APPUNTI dall'Esercitazione su MATLABTM Tenuta dal Prof. Zanzi Politecnico di Milano, 07 - Aprile - 2000

ESEMPIO # 1 - Introduzione; gli operatori .* .^ ./ ; la FFT; comandi per la grafica

ESEMPIO # 2 - Generazione di un rumore bianco

ESEMPIO #3 - Un esempio di filtro a fase lineare

ESEMPIO #4 - Filtri FIR; utilizzo dei comandi fir1, filter, filtfilt

ESEMPIO # 5 - Matrici e vettori: il comando reshape; la convoluzione 2D

ESEMPIO # 6 - Introduzione al 2D: comando meshgrid e comandi per la grafica

ESEMPIO #7 - FFT2; somma di sinusoidi

Autore: Nicola Alagia, lucnik@tiscalinet.it

Nota: Gli appunti non sono stati revisionati da alcun professore.

Iniziamo costruendo un vettore di numeri; in Matlab c'e' un modo comodo per fare questo. Supponiamo di voler costruire un vettore 'v' di 10 numeri ordinati da 1 a 10; il comando e':

```
 v=1:10
```

La variabile 'v' contiene quindi i seguenti elementi

Si noti che Matlab usa come passo predefinito il numero 1; se ne vogliamo usare un altro dobbiamo indicarlo esplicitamente. Ecco un esempio con passo -2 ed un altro con passo 0.5:

```
» v=10:-2:0
v =
        10 8 6 4 2 0

» v=1:0.5:3;
```

Per snellire la scrittura, al posto di 0.5 si puo' indicare semplicemente .5 (v=1:.5:3);

Se alla fine dell'istruzione Matlab viene usato il punto e virgola ';' il risultato dell'operazione non viene visualizzato a schermo; questo e' ovviamente fondamentale quando si usano vettori e matrici di grandi dimensioni perche' evita notevoli perdite di tempo.

Se si vuole mettere un commento dopo un'istruzione bisogna usare il carattere '%'; Matlab ignora il contenuto di tutta la riga dopo il suddetto carattere:

Quando si lavora con matrici e vettori il risultato di una moltiplicazione (righe per colonne) dipende anche dall'ordine degli operandi; ad esempio, indicando con $A_{(m,n)}$ una matrice con m righe ed n colonne, con $V_{(1,n)}$ un vettore riga e con $V_{(n,1)}$ un vettore colonna, e' noto che:

```
\begin{array}{ll} A_{(3,2)}*B_{(2,4)}=C_{(3,4)} & \text{ (il risultato e' una matrice di 3 righe e 4 colonne)} \\ B_{(2,4)}*A_{(3,2)}=?? & \text{ (la moltiplicazione non e' definita)} \\ V1_{(1,n)}*V2_{(n,1)}=k & \text{ (il risultato e' uno scalare)} \\ V2_{(n,1)}*V1_{(1,n)}=D_{(n,n)} & \text{ (il risultato e' una matrice n x n)} \end{array}
```

In Matlab esiste un altro tipo di moltiplicazione (moltiplicazione di array) che, presi due vettori $V_{(1,n)}$ moltiplica elemento per elemento e restituisce come risultato ancora un vettore del tipo $V_{(1,n)}$; per fare questo bisogna usare l'operatore '.*', cioe':

```
V1_{(1,n)}.* V2_{(1,n)} = V3_{(1,n)} (il risultato e' un vettore)
```

Un esempio numerico riesce a chiarire subito; digitiamo al prompt di Matlab i seguenti comandi:

Esistono altri due operatori speciali che sono '.^' e './', cioe' elevamento a potenza e divisione eseguiti elemento per elemento; continuando l'esempio con i due vettori v1 e v2:

Ovviamente, poiche' un vettore e' semplicemente un particolare tipo di matrice, gli operatori '.*', '.^' e './' sono applicabili a matrici delle stesse dimensioni e svolgono l'operazione elemento per elemento. In particolare, detta A una matrice quadrata, possiamo elevarla al quadrato con l'istruzione B=A*A oppure costruire una matrice C che abbia come elementi i quadrati degli elementi di A digitando C=A.*A

Se vogliamo generare un segnale casuale, abbiamo a disposizione due comandi:

rand(1,n) costruisce una matrice (1 x n), cioe' un vettore riga, di numeri casuali distribuiti uniformemente tra 0 e 1; randn(1,n) analogamente costruisce una matrice (1 x n) di numeri casuali distribuiti normalmente con media nulla e varianza unitaria.

Ad esempio, se si vuole generare un vettore casuale di sei elementi distribuito normalmente con media 1 e varianza 4 il comando da utilizzare e'

```
» v=1+2*randn(1,6)
v =
    -0.3930    4.3923    1.1181    4.5941    1.5281    2.7433
```

Prima di incominciare l'esercizio elenchiamo tre comandi che useremo nel seguito: il primo e' whos e serve per sapere quali sono le variabili disponibili in memoria, di che tipo sono, quanto occupano in termini di bytes e se sono di tipo reale o complesso; il secondo comando e' clear, il quale cancella tutte le variabili dalla memoria (mentre il comando clear variabile cancella solo la variabile specificata); l'ultimo comando e' clc e serve per pulire lo schermo (nota che non ha alcun effetto sulle variabili, cioe' non le cancella!)

whos

Name	Size	Elements	Bytes	Density	Complex
ans	1 by 6	6	48	Full	No
V	1 by 6	6	48	Full	No

Vediamo un primo esempio del corso di Elaborazione Numerica dei Segnali: la trasformata di Fourier ed alcune sue proprieta' alle quali e' bene prestare particolare attenzione.

Per i nostri scopi, abbiamo bisogno di un vettore di 16 elementi distribuito normalmente, ~N(0,1)

```
» clc; clear;
» p=randn(1,16);
```

Ne facciamo ora la trasformata di Fourier. Nota che Matlab usa ovviamente l'algoritmo veloce, cioe' la fft; quindi, se la lunghezza del vettore e' una potenza di 2, l'algoritmo risulta piu' veloce.

```
» P=fft(p);
```

Per visualizzare con dei grafici quello che stiamo facendo, usiamo un po' di nuovi comandi.

```
subplot(m,n,x) serve per dividere lo schermo in una matrice di m righe e n colonne in modo da poter disegnare piu' grafici contemporaneamente; il parametro x serve per indicare quindi in quale parte dello schermo vogliamo disegnare (nota che spesso i comandi sono piu' facili da usare che da tentare di spiegare !!!). Chi preferisce puo' usare il comando subplot anche senza le virgole, cioe' subplot(mnx)o addirittura anche senza parentesi, cioe' subplot mnx;

stem (v) traccia il grafico discreto dei valori contenuti nel vettore v

plot (v) simile a stem, ma interpola linearmente i valori tra un campione e l'altro

grid serve per disegnare la griglia nell'ultimo grafico visualizzato

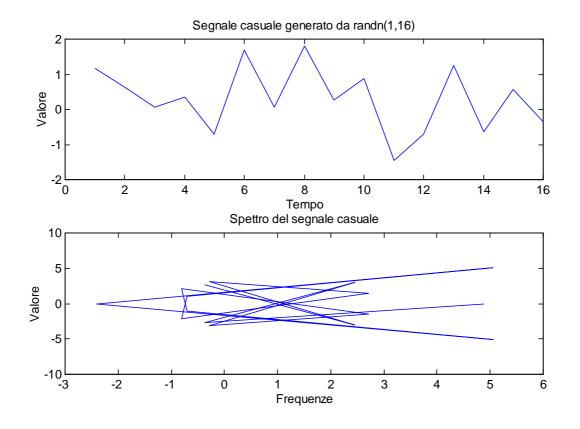
title('titolo del grafico');

xlabel('etichetta asse delle ascisse');

ylabel('etichetta asse delle ordinate');
```

Torniamo al nostro esempio e proviamo a dare i seguenti comandi:

Riportiamo qui sotto i grafici generati da Matlab e notiamo che qualcosa non ha funzionato!



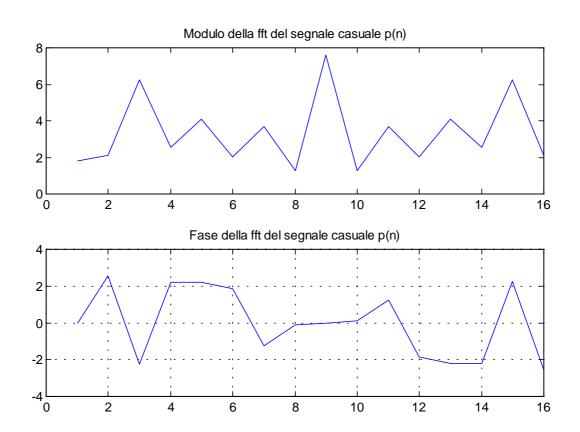
Quale errore abbiamo commesso?

Ci siamo dimenticati che la trasformata di Fourier di un segnale e', a priori, un vettore di numeri complessi e Matlab lo considera come tale. Per questo motivo il comando plot non e' adatto: ci sarebbe bisogno di un grafico in 3 dimensioni! Solitamente, come ben sappiamo, si preferisce rimanere nelle piu' comode 2 dimensioni e disegnare soltanto una parte delle informazioni contenute nel vettore complesso P.

Per fare cio' abbiamo a disposizione i 4 seguenti comandi :

```
abs(P) restituisce il valore assoluto del vettore;
angle(P) restituisce la fase;
real(P) restituisce la parte reale;
imag(P) restituisce la parte immaginaria.
```

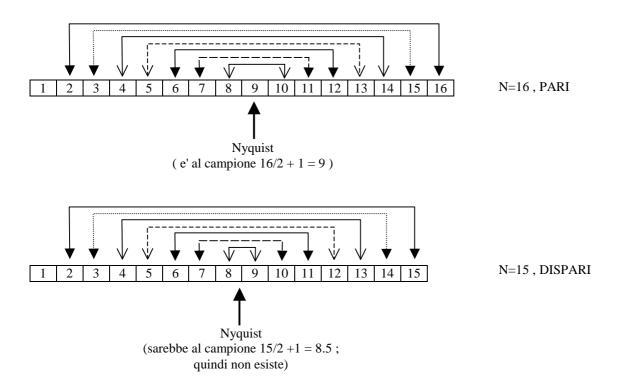
Decidiamo quindi di rappresentare su un nuovo grafico (grazie al comando figure) il modulo e la fase della fft del nostro segnale casuale.



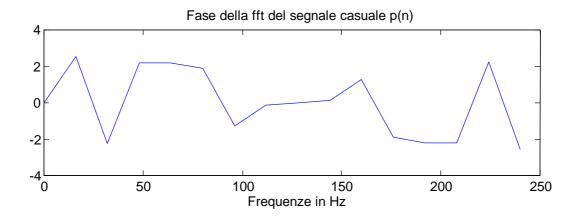
Grazie a questo ultimo plot, possiamo fare delle osservazioni (sperando di evitare errori ricorrenti nei temi d'esame):

- gli indici dei vettori in Matlab incominciano <u>SEMPRE da 1</u> e mai da 0;
- la componente continua e' quindi il campione di posto 1 (visto che quello di posto 0 non esiste);
- il campione al Nyquist occupa la posizione (N/2 + 1) dove N e' ovviamente la lunghezza del vettore. Nel nostro esempio il Nyquist e' quindi il campione numero 9;
- <u>attenzione alle simmetrie!</u> Visto che il segnale nel tempo p(n) era <u>reale</u>, la sua trasformata deve avere <u>modulo</u> <u>con simmetria pari</u> e <u>fase con simmetria dispari</u>; questo rispetto sia alla componente continua che al Nyquist.
 Sempre per questo motivo i <u>campioni 1 e 9 devono essere reali</u> quindi la loro fase deve essere nulla oppure ±π.

Per insistere ancora un minuto sull'importanza delle simmetrie soffermiamoci sui seguenti schemi dove si mettono in evidenza, tramite frecce, i campioni che godono della simmetria complesso coniugata quando si trasforma una sequenza reale. Sono da notare infine due cose: la prima e' che il campione 1 non e' simmetrico con nessuno, mentre la seconda e' che il Nyquist esiste solo se il vettore e' di lunghezza pari (quando esiste, il Nyquist, non deve soddisfare alcuna simmetria). Continua e Nyquist sono sempre reali.



Per completare questo primo esempio vogliamo migliorare ancora la rappresentazione grafica della fft. Supponiamo quindi che il segnale casuale p(n) sia stato generato campionando un segnale analogico con una frequenza di campionamento fc=256 Hz. Questo vuol dire che l'asse delle frequenze va da 0Hz a 256Hz (dopo di che il tutto si ripete periodicamente) e che i 16 campioni della fft saranno quindi spaziati di un df=256/16 Hz. Definiamo quindi il vettore delle frequenze e ridisegnamo la fase della fft del nostro segnale casuale.



Note sulla definizione dell'asse delle frequenze:

- dobbiamo ricordarci sempre che tanto il segnale nel tempo p(n), quanto la sua trasformata P(k)=FFT(p(n)), sono sottointese essere due sequenze periodiche. Per questo motivo la frequenza a 256 Hz viene esclusa: e' infatti del tutto equivalente alla frequenza 0 (256Hz e' cioe' ancora la componente continua);
- l'aver definito il vettore f a partire da zero non ha nulla a che vedere con il discorso fatto in precedenza quando abbiamo sottolineato che l'indice di un vettore parte sempre da 1.

Se vogliamo infatti sapere quanto vale il modulo della fft alla frequenza di Nyquist, qual e' il comando giusto? Possono venire in mente almeno due idee:

```
» P(128) % cioe' il campione a 128 Hz;
» P(9) % cioe' il campione numero 9
```

Quella esatta e' ovviamente la seconda perche' Matlab ragiona sempre in campioni, anche se noi abbiamo definito un asse delle frequenze! La risposta alla nostra richiesta e' quindi:

Quest'ultimo output di Matlab (ans = 7.6027) ci permette di sottolineare un'ultima cosa prima di chiudere il primo esercizio: non avevamo detto che i campioni della fft sono da considerarsi sempre complessi? Non ci dovevamo percio' aspettare una risposta del tipo:

```
ans = 7.6027 + 0.0000i
```

In realta' noi sappiamo bene (e quindi anche Matlab lo sa) che se il segnale nel tempo e' reale, la sua fft ha i campioni alla continua ed al Nyquist anch'essi reali.

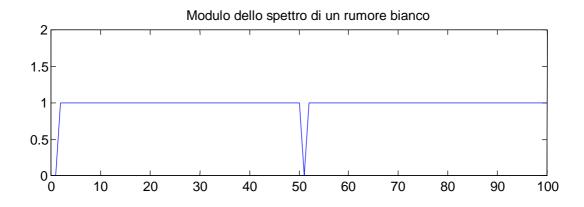
Come conferma, digitiamo 'P' al prompt di Matlab in modo da vederne il contenuto, notando che i campioni 1 e 9 sono numeri reali.

Prima di incominciare il secondo esercizio puliamo la memoria con il comando clear

Vogliamo generare un rumore casuale con spettro costante (cioe' bianco), lavorando nel dominio delle frequenze. Generiamo il modulo e la fase separatamente.

Costruiamo un vettore A lungo 100 elementi per il modulo e imponiamolo unitario a tutte le frequenze tranne che alla continua ed al Nyquist. Usiamo poi il comando plot per visualizzarlo e, poiche' l'output grafico ci piace poco, con il comando axis regoliamo gli intervalli di valori sugli assi X e Y.

Ricordiamo che in Matlab l'indice di un vettore parte sempre da 1, quindi la continua e' A(1), mentre il Nyquist e' A(N/2 + 1) = A(51). Il tutto e' evidenziato dalla figura riportata qui sotto.



Nota che il modulo, essendo costante, soddisfa la simmetria pari (condizione necessaria, ma non sufficiente, per avere il segnale reale nel tempo).

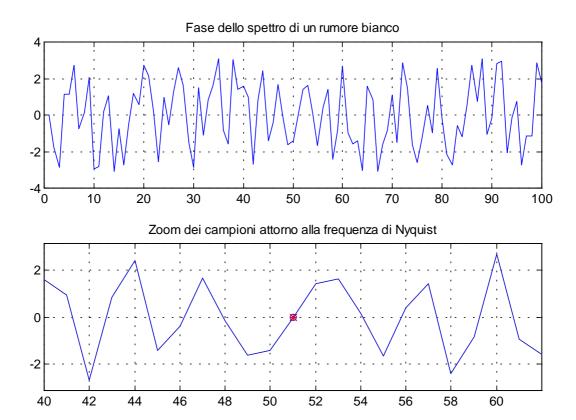
Dobbiamo adesso generare la fase (che in questo problema deve essere casuale), stando quindi attenti a soddisfare la simmetria dispari.

Un modo semplice e' generare la prima meta' dei campioni in modo casuale, forzare la continua e il Nyquist ad essere nulli ed infine forzare la simmetria dispari per la seconda meta' dei campioni.

$$X(k=0) = \sum_{n=1}^{N} x(n)$$
. Quindi, in generale, il valor medio nei tempi e' $\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x(n) = \frac{X(k=0)}{N}$

¹ Ricorda che, dalla formula di trasformazione della DFT, il campione alla componente continua e'

Dal grafico riportato si puo' notare che effettivamente abbiamo soddisfatto le richieste di simmetria.



Costruiamo infine lo spettro combinando modulo e fase.

```
» P=A.*exp(j*a);
```

In Matlab l'unita' immaginaria e' gia' definita, cioe' i=j=sqrt(-1), conviene quindi non usarle come variabili (nei cicli FOR conviene cioe' usare altri indici, come k, n, ecc.).

Da notare che nella moltiplicazione j*a non si e' usato il punto perche' j e' una costante, mentre in A. *exp(j*a) si', perche' A ed exp(j*a) sono due vettori.

Proviamo a vedere che variabili ci sono in memoria (con il comando whos) e notiamo che il vettore P e' effettivamente riconosciuto come complesso (e quindi occupa il doppio della memoria).

» whos

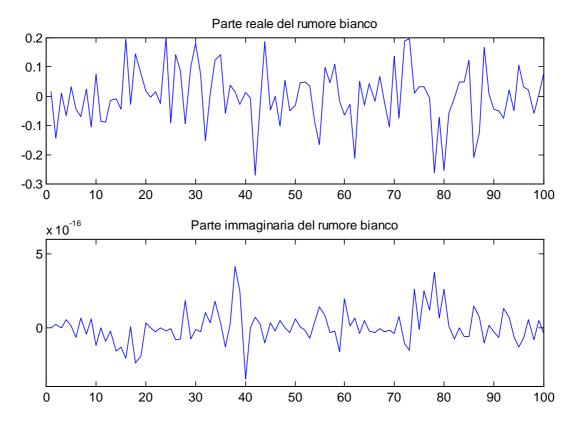
Name	Size	Elements	Bytes	Density	Complex
a	1 by 100	100	800	Full	No
A	1 by 100	100	800	Full	No
P	1 by 100	100	1600	Full	Yes

Calcoliamo adesso l'antitrasformata di P ed otteniamo il nostro rumore bianco.

```
» p=IFFT(P);
```

Noi sappiamo che p e' un vettore con tutte le componenti reali (abbiamo 'faticato' tanto per costruirlo!), ma Matlab non lo sa; calcolando la IFFT di una sequenza complessa si aspetta in generale (come e' giusto che sia) un'altra sequenza complessa. Ma e' proprio vero che la parte immaginaria di p=IFFT(P) e' esattamente nulla? Proviamo a disegnarla:

```
» subplot(211);
» plot(real(p)); title('Parte reale del rumore bianco');
» subplot(212);
» plot(imag(p)); title('Parte immaginaria del rumore bianco');
```



A prima vista la parte immaginaria non e' nulla, ma notiamo che sull'asse delle ordinate e' indicato il numero 10^{-16} : vuol dire che c'e' un piccolissimo errore dovuto alla quantizzazione perche' ovviamente Matlab lavora con un numero grande ma finito di bit. L'entita' dell'errore dipende anche dalla capacita' del Personal Computer che si sta usando, comunque deve essere $<10^{-12}$.

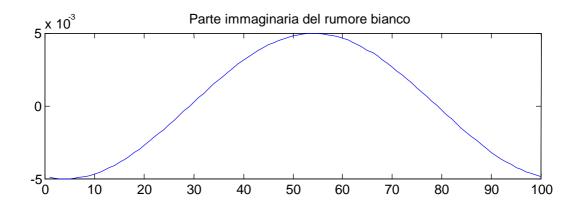
Per risolvere questo problema e' bene estrarre la sola parte reale del vettore p:

```
» p=real(p);
```

Vediamo infine cosa succede se viene a mancare la simmetria coniugata dello spettro anche per colpa di un solo elemento del vettore. Poniamo quindi, a titolo di esempio, A(100)=0.5 e ripetiamo i comandi necessari fino a visualizzare la parte immaginaria della sequenza nei tempi.

```
» A(100)=0.5;
» P=A.*exp(j*a);
» p=IFFT(P);

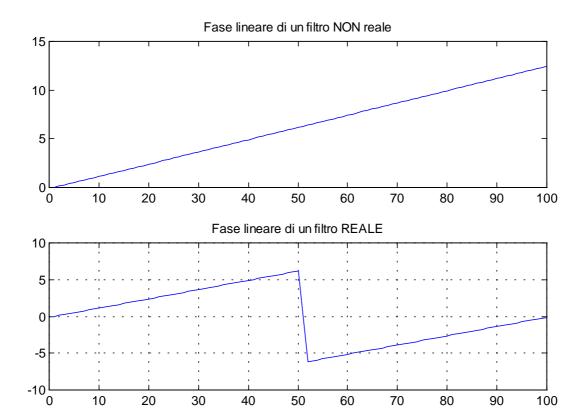
» subplot(211);
» plot(real(p));
» title('Parte reale del rumore bianco');
» subplot(212);
» plot(imag(p));
» title('Parte immaginaria del rumore bianco');
```



Si vede chiaramente che e' comparsa una sinusoide di ampiezza 5*10⁻³, e quindi non trascurabile.

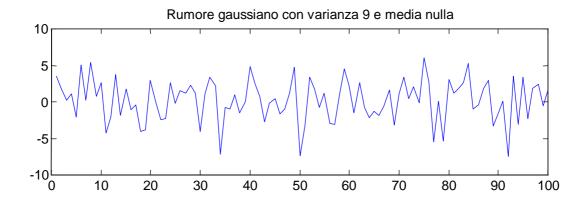
Vogliamo creare un filtro a fase lineare. Per fare questo, analogamente a quanto fatto nell'esempio precedente, lavoriamo nel dominio della FFT stando sempre attenti alle simmetrie; vediamo in particolare quella dispari della fase. Definiamo un asse delle frequenze di 100 elementi, creiamo il vettore a con fase lineare e poi aggiustiamo le simmetrie per avere un filtro reale.

```
  k=0:99; 
                              % asse delle frequenze
» tau=0.02;
                                e' il ritardo
» a=2*pi*k*tau;
                                fase del filtro
» subplot(211);
» plot(a);
                              % disegno la fase del filtro non reale
» title('Fase lineare di un filtro NON reale');
 = a(1) = 0; 
                              % forzo la continua
 * a(51)=0; 
                              % forzo il Nyquist
» a(52:100)=-fliplr(a(2:50)); % a(52:100)=-a(50:-1:2)
                                                        forzo la simmetria
                                dispari rispetto al Nyquist
» subplot(212);
                        % disegno la fase del filtro reale (per costruzione)
» plot(a);
» title('Fase lineare di un filtro REALE');
» grid;
```



In questo esercizio vediamo come costruire un filtro FIR (con il comando fir1) e come filtrare una sequenza casuale con i comandi filter e filtfilt.

Per prima cosa generiamo una sequenza di 100 campioni casuali distribuiti normalmente con media nulla e varianza 9 e la visualizziamo.



Prima di proseguire, e' bene sapere che il comando firl(N,w) costruisce un filtro (di default e' un passa-basso) utilizzando il metodo delle finestre (di default usa la finestra di Hamming). I parametri da specificare sono l'ordine del filtro N (il filtro sara' lungo quindi N+1 campioni) e la frequenza di taglio w ($con 0 \le w \le 1$, dove w=1 e' la pulsazione di Nyquist)

Se w e' un vettore, w= [w1 w2], allora indica un intervallo e Matlab realizza un filtro passa-banda con banda passante da w1 a w2.

 $Con\ il\ comando\ \texttt{firl}\ (\texttt{N}\ , \texttt{[wl w2]}\ , \texttt{'stop'}\)\ realizziamo\ un\ arresta-banda\ (nota\ che\ N\ deve\ essere\ pari).$

Usando invece fir1(N,w,'high') il risultato e' un passa-alto (anche qui N deve essere pari).

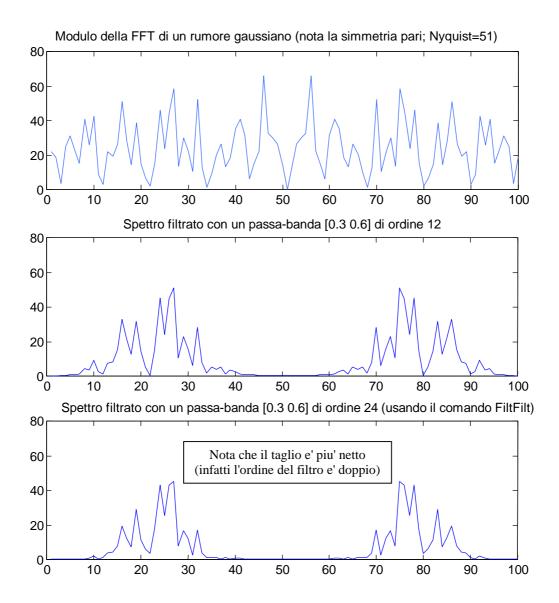
Esiste anche un comando fir2 per creare un filtro specificando alcune frequenze ed il valore che il modulo deve avere Per maggiori dettagli digitare help fir1 e help fir2 al prompt di Matlab.

Siamo pronti per creare un filtro FIR di ordine 12, passabanda tra $0.3 \, e \, 0.6$ rispetto alla frequenza di Nyquist. Dopo filtreremo con il comando y=filter(num,den,x), dove num e den sono numeratore e denominatore del filtro, x e' il segnale in ingresso e y quello in uscita.

```
» b=fir1(12,[0.3 0.6]);
                              % costruisco il filtro
» p=filter(b,1,s);
                              % filtro la sequenza s(n) ed ottengo p(n)
» figure;
 subplot(311);
 plot(abs(fft(s)));
                              % disegno lo spettro della sequenza originaria
» axis([0 100 0 80]);
» title('Modulo della FFT di un rumore gaussiano (nota la simmetria pari;...
        Nyquist=51)');
» subplot(312);
 plot(abs(fft(p)));
                              % disegno lo spettro della sequenza filtrata
» title('Spettro filtrato con un passa-banda [0.3 0.6] di ordine 12');
» axis([0 100 0 80]);
```

Se vogliamo un filtraggio a fase zero dobbiamo filtrare due volte (da destra a sinistra e viceversa); fortunatamente esiste il comando filtfilt che fa proprio questo.

Attenzione: cosi' facendo abbiamo raddoppiato l'ordine del filtro.



Vediamo alcuni comandi per costruire vettori a partire da matrici e viceversa. Incominciamo definendo la seguente matrice A:

Se ad esempio vogliamo estrarre la 2 riga di A ed assegnarla al vettore B, dobbiamo usare il seguente comando:

```
» B=A(2,:)
B =
3
```

Se invece vogliamo la prima colonna...

Infine, se vogliamo che tutta la matrice venga rimappata in un vettore possiamo usare il seguente:

Si noti che D e' stato costruito prendendo i valori di A colonna per colonna; se invece vogliamo prenderli riga per riga dobbiamo prima calcolare la trasposta di A:

Esiste anche un comando specifico, reshape (A, m, n), che, partendo dalla matrice A ne crea un'altra di dimensioni a scelta m x m (ovviamente la matrice A deve avere m*n elementi):; si noti che anche in questo caso l'estrazione degli elementi di A viene fatta colonna per colonna, quindi, se vogliamo estrarre riga per riga, dobbiamo usare reshape (A', m, n)

Questo comando puo' essere utilizzato anche per creare una matrice partendo da un vettore, ad esempio:

una possibile applicazione e' la seguente: vogliamo costruire una matrice M quadrata 6x6, di tutti zeri tranne tre elementi a caso che devono avere valore unitario; i passi da compiere sono:

```
» V=zeros(1,36);
                         % parto da un vettore riga di 6*6 elementi
» x=ceil(36*rand(1,3)); % scelgo 3 numeri a caso tra 1 e 36
 > V(x) = ones(1,3); 
                         % utilizzo i tre precedenti numeri casuali come indice
                           per scegliere quali elementi di V devono avere valore
                           unitario
                         % costruisco la matrice 6x6 a partire dal vettore V
» M=reshape(V,6,6)
M =
     0
                                      0
            0
                  0
                         0
                               1
     0
            0
                                      0
                  0
                         0
                               0
     0
            1
                         0
                                      0
                  0
                               0
     0
            0
                         0
                                      0
                  0
                               0
     0
            0
                  0
                         1
                               0
                                      0
     0
            0
                         0
                                      0
                  0
                               0
```

Definiamo ora la seguente matrice C a croce

```
» C=[0 2 0 ; 2 2 2 ; 0 2 0]
C =

0 2 0
2 2 2
0 2 0
```

e replichiamola in M centrata proprio nelle posizioni indicate dagli 1 (che sono quindi degli spilli); per farlo convolviamo C ed M con il comando Y=conv2 (M,C,'same') dove l'opzione 'same' obbliga Y a mantenere le stesse dimensioni di M

```
» Y=conv2(M,C,'same')
Y =
      0
              0
                      0
                             2
                                     2
                                             2
      0
              2
                      0
                             0
                                             0
                                     2
      2
              2
                      2
                             0
                                     0
                                             0
      0
              2
                      0
                             2
                                     0
                                             0
      0
              0
                      2
                             2
                                     2
                                             0
      0
              0
                      0
                             2
                                     0
                                             0
```

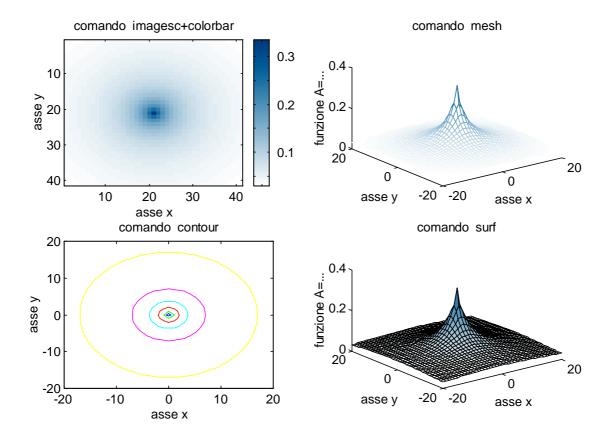
Vediamo in questo esercizio alcuni comandi caratteristici dei segnali in due dimensioni. Analogamente al caso 1D, dove definivamo l'asse temporale per poi costruire il segnale da elaborare, nel caso 2D bisogna definire una matrice di punti; per fare questo potremmo ad esempio usare due cicli for:

e' sicuramente piu' veloce il comando meshgrid, che si usa nel seguente modo:

nota che gli indici del ciclo for devono per forza essere numeri naturali positivi, poiche' servono anche come indici della matrice a(x,y); nel secondo esempio invece, e' stato messo in evidenza che questa restrizione e' superata, quindi: non si usano i cicli for ma il comando meshgrid.

Se vogliamo visualizzare su schermo i segnali 2D abbiamo a disposizione diversi comandi, tra i quali imagesc, mesh, contour, surf, plot3; vediamone l'ouput grafico.

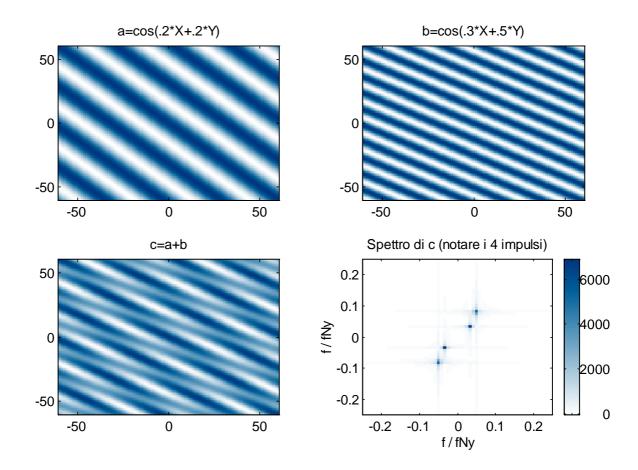
```
» figure;
» subplot 221;
» imagesc(A);
» colorbar;
                        % serve per avere la legenda a fianco del grafico
» xlabel('asse x');
» ylabel('asse y');
» zlabel('funzione A=...');
» title('comando imagesc');
» subplot 222;
» mesh(X,Y,A);
» xlabel('asse x'); ylabel('asse y'); zlabel('funzione A=...');
» title('comando mesh');
» subplot 223;
» contour(X,Y,A);
                              % per disegnare i livelli
» xlabel('asse x'); ylabel('asse y'); zlabel('funzione A=...');
» title('comando contour');
» subplot 224;
» surf(X,Y,A);
                              % per disegnare la superficie
» xlabel('asse x'); ylabel('asse y'); zlabel('funzione A=...');
» title('comando surf');
```



Si noti che il comando imagesc visualizza la matrice con l'origine degli assi in alto a sinistra, l'asse x orientato verso destra e l'asse y verso il basso; se vogliamo utilizzare il consueto sistema cartesiano con origine in basso a sinistra ed asse y orientato verso l'alto bisogna utilizzare imagesc(x,-y,A). Ovviamente, data la simmetria circolare del segnale A, in questo caso non si vede alcuna differenza; quando lavoreremo con le sinusoidi potra' invece essere fonte di equivoci...

Generiamo ora due sinusoidi 2D, le sommiamo e ne facciamo la trasformata di Fourier bidimensionale.

```
 > N = 60; 
 x=-N:N;
                              % vettore lungo le x
 y=-N:N;
                                vettore lungo le y
  [X,Y] = meshgrid(x,y);
 a = cos(.2*X+.2*Y);
                              % coseno con azimuth +45 deg
 b = cos(.3*X+.5*Y);
                                coseno con azimuth atan(.5/.3)=63 deg
 c=a+b;
                                somma di sinusoidi
 C=fft2(c);
                                DFT in due dimensioni
» figure;
» subplot 221; imagesc(x,-y,a);
                                 title(' a=cos(.2*X+.2*Y) ');
» subplot 222; imagesc(x,-y,b);
                                 title(' b=cos(.3*X+.6*Y) ');
» subplot 223; imagesc(x,-y,c);
                                 title(' c=a+b ');
» subplot 224; imagesc(x/N,-y/N,fftshift(abs(C)));
» title('Spettro di c (notare i 4 impulsi)');
» axis([-30 30 -30 30]/N); grid; xlabel('f / fNy'); ylabel('f / fNy');
```



Il comando fftshift serve per traslare lo spettro (ricorda che la traslazione e' circolare, data la periodicita') in modo da visualizzare la componente alla frequenza nulla nel centro dello schermo.