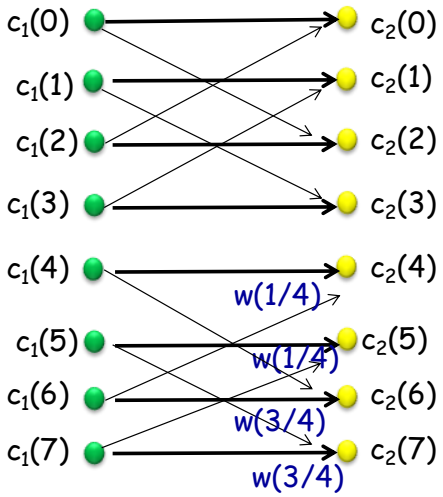
di 8 componenti che si ottengono al variare di $q_2,q_1,p_2 = 0,1$. Più in dettaglio:

$c_2(0)=c_2(0,0,0)=c_1(0,0,0)+c_1(0,1,0)w(0)=c_1(0)+c_1(2)$
$c_2(1)=c_2(0,0,1)=c_1(0,0,1)+c_1(0,1,1)w(0)=c_1(1)+c_1(3)$
$c_2(2)=c_2(0,1,0)=c_1(0,0,0)+c_1(0,1,0)w(1/2)=c_1(0)-c_1(2)$
$c_2(3)=c_2(0,1,1)=c_1(0,0,1)+c_1(0,1,1)w(1/2)=c_1(1)-c_1(3)$
$c_2(4)=c_2(1,0,0)=c_1(1,0,0)+c_1(1,1,0)w(1/4)=c_1(4)+c_1(6)w(1/4)$
$c_2(5)=c_2(1,0,1)=c_1(1,0,1)+c_1(1,1,1)w(1/4)=c_1(5)+c_1(7)w(1/4)$
$c_2(6)=c_2(1,1,0)=c_1(1,0,0)+c_1(1,1,0)w(3/4)=c_1(4)-c_1(6)w(1/4)$
$c_2(7)=c_2(1,1,1)=c_1(1,0,1)+c_1(1,1,1)w(3/4)=c_1(5)-c_1(7)w(1/4)$

Ricordiamo che, per la proprietà 4) $w(1/2)=-w(0)= -1$, $w(1/4)= e^{-2\pi i/4}=\cos\pi/2-i\,\mathrm{sen}\pi/2=-i$ $w(3/4)=-w(1/4)$. Analogamente al passo precedente si ha:

$(c_2(0),c_2(2))$ =DFT del vettore $(c_1(0),c_1(2))$
$(c_2(1),c_2(3))$ =DFT del vettore $(c_1(1),c_1(3))$
$(c_2(4),c_2(6))$ =DFT del vettore $(c_1(4),c_1(6))$
$(c_2(5),c_2(7))$ =DFT del vettore $(c_1(5),c_1(7))$

Si ha quindi che per ottenere il vettore c_2 è necessario calcolare *4 DFT di lunghezza 2*.



Abbiamo ottenuto:

$$F_k = F(q_0,q_1,q_2) = \sum_{p_2=0}^{1} c_2(q_2,q_1,p_2)w(\frac{kp_2}{8})$$

Passo 3 (ultimo)

Calcoliamo il vettore

$$c_3(q_2,q_1,q_0) = \sum_{p_2=0}^{1} c_2(q_2,q_1,p_2)w(\frac{kp_2}{8})$$ = $c_2(q_2,q_1,0)w(0)+ c_2(q_2,q_1,1)w((4q_0+2q_1+q_2)/8)$

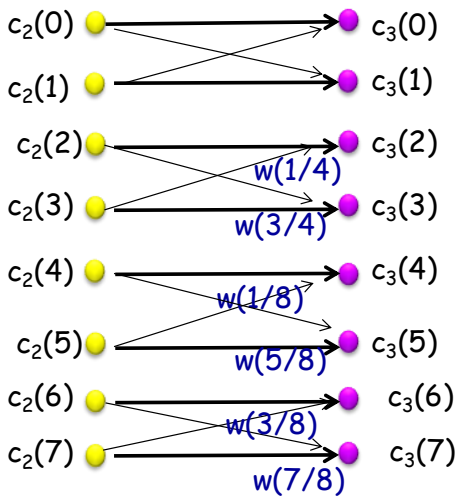
di 8 componenti che si ottengono al variare di q_2,q_1,q_0 = 0,1. Più in dettaglio:

$c_3(0)=c_3(0,0,0)=c_2(0,0,0)+c_2(0,0,1)w(0)=c_2(0)+c_2(1)$
$c_3(1)=c_3(0,0,1)=c_2(0,0,0)+c_2(0,0,1)w(1/2)=c_2(0)-c_2(1)$
$c_3(2)=c_3(0,1,0)=c_2(0,1,0)+c_2(0,1,1)w(1/4)=c_2(2)+c_2(3)w(1/4)$
$c_3(3)=c_3(0,1,1)=c_2(0,1,0)+c_2(0,1,1)w(3/4)=c_2(2)-c_2(3)w(1/4)$
$c_3(4)=c_3(1,0,0)=c_2(1,0,0)+c_2(1,0,1)w(1/8)=c_2(4)+c_2(5)w(1/8)$
$c_3(5)=c_3(1,0,1)=c_2(1,0,0)+c_2(1,0,1)w(5/8)=c_2(4)-c_2(5)w(1/8)$
$c_3(6)=c_3(1,1,0)=c_2(1,1,0)+c_2(1,1,1)w(3/8)=c_2(6)+c_2(7)w(3/8)$
$c_3(7)=c_3(1,1,1)=c_2(1,1,0)+c_2(1,1,1)w(7/8)=c_2(6)-c_2(7)w(3/8)$

con :$w(1/2)=-w(0)=-1$;$w(3/4)=-w(1/4)$;$w(5/8)=-w(1/8)$;$w(7/8)=-w(3/8)$.
Analogamente ai passi precedenti si ha:

$(c_3(0),c_3(1))$ =DFT del vettore $(c_2(0),c_2(1))$
$(c_3(2),c_3(3))$ =DFT del vettore $(c_2(2),c_2(3))$
$(c_3(4),c_3(5))$ =DFT del vettore $(c_2(4),c_2(5))$
$(c_3(6),c_3(7))$ =DFT del vettore $(c_2(6),c_2(7))$

Si ha quindi che per ottenere il vettore c_3 è necessario calcolare *4 DFT di lunghezza 2.*



Riassumendo:

passo 1 passo 2 passo 3

$f(p_0,p_1,p_2) \rightarrow c_1(q_2,p_1,p_2) \rightarrow c_2(q_2,q_1,p_2) \rightarrow c_3(q_2,q_1,q_0) = F(q_0,q_1,q_2)$

Ossia il vettore F ha le stesse componenti, ma in ordine diverso, del vettore c_3:

$c_3(2^2q_2 + 2q_1 + q_0) = F(2^2q_0 + 2q_1 + q_2) \Rightarrow$
$c_3(0)=F(0)$; $c_3(1)=F(4)$; $c_3(2)=F(2)$; $c_3(3)=F(6)$;
$c_3(4)=F(1)$; $c_3(5)=F(5)$; $c_3(6)=F(3)$; $c_3(7)=F(7)$.

E' necessario quindi "riordinare" le componenti del vettore c_3 (operazione di bit reversal).
Per valutare la complessità dell'algoritmo esposto consideriamo che esso calcola una DFT di lunghezza N=8 effettuando:

- al primo passo *4 DFT di lunghezza 2*
- al secondo passo *4 DFT di lunghezza 2*
- al terzo passo *4 DFT di lunghezza 2*

In totale quindi calcola: 3×4 DFT di lunghezza 2 $= (N/2)\log_2 N$ DFT di lunghezza 2

Algoritmo FFT radix-2. Caso generale $N=2^m$

A questo punto l'algoritmo precedentemente esposto si può generalizzare al caso di $N=2^m$. Posto:

$j=2^{m-1}\times p_0 + 2^{m-2} \times p_1 + \ldots + 2p_{m-2} + p_{m-1} = (p_0, p_1, \ldots, p_{m-1})_2$
$k=2^{m-1}\times q_0 + 2^{m-2} \times q_1 + \ldots + 2q_{m-2+} q_{m-1} = (q_0, q_1, \ldots, q_{m-1})_2 \qquad p_i, q_i = 0,1; i=0,..,m-1$

Si ha:

$$F_k = \sum_{j=0}^{N-1} f_j w(jk/N) \Leftrightarrow F_k = F(q_0,q_1,...,q_{m-1}) = \sum_{p_{m-1}=0}^{1} .... \sum_{p_1=0}^{1} \sum_{p_0=0}^{1} f(p_0,p_1,...,p_{m-1}) w\left(\frac{kp_0 2^{m-1} + ... + kp_{m-1}}{2^m}\right)$$

Ricordando la proprietà 2) di w si ha :

$$F_k = F(q_0,q_1,...,q_{m-1}) = \sum_{p_{m-1}=0}^{1} w\left(\frac{kp_{m-1}}{2^m}\right) .... \sum_{p_1=0}^{1} w\left(\frac{kp_1}{2^2}\right) \sum_{p_0=0}^{1} f(p_0,...,p_{m-1}) w\left(\frac{p_0 k}{2}\right)$$

Consideriamo l'esponenziale relativo alla somma più interna. Ricordando che w(r) =1 per r intero si ha:

$w(kp_0/2) = w(p_0(2^{m-1}q_0 + 2^{m-2} q_1 + \ldots + 2q_{m-2+} q_{m-1})/2) =$

$$w(2^{m-2}p_0q_0)\ w(2^{m-3}p_0q_1)\ldots\ w(p_0q_{m-2})\ w(p_0q_{m-1}/2)= w(p_0q_{m-1}/2)$$

$$F_k = F(q_0, q_1,..., q_{m-1}) = \sum_{p_{m-1}=0}^{1} w(\frac{kp_{m-1}}{2^m})... \sum_{p_1=0}^{1} w(\frac{kp_1}{2^2}) \sum_{p_0=0}^{1} f(p_0, p_1,.., p_{m-1}) w(\frac{p_0 q_{m-1}}{2})$$

Il calcolo del vettore F si effettua in m passi, a partire dalla somma più interna. Ad ogni passo, per ciascuna delle componenti del vettore F, l'algoritmo calcola una "somma" costituita da due addendi. Analizziamo l'algoritmo passo per passo.

Passo 1:

si calcola il vettore di N componenti:

$$c_1(q_{m-1}, p_1,..., p_{m-1}) = \sum_{p_0=0}^{1} f(p_0, p_1,..., p_{m-1}) w(\frac{p_0 q_{m-1}}{2}) =$$

$$= f(0,p_{1,...,}p_{m-1})w(0)+f(1,p_{1,...,}p_{m-1})w(q_{m-1}/2)$$

ottenute al variare di $q_{m-1}=0,1; p_i=0,1, i=1,\ldots,m-1$. Analogamente all'esempio si ha:

$c_1(0)=c_1(0,0,..,0) = f(0,0,..,0)+f(1,0,..,0)w(0)=f_0+f_{N/2}$
$c_1(1)=c_1(0,0,..,1) = f(0,0,..,1)+f(1,0,..,1)w(0)=f_1+f_{N/2+1}$
….
$c_1(N/2)=c_1(1,0,..,0)=f(0,..,0)+f(1,0,..,0)w(1/2)=f_0-f_{N/2}$
$c_1(N/2+1)=c_1(1,0,..,1)=f(0,..,1)+f(1,0,..,1)w(1/2)=f_1-f_{N/2+1}$
….

Come verificato nell'esempio, anche nel caso generale si ottiene che:

$(c_1(0), c_1(N/2))$=DFT del vettore $(f_0,f_{N/2})$
$(c_1(1), c_1(N/2+1))$=DFT del vettore $(f_1,f_{N/2+1})$
……

Si ha quindi che per ottenere il vettore c_1 è necessario calcolare *N/2 DFT di lunghezza 2*.
Dopo il primo passo , si ottiene:

$$F_k = F(q_0, q_1,..., q_{m-1}) = \sum_{p_{m-1}=0}^{1} w(\frac{kp_{m-1}}{2^m}).... \sum_{p_1=0}^{1} w(\frac{kp_1}{2^2}) c_1(q_{m-1}, p_1,..., p_{m-1})$$

Poiché (proprietà 1) di w):

$$w(kp_1/2^2)= w((2q_{m-2}p_1+q_{m-1}p_1)/2^2)$$

si ha :

$$F_k = F(q_0, q_1, q_{m-1}) = \sum_{p_{m-1}=0}^{1} w(\frac{kp_{m-1}}{2^m})..... \sum_{p_1=0}^{1} w(\frac{2q_{m-2}p_1 + q_{m-1}p_1}{2^2}) c_1(q_{m-1}, p_1,..., p_{m-1})$$

Passo 2:

si calcola il vettore di N componenti:

$$c_2(q_{m-1}, q_{m-2}, p_2,..., p_{m-1}) = \sum_{p_1=0}^{1} c_1(q_{m-1}, p_1,..., p_{m-1}) w(\frac{2p_1q_{m-2} + p_1q_{m-1}}{2^2}) =$$

$$= c_1(q_{m-1},0,p_{2,...,}p_{m-1})w(0)+c_1(q_{m-1},1,p_{2,...,}p_{m-1})w((2q_{m-2}+q_{m-1})/2^2)$$

ottenute al variare di $q_{m-1},q_{m-2}=0,1; p_i=0,1, i=2,\ldots,m-1$. Si ha che:

$c_2(0)=c_2(0,..,0)=c_1(0,..,0)+c_1(0,1,..,0)w(0)=c_1(0)+c_1(N/4)$
$c_2(1)=c_2(0,..,1)=c_1(0,..,1)+c_1(0,1,..,1)w(0)=c_1(1)+c_1(N/4+1)$
….
$c_2(N/4)=c_2(0,1,..,0)=c_1(0,..,0)+c_1(0,1,..,0)w(1/2)=c_1(0)-c_1(N/4)$
$c_2(N/4+1)=c_2(0,1,..,1)=c_1(0,0,..,1)+c_1(0,1,..,1)w(1/2)=c_1(1)-c_1(N/4+1)$
….

Si ottiene che:

($c_2(0)$, $c_2(N/4)$)=DFT del vettore ($c_1(0)$,$c_1(N/4)$)
($c_2(1)$, $c_2(N/4+1)$)=DFT del vettore ($c_1(1)$,$c_1(N/4+1)$)
……

Si ha quindi che per ottenere il vettore c_2 è necessario calcolare *N/2 DFT di lunghezza 2*.
In generale, al **Passo s** si ha:

$$F_k = F(q_0, q_1,..., q_{m-1}) = \sum_{p_{m-1}=0}^{1} w(\frac{kp_{m-1}}{2^m})....\sum_{p_{s-1}=0}^{1} w(\frac{kp_{s-1}}{2^s}) c_{s-1}(q_{m-1},...q_{m-s+1}, p_{s-1},..., p_{m-1})$$

Bisogna calcolare il vettore:

$c_s(q_{m-1},..,q_{m-s},p_{s,...,}p_{m-1}) = c_{s-1}(q_{m-1},..,q_{m-s+1},0,p_{s,...,}p_{m-1})w(0)+$
$+c_{s-1}(q_{m-1},..,q_{m-s+1},1,p_{s,..,}p_{m-1})w((2^{s-1}q_{m-s}+..+q_{m-1})/2^s)$

Come nei passi precedenti, per ottenere il vettore c_s è necessario calcolare *N/2 DFT di lunghezza 2*. Dopo m=$\log_2 N$ passi si ha:

$F(k) = F(q_0,q_1,\ldots,q_{m-1}) = c_m(q_{m-1},\ldots,q_1,q_0) = c_m(j)$ $\quad q_i=0,1,\ i=0,\ldots,m-1$

I due vettori hanno le stesse componenti, ma non nello stesso ordine. Una semplice operazione di **bit reversal** permette, noto j (o k) di ottenere k (o j):

- Convertire j in binario
- Invertire l'ordine delle cifre binarie che rappresentano j
- Considerare il numero binario ottenuto.Tale numero è la rappresentazione binaria di k.

Complessità di tempo dell'algoritmo FFT radix-2

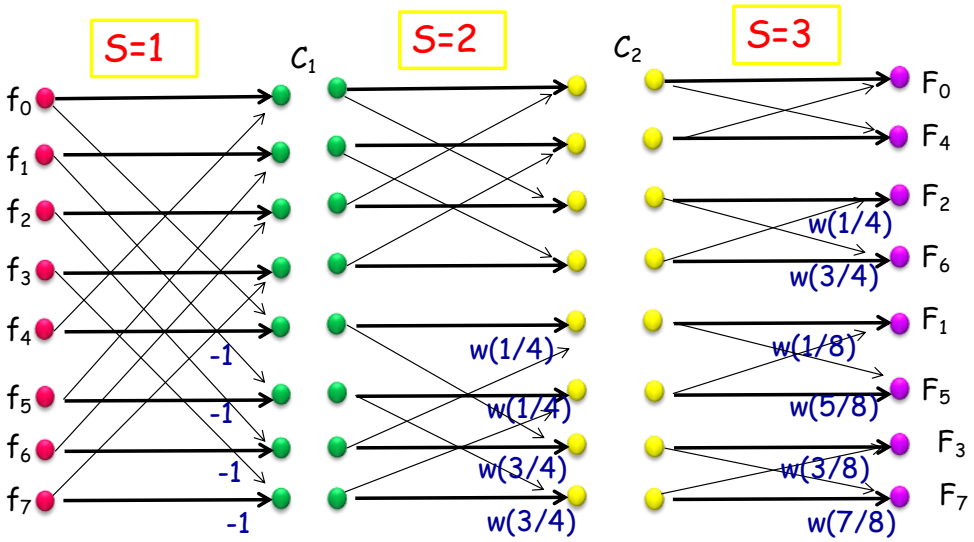
Il vettore F si calcola costruendo ricorsivamente i vettori c_s,s=1,2,..,$\log_2$N,ognuno dei quali è ottenuto dal calcolo di N/2 DFT di lunghezza 2.Il calcolo di una DFT di lunghezza N =2^m si effettua mediante il calcolo di : N$\log_2$N *DFT di lunghezza 2*
da cui: $T_{FFT}(N) = (N\log_2 N)/2 \times T_{DFT}(2) = (N\log_2 N)/2 \times O(2) = O(N\log_2 N)$
(le operazioni considerate sono addizioni e moltiplicazioni complesse).

Implementazione dell'algoritmo FFT radix-2

Come si è visto precedentemente, il calcolo di una DFT di lunghezza N=2^mconsiste fondamentalmente nel calcolo dei vettori c_s,s=1,2,..,$\log_2$N=m, e degli esponenziali complessi che compaiono nella loro espressione. Per una efficiente implementazione dell'algoritmo bisogna sfruttare alcune proprietà di tali vettori e degli esponenziali complessi ad essi collegati.

Costruzione dei vettori c_s , s=1,..,m

Per analizzare in dettaglio la costruzione dei vettori c_s,s=1,2,..,$\log_2$N, ritorniamo all'esempio con N=8. L'algoritmo procede come nella figura seguente:



Osserviamo che, per ottenere i nodi $c_1(0)$ e $c_1(4)$ sono necessari solo f(0) e f(4) e questi ultimi non sono più utilizzati nel calcolo delle altre componenti del vettore c_1. Analogamente si ha per le coppie (c_1 (1), c_1 (5)),(c_1 (2), c_1 (6)),…,e per le coppie dei passi successivi ($c_2(0),c_2(2)$),($c_2(5),c_2(7)$),.., ossia per le coppie le cui DFT di ordine 2 producono la coppia del passo successivo. Ciascuna di tali coppie è detta coppia di nodi duali.
Osserviamo che una coppia di nodi duali al passo s (in colonna s) è generata dalla coppia di nodi del passo s-1(in colonna s-1) occupanti la stessa posizione. Bisogna determinare a questo punto la distanza, ad ogni passo, tra due nodi duali.
Osservando la figura precedente, relativa ad N=8, otteniamo ad esempio:

Passo 1 (c_1 (0), $c_1(4)$) → distanza=4=N/2
Passo 2 (c_2 (0), $c_2(2)$) → distanza=2=$N/2^2$
Passo 3 (c_3 (0), $c_3(1)$) → distanza=1=$N/2^3$

Ciò vale in generale , cioè i nodi duali nella s-ma colonna distano tra loro :

$$\text{distanza} = d = N/2^s = 2^{m-s} \quad , s=1,2,\ldots, \log_2 N.$$

Ricordiamo a questo punto che una coppia di nodi duali al passo s è data da $q_{m-s}=0$ per il primo nodo e $q_{m-s}=1$ per il secondo:

$$c_s(q_{m-1},..,q_{m-s},p_{s,\ldots,}p_{m-1}) = c_{s-1}(q_{m-1},..,q_{m-s+1},0,p_{s,\ldots,}p_{m-1})w(0)+$$
$$+c_{s-1}(q_{m-1},..,q_{m-s+1},1,p_{s,..,}p_{m-1})w((2^{s-1}q_{m-s}+..+q_{m-1})/2^s)$$
$$q_{m-s}=0,1.$$

Se per la stringa di bit relativa al primo nodo si ha:

$$(q_{m-1},\ldots,q_{m-s+1},0,p_s,\ldots,p_{m-1}) = r$$

allora la stringa di bit relativa al suo duale si ottiene:

$$(q_{m-1},\ldots,q_{m-s+1},1,p_s,\ldots,p_{m-1}) = r+2^{m-s} = r + \text{distanza tra i 2 nodi}$$

Si può ottenere a questo punto una formula *ricorrente* per il calcolo dei vettori c_s,s=1,2,..,$\log_2 N$,che , per una coppia di nodi duali è:

- $c_s(r) = c_{s-1}(r) + c_{s-1}(r+2^{m-s})w(z/2^s)$
- $c_s(r+2^{m-s}) = c_{s-1}(r) + c_{s-1}(r+2^{m-s})w(z'/2^s)$

dove:

$$w(z/2^s) = w((2^{s-1}\times 0+\ldots+2q_{m-2}+q_{m-1})/2^s)$$

$$w(z'/2^s) = w((2^{s-1}\times 1+\ldots+2q_{m-2}+q_{m-1})/2^s)$$

cioè $z'=z+2^{s-1}$ da cui si ottiene:

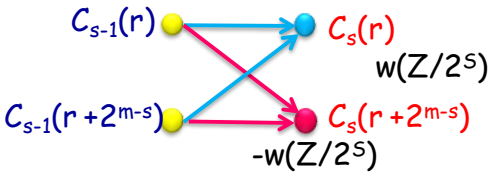
$$w(z'/2^s)=w((z+2^{s-1})/2^s)= w(z/2^s)w(1/2)= - w(z/2^s)$$

e quindi la formula ricorrente diventa:

A) $c_s(r) = c_{s-1}(r) +c_{s-1}(r+2^{m-s})w(z/2^s)$

B) $c_s(r+2^{m-s}) = c_{s-1}(r) -c_{s-1}(r+2^{m-s})w(z/2^s)$

le formule precedenti costituiscono *l'operazione di base dell'algoritmo FFT* e sono visualizzate dal seguente diagramma "*butterfly*" (a farfalla):



Le considerazioni precedenti consentono una implementazione efficiente basata sul calcolo, al passo s ,del vettore $c_s(r)$,r=0,1,..,N-1, mediante la valutazione di coppie di nodi duali,distanti tra loro $N/2^s=2^{m-s}$,a partire dal nodo $c_s(0)$. Consideriamo ad esempio $N=8=2^3$.

Passo *s=1*. A partire da $c_1(0)$ si calcolano, per j=0,1,2,3, $N/2^s$ =4 nodi e contemporaneamente i relativi duali(8 nodi):

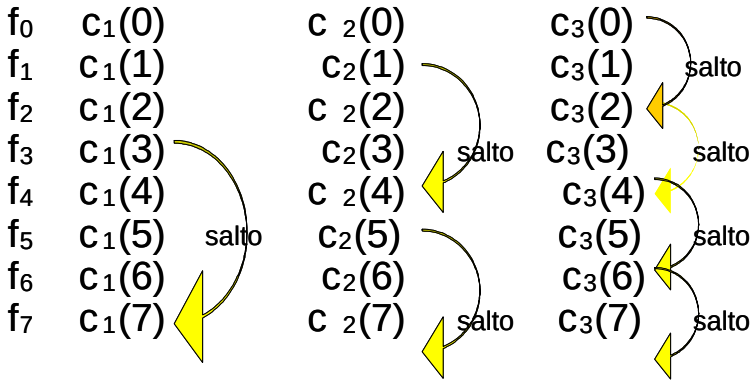
j=0 $c_1(0)$ e $c_1(4)$
j=1 $c_1(1)$ e $c_1(5)$
j=2 $c_1(2)$ e $c_1(6)$
j=3 $c_1(3)$ e $c_1(7)$

Il prossimo nodo da calcolare è $j=0+2\times 4=8 \Rightarrow$ fine passo 1.

Passo *s=2*. A partire da $c_2(0)$ si calcolano,per $j=0,1,N/2^s=2$ nodi e contemporaneamente i relativi duali(4 nodi):

j=0 $c_2(0)$ e $c_2(2)$
j=1 $c_2(1)$ e $c_2(3)$

Bisogna saltare i nodi 2,3 già calcolati e proseguire da $j=0+2\times 2=4 \Rightarrow c_2(4)$, e così via.



In generale, a partire dal nodo $c_s(0)$, l'algoritmo calcola $N/2^s$nodi e contemporaneamente duali. Per evitare di calcolare nuovamente tali duali è necessario effettuare un "salto" alla prima componente del vettore non ancora calcolata:

"salto"= $N/2^s=2^{m-s}$ nodi

In altre parole, per ogni s fissato, si calcola un vettore $C(r)=c_s(r)$ mediante le formule A),B),ossia

$$C(r)=C(r)+E\times A(n)$$

$$C(n)=C(r)-E\times A(n) \qquad \text{dove } n=r+2^{m-s},\ E=w(z/2^{s})$$

Valutazione degli esponenziali complessi.

Osserviamo che *gli esponenziali si ripetono da un passo all'altro*. Per verificare ciò torniamo all'esempio N=8 .

Ricordando la proprietà 4) di w, gli esponenziali da calcolare sono:

passo 1 1 esponenziale complesso : w(0)

passo 2 2 esponenziali complessi : w(0), w(1/4)

passo 3 4 esponenziali complessi : w(0),w(1/8),w(1/4),w(3/8)

In generale si ha che:

al passo s intervengono 2^{s-1} esponenziali diversi e si ripetono da un passo all'altro.

In totale quindi il numero di esponenziali diversi da calcolare è quello dell'ultimo passo, ossia è $2^{m-1}=N/2$ (w(0),w(1/N),w(2/N),…,w(((N/2)-1)/N). Inoltre all'interno di ogni "salto" si usa lo stesso esponenziale $\Rightarrow$ numero di esponenziali diversi ad ogni passo = numero di salti.

Bisogna ora ottenere una formula efficiente per il calcolo del fattore moltiplicativo $w(z/2^{s})$. Ricordando che il valore di *z* ed r, espressi in binario, è:

$z=2^{s-1}q_{m-s}+\ldots+2q_{m-2}+q_{m-1}$

$r=2^{m-1}q_{m-1}+\ldots+2^{m-s}q_{m-s}+2^{m-s-1}p_{s}+\ldots+2p_{m-2}+p_{m-1}$

si può ottenere *z* dalla stringa di bit che rappresenta r mediante il seguente algoritmo:

- *si converte r in binario su m bit* $\rightarrow(q_{m-1},\ldots,q_{m-s},p_{s},\ldots,p_{m-1})=r$
- *si estraggono i primi s bit* $\rightarrow(q_{m-1},\ldots,q_{m-s})$
- *si inverte l'ordine* $\rightarrow(q_{m-s},\ldots,q_{m-1})$
- *si ottiene z su s bit* $\rightarrow(q_{m-s},\ldots,q_{m-1})=z$

Complessità di spazio

Ad ogni passo una coppia di nodi è utilizzata solo per calcolare la coppia di nodi duali , nella colonna successiva, che occupa la stessa posizione. Ciò implica che è possibile implementare l'algoritmo FFT radix-2 "in place", utilizzando un solo array di lunghezza N. Tale array contiene:

- in input il vettore f
- ad ogni passo s il vettore c_s, s=1,2,…,m
- all'ultimo passo il vettore c_m, che ha le stesse componenti del vettore F, ma in ordine diverso.

Si ha quindi che: $S_{FFT}(N)=O(N)$

Algoritmo FFT radix-2

L'algoritmo fondamentalmente consiste in:

Input : $N=2^{m}$*; f(j), j=0,…,N-1*

Output: *F(k), k=0,…,N-1 (F= DFT di f).*

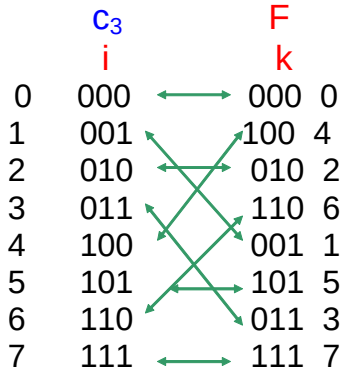
- 1. Calcolo dei vettori c_s, s=1,…,m

- 2. Riordino delle componenti di c_m per ottenere il vettore F (*bit reversal)*

Calcolo dei vettori c_s , s=1,..,m

```
for  s=1,m                 numero passi
     d= N/2^s              distanza nodi duali
     nesp=2^(s-1)          numero esponenziali diversi al passo s
     k=0
  for i=1,nesp             ciclo sugli esponenziali diversi(=numero di salti)
      r=k                  indice I el. di c_s che usa uno dei nesp esponenziali
     calcolo  w(z/2^s)
    for l=0,d-1            calcolo coppie di duali che utilizzano w(z/2^s)
        r=l+k              indice I elemento coppia
        r1=r+d             indice II elemento coppia
        t= w(z/2^s)f(r1)
        f(r1)=f(r)-t       calcolo coppia di nodi duali
        f(r)=f(r)+t
     endfor
     k=k+2d                indice primo elemento non ancora calcolato
  endfor
endfor
```

Osserviamo che t,f(r), f(r1) sono numeri complessi e si calcolano separatamente parte reale e coefficiente dell'immaginario. Bisogna inoltre ricordare che e^{ix}=cosx+isinx.
E' necessario a questo punto riordinare le componenti di c_m per ottenere F con un'operazione di bit reversal.



Nell'algoritmo precedente gli esponenziali w sono calcolati ad ogni passo e ciò comporta il calcolo di uno stesso esponenziale più di una volta:

- vantaggio : non è necessaria area di memoria ausiliaria
- svantaggio : calcolo dello stesso esponenziale più volte , ricalcolo di tutti gli esponenziali se si richiedono più DFT di dimensione N

Una strategia alternativa consiste nel calcolare una sola volta tutti gli N/2 w e memorizzarli in un array:

- vantaggio : si evitano calcoli di stessi esponenziali e si calcolano una sola volta nel caso di più DFT di ordine N
- svantaggio : necessità di un'area di memoria ausiliaria di dimensione O(N).

Poiché in molte applicazioni scientifiche è richiesto il calcolo di più DFT della stessa dimensione, molte librerie (NAG,IMSL,MATLAB,..) utilizzano tale strategia.
L'importanza della DFT nelle applicazioni ha portato alla progettazione di speciali dispositivi Hardware per l'implementazione degli algoritmi FFT : BBN Butterfly, Calcolatori paralleli basati su reti di transputer,…
Esistono algoritmi FFT radix-r ($N=r^k$) con la stessa complessità dell'algoritmo radix-2 e più in generale esistono algoritmi FFT mixed-radix con $N= r_1+r_2+..+r_n$, con complessità $O(N(r_1+r_2+..+r_n))$.Gli algoritmi FFT radix-2 sono i più utilizzati perché:

- un elaboratore utilizza il genere un sistema aritmetico floating point in base 2
- per N potenza di 2 alcuni esponenziali sono tali da evitare l'esecuzione effettiva delle moltiplicazioni ($\pm1,\pm i,\ldots$)

Si ha che il calcolo è più veloce se N è potenza di due, anche se la sequenza è più lunga, come si vede dal seguente esempio.

```
>> t=linspace(0,4*pi,4096);
>> y=sin(100*t)+rand(1,4096).*sin(50*t)-0.05;
>> t=cputime;
>> fft(y);
>> cputime-t
ans =
    0.0781
>> t=linspace(0,4*pi,3137);
>> y=sin(100*t)+rand(1,3137).*sin(50*t)-0.05;
>> t=cputime;
>> fft(y);
>> cputime-t
ans =
    0.1406
```

Infine, l'algoritmo FFT radix-2 risulta numericamente stabile e più accurato della valutazione diretta di una DFT.

Funzioni matlab fft, ifft

y=fft(x,n)
calcola la DFT di un vettore
x= vettore di input
n= facoltativo,calcola la DFT lunga n,se n>dimensione di x aggiunge zeri, se è minore taglia x ad n elementi
y=DFT di x
y=ifft(x,n)
calcola la IDFT di un vettore
x= vettore di input
n= facoltativo,calcola la IDFT lunga n,se n>dimensione di x aggiunge zeri, se è minore taglia x ad n elementi
y=IDFT di x
Le funzioni precedenti sono basate sulla libreria FFTW(1998), attualmente il miglior software per il calcolo della trasformata di Fourier.

Funzioni matlab di manipolazione di bit

bitrevorder

Permute data into bit-reversed order

y = bitrevorder(x)

returns the input data in bit-reversed order in vector or matrix y. The length of x must be an integer power of 2. If x is a matrix, the bit-reversal occurs on the first dimension of x with size greater than 1. y is the same size as x.

```
  x=[0:7]';              % Create a column vector
[x,bitrevorder(x)]
ans =
    0    0
    1    4
    2    2
    3    6
    4    1
    5    5
    6    3
    7    7
```

de2bi

Convert decimal numbers to binary vectors

b = de2bi(d)

b = de2bi(d) converts a nonnegative decimal integer d to a binary row vector. If d is a vector, the output b is a matrix, each row of which is the binary form of the corresponding element in d.

b = de2bi(d,n) is the same as b = de2bi(d), except that its output has n columns, where n is a positive integer.

b = de2bi(d,..,flg) uses the string flg to determine whether the first column of b contains the lowest-order or highest-order digits. Values for flg are 'right-msb' and 'left-msb'. The value 'right-msb' produces the default behavior.

```
>> de2bi(3,3)
ans =
    1    1    0
>> de2bi(3,3,'left-msb')
ans =
    0    1    1
```