UNIVERSITÀ DEGLI STUDI DI CAGLIARI

La Trasformata Discreta di Fourier in Python

Docente
Prof. Giuseppe Rodriguez

Ambra Demontis

Anno 2015

Indice

1 Introduzione **1**

2 Approssimazione ai minimi quadrati **3**
2.1 La formulazione generale 4
2.2 Un esempio in R^n 6
2.3 Approssimazione di funzioni 6
2.3.1 La serie di Fourier 7
2.3.2 La Trasformata di Fourier 9

3 Interpolazione **11**
3.1 Discrete Fourier transform (DFT) 11
3.1.1 La musica 13
3.2 2D Discrete Fourier transform 15
3.2.1 Le immagini 15
3.2.2 convoluzione 17

4 Codice python e figure generate **21**
4.1 interpolazione 26
4.2 Serie di Fourier 28
4.3 Trasformata di Fourier 33
4.4 2D Discrete Fourier Transform 37

Bibliografia **51**

Capitolo 1

Introduzione

Il mondo che ci circonda è pieno di fenomeni rappresentabili con funzioni continue, come i suoni e le immagini. Tuttavia per poter digitalizzare questi fenomeni occorre discretizzarli. La Trasformata Discreta di Fourier ci permette, dati i valori di un insieme di campioni, di ottenere un'approssimazione del fenomeno continuo che li ha generati. La maggior parte delle librerie che offrono funzionalità matematiche implementano la FFT (Fast Fourier Transform), che è un algoritmo veloce per il calcolo della DFT. Nei capitoli seguenti è stato riportato un riassunto della teoria riguardante la DFT e il codice python con il quale sono stati realizzati i plot allegati, realizzati con l'intento di chiarire tramite immagini il funzionamento dei vari parametri che regolano la trasformata.

Capitolo 2

Approssimazione ai minimi quadrati

La soluzione di tantissimi problemi richiede la risoluzione di sistemi lineari

$$A\mathbf{x} = b \tag{2.1}$$

con: A matrice $m \times n$, $\mathbf{b} \in R^m$, $\mathbf{x} \in R^n$
Il problema é che generalmente il numero di equazioni é differente dal numero delle incognite. Supponendo che A sia a rango pieno si possono avere due casi problematici:
$m > n$: sistema sovra-determinato. Si hanno più equazioni (vincoli) che incognite. Il problema potrebbe non avere soluzione.
(Notiamo che in casi in cui la matrice dei dati é affetta da errore oppure i nostri dati sono solo dei campioni prelevati da una distribuzione, e quindi vorremmo che la soluzione fosse corretta non solo per l'insieme dei dati campionato ma per tutta la distribuzione, questo é il caso più auspicabile).
$m < n$: sistema sotto-determinato. Si hanno più incognite che equazioni. Il problema può avere infinite soluzioni.

In questi casi il problema é mal posto! (potrebbe non avere un'unica soluzione o un piccolo errore nei dati potrebbe portare ad un notevole errore nella soluzione). Nei casi in cui i dati siano affetti da grossi errori, oppure nei casi in cui lo scopo é trovare una funzione generale che rappresenti bene anche nuovi dati, come nel problema mostrato nella Figura 2.1, non si effettua l'interpolazione ma piuttosto si cerca una soluzione che minimizza la differenza tra il primo ed il secondo membro del sistema. Generalmente si minimizza lo scarto quadratico medio (la varianza). Il problema diventa quindi:

$$\min_{\mathbf{x} \in R^n} = \left\| A\mathbf{x} - \mathbf{b} \right\| \tag{2.2}$$

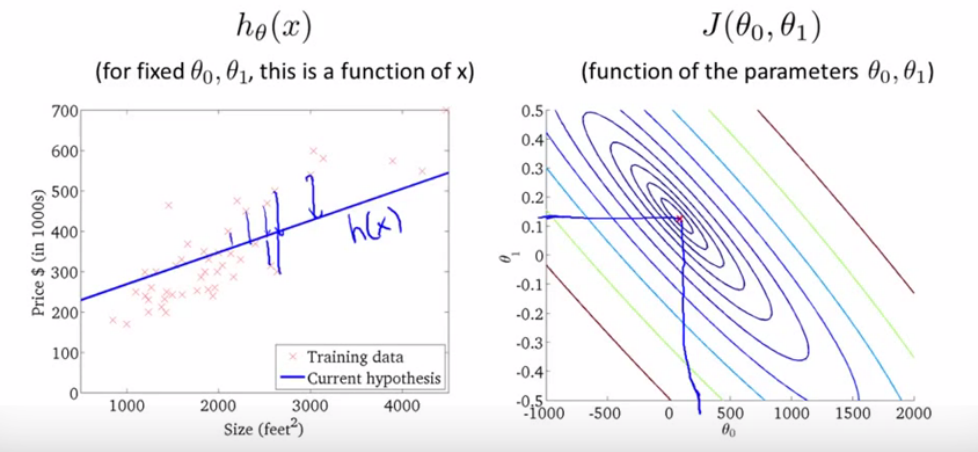

Figura 2.1: Avendo un'insieme di case delle quali conosciamo la dimensione in metri quadri e il prezzo, vogliamo trovare i parametri: θ_0 che rappresenta il bias e θ_1 che rappresenta l'inclinazione della retta $h(x) = \theta_0 + \theta_1 \cdot x$ che approssima il variare del prezzo al variare dei metri quadri in modo tale da poter, dati i metri quadri di una casa stimarne il prezzo.

2.1 La formulazione generale

Sia H uno spazio di Hilbert cioè uno spazio vettoriale normato dotato del prodotto scalare e della norma da esso indotta:

$$\|f\| = \langle f, f \rangle^{\frac{1}{2}}, \text{con } f \text{ in } V \tag{2.3}$$

Supponiamo di voler approssimare un vettore **v** appartenente allo spazio V con un vettore **w*** appartenente allo spazio H con H sottospazio di V. Il vettore che ci darà la migliore approssimazione nel senso dei minimi quadrati sarà il vettore che minimizza il quadrato della norma della differenza tra il vettore originale e il vettore approssimato:

$$\mathbf{w}^* = \min_{\mathbf{w} \in H} \|v - w\|^2 = \langle \mathbf{v} - \mathbf{w}, \mathbf{v} - \mathbf{w} \rangle \tag{2.4}$$

Un teorema afferma che il vettore $\mathbf{w}*$ è il vettore di migliore approssimazione in H di **v** in V se e solo se il vettore dell'errore tra il vettore di migliore approssimazione **w** e quello da approssimare **v** è ortogonale ad ogni vettore del sottospazio H nel quale vogliamo proiettarlo:

$$\langle \mathbf{v} - \mathbf{w}^*, \mathbf{w} \rangle = 0, \ \forall \mathbf{w} \in H \tag{2.5}$$

Per trovare tale vettore dovremmo risolvere il sistema lineare:

$$W\mathbf{w}^* = v \tag{2.6}$$

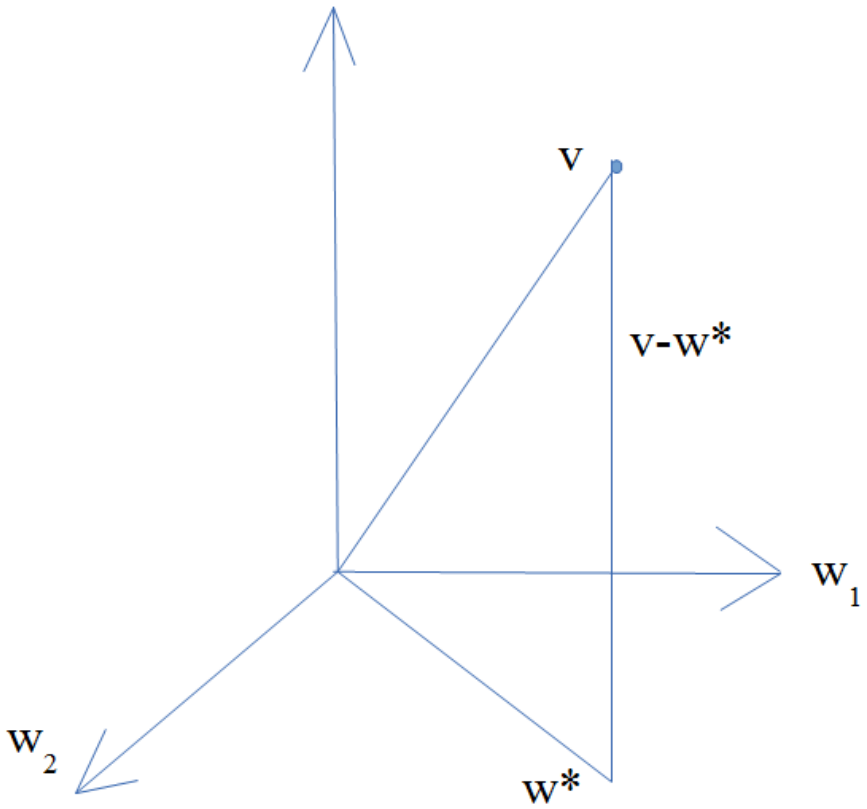

Figura 2.2: Avendo il vettore v $\in R^3$ trovare la sua migliore approssimazione nel senso dei minimi quadrati in R^2.

Essendo W la base del sottospazio H nel quale vogliamo proiettare il vettore. Poiché tale sistema é spesso sotto o sovradeterminato, troviamo il vettore risolvendo il sistema normale (con il rango della matrice incompleta (che nel sistema precedente era W) uguale al numero delle sue equazioni)

$$W^T W\mathbf{w} = W^T\mathbf{v} \tag{2.7}$$

La matrice W^TW viene detta matrice di Gram e ogni suo elemento é il prodotto scalare tra due vettori della base $\langle \mathbf{w}_i, \mathbf{w}_j \rangle$ Mentre il vettore colonna $W^T\mathbf{v}$ viene detto vettore dei momenti o vettore dei coefficienti di Fourier ed ogni suo elemento é la proiezione di v sui vettori della base $\langle \mathbf{v}, \mathbf{w}_i \rangle$.

2.2 Un esempio in R^n

Nella fig. 2.2 é mostrato un esempio in R^n preso da [4]. Abbiamo il vettore v in R^3 di coordinate:

$$\mathbf{v} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{2.8}$$

e vogliamo approssimarlo in R^2 essendo la base di R^3 composta da i due vettori colonna $\mathbf{v}_1$ e $\mathbf{v}_2$:

$$W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{2.9}$$

Sostituendo i valori nell'equazione n. 2.7 avremo quindi:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{w}_1^* \\ \mathbf{w}_2^* \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.10}$$

cioé:

$$\mathbf{w}^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.11}$$

2.3 Approssimazione di funzioni

Una funzione può essere approssimata con un polinomio trigonometrico. Un polinomio trigonometrico di ordine n é rappresentato da una funzione del tipo:

$$T_n = a_0 + \sum_{1}^{n} (a_k cos(kwx) + b_k sin(kwx)) \tag{2.12}$$

con a_0, a_k, b_k, e w numeri reali e con $w \not\equiv 0$.
Un polinomio trigonometrico é quindi una combinazione lineare delle $2n+1$ funzioni elementari:

$$1, cos(wx), sin(wx), ..., cos(nwx), sin(nwx) \tag{2.13}$$

tali funzioni sono dette **armoniche elementari** e hanno periodo $T = \frac{2\pi}{w}$ e periodo fondamentale $T = \frac{2\pi}{kw}$. Essendo una loro combinazione lineare, anche il polinomio trigonometrico sarà una funzione periodica di periodo $T = \frac{2\pi}{w}$.

Come possiamo notare dalla Figura 4.1, per quanto vengano chiamate tutte armoniche elementari a parte la funzione costante, tutte le altre possono essere generate a partire dalla funzione seno, alterandone la frequenza (nel caso degli altri seni), o fase e frequenza nel caso dei coseni. Il coseno infatti non é altro che la funzione seno con una fase di 1/2 pigreco. Tuttavia nella combinazione lineare i coefficienti (quindi ciò che possiamo variare per riprodurre il polinomio trigonometrico) rappresentano l'ampiezza, mentre la fase é fissa. Le armoniche elementari sono tra loro ortogonali. Due funzioni f e g sono ortogonali nel loro intervallo di definizione [a,b] se il loro prodotto interno in $L^2[a,b]$ é nullo:

$$\langle f, g \rangle = \int_b^a f(x)g(x)\,dx \tag{2.14}$$

Possiamo vedere graficamente la proprietà di ortogonalità osservando i grafici nella Figura 4.2. Le armoniche elementari costituiscono quindi uno spazio completo, il che significa che con una combinazione lineare delle prime 2n +1 armoniche elementari possiamo generare qualsiasi polinomio trigonometrico di ordine n.

2.3.1 La serie di Fourier

È un polinomio trigonometrico di ordine infinito, definito su tutta la retta reale del tipo:

$$Sf(x) = \frac{a_0}{2} + \sum_{k=1}^{\inf}[a_k cos(kwx) + b_k sin(kwx)] \tag{2.15}$$

con $w = \frac{\pi}{L}$.
Data una funzione integrabile in [-L,L], con periodo $T = 2L$, ad essa é associabile in modo univoco la serie di Fourier con coefficienti:

$$a_0 = \frac{1}{2L}\int_{-L}^{L} f(x)\,dx \tag{2.16}$$

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cdot cos(n\cdot\frac{\pi}{L}x)\,dx \tag{2.17}$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\cdot sin(n\cdot\frac{\pi}{L}x)\,dx \tag{2.18}$$

Che definisce l'estensione periodica, con periodo 2L della funzione su tutta la retta reale. La forma che abbiamo visto prima della serie di Fourier é

detta **forma trigonometrica**. La serie di Fourier può essere riscritta in modo più compatto nella **forma complessa**:

$$Sf(x) = \sum_{k=-\inf}^{\inf} c_k \cdot e^{ikwx} \tag{2.19}$$

con

$$c_k = \frac{1}{2\pi} \int_{-L}^{L} f(x) \cdot e^{-ikwx}\, dx \tag{2.20}$$

Nel caso in cui l'intervallo di definizione non sia centrato sullo zero ma sia un generico intervallo [a,b] possiamo traslare la funzione sommandole L, con $L = \frac{b-a}{2}$.
Scelto l'ordine del polinomio (quindi il numero di armoniche elementari) con il quale vogliamo approssimare la funzione, i coefficienti della serie di Fourier ci forniscono il polinomio di grado n che ci costituisce la miglior approssimazione della funzione nel senso dei minimi quadrati, detta anche della minima differenza di energia, essendo l'energia di una funzione la sua norma:

$$\|f\| = \left(\int_0^T |f(x)|^2 \, dx \right)^{\frac{1}{2}} \tag{2.21}$$

Per le funzioni simmetriche rispetto all'asse delle y dette **pari** (f(x) = f(-x)) i coefficienti diversi da zero sono a_0 e i coefficienti delle funzioni coseno (a_k) nella forma trigonometrica che corrispondono ai c_k reali nella forma complessa.
Per le funzioni che hanno una simmetria centrale rispetto all'origine dette **dispari** (f(x) = -f(-x)) i coefficienti diversi da zero sono quelli delle funzioni seno cioè i b_k nella forma trigonometrica che corrispondono ai c_k immaginari nella forma complessa.

Alcune proprietà che riguardano le capacità di approssimazione della serie di Fourier sono:
-la **convergenza dei coefficienti**:
Per il teorema di Riemann-Lebesgue se la funzione è regolare a tratti nell'intervallo [-L,L] i coefficienti della serie di Fourier a_k, b_k convergono a zero per k che tende all'infinito. La rapidità con la quale convergono a zero dipende dalla regolarità della funzione. Con infiniti coefficienti possiamo quindi riprodurre alla perfezione la funzione originale.
Ricordiamo che una funzione è **regolare a tratti** se esistono e sono finite le

derivate destra e sinistra nei punti estremi e non esistono all'interno punti con derivata destra o sinistra infiniti, cioè cuspidi o flessi a tangente verticale.
-convergenza puntuale della serie:
Per Dirichlet-Jordan, se la funzione è regolare a tratti la serie di Fourier converge al valore vero dove la funzione è continua, e nei punti nei quali è discontinua al valore medio dei due valori che assume la funzione in tali punti.
-Per la disuguaglianza di Bessel l'energia della serie di Fourier che meglio approssima una funzione è sempre inferiore a quella della funzione stessa.

2.3.2 La Trasformata di Fourier

Per approssimare una funzione periodica mediante la serie di Fourier sono necessari un numero esiguo di diverse frequenze e quindi di coefficienti. Nel caso in cui la funzione da approssimare sia una funzione non periodica il numero dei coefficienti necessari è notevolmente più alto. In questo caso la funzione non viene pertanto approssimata mediante la serie di Fourier ma utilizzando la trasformata di Fourier. La trasformata di Fourier non ci fornisce quindi una serie di coefficienti che rappresentano l'ampiezza di ogni armonica elementare ma una funzione $\mathcal{F}(f)$ che rappresenta l'ampiezza di ciascuna frequenza $k\,in\,R$ nella funzione f, ovvero. La trasformata di Fourier è definita come:

$$\mathcal{F}(f) = \int_{-\infty}^{\infty} e^{-ikx} \cdot f(x)\,dx \tag{2.22}$$

Il calcolo della trasformata di Fourier viene chiamato **analisi dello spettro delle frequenze**. Il processo di ricostruzione della funzione originale a partire dalla sua trasformata viene chiamato **sintesi del segnale** e si effettua utilizzando la trasformata inversa di Fourier:

$$\mathcal{F}^{-1}(F) = \frac{1}{2 * \pi} \cdot \int_{-\infty}^{\infty} e^{ikx} \cdot f(x)\,dx \tag{2.23}$$

Capitolo 3

Interpolazione

Dati i valori di una funzione in n + 1 punti, lo scopo dell'interpolazione è quello di trovare una funzione approssimante che assuma quei valori. La funzione deve passare esattamente in quei punti (a differenza della regressione nella quale si richiede solo che la funzione si avvicini il più possibile ai punti dati). La funzione approssimante sarà una combinazione lineare di funzioni semplici e linearmente indipendenti. Per trovare i coefficienti della combinazione lineare si impongono le condizioni di interpolazione (una per ogni punto del quale conosciamo i valori), ottenendo un sistema. Tale sistema avrà un unica soluzione se la matrice dei coefficienti è non singolare (cioè se ha determinante diverso da zero) e in tal caso viene detto sistema di Chebychev. Le funzioni più frequentemente usate come base per l'approssimazione sono i polinomi.

3.1 Discrete Fourier transform (DFT)

Per approssimare funzioni periodiche a tempo discreto, cioè funzioni delle quali conosciamo il valore solo in un insieme finito di punti a distanza costante, non si utilizza la serie di Fourier ma la DFT. Tipicamente queste funzioni sono segnali digitali che si ottengono dal campionamento di segnali analogici (ad esempio dei suoni). I coefficienti della DFT si ricavano cercando di interpolare una funzione definita in un intervallo [0,T) della quale conosciamo n campioni $x = 0, \frac{T}{n}, ...n-1 \cdot \frac{T}{n}$ prelevati ad intervalli $\frac{T}{n}$. Supponiamo di voler approssimare una funzione, della quale conosciamo il valore

in n punti, con la sua serie di Fourier:

$$Sf(x) = \sum_{k=0}^{L}[a_k cos(w_k x) + b_j sin(w_k x)] \tag{3.1}$$

$$L = \ parte\ intera\ di\ \frac{T}{2} \tag{3.2}$$

$$w_k = \frac{\pi \cdot k}{L} \tag{3.3}$$

Come spiegato in [2] per le formule di Eulero può essere riscritta come:

$$Sf(x) = \sum_{k=0}^{L}[\frac{a_k}{2} \cdot (e^{iw_k x} + e^{iw_k x}) + \frac{ib_k}{2} \cdot (e^{iw_k x} + e^{-iw_k x})] \tag{3.4}$$

$$= \sum_{k=0}^{L}[\frac{1}{2}\bar{p_j} \cdot e^{iw_k x} + \frac{1}{2}p_j \cdot e^{-iw_k x}] \ con\ p_j = a_j + ib_j \tag{3.5}$$

$$= \frac{\bar{p_0}}{2} + \frac{1}{2} \cdot \sum_{k=1}^{L}[\bar{p_j} \cdot e^{iw_k x}] + \frac{\bar{p_0}}{2} + \frac{1}{2}\sum_{k=1}^{L}[p_j \cdot e^{-iw_k x}] \tag{3.6}$$

$$= Sf(x) = a_0 + \frac{1}{2}\sum_{j=1}^{L}\bar{p_j} \cdot e^{i \cdot w_j \cdot x} + \frac{1}{2}\sum_{k=L+1}^{2L}\bar{p_k} \cdot e^{i \cdot w_{L-k} \cdot x} \tag{3.7}$$

Trovare i coefficienti della serie di Fourier corrisponde a risolvere il seguente sistema lineare:

$$\mathbf{x} = \bar{W} \cdot \mathbf{y} \tag{3.8}$$

dove:
il vettore x contiene i valori della funzione nei punti campionati

$$\mathbf{x} = (f_0, f_1 ... f_{2L}) \ \in \ C^n \tag{3.9}$$

il vettore y è il vettore che contiene i coefficienti della DFT

$$\mathbf{y} = (a_0, \frac{\bar{p_1}}{2}, \frac{\bar{p_2}}{2} ... \frac{\bar{p_L}}{2}, \frac{p_L}{2}, ... \frac{p_2}{2}, \frac{p_1}{2}) \ \in \ C^n \tag{3.10}$$

e $\bar{W}$ è una matrice $n \times n$ con ogni elemento:

$$\bar{w}_{sr} = e^{\frac{2\pi i}{n} s \cdot r} \tag{3.11}$$

Tali elementi sono i complessi coniugati delle radici n-esime dell'unità. W viene detta **Matrice di Fourier** ed ogni suo elemento è del tipo:

$$w_s r = e^{\frac{2\pi i}{n} s \cdot r} \tag{3.12}$$

Si dimostra che $\bar{W}$ è invertibile, con inversa:
$(\bar{W})^{-1}) = \frac{1}{n} W$
Il sistema lineare ha quindi un'unica soluzione:

$$\mathbf{y} = \frac{1}{n} W \cdot \mathbf{x} \tag{3.13}$$

Utilizzando la notazione complessa la formula per il calcolo della DFT risulta quindi:

$$F(u) = \frac{1}{n} \cdot \sum_{x=0}^{n-1} f(x) \cdot e^{\frac{-2\pi jux}{n}} \tag{3.14}$$

con u=0,...,n-1
e la IDFT risulta quindi:

$$f(x) = \sum_{x=0}^{n-1} F(u) \cdot e^{\frac{-2\pi jux}{n}} \tag{3.15}$$

con x=0,...,n-1

3.1.1 La musica

Ricordiamo che le principali grandezze che vengono utilizzate per misurare le onde sono: Aggiungiamo inoltre che per una generica onda
-la **velocità angolare**:

$$w = 2 \cdot \pi \cdot f = \frac{2\pi}{T} \tag{3.16}$$

indica quanti periodi ci sono in un intervallo 2π
-la **frequenza**:

$$f = \frac{1}{T} \tag{3.17}$$

indica quante volte durante il tempo T l'onda si ripete.
-Il **periodo fondamentale**

$$T = \frac{1}{f} \tag{3.18}$$

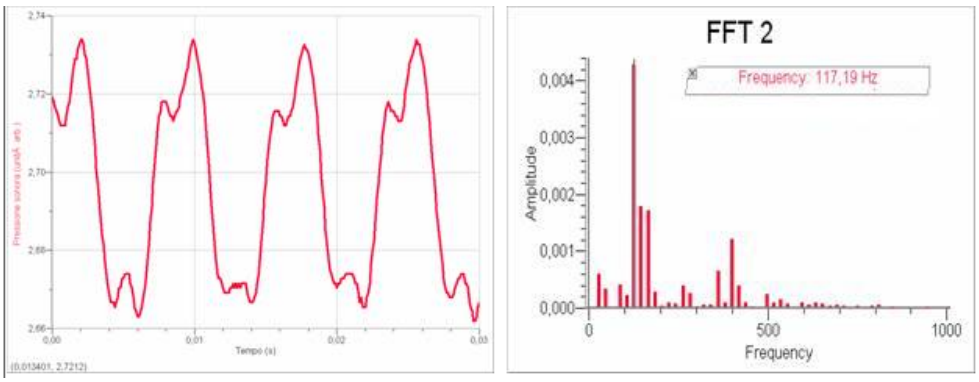

Figura 3.1: L'onda che rappresenta il suono prodotto da una chitarra.

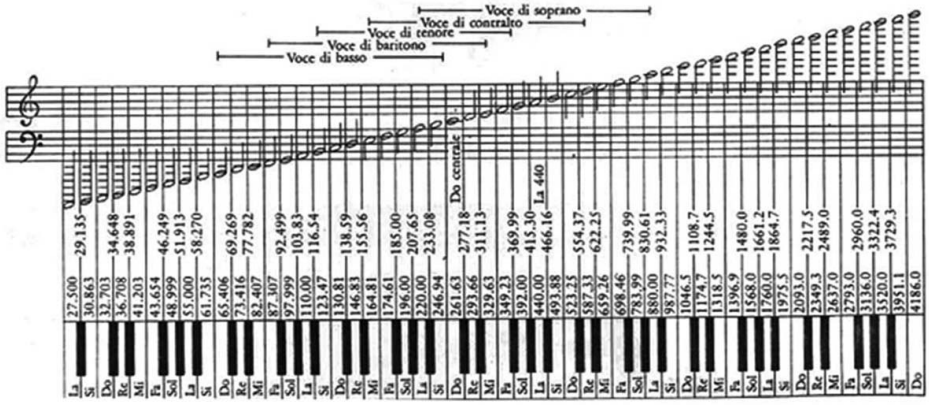

Figura 3.2: Ogni nota ha una sua frequenza caratteristica, nell'immagine sono mostrate le frequenze delle diverse note

indica dopo quanto l'onda si ripete ed è il reciproco della frequenza fondamentale (la prima armonica). Generalmente la frequenza dei suoni viene misurata in Hertz (Hz) che è l'inverso del secondo (s)

$$1HZ = \frac{1}{s} \tag{3.19}$$

Una nota non è fatta da una sola sinusoide semplice (il suono emesso dal telefono quando è libero lo è).

Il modo in cui è fatta (la sua forma) viene detto timbro e dipende dallo strumento che la emette, ad esempio la Figura 3.1 mostra il DO emesso da una chitarra. Mentre il timbro è caratteristico dello strumento che emette la nota, la frequenza (ogni quanto si ripete) è caratteristica della nota. Nella Figura 3.2 sono mostrate le frequenze in Hz delle diverse note musicali. Nella musica inoltre:
-L'ampiezza rappresenta il volume del suono (la sua intensità).
-La fase (quando il suono comincia ad essere emesso) non è riconoscibile

dall'orecchio umano.
-La frequenza fondamentale determina l'intonazione del suono. Alte frequenze hanno un suono acuto.

3.2 2D Discrete Fourier transform

La trasformata di Fourier 2D è analoga a quella in 1D ma effettua la trasformazione per funzioni di due variabili. La sua espressione è quindi:

$$\mathcal{F}(w_x, w_y) = \iint_{-\infty}^{\infty} f(x,y) \cdot e^{-iw_x x + -iw_y x}\, dx\, dy \tag{3.20}$$

Data l'indipendenza delle due variabili si può calcolare in due step effettuando prima la trasformata rispetto alla variabile x e trasformando il risultato rispetto alla variabile y:

$$\mathcal{F}^{-1}(w_x, w_y) = \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} f(x,y) \cdot e^{-iw_x x}\, dx \right) \cdot e^{-iw_y y}\, dy \tag{3.21}$$

o viceversa.
Analogamente la formula della DFT in due variabili sarà

$$\mathcal{F}(u,v) = \frac{1}{n \cdot m} \cdot \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} f(x,y) \cdot e^{-2\pi j \cdot \left(\frac{ux}{n} + \frac{vj}{m}\right)} \tag{3.22}$$

$$\tag{3.23}$$

con u=0,...,n-1 e v=0,...,m-1.
la IDFT:

$$\mathcal{F}^{-1}(x,y) = \cdot \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} \mathcal{F}(u,v) \cdot e^{2\pi j \cdot \left(\frac{ux}{n} + \frac{vj}{m}\right)} \tag{3.24}$$

con u=0,...,n-1 e v=0,...,m-1.

3.2.1 Le immagini

I coefficienti della trasformata, come nel caso 1D sono numeri complessi, per cui ognuno di essi ha un modulo ed una fase. La DFT 1D viene spesso

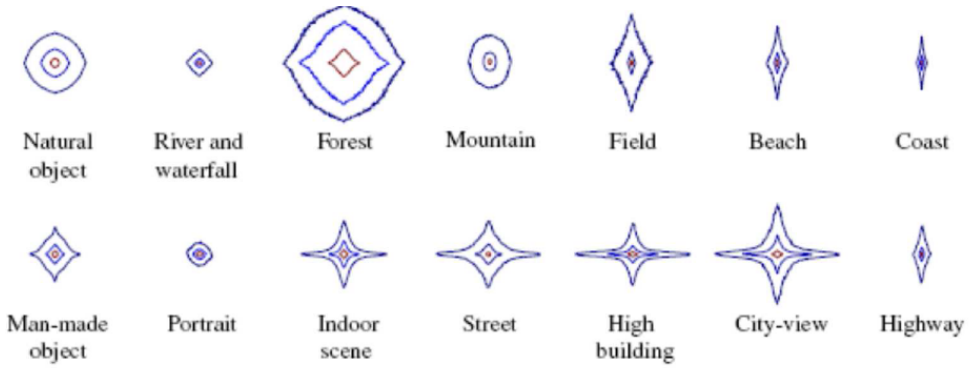

Figura 3.3: L'immagine mostra la firma spettrale di alcune immagini naturali

utilizzata per l'analisi del suono nel quale la fase non è riconoscibile. La DFT 2D viene spesso utilizzata per l'analisi delle immagini. In questo caso le grandezze che vengono prese in esame sono:
- lo **spettro della trasformata**, o modulo

$$|\mathcal{F}(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)} \tag{3.25}$$

che contiene l'informazione relativa alle strutture periodiche presenti nell'immagine. - **l'angolo di fase**:

$$\phi(u,v) = \tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right] \tag{3.26}$$

che contiene l'informazione relativa a dove le strutture periodiche evidenziate dalla DFT sono collocate.
- **la potenza spettrale**

$$|\mathcal{F}(u,v)|^2 = R^2(u,v) + I^2(u,v) \tag{3.27}$$

Sebbene esclusi i casi di immagini banali normalmente sia impossibile ricondurre parti della trasformata alle corrispondenti parti dell'immagine, poiché le alte frequenze rappresentano brusche variazioni nel colore, queste sono associabili ai bordi, mentre le basse frequenze sono associabili alle zone uniformi dell'immagine. Come mostrato in [1] è interessante osservare che alcuni ricercatori del MIT nel 2003 hanno pubblicato un articolo nel quale mostrano che le immagini naturali presentano una sorta di firma spettrale. Nella figura 3.3 sono mostrate le firme spettrali di alcune immagini naturali che sono state da loro ottenute come mostrato in 3.4 tracciando delle linee nello spettro in modo da racchiudere il 60, l'80 e il 90 percento dell'energia totale. Nella figura 3.5 sono state mostrate alcune immagini e le loro relative firme spettrali.

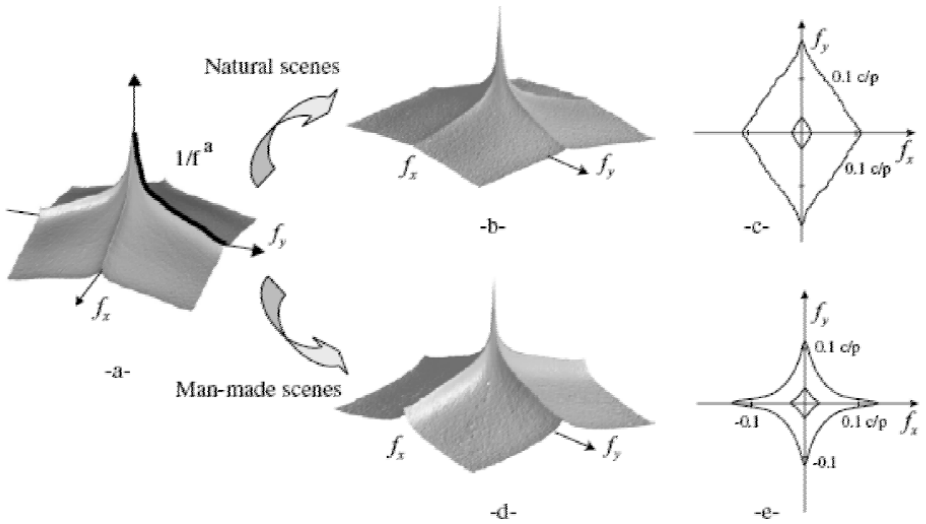

Figura 3.4: L'immagine mostra come viene ricavata la firma spettrale

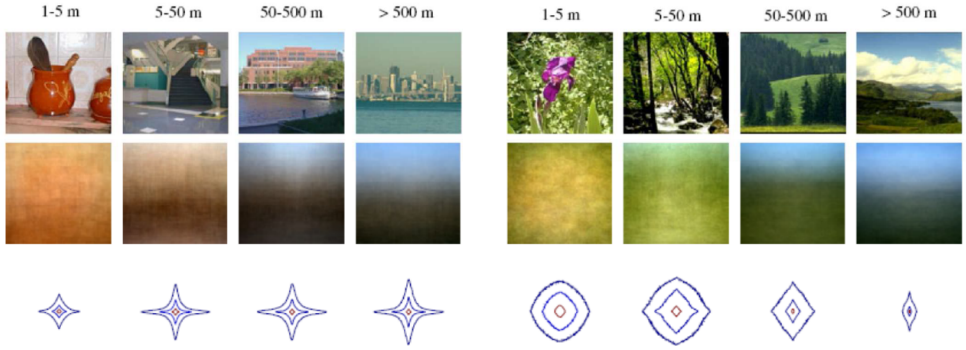

Figura 3.5: Alcune immagini e le loro firme spettrali

3.2.2 convoluzione

Per modificare l'immagine, ad esempio per renderla più uniforme smussando i bordi si utilizzano dei filtri. Per applicare i filtri bisogna effettuare la **convoluzione** tra l immagine ed il filtro. Siano f e g due funzioni integrabili in $\mathbb{R}$. La loro **convoluzione** è la funzione:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(x-a) \cdot g(a)\, da \tag{3.28}$$

in 2D:

$$(f * g)(x,y) = \iint_{-\infty}^{\infty} f(x-a, y-b) \cdot g(a,b)\, da\, db \tag{3.29}$$

Effettuare la convoluzione di un'immagine nel dominio spaziale significa sovrapporre il filtro ad ogni pixel dell'immagine e calcolare il nuovo valore del

pixel in base al risultato dell'applicazione del filtro ai pixel nel suo vicinato. Tale operazione può essere effettuata più facilmente nel dominio delle frequenze. Per il **Teorema della Convoluzione**:

$$f(x,y) * g(x,y) = \mathcal{F}(u,v) \cdot \mathcal{G}(u,v) \tag{3.30}$$

La convoluzione di due funzioni f e g ha come trasformata di Fourier il prodotto delle trasformate delle due funzioni, ovvero la convoluzione nel dominio spaziale corrisponde al prodotto nel dominio delle frequenze. Analogamente la convoluzione nel dominio delle frequenze corrisponde al prodotto nel dominio spaziale:

$$f(x,y) \cdot g(x,y) = \mathcal{F}(u,v) * \mathcal{G}(u,v) \tag{3.31}$$

Capitolo 4

Codice python e figure generate

Il codice 4.1 mostra le armoniche elementari e come siano tutte (a parte la funzione costante) una variazione della funzione seno. Ricordiamo che le funzioni seno e coseno hanno periodo fondamentale

$$T = \frac{1}{f} = \frac{2 \cdot \pi}{w} \tag{4.1}$$

e quindi frequenza:

$$f = \frac{1}{T} = \frac{w}{2 \cdot \pi} \tag{4.2}$$

I plot generati sono mostrati nella Figura 4.1.

Il codice 4.2 mostra graficamente la proprieta di ortogonalita fra le armoniche elementari con i plot 4.2. Come possiamo vedere l'area sottesa dalla funzione sopra l'asse delle x è la stessa di quella sotto tale asse.

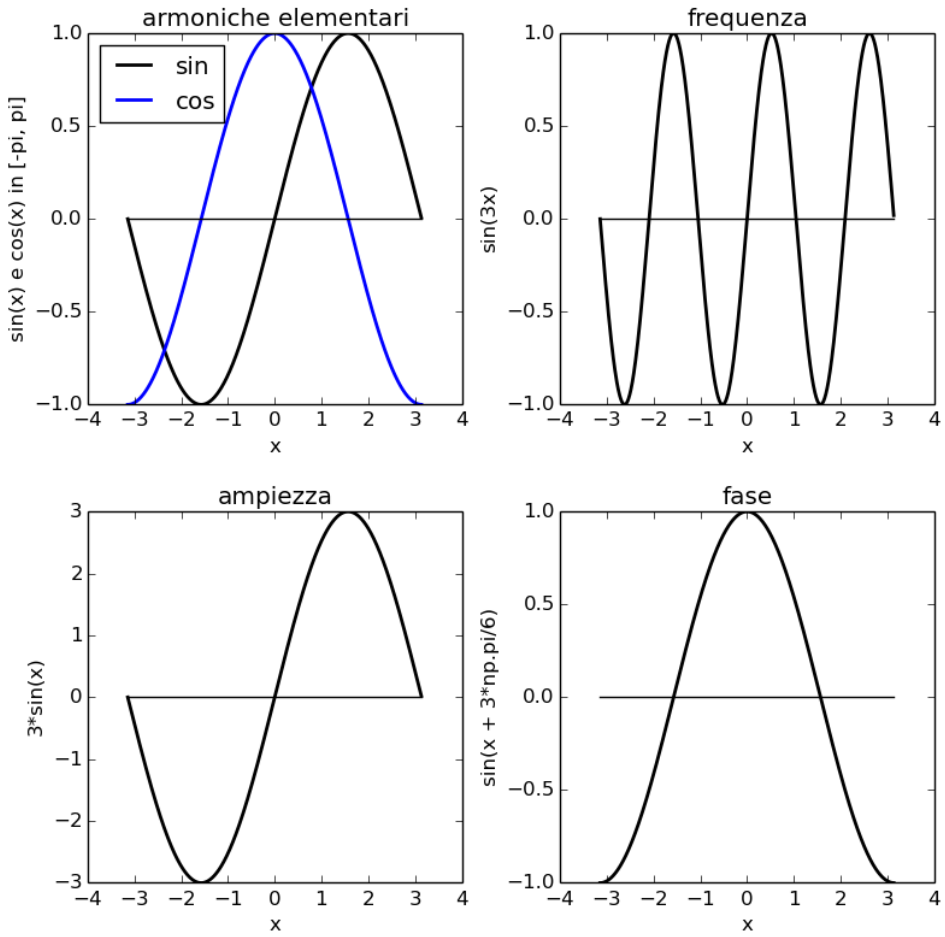

Figura 4.1: La figura mostra le Armoniche elementari e mette in evidenza come siano tutte delle variazioni della funzione seno.

Listing 4.1: Programma, armoniche elementari

```
1
import numpy as np
3 from numpy import linspace
import matplotlib
5 import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
7 width = 8
height = 8
9 fig = plt.figure(figsize=(width, height))
grid_spec = gridspec.GridSpec(2,2)
11

13 n = 1000 #point on x axis
intervallo = 2*np.pi
15 t = np.arange( 0, n ).astype(float) / n * intervallo - intervallo/2 #bisogna ricordarsi di convertire a float! altrimenti
       la divisione da un vettore di [0...1]
#vogliamo vedere la funzione in [-pi , pi]
17 #arange crea un array di interi con intervallo 1, da 0 a n
#dividiamo per n in modo da avere n reali equispaziati da 0 a 1
19 #moltiplichiamo per intervallo in modo da avere n reali equispaziati in 0, intervallo
# sottraiamo -np.pi per shiftarlo indietro in modo che vada da - intervallo/2 a intervallo/2 cioe' da - pigreco a
       pigreco
21 fsin = np.sin( t )
fcos = np.cos( t )
23 c = np.zeros(n) #crea una funzione costante in zero
sp = fig.add_subplot(grid_spec[0, 0])
25 sp.plot ( t, fsin, color='black', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione seno
sp.plot ( t, fcos, color='blue', alpha=1, linewidth=2.0, linestyle='-') #plottiamo la funzione coseno
27 sp.legend(['sin','cos'],loc ='northwest')
sp.plot ( t, c, color='black', alpha=1, linewidth=1.0, linestyle='-' ) #plottiamo una funzione costante in zero
29 sp.set_xlabel("x")
sp.set_ylabel("sin(x) e cos(x) in [-pi, pi]")
31 sp.set_title("armoniche elementari")

33
sp = fig.add_subplot(grid_spec[0,1])
35 fsin = np.sin( 3*t )
sp.plot ( t, fsin, color='black', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione seno
37 sp.plot ( t, c, color='black', alpha=1, linewidth=1.0, linestyle='-' ) #plottiamo una funzione costante in zero
sp.set_xlabel("x")
39 sp.set_ylabel("sin(3x)")
sp.set_title("frequenza")
41

43 sp = fig.add_subplot(grid_spec[1,0])
fsin = 3 * np.sin( t )
45 sp.plot ( t, fsin, color='black', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione seno
sp.plot ( t, c, color='black', alpha=1, linewidth=1.0, linestyle='-' ) #plottiamo una funzione costante in zero
47 sp.set_xlabel("x")
sp.set_ylabel("3*sin(x)")
49 sp.set_title("ampiezza")

51
sp = fig.add_subplot(grid_spec[1,1])
53 fsin = np.sin( t + 3 * np.pi / 6 )
sp.plot ( t, fsin, color='black', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione seno
55 sp.plot ( t, c, color='black', alpha=1, linewidth=1.0, linestyle='-' ) #plottiamo una funzione costante in zero
sp.set_xlabel("x")
57 sp.set_ylabel("sin(x + 3*np.pi/6)")
sp.set_title("fase")
59 print "la funzione coseno e' uguale a seno shiftato di 1/2 pigreco, alterandone la fase"

61 fig.tight_layout(w_pad=0.5)

63 plt.show(block=True)
```

Listing 4.2: Programma, orgogonalita

```
import numpy as np
from numpy import linspace
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
width = 11
height = 11
fig = plt.figure(figsize=(width, height))
grid_spec = gridspec.GridSpec(2,2)

n = 1000 #point on x axis
intervallo = 2*np.pi
t = np.arange( 0, n ).astype(float) / n * intervallo - intervallo/2 #bisogna ricordarsi di convertire a float! altrimenti
    la divisione da un vettore di [0...1]
fsin = np.sin( t )
fcos = np.cos( t )
f = fsin * fcos
c = np.zeros(n) #create a constant function
sp = fig.add_subplot(grid_spec[0, 0])
sp.plot ( t, c, color='black', alpha=1, linewidth=1.0, linestyle='-' ) #plottiamo una funzione costante in zero
sp.plot ( t, fsin, color='black', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione seno
sp.plot ( t, fcos, color='blue', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione coseno
sp.plot ( t, f, color='red', alpha=1, linewidth=5.0, linestyle='-' ) #plottiamo la funzione composta
sp.set_xlabel("x")
sp.set_ylabel("in rosso f(x) = sin(x)*cos(x)" )
sp.set_title("ortogonalita di sin(x) e cos(x)")
print "verifichiamo l ortogonalita di sin(x) e cos(x) "
sp.axvline(-np.pi, c='black', ymin=-1.5, ymax=1.5)
sp.axvline(np.pi, c='black', ymin=-1.5, ymax=1.5)

fsin = np.sin( t )
const = np.ones((n)) * 2
f = fsin * const
sp = fig.add_subplot(grid_spec[0, 1])
c = np.zeros(n) #create a constant function
sp.plot ( t, const, color='black', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo una funzione costante in 2
sp.plot ( t, c, color='black', alpha=1, linewidth=1.0, linestyle='-' ) #plottiamo una funzione costante in zero
sp.plot ( t, fsin, color='black', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione seno
sp.plot ( t, f, color='red', alpha=1, linewidth=5.0, linestyle='-' ) #plottiamo la funzione composta
sp.set_ylim((-3,+3))
sp.set_xlabel("x")
sp.set_ylabel("in rosso f(x) = sin(x)* np.ones((n)) * 2" )
sp.set_title("ortogonalita di sin e della funzione costante")
print "verifichiamo l ortogonalita di sin e della funzione costante "
sp.axvline(-np.pi, c='black', ymin=-1.5, ymax=1.5)
sp.axvline(np.pi, c='black', ymin=-1.5, ymax=1.5)

fsin = np.sin( t )
fcos = np.cos( 2*t )
f = fsin * fcos
c = np.zeros(n) #create a constant function
sp = fig.add_subplot(grid_spec[1, 0])
sp.plot ( t, c, color='black', alpha=1, linewidth=1.0, linestyle='-' ) #plottiamo una funzione costante in zero
sp.plot ( t, fsin, color='black', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione seno
sp.plot ( t, fcos, color='blue', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione coseno
sp.plot ( t, f, color='red', alpha=1, linewidth=5.0, linestyle='-' ) #plottiamo la funzione composta
sp.set_xlabel("x")
sp.set_ylabel("in rosso f(x) = sin(x)*cos(2x)" )
sp.set_title("ortogonalita di sin(x) e cos(2x)")
print "verifichiamo l ortogonalita di sin e cos "
sp.axvline(-np.pi, c='black', ymin=-1.5, ymax=1.5)
sp.axvline(np.pi, c='black', ymin=-1.5, ymax=1.5)

fstring = 'np.sin( 2 * x ) + .5 * np.cos ( 4 * x )'
fsin = np.sin( 2* t )
fcos = .5 * np.cos( 4 * t )
f = fsin + fcos
sp.set_xlabel("x")
c = np.zeros(n) #create a constant function
sp = fig.add_subplot(grid_spec[1, 1])
sp.plot ( t, c, color='black', alpha=1, linewidth=1.0, linestyle='-' ) #plottiamo una funzione costante in zero
sp.plot ( t, fsin, color='black', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione seno
sp.plot ( t, fcos, color='blue', alpha=1, linewidth=2.0, linestyle='-' ) #plottiamo la funzione coseno
sp.plot ( t, f, color='red', alpha=1, linewidth=5.0, linestyle='-' ) #plottiamo la funzione composta
sp.axvline(-np.pi, c='black', ymin=-1.5, ymax=1.5)
sp.axvline(np.pi, c='black', ymin=-1.5, ymax=1.5)
sp.set_xlabel("t")
sp.set_ylabel("f(x) = " + fstring )
sp.set_title("funzione composta da armoniche elementari")

fig.tight_layout(w_pad=0.5)

plt.show(block=True)
```

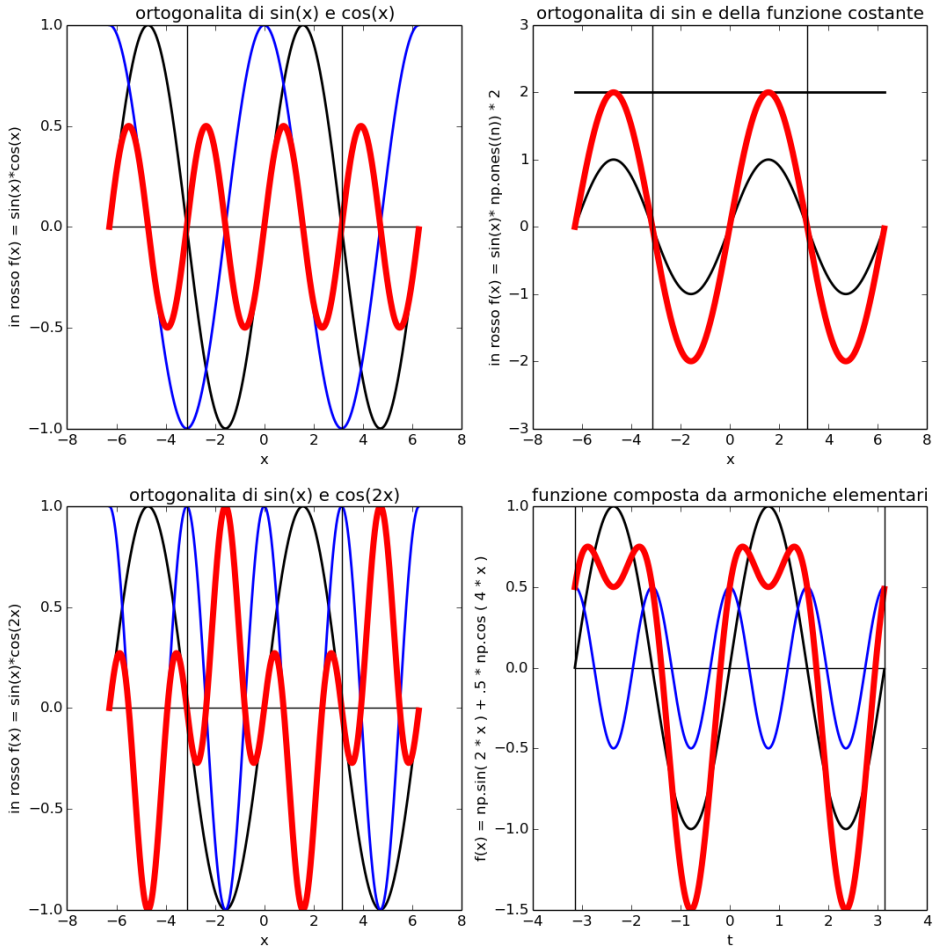

Figura 4.2: La figura mostra come le armoniche elementari siano ortogonali. L'area sottesa dalla curva sopra l'asse delle x risulta infatti identica a quella sottesa dalla curva sotto l'asse delle x.

4.1 interpolazione

Il codice 4.3 mostra la ricostruzione di una funzione, dato il suo valore in cinque punti, sia mediante la risoluzione del sistema 3.13, sia mediante l'utilizzo della funzione fft (Fast Fourier Transform) di python che calcola in modo veloce la DFT. Nel codice abbiamo moltiplicato la trasformata per $\frac{1}{n}$ e la ricostruzione per n per ottenere risultati coerenti con la definizione di DFT e di IDFT data, in quanto python come anche matlab moltiplica per $\frac{1}{n}$ i coefficienti durante la ifft.

Listing 4.3: Programma, interpolazione

```
from scipy.integrate import simps
import numpy as np
from numpy import linspace
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from numpy.fft import *
width = 12
height = 12
fig = plt.figure(figsize=(width, height))
grid_spec = gridspec.GridSpec(1,2)

sp = fig.add_subplot(grid_spec[0, 0])
print "prendiamo 5 campioni e cerchiamo di ricostruirla"
n = 5 #numero dei campionamenti
t = np.array([0,1,2,3,4])
y = np.array([1,1,1,2,2]) #valutiamo la funzione in quei 5 punti (in modo da avere i valori dei nostri campionamenti)

#calcoliamo i coefficienti risolvendo il sistema
#avendo 5 campioni proviamo ad approssimarli con un polinomio trigonometrico di ordine 5
#costruiamo la matrice di fourier
W = np.ones((n,n)).astype(complex)
esp = np.array(-2.0*np.pi/(n) * 1j )
for r in xrange (0,n):
for c in xrange (0,n):
W[r,c] = np.exp( esp )
W[r,c] = np.power(W[r,c], (r*c))
#troviamo i coefficienti della DFT risolvendo il sistema lineare
c_m = 1.0/n * np.dot(W,y.T)
print "i coefficienti trovati risolvendo il sistema sono :",c_m
g = (n*ifft(c_m)) #g e' la funzione ricostruita #
sp.plot ( t, g, color='red', alpha=1, linewidth=1.0, linestyle='-' )
sp.scatter ( t, y, c='yellow', s=18.0) #plottiamo i punti interpolati
sp.set_title("FFT ottenuti risolvendo il sistema" )

#usiamo numpy per calcolare i coefficienti
sp = fig.add_subplot(grid_spec[0, 1])
c = (1./n * fft(y) )
print "i coefficienti trovati con la funzione FFT di numpy sono :",c
g = (n*ifft(c)) #g e' la funzione ricostruita #
sp.plot ( t, g, color='red', alpha=1, linewidth=1.0, linestyle='-' )
sp.scatter ( t, y, c='yellow', s=18.0) #plottiamo i punti interpolati
sp.set_title("FFT calcolati numpy" )
plt.show(block=True)
```

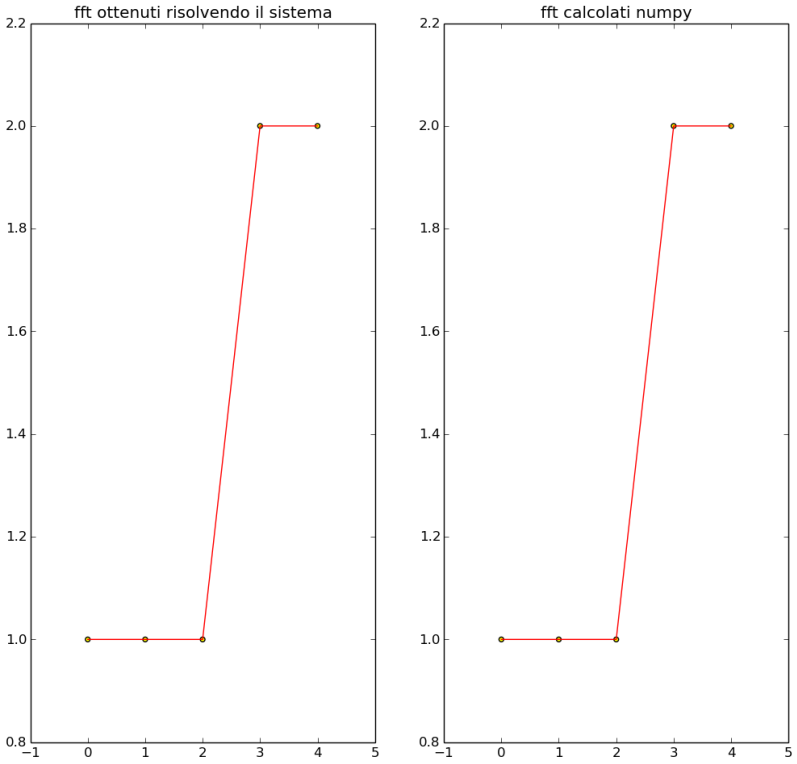

Figura 4.3: Interpolazione della funzione generatrice dei 5 punti mostrati in giallo utilizzando la DFT.

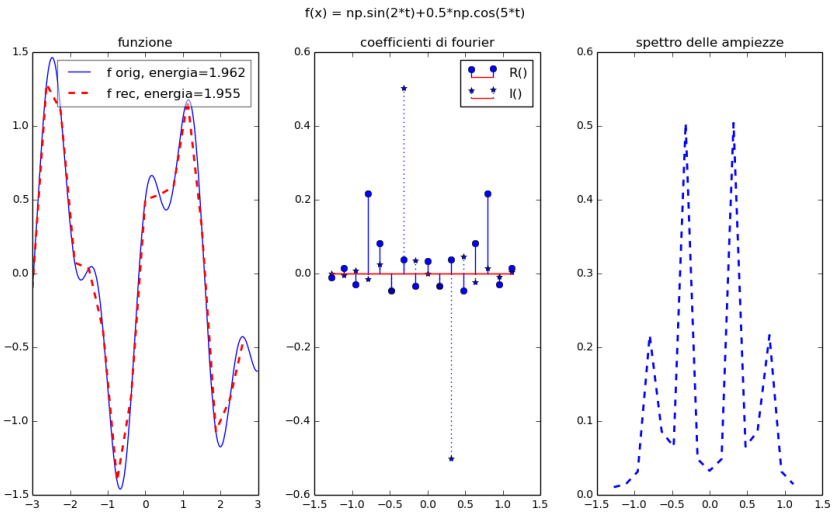

Figura 4.4: Nella Figura a sinistra è mostrata in rosso la ricostruzione della funzione in blu campionata in 16 punti. Nella Figura centrale sono mostrati i coefficienti di Fourier (con la linea continua la loro parte reale, con linea tratteggiata la loro parte immaginaria), mentre la Figura a destra mostra lo spettro dei coefficienti

4.2 Serie di Fourier

Per calcolare la serie di Fourier di una funzione possiamo campionarla ed utilizzare poi le funzioni di numpy per il calcolo veloce della DFT. Le funzioni fft e ifft di python richiedono che i valori della funzione siano campionati nell'intervallo [0,T) pertanto nel caso in cui l'intervallo sia centrato sullo zero è necessario utilizzare ifftshift. Lo spazio di campionamento, detto anche periodo di campionamento (intervallo di campionamento / numero dei campioni), che è l inverso della frequenza di campionamento , non deve essere necessariamente uno, tuttavia nel caso in cui sia differente bisogna ricordarsi di settare il parametro d della funzione fftfreq che dato un intero n, che rappresenta il numero di sample, fornisce un array di n elementi contenenti le frequenze:
$f = [0, 1, ..., n/2 - 1, -n/2, ..., -1]/(d * n)$ se n è pari
$f = [0, 1, ..., (n-1)/2, -(n-1)/2, ..., -1]/(d * n)$ se n è dispari

4.4 è il codice python per il calcolo dei coefficienti di Fourier utilizzato per produrre i grafici delle Figure 4.2, 4.2, 4.6, 4.2 nei quali vengono mostrati i coefficienti della serie di Fourier.

Quando si sceglie la **frequenza di campionamento** occorre tenere conto del teorema di campionamento secondo il quale: Se un segnale è campio-

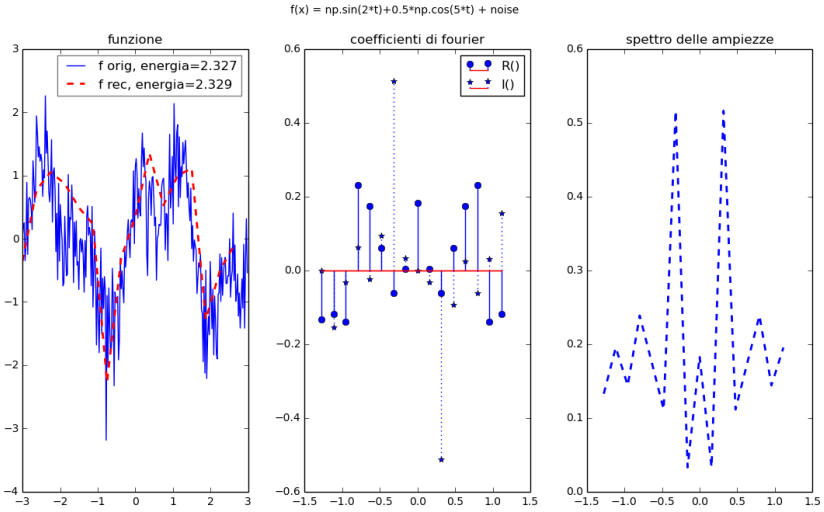

Figura 4.5: Nella Figura a sinistra è mostrata in rosso la ricostruzione della funzione rumorosa in blu campionata in 16 punti. Nella Figura centrale sono mostrati i coefficienti di Fourier (con la linea continua la loro parte reale, con linea tratteggiata la loro parte immaginaria), mentre la Figura a destra mostra lo spettro dei coefficienti

nato con una frequenza almeno doppia rispetto al suo massimo contenuto in frequenza il campionamento non induce errori. Se questo criterio non viene rispettato si incorre nel fenomeno dell'**aliasing**, cioè il contributo energetico delle frequenze per le quali il teorema di campionamento non è rispettato appaiono a frequenza inferiore.
Vediamo un esempio dal sito [3] : Considerando il segnale periodico a banda limitata, mostrato in figura 4.8, formato da 5 frequenze da 100, 200, 300, 400 e 500 Hz, il suo periodo è il reciproco della frequenza della prima armonica (la fondamentale), perciò $T = 1/100 = 10ms$. La massima frequenza del segnale è 500 Hz. Per il teorema, la frequenza di campionamento dev'essere pari o superiore a 1000 Hz. Il corrispondente campionamento è mostrato nella figura 4.9

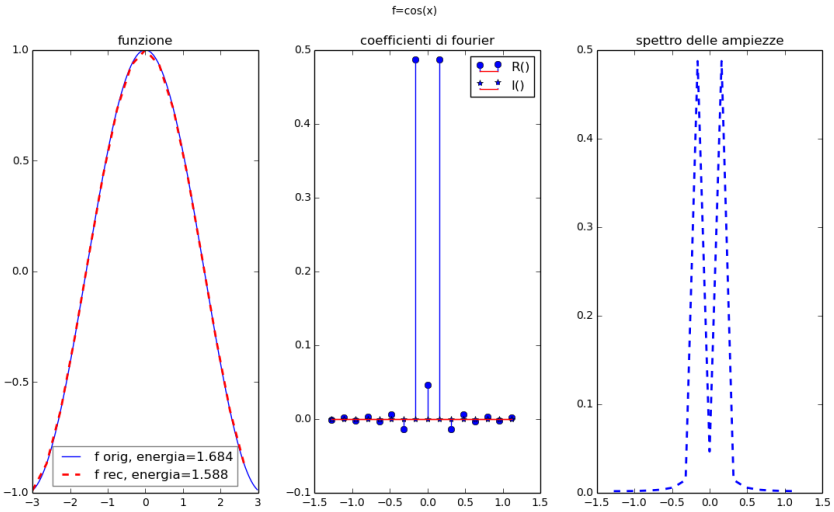

Figura 4.6: Nella figura a sinistra è mostrata in rosso la ricostruzione della funzione coseno in blu campionata in 16 punti. Nella Figura centrale sono mostrati i coefficienti di Fourier (con la linea continua la loro parte reale, con linea tratteggiata la loro parte immaginaria), mentre la Figura a destra mostra lo spettro dei coefficienti. Si può notare che la parte immaginaria dei coefficienti è nulla.

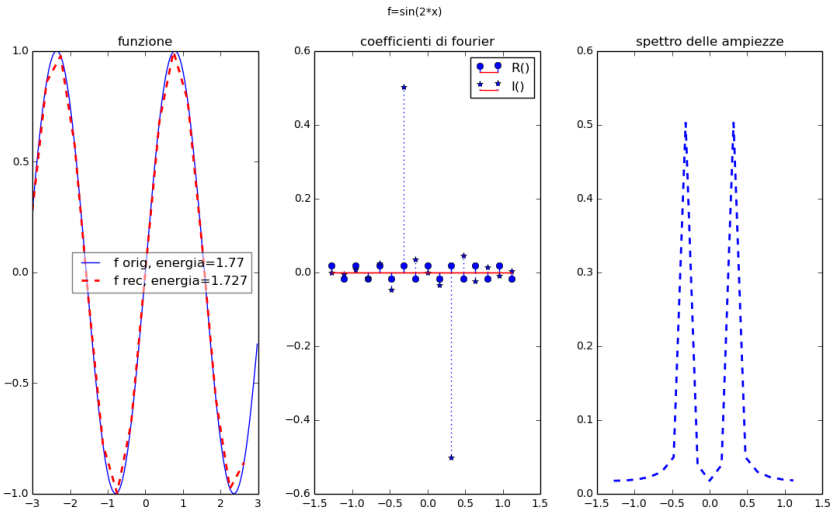

Figura 4.7: Nella Figura a sinistra è mostrata in rosso la ricostruzione della funzione seno in blu campionata in 16 punti. Nella Figura centrale sono mostrati i coefficienti di Fourier (con la linea continua la loro parte reale, con linea tratteggiata la loro parte immaginaria), mentre la Figura a destra mostra lo spettro dei coefficienti. Si può notare che la parte reale dei coefficienti è nulla.

Listing 4.4: Programma, serie di fourier

```
from scipy.integrate import simps
import matplotlib.pyplot as plt
import numpy as np
from numpy.fft import *
from matplotlib import pyplot
import matplotlib.gridspec as gridspec
from matplotlib import rcParams

n=16 #numero di campioni da usare per il calcolo dei coefficienti
T = 2*np.pi #larghezza periodo
nat_order = True #true se i valori dell intervallo sono centrati in zero
def f0(t):
    return np.sin(2*t)+0.5*np.cos(5*t)
function_name = 'f(x) = np.sin(2*t)+0.5*np.cos(5*t)'
fs_data = [[f0,n,T,nat_order,function_name]]

def f1(t):
    n = t.size
    return np.sin(2*t)+0.5*np.cos(5*t) + 0.5* np.random.normal(size=n)
function_name += " + noise"
fs_data += [[f1,n,T,nat_order,function_name]]

f2 = np.cos
function_name = "f=cos(x)"
fs_data += [[f2,n,T,nat_order,function_name]]

def f3(t):
    return np.sin(2*t)
function_name = "f=sin(2*x)"
fs_data += [[f3,n,T,nat_order,function_name]]

def plot_dft(f,n,T,nat_order,function_name=None): #y e' la funzione vera
    width = 13
    height = 7
    n_lrg = 256 #numero di campioni da usare per plottare la funzione vera
    if nat_order == True:
        L=int(T/2.0)
        t = np.linspace(-L, L, num=n, endpoint=False)
        t_lrg = np.linspace(-L, L, num=n_lrg, endpoint=False)
    else:
        t = np.linspace(0, n, num=n, endpoint=False)
        t_lrg = np.linspace(0, n, num=n_lrg, endpoint=False)
    y = f(t)
    y_lrg = f(t_lrg)
    fig = plt.figure(figsize=(width, height))
    if function_name != None:
        fig.suptitle(function_name)
    g_s = gridspec.GridSpec(1,3)
    sp_func = fig.add_subplot(g_s[0, 0])
    sp_func.set_title("funzione" )
    sp_spettro = fig.add_subplot(g_s[0, 2]) #plottiamo la trasformata
    sp_spettro.set_title("spettro delle ampiezze")
    sp_coeff = fig.add_subplot(g_s[0, 1]) #qui plottiamo le parti reali dei coefficienti
    sp_coeff.set_title("coefficienti di fourier" )
    if nat_order == False :
        ck = 1.0/n * fftshift(fft(y)) #shiftiamo per avere la trasformata con il bias (a0)centrato
        w = fftshift(fftfreq(len(y) ,d=T/(n+0.0)))
        rec_y = n * ifft(ifftshift(ck))
    else :
        ck = 1.0/n * fftshift(fft(ifftshift(y))) #se la funzione e' centrata in zero dobbiamo prima usare l inverse shift
        w = fftshift(fftfreq(len(y),d=T/(n+0.0)))
        rec_y = n * fftshift( ifft(ifftshift(ck)))
    sp_coeff.stem(w, np.real(ck), 'b', linewidth=2.0, linestyle='--', label='R()')
    sp_coeff.stem(w, np.imag(ck), 'b:', markerfmt='*', label='I()')
    sp_coeff.legend()
    sp_spettro.plot(w, np.abs(ck), 'b', linewidth=2.0, linestyle='--', label='spettro')
    print "la frequenza con la massima ampliezza e ",w[np.argmax(np.abs(ck))]
    print "controlliamo che la disuguaglianza di Bessel sia rispettata (l'energia dal polinomio approssimante deve
        essere minore di quella del polinomio approssimato"
    energia_orig_f = np.sqrt (simps( np.power(abs(y_lrg),2) , t_lrg))
    print "l area sottesa dal polinomio approssimato e' di ",energia_orig_f
    energia_rec_f = np.sqrt (simps( np.power(abs(y),2) , t))
    print "l area sottesa dal polinomio approssimante e' di ",energia_rec_f
    sp_func.plot(t_lrg, y_lrg, 'b', label='f orig, energia='+str(np.round(energia_orig_f,3))) #plottiamo la funzione
        vera
    sp_func.plot(t, rec_y, 'r', linewidth=2.0, linestyle='--', label='f rec, energia='+str(np.round(energia_rec_f,3))
        ) #rec_y e' la funzione ricostruita
    sp_func.legend(loc='best',framealpha=0.5)
    g_s.update(wspace=0.25, hspace=0.3)
    rcParams.update({'font.size': 10, 'legend.font.size':10})
    plt.show(block=True)

function_number =len(fs_data)
for fn in xrange(function_number):
    f,n,T,nat_order,function_name = fs_data[fn]
    plot_dft(f,n,T,nat_order,function_name)
```

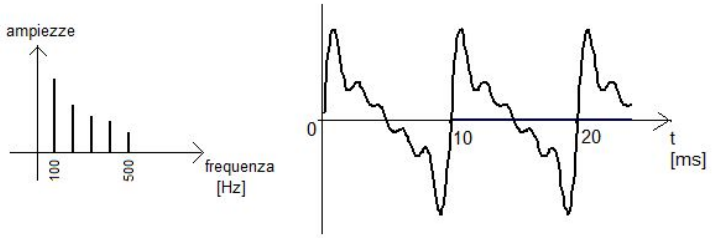

Figura 4.8: Esempio di funzione

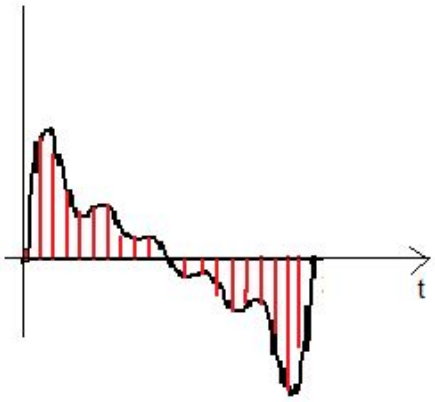

Figura 4.9: Esempio del campionamento da effettuare per evitare il fenomeno dell'aliasing.

4.3 Trasformata di Fourier

La trasformata di Fourier 2.22 può essere calcolata utilizzando la DFT. Infatti poiché la trasformata di Fourier serve per approssimare funzioni non periodiche abbiamo che:

$$\lim_{n \to \pm\infty} f(x) = 0 \tag{4.3}$$

quindi possiamo approssimare l'integrale troncando l'intervallo di integrazione:

$$\mathcal{F}(f) = \int_{-A/2}^{A/2} e^{-ikx} \cdot f(x)\, dx \tag{4.4}$$

A questo punto scegliendo un numero di intervalli pari ad n, si può discretizzare l'intervallo e calcolare l'integrale con la formula dei trapezi composta. Si può notare che la trasformata di Fourier è A volte la DFT che si può ottenere calcolando la DFT della funzione su n intervalli equispaziati. Notiamo anche che A è il massimo periodo possibile per l'onda e quindi $\frac{1}{A}$ è la più bassa frequenza ammissibile.

Il codice 4.5 è uno script per il plot dei grafici riguardanti la trasformata di Fourier. Nei grafici prodotti con questo script 4.10 e 4.11 in blu è sono state plottate la funzione originale e la vera trasformata, mentre in rosso quelle calcolate con le funzioni della libreria numpy.

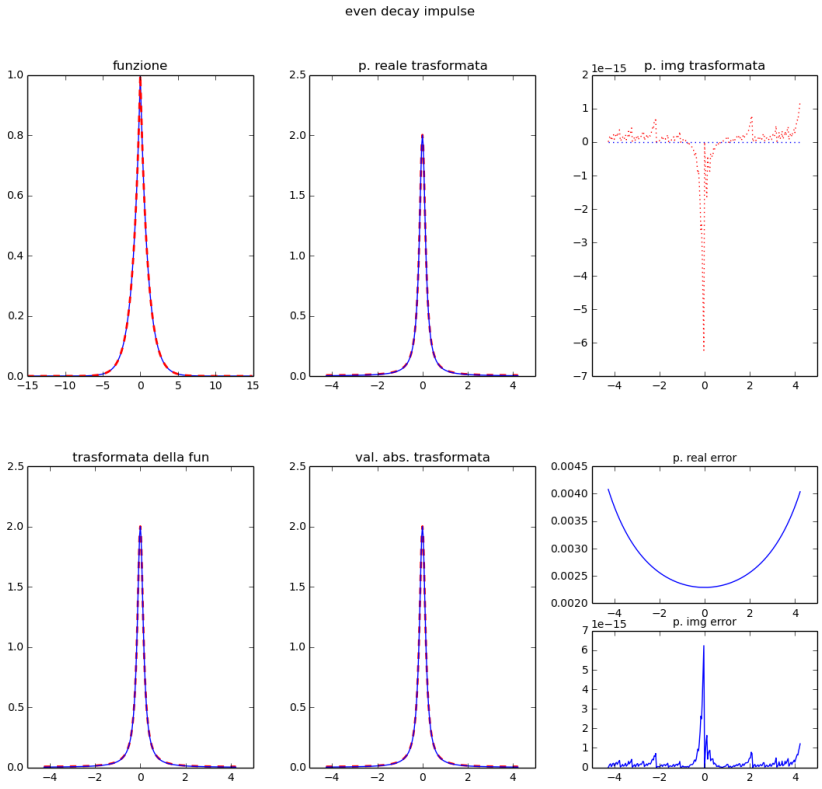

Figura 4.10: Nella figura in alto a sinistra è mostrata in blu la funzione originale e in rosso la sua ricostruzione a partire dal campionamento della funzione in 256 punti equi-spaziati. Negli altri grafici è mostrata la trasformata di Fourier della funzione (in blu quella reale calcolata analiticamente, in rosso quella calcolata a partire dal campionamento, utilizzando python)

Listing 4.5: Programma, trasformata

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.fft import *
from matplotlib import pyplot
import matplotlib.gridspec as gridspec
from matplotlib import rcParams

n=256 #numero di campioni
function_name ="even decay impulse"
def even_decay_true_trasf(w):
    return 2.0/(1+(2.0*np.pi*w)**2)
T = 30 #larghezza periodo
L=int(T/2.0)
t = np.linspace(-L, L, num=n, endpoint=False)
nat_order = True
y = np.exp(-abs(t))
fs_data = [[t,y,n,T,nat_order,function_name,even_decay_true_trasf]]

function_name ="odd decay impulse"
def odd_decay_true_trasf(w):
    return -4.0j*np.pi*w/(1+(2.0*np.pi*w)**2)
T=8
L=int(T/2)
t = np.linspace(-L, L, num=n, endpoint=False)
y = np.exp(-t)
y[t<0] = -np.exp(t[t<0])
fs_data += [[t,y,n,T,nat_order,function_name,odd_decay_true_trasf]]

def plot_dft(t,y,n,T,nat_order,function_name=None,func_true_trasf=None): #y e' la funzione vera #
      func_true_trasf e' la trasformata vera
    width = 13
    height = 15
    fig = plt.figure(figsize=(width, height))
    if function_name != None:
        fig.suptitle(function_name)
    g_s = gridspec.GridSpec(2,3)
    sp_func = fig.add_subplot(g_s[0, 0])
    sp_func.set_title("funzione" )
    sp_transf = fig.add_subplot(g_s[1, 0]) #plottiamo la trasformata
    sp_transf.set_title("trasformata della fun")
    sp_abs = fig.add_subplot(g_s[1, 1]) #qui plottiamo il valore assoluto dei coeff ricostruiti
    sp_abs.set_title("val. abs. trasformata" )
    sp_r = fig.add_subplot(g_s[0, 1]) #qui plottiamo le parti reali dei coefficienti
    sp_r.set_title("p. reale trasformata" )
    sp_i = fig.add_subplot(g_s[0, 2]) #qui plottiamo le parti immaginarie dei coefficienti
    sp_i.set_title("p. img trasformata" )
    sp_func.plot(t, y, 'b', label='f orig') #plottiamo la funzione vera
    if nat_order == False :
        trasf = (T+0.0)/n * fftshift(fft(y)) #shiftiamo per avere la trasformata con il bias (a0) centrato
        w = fftshift(fftfreq(len(y),d=T/(n+0.0)))
        rec_y = ( (n+0.0)/T ) * ifft(ifftshift(trasf))
    else :
        trasf = (T+0.0)/n * fftshift(fft(ifftshift(y))) #se la funzione e' centrata in zero dobbiamo prima usare l inverse
              shift
        w = fftshift(fftfreq(len(y),d=T/(n+0.0)))
        rec_y = ( (n+0.0)/T ) * fftshift( ifft(ifftshift(trasf)))
    sp_transf.plot(w, trasf, 'r', linewidth=2.0, linestyle='--', label='computed')
    sp_r.plot(w, np.real(trasf), 'r', linewidth=2.0, linestyle='--', label='R()')
    sp_i.plot(w, np.imag(trasf), 'r:', label='I()')
    sp_abs.plot(w, np.abs(trasf), 'r', linewidth=2.0, linestyle='--', label='abs()')
    sp_func.plot(t, rec_y, 'r', linewidth=2.0, linestyle='--', label='f rec') #rec_y e' la funzione ricostruita
    if trasf != None:
        gs12 = gridspec.GridSpecFromSubplotSpec(2, 1, subplot_spec=g_s[1, 2]) #qui plottiamo gli errori di
              ricostruzione
        ax1 = plt.Subplot(fig, gs12[0, 0])
        sp_err_real = fig.add_subplot(ax1)
        ax1 = plt.Subplot(fig, gs12[1, 0])
        sp_err_imm = fig.add_subplot(ax1)
        true_trasf = func_true_trasf(w)
        sp_transf.plot(w, true_trasf, 'b', label='true')
        sp_r.plot(w, np.real(true_trasf), 'b', label='true_R()')
        sp_i.plot(w, np.imag(true_trasf), 'b:', label='true_I()')
        sp_abs.plot(w, np.abs(true_trasf), 'b', label='true_abs()')
        sp_err_real.plot(w, abs(np.real(trasf-true_trasf)), 'b')
        sp_err_imm.plot(w, abs(np.imag(trasf-true_trasf)), 'b')
        sp_err_real.set_title("p. real error",fontsize=10)
        sp_err_imm.set_title("p. img error",fontsize=10)
    g_s.update(wspace=0.25, hspace=0.3)
    rcParams.update({'font.size': 10, 'legend.font.size':10})
    plt.show(block=True)

function_number =len(fs_data)
for fn in xrange(function_number):
    t,y,n,T,nat_order,function_name,function_true_trasf = fs_data[fn]
    plot_dft(t,y,n,T,nat_order,function_name,function_true_trasf)
```

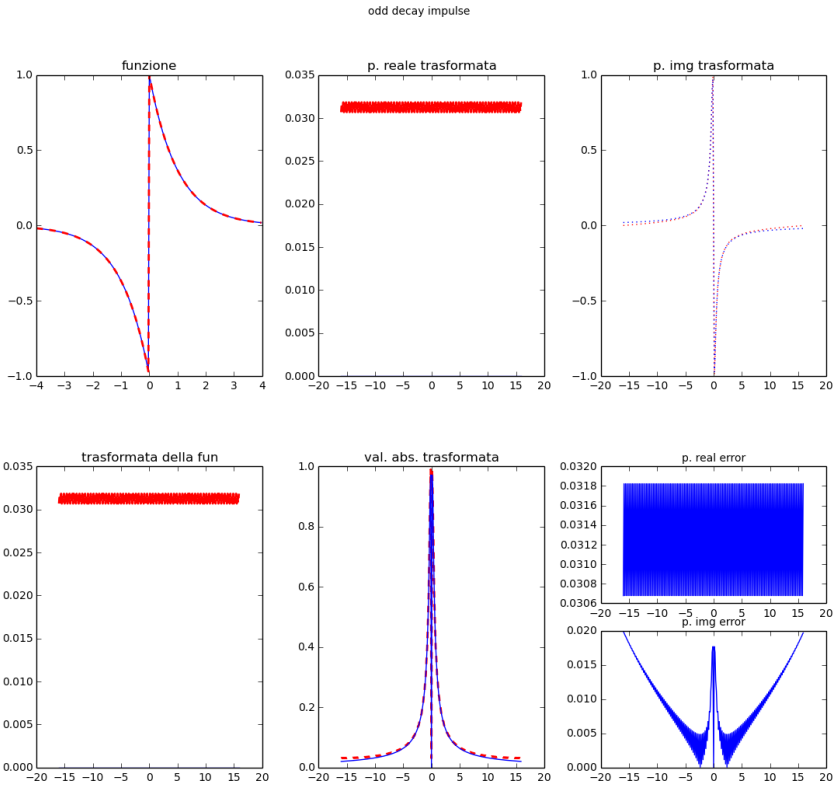

Figura 4.11: Nella figura in alto a sinistra è mostrata in blu la funzione originale e in rosso la sua ricostruzione a partire dal campionamento della funzione in 256 punti equi-spaziati. Negli altri grafici è mostrata la trasformata di Fourier della funzione (in blu quella reale calcolata analiticamente, in rosso quella calcolata a partire dal campionamento, utilizzando python)

4.4 2D Discrete Fourier Transform

Nel capitolo 3 abbiamo visto che la DFT può essere applicata anche a funzioni di due variabili e che viene spesso impiegata per l'analisi delle immagini. Come è stato precedentemente accennato, nel caso delle immagini vengono analizzati modulo e fase dei coefficienti di fourier.

Listing 4.6: Programma, trasformata 2D plot

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np
from numpy.fft import *
from matplotlib import pyplot
import matplotlib.gridspec as gridspec
from matplotlib import rcParams
import matplotlib.image as mpimg
from skimage import color
import os
from kivy.graphics.context_instructions import LineWidth
img_folder_path = os.path.dirname(os.path.abspath(__file__)) + "/imgs"
imgs=['img3.png','img4.png','img5.png','img6.png','img7.png','img8.png','img9.png'] #i nomi delle immagini nella
    cartella

def altera_trasformata(trasf):
    modulo = np.abs(trasf)
    fase = np.angle(trasf)
    trasf = np.nan_to_num(modulo * np.exp(1j*fase))
    return trasf,fase,modulo

def plot_dft(img,img_name=None):
    width = 13
    height = 15
    fig = plt.figure(figsize=(width, height))
    fig.suptitle("immagine non alterata",fontsize = 24)
    if img_name != None:
        fig.suptitle(img_name)
    g_s = gridspec.GridSpec(2,3)
    sp_img = fig.add_subplot(g_s[0, 0])
    imgplot = plt.imshow(img,plt.get_cmap('gray'))
    sp_img.set_title("immagine" )
    m,n = img.shape
    trasf = fftn(img) #shiftiamo per avere la trasformata con il bias (a0) centrato
    m_idx=np.linspace(0, m, 5, endpoint=True).astype(int)
    n_idx=np.linspace(0, n, 5, endpoint=True).astype(int)
    m_idx_shift = fftshift(m_idx)
    n_idx_shift = fftshift(n_idx)
    trasf,fase,modulo = altera_trasformata(trasf)
    X = np.arange(0, m, 1)
    Y = np.arange(0, n, 1)
    X, Y = np.meshgrid(X, Y)
    sp_modulo = fig.add_subplot(g_s[1, 1],projection='3d') #qui plottiamo il modulo dei coeff
    sp_modulo.set_title("modulo trasformata" )
    surf= sp_modulo.plot_surface(X, Y, fftshift(modulo))
    sp_modulo.set_xticklabels( (m_idx_shift).astype(str).tolist())
    sp_modulo.set_yticklabels( (n_idx_shift).astype(str).tolist())
    sp_fase = fig.add_subplot(g_s[1, 2],projection='3d') #qui plottiamo la fase di coeff
    sp_fase.set_title("fase trasformata" )
    sp_fase.plot_surface(X, Y, fftshift(fase), cmap=cm.coolwarm)
    sp_fase.set_xticklabels( (m_idx_shift).astype(str).tolist())
    sp_fase.set_yticklabels( (n_idx_shift).astype(str).tolist())
    sp_modulo = fig.add_subplot(g_s[0, 1],projection='3d') #qui plottiamo il modulo dei coeff
    sp_modulo.set_title("modulo ns transf." ) #ns sta per non shiftata
    sp_modulo.set_xticklabels(m_idx)
    sp_modulo.set_yticklabels(n_idx)
    sp_modulo.plot_surface(X, Y, modulo)
    sp_modulo.set_xticklabels( (m_idx).astype(str).tolist() )
    sp_modulo.set_yticklabels((n_idx).astype(str).tolist() )
    sp_fase = fig.add_subplot(g_s[0, 2],projection='3d') #qui plottiamo la fase di coeff
    sp_fase.set_title("fase ns transf." )
    sp_fase.set_xticklabels(m_idx)
    sp_fase.set_yticklabels(n_idx)
    sp_fase.plot_surface(X, Y, fase, cmap=cm.coolwarm)
    sp_fase.set_ylabel("y")
    sp_fase.set_xlabel("x")
    rec_y = ifftn(trasf)
    sp_shiftex_imm = fig.add_subplot(g_s[1, 0]) #immagine ricostruita
    sp_shiftex_imm.set_title("immagine ricostruita")
    imgplot = plt.imshow(rec_y.astype(np.float64),plt.get_cmap('gray'))
    fig.tight_layout(w_pad=0.5,h_pad=0.5)
    rcParams.update({'font.size': 10})
    plt.subplots_adjust(top=0.95)
    plt.show(block=True)

img_number =len(imgs)
for fn in xrange(img_number):
    path_immagine = img_folder_path+"/"+imgs[fn]
    immagine = color.rgb2gray(mpimg.imread(path_immagine)); #legge un immagine e la converte in bianco e nero
    plot_dft(immagine)
```

I plot seguenti sono stati realizzato utilizzando lo script python 4.6. Nei plot della fase il colore rappresenta l'azze Z (il valore della fase), i toni freddi rappresentano valori più bassi mentre i toni caldi i valori più alti.
Quando diciamo che la trasformata è stata shiftata significa che la matrice dei coefficienti è stata alterata come mostrato in in modo che i coefficienti delle onde ad ampiezze maggiori si trovino al centro del grafico. Dai grafici prodotti è stato possibile osservare, per quanto riguarda lo **spettro**:
- Sappiamo che esso è proporzionale all'ampiezza dei coefficienti delle diverse frequenze. Dal confronto dei grafici nelle figure 4.12 e 4.13 possiamo vedere come nel caso di immagini che contengono cambiamenti soft dei toni di grigio, come la prima citata, le onde presenti siano quelle con frequenze più basse, mentre nel caso di immagini con cambiamenti più repentini, come la seconda citata, siano presenti anche frequenze più alte.

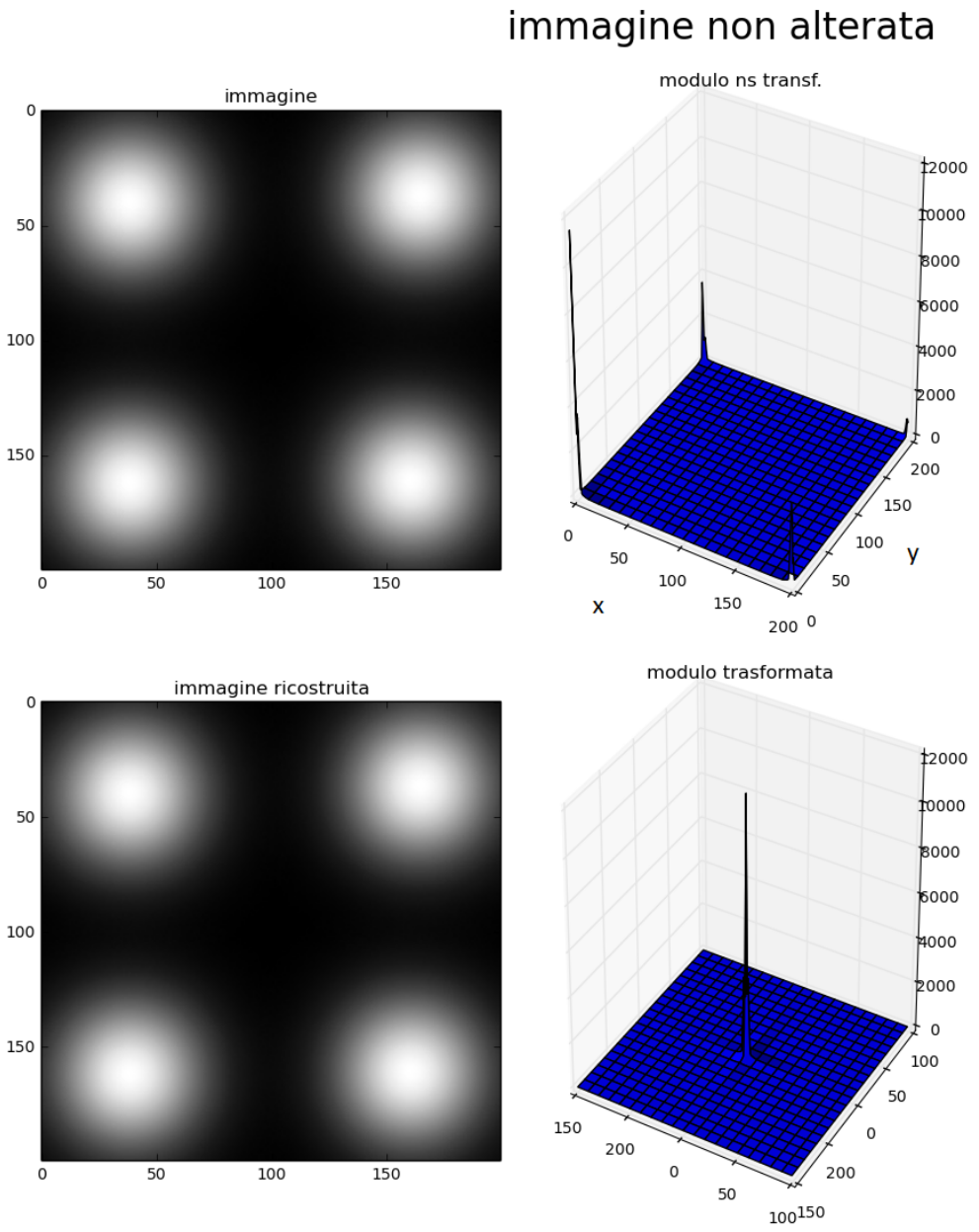

Figura 4.12: Nella figura in alto sono mostrati l'immagine originale e la sua trasformata. Nella riga in basso sono mostrati la figura ricostruita mediante la trasformata inversa ed il modulo della sua trasformata shiftato in modo che i coefficienti delle onde ad ampiezze maggiori si trovino al centro del grafico.

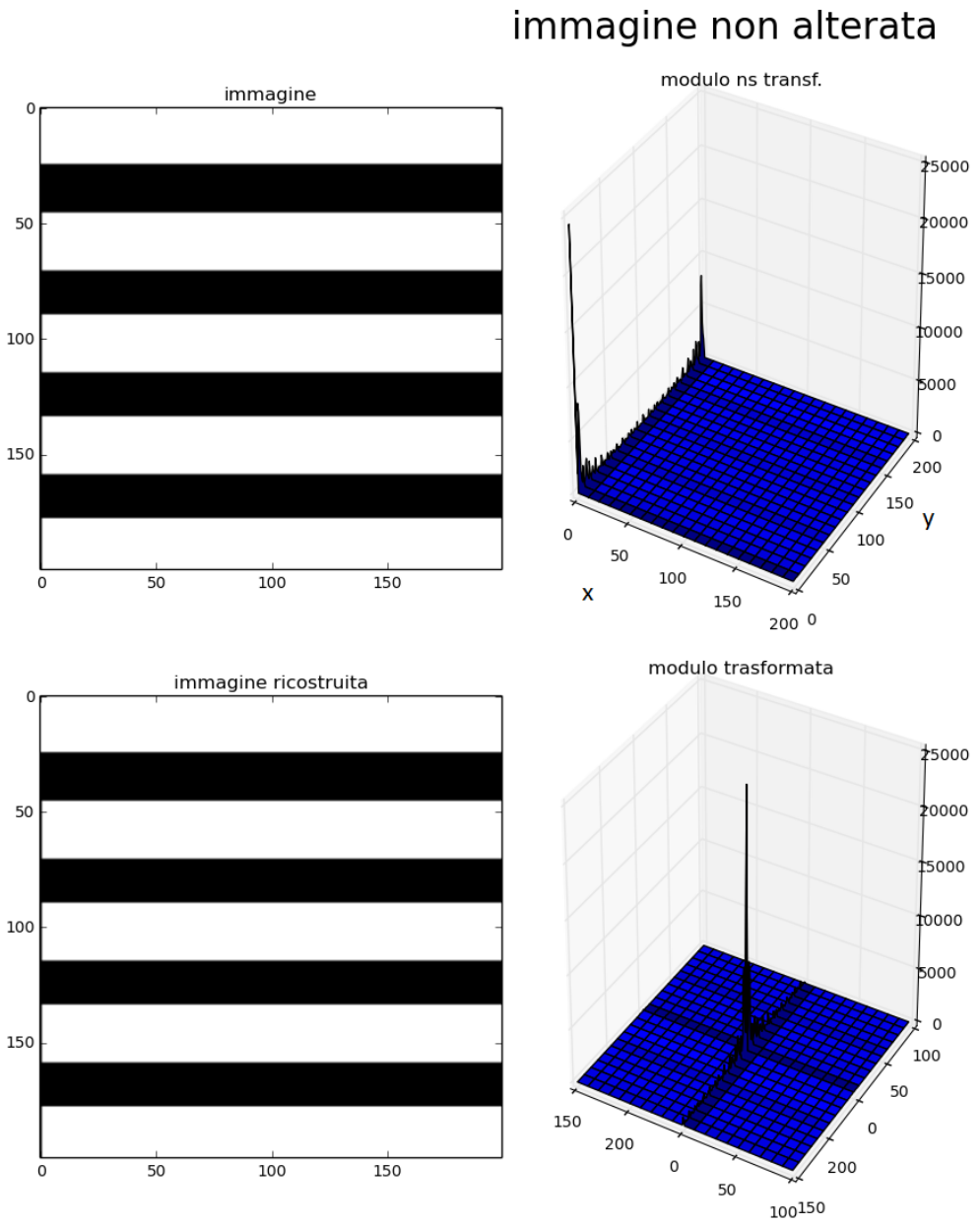

Figura 4.13: Nella figura in alto sono mostrati l'immagine originale e la sua trasformata. Nella riga in basso sono mostrati la figura ricostruita mediante la trasformata inversa ed il modulo della sua trasformata shiftato in modo che i coefficienti delle onde ad ampiezze maggiori si trovino al centro del grafico.

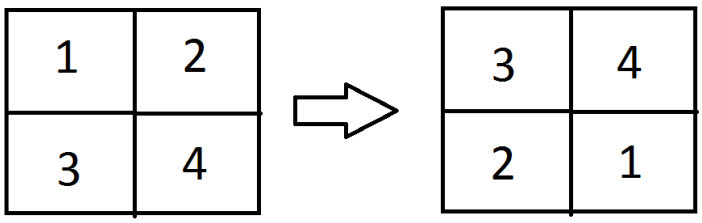

Figura 4.14: La figura mostra come vine effettuato lo shift della trasformata, i quadranti della matrice dei coefficienti vengono spostati in modo tale che le onde ad ampiezze maggiori si trovino al centro del grafico.

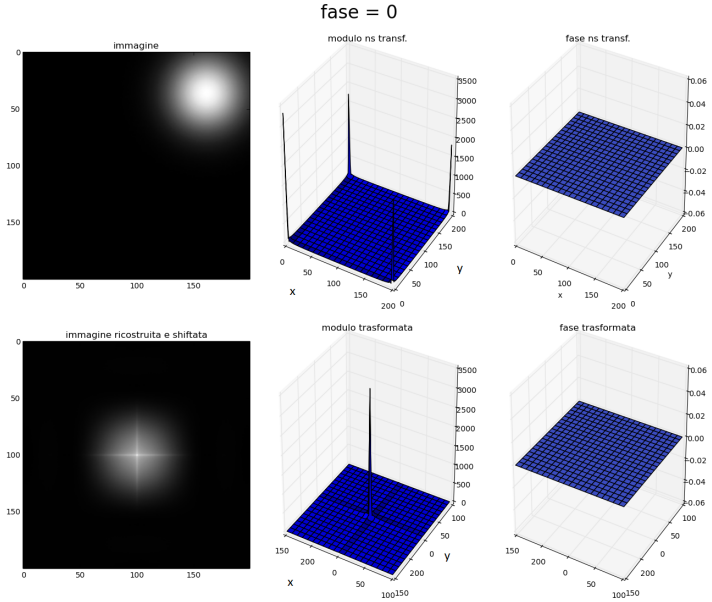

Figura 4.15: Nella figura in alto è mostrata l'immagine originale. Il modulo della sua trasformata e la sua fase (in basso replicati a seguito dello shift). In basso a sinistra è mostrata l'immagine ricostruita con la trasformata inversa. La trasformata è stata calcolata ponendo la fase dei suoi coefficienti uguale a zero in modo da mettere in evidenza solo il suo spettro

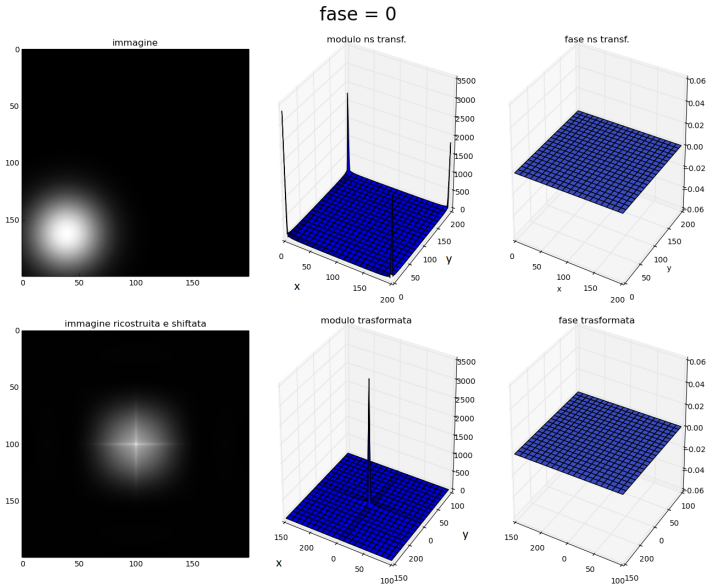

Figura 4.16: Nella figura in alto è mostrata l'immagine originale. Il modulo della sua trasformata e la sua fase (in basso replicati a seguito dello shift). In basso a sinistra è mostrata l'immagine ricostruita con la trasformata inversa. La trasformata è stata calcolata ponendo la fase dei suoi coefficienti uguale a zero in modo da mettere in evidenza solo il suo spettro

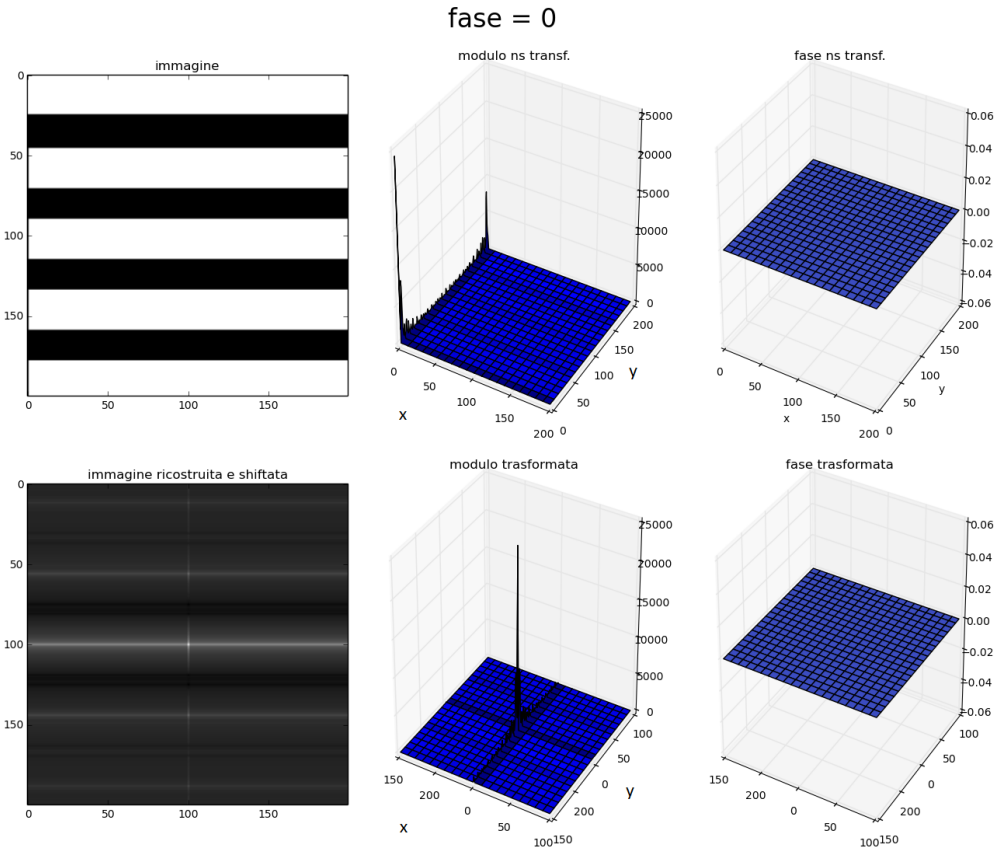

Figura 4.17: Nella figura in alto è mostrata l'immagine originale. Il modulo della sua trasformata e la sua fase (in basso replicati a seguito dello shift). In basso a sinistra è mostrata l'immagine ricostruita con la trasformata inversa. La trasformata è stata calcolata ponendo la fase dei suoi coefficienti uguale a zero in modo da mettere in evidenza solo il suo spettro

- L'ampiezza è parzialmente dipendente dalla posizione delle zone chiare nell'immagine. Infatti osservando le immagini 4.15 e 4.16 possiamo vedere che nonostante la sfera luminosa si trovi in due zone diverse dell'immagine il modulo risulta identico. Osservando le immagini 4.17 e 4.18 tuttavia possiamo notare che lo spettro non è del tutto indipendente dalla posizione. Mentre l'energia complessiva risulta la stessa, risultano differenti gli indici dei coefficienti che la posseggono. L'immagine 4.17 presenta delle righe orizzontali, i coefficienti che contengono l'energia sono infatti quelli corrispondenti alle onde verticali, che sommate formeranno le righe nell'immagine. La riga bianca orizzontale al centro dello spettro è dovuta al fatto che in quel punto alle altre onde viene sommata anche quella ad ampiezza maggiore (quella con frequenza minore) pertanto la somma delle ampiezze delle onde assume in quel punto un valore più alto. Nelle immagini precedenti non si notava al-

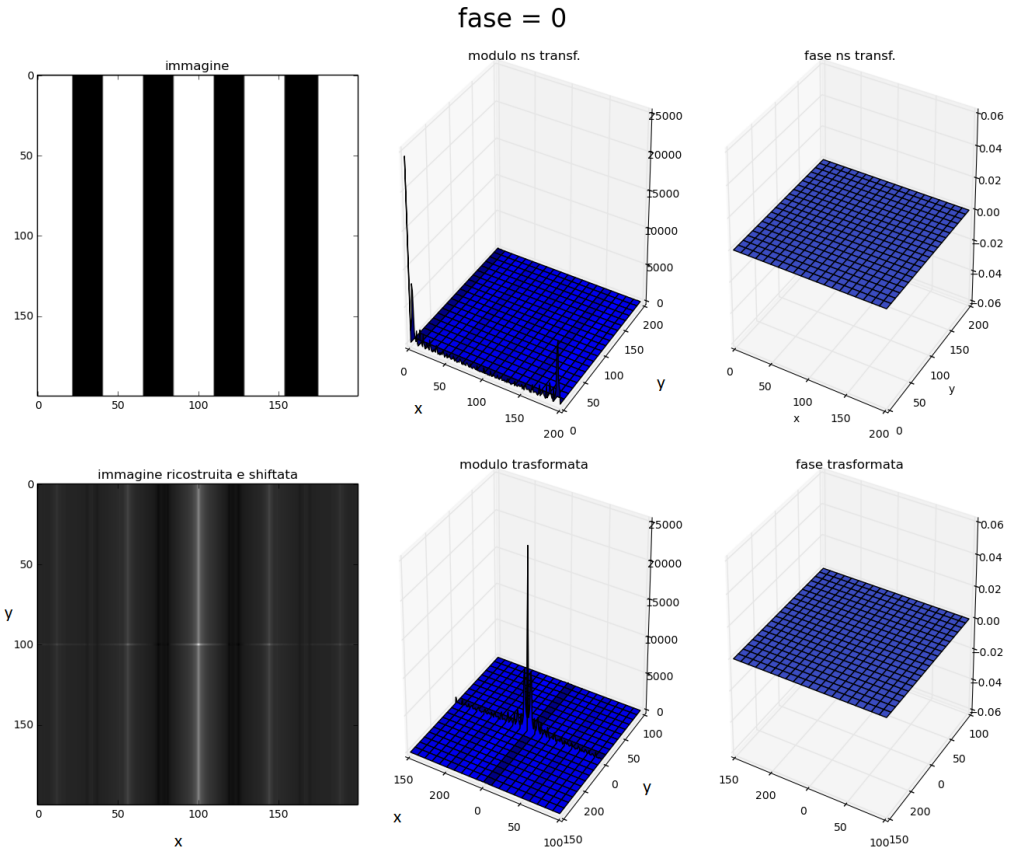

Figura 4.18: Nella Figura in alto è mostrata l'immagine originale. Il modulo della sua trasformata e la sua fase (in basso replicati a seguito dello shift). In basso a sinistra è mostrata l'immagine ricostruita con la trasformata inversa. La trasformata è stata calcolata ponendo la fase dei suoi coefficienti uguale a zero in modo da mettere in evidenza solo il suo spettro

cuna differenza perché l'ampiezza delle onde che partono dall'asse x e quelle che partono dall'asse y risultava simmetrica (solo sfasata).

Ricapitolando lo spettro fornisce l'informazione relativa ad ampiezza e frequenza delle onde che partono dall'asse x e di quelle che partono dall'asse y.

Per quanto riguarda la **fase**:
- Come possiamo vedere nelle figure 4.19, 4.20, 4.21, essa dipende dalla posizione delle zone chiare nell'immagine (del resto ci dice da dove partono le onde alle diverse frequenze).
- Tuttavia come possiamo notare da 4.22 da essa dipende anche l'intensità del colore dell'immagine (poiché se spostiamo alcune onde alteriamo il risultato della loro somma).

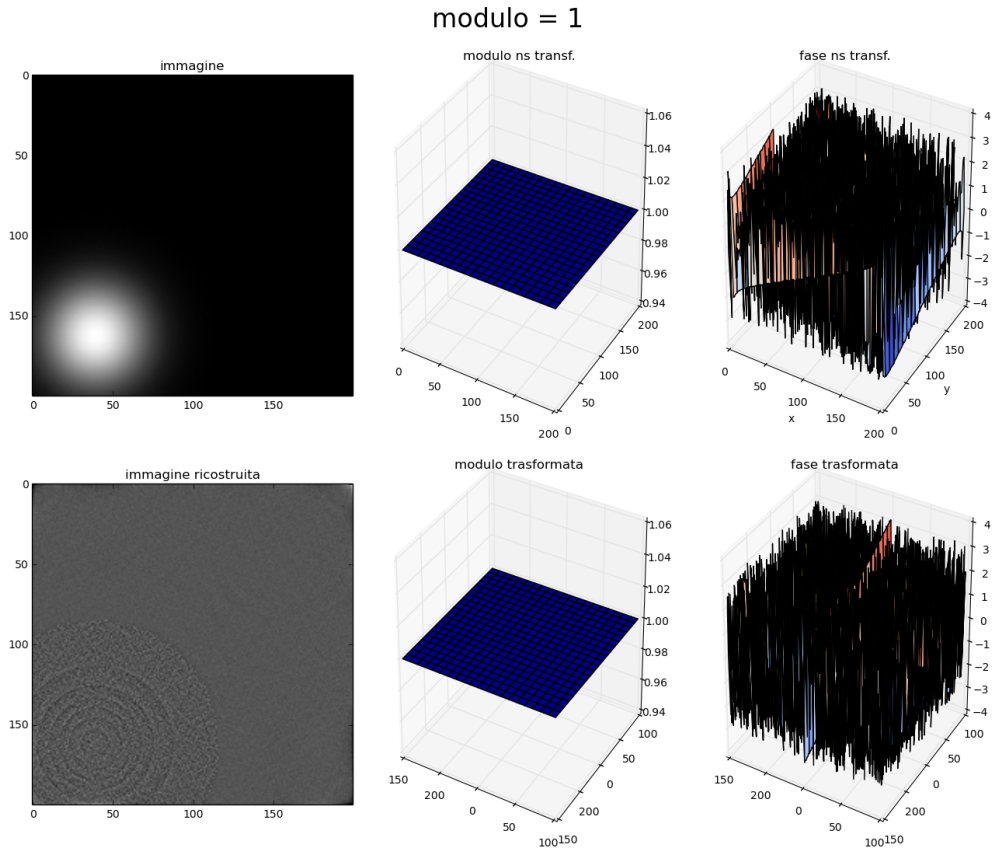

Figura 4.19: Nella figura in alto è mostrata l'immagine originale. Il modulo della sua trasformata e la sua fase (in basso replicati a seguito dello shift). In basso a sinistra è mostrata l'immagine ricostruita con la trasformata inversa e shiftata. La trasformata è stata calcolata ponendo il modulo dei suoi coefficienti uguale a uno in modo da mettere in evidenza solo la sua fase

In conclusione dagli esperimenti sulle immagini proposti possiamo vedere che in 2D lo spettro dei coefficienti corrisponde alle ampiezze di una serie di onde verticali e di una serie di onde orizzontali che sommate generano l'immagine.
Mentre dalla fase dipende una sorta di shift spaziale nell'asse x e nell'asse y delle onde.

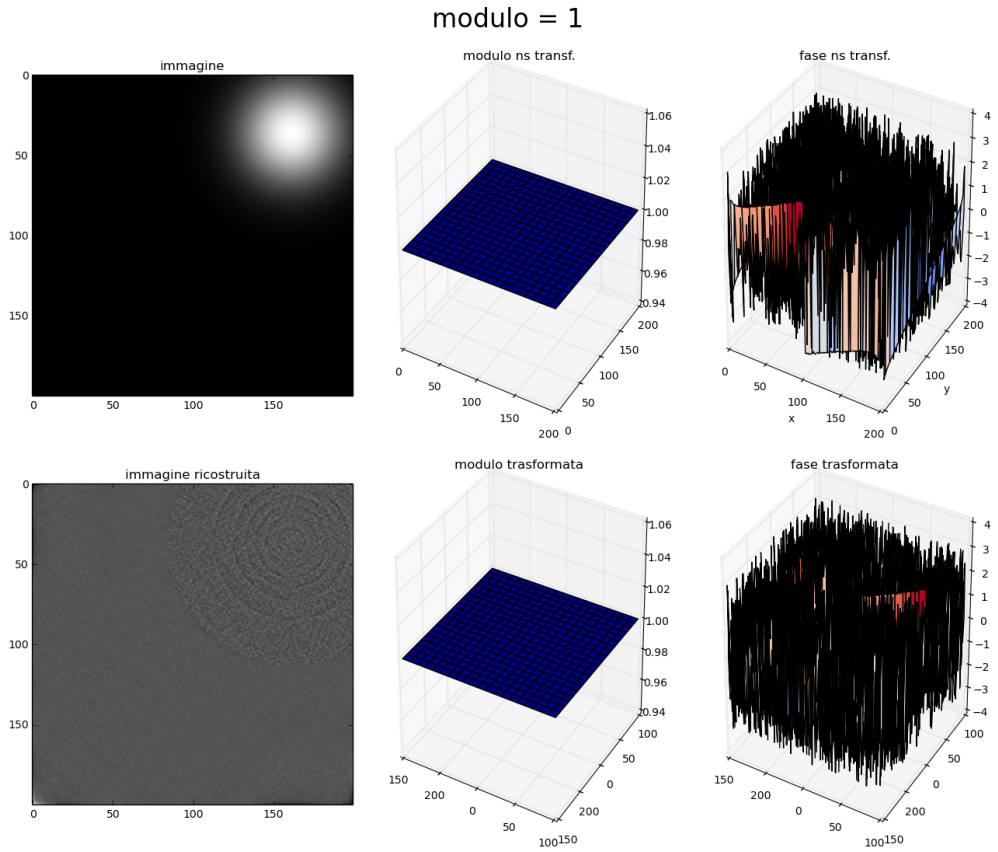

Figura 4.20: Nella figura in alto è mostrata l'immagine originale. Il modulo della sua trasformata e la sua fase (in basso replicati a seguito dello shift). In basso a sinistra è mostrata l'immagine ricostruita con la trasformata inversa. La trasformata è stata calcolata ponendo il modulo dei suoi coefficienti uguale a uno in modo da mettere in evidenza solo la sua fase

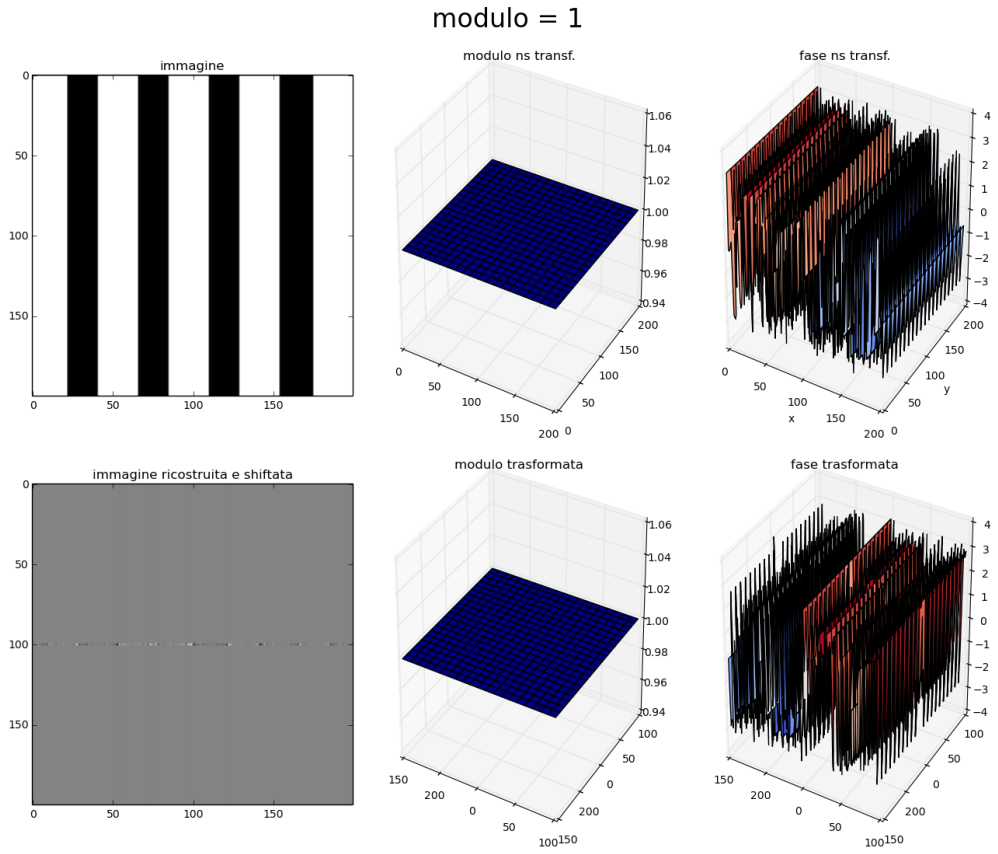

Figura 4.21: Nella figura in alto è mostrata l'immagine originale. Il modulo della sua trasformata e la sua fase (in basso replicati a seguito dello shift). In basso a sinistra è mostrata l'immagine ricostruita con la trasformata inversa. La trasformata è stata calcolata ponendo il modulo dei suoi coefficienti uguale a uno in modo da mettere in evidenza solo la sua fase

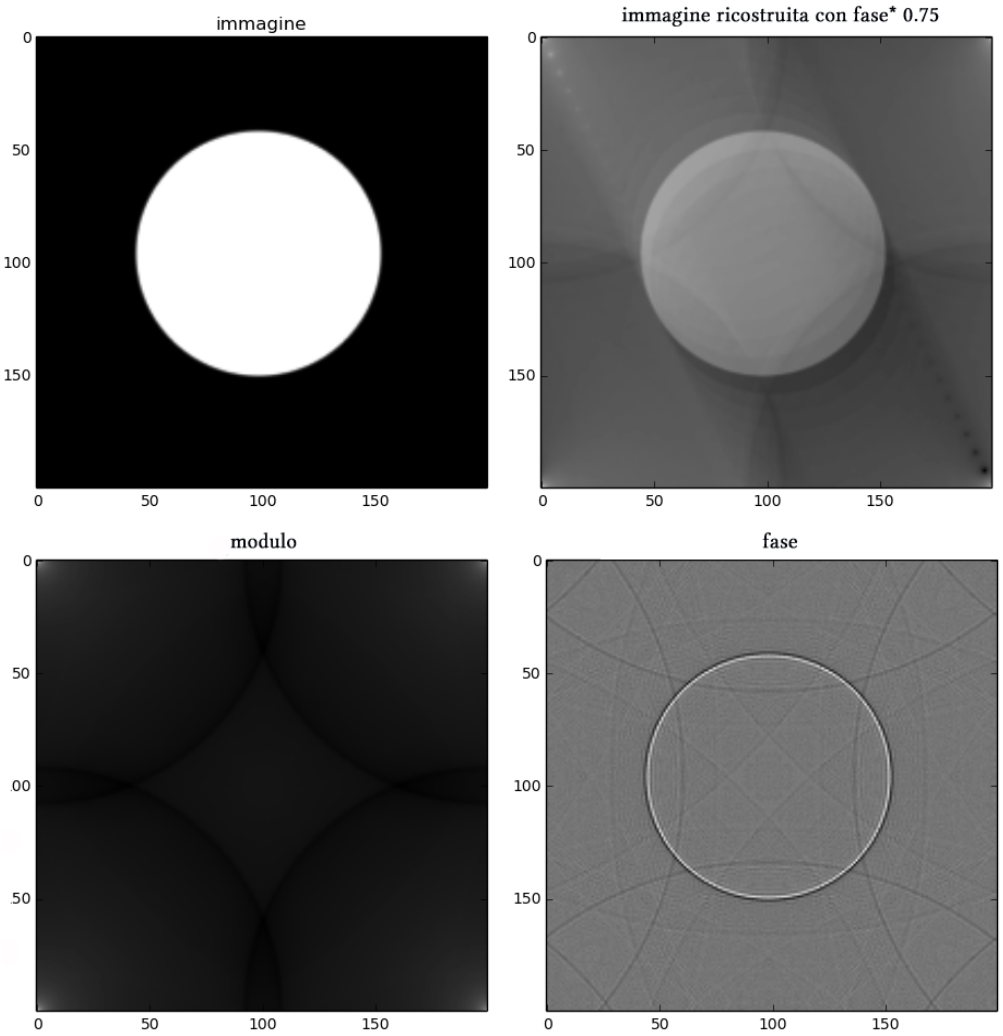

Figura 4.22: Nella figura in alto a sinistra è mostrata l'immagine originale. In alto a destra l'immagine ricostruita riducendo la fase della trasformata (moltiplicandola per 0.75). In basso è mostrata sempre l'immagine ricostruita alterando la sua trasformata: a sinistra ponendo la fase dei suoi coefficienti uguale a zero in modo da evidenziarne il modulo e a destra ponendo il modulo uguale ad uno in modo da evidenziarne la fase.

Bibliografia

[1] Sebastiano Battiato. Filtraggio nel dominio della frequenza.

[2] Sergio Invernizzi. Applicazioni matlab dft.

[3] Giancarlo Perlo. elemania.

[4] Giuseppe Rodriguez. Appunti sulla dft Rodriguez.