

Sistemi di Telecomunicazioni

Docente: Ing. Andrea Conti
a.conti@ieee.org

Breve introduzione a MATLAB

Realizzata da: Ing. Andrea Giorgetti

1 Questa breve guida . . .

Lo scopo di questa guida è quello di offrire un ausilio a chi muove i primi passi nell' ambiente MATLAB. Dettagli sulle funzionalità e sull'uso dei comandi esulano dagli scopi di questa dispensa. L'approccio seguito è diretto e basato su esempi esplicativi. Si consiglia sempre di consultare l'help in linea e la manualistica per dettagli sull'uso delle funzioni e dei comandi.

2 Generalità

MATLAB è una piattaforma per il calcolo scientifico, l'elaborazione dei dati e la loro visualizzazione. Il suo nome deriva da MATrix LABoratory. Molte informazioni utili possono essere reperite su:

- Siti ufficiali del produttore: www.mathworks.com o www.mathworks.it
- Tool sviluppati su piattaforma MATLAB: www.mathtools.com
- Tool sviluppati da utenti MATLAB (third-party) www.mathworks.it/matlabcentral

Brevi ma utili guide per iniziare ad utilizzare MATLAB (oltre a questa):

- K. Sigmon, *MATLAB Primer*, 3rd Edition, Dept. of Math., Univ. of Florida.
- K. Sigmon, T. A. Davis, *MATLAB Primer*, 6th Edition, Chapman & Hall, 2001.
- M. Tibaldi, *Note Introduttive a MATLAB e Control System Toolbox*, Esculapio, Bologna.

Libri dedicati al Signal Processing in ambiente MATLAB:

- V. K. Ingle, J. G. Proakis, *Digital Signal Processing using MATLAB*, Brooks/Cole, 2000.
- J. G. Proakis, M. Salehi, *Contemporary Communication Systems using MATLAB*, Brooks/Cole, 2000.
- C. S. Burrus *et al.*, *Computer exercise for signal processing using MATLAB*, Prentice-Hall, 1994.

3 Signal Processing con MATLAB

MATLAB implementa numerosi comandi che svolgono azioni comuni nella elaborazione numerica dei segnali. Alcuni di questi sono presenti nel pacchetto base, altri sono disponibili tramite i vari Toolbox: *Signal Processing Toolbox*, *Image Processing Toolbox* etc.

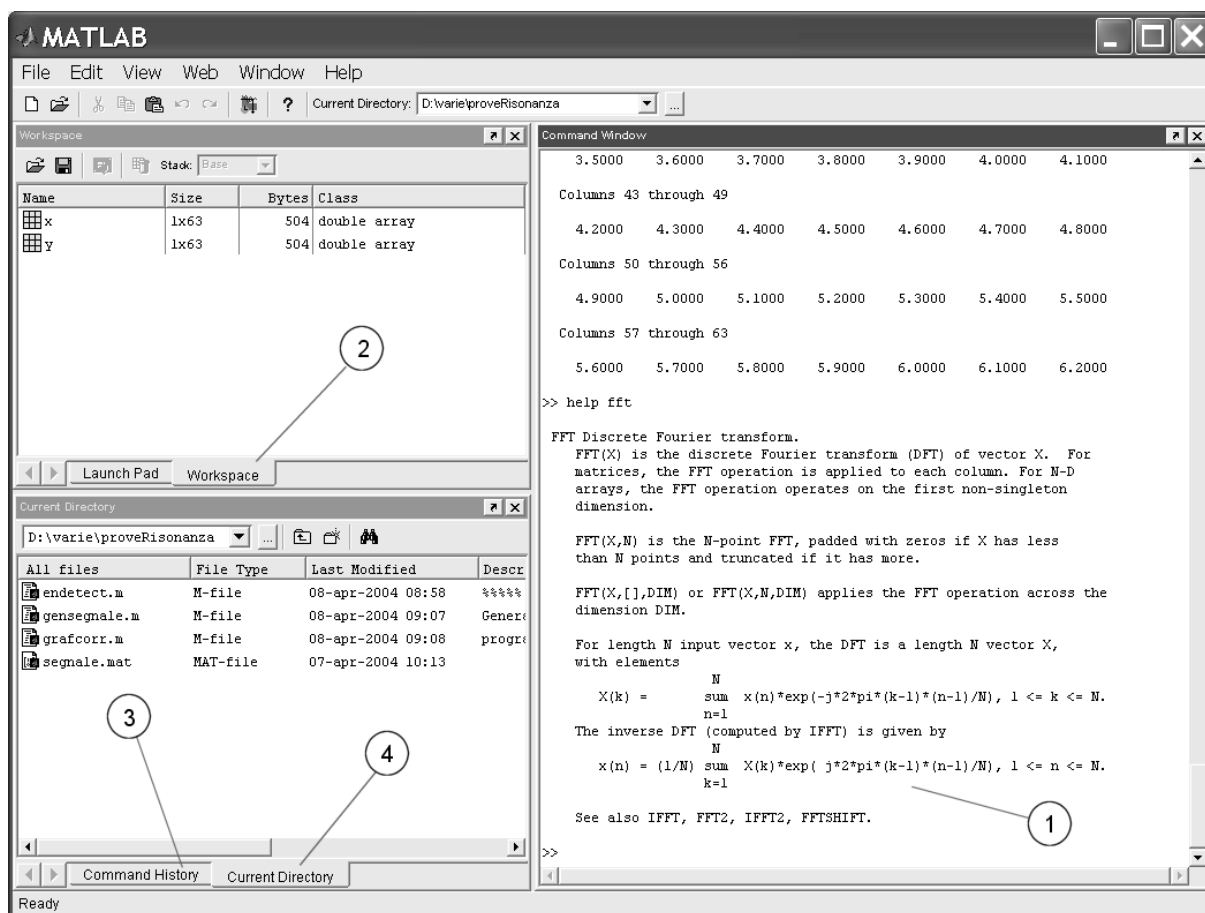


Figura 1: Ecco come appare MATLAB al suo avvio.

4 L'ambiente

L'ambiente MATLAB si presenta come un'area di lavoro in cui si possono individuare quattro finestre come mostrato in Fig. 1:

1. *Command Window.* È l'area in cui vengono digitati i comandi.
2. *Workspace.* Vengono visualizzate tutte le variabili definite dall'utente, il tipo e la loro occupazione di memoria.
3. *Command History.* Riporta l'elenco in ordine cronologico dei comandi che sono stati digitati nella Command Window.
4. *Current Directory.* Mostra i file della cartella di lavoro corrente.

Nella *Command Window* è possibile digitare comandi e premere invio per avviare l'elaborazione e visualizzare l'eventuale risultato. Inoltre, con le frecce è possibile richiamare comandi precedentemente digitati in alternativa alla *Command History*.

Uno strumento comodo e veloce per imparare ad utilizzare le funzioni e le routine di MATLAB è l'*help* in linea:

```
>> help fft
```

FFT Discrete Fourier transform.

FFT(X) is the discrete Fourier transform (DFT) of vector X. For matrices, the FFT operation is applied to each column. For N-D arrays, the FFT operation operates on the first non-singleton dimension.

FFT(X,N) is the N-point FFT, padded with zeros if X has less than N points and truncated if it has more.

FFT(X,[],DIM) or FFT(X,N,DIM) applies the FFT operation across the dimension DIM.

For length N input vector x, the DFT is a length N vector X, with elements

$$X(k) = \sum_{n=1}^N x(n) \exp(-j*2*\pi*(k-1)*(n-1)/N), \quad 1 \leq k \leq N.$$

The inverse DFT (computed by IFFT) is given by

$$x(n) = (1/N) \sum_{k=1}^N X(k) \exp(j*2*\pi*(k-1)*(n-1)/N), \quad 1 \leq n \leq N.$$

See also IFFT, FFT2, IFFT2, FFTSHIFT.

Come si può notare, l'*help* è suddiviso in diverse parti:

- Uso del comando con eventuali opzioni.
- Cenni alla implementazione del comando (molto utile per essere sicuri che una data funzione con un nome a noi familiare svolga il compito da noi richiesto).
- Eventuale esempio di utilizzo.
- Funzioni correlate.

Per una esplorazione completa e strutturata dell'*help* occorre accedere ad esso tramite la barra dei menu.

5 Manipolare variabili

Assegnare un valore ad una variabile x :

```
>> x=2
x =
    2
```

Assegnare il valore della variabile x alla variabile y :

```
>> y=x
y =
    2
```

Eseguire somme “+”, sottrazioni “-”, divisioni “/” e moltiplicazioni “*“:

```
>> x*y
ans =
    4
```

```
>> x+y
ans =
    4
```

```
>> x/y+5
ans =
    6
```

Ogni volta che si vuole eseguire un comando come un semplice assegnamento ma non si vuole visualizzare l’output é sufficiente aggiungere “;” al termine del comando. Per esempio se vogliamo calcolare $z = x^{2.5}$:

```
>> z=x^2.5
z =
    5.6569
```

```
>> z=x^2.5;
>>
```

Se vogliamo conoscere il valore che assume una variabile in un dato momento è sufficiente digitare il suo nome seguito da invio:

```
>> z
z =
    5.6569
```

o in alternativa “doppio clic” sulla variabile che compare nella *Workspace Window* (Sez. 4 in Fig. 1). Provare!!

5.1 Cancellare variabili

Giunti a questo punto nel workspace saranno presenti le tre variabili x , y e z . Se si vuole eliminare una variabile non più utile (risparmiare memoria non fa mai male specie per calcoli che richiedono matrici e vettori di grandi dimensioni) esiste il comando `clear`:

```
>> clear z
```

Si noti che la variabile z non è più presente nel workspace.

Quando si vuole ripulire l'intero workspace è sufficiente digitare il comando `clear` seguito da invio:

```
>> clear
```

ATTENZIONE. I dati persi non potranno più essere recuperati!

5.2 Pulizia della *Command Window*

Se invece vogliamo ripulire la *Command Window* (senza cancellare le variabili) affidarsi al comando `clc`:

```
>> clc
```

5.3 Numeri complessi

MATLAB è in grado di svolgere operazioni con numeri complessi. Cominciamo con il digitare la lettera “i” o la lettera “j” seguita da invio:

```
>> i
ans =
      0 + 1.0000i
```

È evidente che MATLAB assegna (by default) l'unità immaginaria $i = \sqrt{-1}$ proprio alle variabili i e j . Attenzione, se assegnate a queste variabili un valore diverso (chi non ha mai usato la variabile i in un ciclo `for`?):

```
>> i=10
i =
     10
```

perderete il suo valore originale! Non è grave, possiamo definire una variabile ii :

```
>> ii=sqrt(-1)
ii =
      0 + 1.0000i
```

Anche la costante π non può mancare:

```
>> pi
ans =
    3.1416
```

Per i numeri complessi sono già definite somme, sottrazioni, moltiplicazioni e divisioni:

```
>> clear
>> x=1+i;
>> y=2+2*i;

>> x*y
ans =
    0 + 4.0000i
```

E ovviamente non possono mancare parte reale, parte immaginaria, modulo e fase (in radianti):

```
>> x = 1 + i;
>> real(x)
ans =
    1

>> imag(x)
ans =
    1

>> abs(x)
ans =
    1.4142

>> angle(x)
ans =
    0.7854
```

Per chi avesse dubbi digitare `help nomecomando`.

5.4 Notazione esponenziale

Nel caso si vogliano introdurre numeri con esponente valgono le seguenti equivalenze:

$$9.8*10^{(-12)} = 9.8e-12 = 9.8E-12$$
$$-2.3*10^5 = -2.3e5 = -2.3E5 = -2.3E+5$$

6 Vettori

È possibile generare un vettore riga nei seguenti modi:

```
>> a=[1 2 3]
a =
     1     2     3

>> a=[1,2,3]
a =
     1     2     3
```

ove la virgola o lo spazio sono interpretati come separatori di colonna, mentre il punto e virgola agisce da separatore di riga:

```
>> b=[1;2;3]
b =
     1
     2
     3
```

generando così un vettore colonna. Spesso è necessario generare un vettore di numeri equispaziati da x_{min} a x_{max} con passo d_x . Ad esempio, per generare automaticamente un vettore da 1 a 2 con passo 0.1:

```
>> x=1:0.2:2
x =
     1.0000     1.2000     1.4000     1.6000     1.8000     2.0000
```

e in caso di variabili simboliche MATLAB provvederà alla opportuna sostituzione:

```
>> xmin=1;
>> xmax=2;
>> dx=0.2;
>> x=[xmin:dx:xmax];
>> x
x =
     1.0000     1.2000     1.4000     1.6000     1.8000     2.0000
```

Cosa succede se digitiamo $x=[xmax:-dx:xmin]$?

6.1 Manipolare vettori

Per effettuare la trasposizione di un vettore basta appendere “.’”:

```
>> x=[1+i,2+2*i,3-i]
x =
    1.0000 + 1.0000i    2.0000 + 2.0000i    3.0000 - 1.0000i
>> x.'
ans =
    1.0000 + 1.0000i
    2.0000 + 2.0000i
    3.0000 - 1.0000i
```

se invece non mettiamo il punto:

```
>> x'
ans =
    1.0000 - 1.0000i
    2.0000 - 2.0000i
    3.0000 + 1.0000i
```

otteniamo il trasposto coniugato!!

Un modo veloce per sommare tutti gli elementi di un vettore è:

```
>> x=[1 2 4];
>> sum(x)
ans =
    7
```

Estrarre parti di un vettore è una operazione molto comune che MATLAB rende alquanto semplice:

```
>> x=[1 3 5 7 9 11 13];
>> x(1:3)
ans =
    1    3    5

>> x(2:3)
ans =
    3    5

>> x(3:length(x))
ans =
    5    7    9   11   13
```

dove `length()` restituisce la lunghezza del vettore.

Ci sono molte altre funzioni MATLAB per manipolare vettori tra le quali:

```
>> v=[1 2 9 4 6 5 7 8 3];
>> min(v)
ans =
     1

>> max(v)
ans =
     9

>> mean(v)
ans =
     5

>> std(v)
ans =
    2.7386

>> sort(v)
ans =
     1     2     3     4     5     6     7     8     9
```

Spesso è necessario importare ed esportare vettori da MATLAB ad altri ambienti ed in diversi formati. Per fare questo esistono le funzioni `save` e `load` a cui rimandiamo all'*help* per eventuali dettagli sui formati e le opzioni possibili.

Per esempio se vogliamo salvare il vettore v in un file di testo in formato ASCII possiamo digitare il comando:

```
>> v=[1.2e-10; 3.1e-8; 2.4e-5];
>> save prova.dat -ascii v
```

Aprire il file *prova.dat* con un editor di testo e controllare il buon esito dell'operazione! Allo stesso modo se vogliamo importare il vettore dal file *prova.dat* in MATLAB:

```
>> w=load('prova.dat')
w =
    1.0e-004 *
    0.0000
    0.0003
    0.2400
```

il contenuto del file verrà caricato nel vettore w .

7 Matrici

L'introduzione di matrici segue le regole viste per i vettori; spazi o virgole per le colonne e punti e virgola per le righe:

```
>> a=[1 2 3; 4 5 6]
a =
     1     2     3
     4     5     6
```

come per i vettori è possibile risalire alle dimensioni di una matrice:

```
>> size(a)
ans =
     2     3
```

ed è anche possibile salvare o importare matrici con i comandi `save` e `load`. È possibile generare automaticamente alcune matrici di uso comune:

```
>> eye(2)
ans =
     1     0
     0     1

>> zeros(2)
ans =
     0     0
     0     0

>> ones(1,4)
ans =
     1     1     1     1
```

7.1 Manipolare Matrici

In MATrix LABoratory non possono mancare tutte le operazioni standard per matrici: addizione “+”, sottrazione “-”, moltiplicazione “*”, divisione “\ o /”, potenza “^” e trasposto coniugato “'”. Ad esempio:

```
>> a=[1 2; 3 4]+i*[0 1; -1 0]
a =
 1.0000          2.0000 + 1.0000i
 3.0000 - 1.0000i  4.0000

>> a'
ans =
 1.0000          3.0000 + 1.0000i
 2.0000 - 1.0000i  4.0000
```

ATTENZIONE: è cura dell'utente garantire che le matrici abbiano dimensioni compatibili!! Altrimenti ecco cosa potrebbe succedere:

```
>> clear
>> a=eye(2)
a =
     1     0
     0     1

>> b=zeros(3)
b =
     0     0     0
     0     0     0
     0     0     0

>> a+b
??? Error using ==> + Matrix dimensions must agree.

>> a*b
??? Error using ==> * Inner matrix dimensions must agree.
```

Un altro errore assai frequente è il seguente:

```
>> a=eye(3)
a =
     1     0     0
     0     1     0
     0     0     1

>> b=[1 2 3]
b =
     1     2     3

>> a*b
??? Error using ==> * Inner matrix dimensions must agree.
```

ed è ovvio poiché non esiste la moltiplicazione di una matrice per un vettore riga, mentre:

```
>> b*a
ans =
     1     2     3

>> a*(b')
ans =
     1
     2
     3
```

sono corrette!

8 Funzioni matematiche elementari

Una funzionalità molto comoda di MATLAB è la possibilità di effettuare operazioni sui *singoli elementi* di vettori e matrici (*element-wise*) semplicemente anteponendo il “.” all’operatore:

```
>> x=[1 2 3 4];
>> y=[1 0 1 0];
>> x.*y
ans =
     1     0     3     0

>> x.^2
ans =
     1     4     9    16
```

idem per le matrici ... fare qualche prova!

MATLAB include tutte le funzioni matematiche elementari `abs`, `angle`, `real`, `imag`, `sqrt` che abbiamo visto più altre non meno importanti quali `exp(x)=ex`, `log(x)=ln x` e `log10(x)=log10 x` che rappresentano rispettivamente l’esponenziale, il logaritmo naturale ed il logaritmo in base 10.

Poi ci sono le funzioni trigonometriche `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `sinh`, `cosh`, `tanh`,... e tante altre funzioni cosiddette *speciali* di uso comune come `bessel`, `gamma`, `erfc`, `inverfc` i cui dettagli possono essere reperiti con il consueto `help nomecomando`.

È molto utile ricordare che queste funzioni agiscono sui singoli elementi dei vettori (sono *element wise*), così se chiediamo a MATLAB di calcolare il “seno di un vettore” ci risponderà:

```
>> clear
>> x=0:pi/5:pi

x =
     0     0.6283     1.2566     1.8850     2.5133     3.1416

>> sin(x)

ans =
     0     0.5878     0.9511     0.9511     0.5878     0.0000
```

con il seno di ogni suo elemento.

Si noti che le funzioni definite dall’utente NON sono di tipo *element wise*.

9 Istruzioni condizionali: IF...ELSE ...END

Come tutti i linguaggi di programmazione, anche MATLAB incorpora il costrutto `if`. La forma più semplice è la seguente:

```
>> a=5;
>> if a > 3 b=2; end
>> b
b =
    2
```

se è necessario aggiungere `else`:

```
if a~=5 b=2; else b=10; end
>> b
b =
    10
```

digitare `help if` per informazioni aggiuntive, `elseif...`

Gli operatori relazionali sono:

```
<  minore di
>  maggiore di
<= minore o uguale di
>= maggiore o uguale di
==  uguale
~=  diverso
```

Non confondere mai `==` con `=`, il primo è un operatore relazionale, il secondo è un assegnamento!

10 I/O su Video

Per la stampa a video MATLAB segue una sintassi C-like:

```
>> fprintf('Il valore di P greco risulta %f \n',pi);
Il valore di P greco risulta 3.141593
```

Mentre se si devono immettere dati da input:

```
>> a=input('Immettere il valore della temperatura = ')
Immettere il valore della temperatura = 21.4
a =
    21.4000
```

11 Cicli For, While ...

Partiamo subito con un esempio:

```
>> for i=1:10, x(i)=i; end
>> x
x =
     1     2     3     4     5     6     7     8     9    10
```

che permette di generare un vettore x di 10 elementi. Se si vuole rendere più leggibile il codice si può anche andare a capo senza la virgola:

```
>> for i=1:10
      x(i)=i;
    end
>> x
x =
     1     2     3     4     5     6     7     8     9    10
```

Analogamente possiamo innestare più cicli:

```
>> m=3;
>> n=2;
>> for i=0:m-1
      for j=0:n-1
          a(i,j)=i+j;
      end
    end
??? Index into matrix is negative or zero.
```

Attenzione gli indici di vettori e matrici partono da 1!! Con una semplice traslazione degli indici:

```
>> m=3;
>> n=2;
>> for i=1:m
      for j=1:n
          a(i,j)=(i-1)+(j-1);
      end
    end
>> a
a =
     0     1
     1     2
     2     3
```

otteniamo il risultato desiderato. In MATLAB esiste anche il costrutto `while...`

12 Efficienza...

MATLAB è un linguaggio non strutturato e in quanto tale non necessita di allocare variabili, vettori o matrici prima dell'uso. Questo semplifica il lavoro del programmatore ma può essere fonte di errori ed inefficienze.

Ad esempio, se desideriamo generare un vettore del tipo $x = (1, 2, 3, 4, \dots, 10^6)$ è sufficiente digitare:

```
>> for i=1:10^6, x(i)=i; end
```

ma vi accorgete che la generazione del vettore x richiede molto tempo. Il motivo di tale inefficienza risiede nel fatto che MATLAB non conosce a priori la dimensione del vettore, perciò ad ogni ciclo alloca un nuovo elemento in coda al vettore generato ai cicli precedenti. Tutti sanno che l'operazione di allocazione ha un costo elevato in quanto coinvolge primitive del sistema operativo che devono cercare una locazione di memoria libera ecc. In questo caso MATLAB deve eseguire 10^6 allocazioni! Per ovviare a questo inconveniente si può forzare MATLAB ad allocare (ed inizializzare) il vettore x (od eventualmente una matrice) prima del suo utilizzo tramite la funzione **zeros**:

```
>> x=zeros(1,10^6);  
>> for i=1:10^6, x(i)=i; end;
```

In questo modo la generazione è immediata in quanto l'allocazione avviene in un colpo solo, in modo analogo alla *malloc* del C.

13 Script files

Sino ad ora abbiamo digitato comandi direttamente nella *Command Window*. Nel caso si debbano eseguire ripetutamente lunghe sequenze di operazioni risulta molto comodo usare gli *script files*. Uno script file non è altro che un file di testo con estensione `.m` contenente la sequenza di comandi che si voglio fare eseguire a MATLAB.

Editare script files è semplice, basta un qualunque editor di testo ascii o più semplicemente seguire il percorso del menu *File* – *>* *New* – *>* *M-file*, dal quale si apre l'*M-file editor*. Vediamo un esempio:

```
% ESEMPIO DI SCRIPT FILE. 22 aprile 2004

assex=[0:0.1:1]';      % genero l'asse x
assey=exp(assex);     % calcolo la funzione esponenziale
matrice=[assex assey] % genero una matrice dei punti
```

e salvare il file mediante *File* – *>* *Save As* – *>* *esempio.m* ...

Digitare il comando `esempio` nella *Command Window* ed apparirà a video il risultato dell'esecuzione (vedi Fig. 2).

Uno script file può chiamare al suo interno altri script file, e persino se stesso (routine ricorsive).

Le variabili definite all'interno di uno script file sono GLOBALI cioè visibili da ogni altro script o funzione.

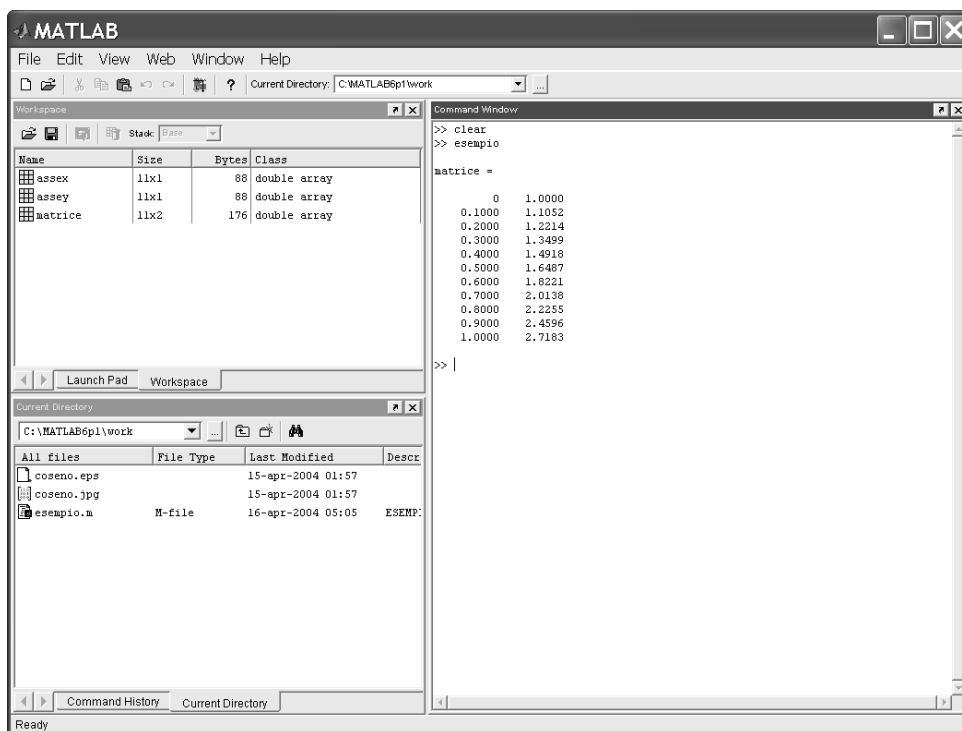


Figura 2: Si noti il file `esempio.m` nella finestra *Current Directory* ed il risultato del comando `esempio` nella *Command Window* e nel *Workspace*.

14 Funzioni definite dall'utente

In MATLAB è possibile definire proprie funzioni specificando i parametri in ingresso, quelli d'uscita e definendo variabili che risultano LOCALI alla funzione.

Le funzioni sono file di testo così come gli script file e possono essere editate con l'*M-file editor* (sempre dal percorso *File* - > *New* - > *M-file*). Vediamo un esempio:

```
function ris = somma(x,y)
% somma(x,y) restituisce la somma di due numeri x,y
% siano essi reali o immaginari, vettori o matrici.
ris = x + y;
```

Si noti che la variabile `ris` compare nella dichiarazione e nel corpo della funzione. Occorre salvare la funzione con il nome `somma.m` (lo stesso nome della funzione) con estensione `.m` e MATLAB la riconoscerà come tale. A questo punto la funzione `somma` è disponibile all'utente. Un esempio di utilizzo:

```
>> somma(2+i,3)
ans =
    5.0000 + 1.0000i

>> help somma
    somma(x,y) restituisce la somma di due numeri x,y
    siano essi reali o immaginari, vettori o matrici.
```

Si noti come le righe commentate compaiano come *help!*

Si possono definire funzioni con più parametri in uscita. Un esempio:

```
function [vett1, vett2] = separa(x)
% separa(x) separa un vettore in due sottovettori.
lun=length(x);
if rem(lun, 2)==0 % resto della divisione
    vett1=x(1:lun/2);
    vett2=x(lun/2+1:lun);
end
```

Un esempio di uso:

```
>> x=[1 2 3 4];
>> [a,b]=separa(x)
a =
     1     2
b =
     3     4
```

15 Grafici

Vediamo ora come realizzare grafici 2D in MATLAB. La sperimentazione di molte delle potenzialità grafiche di MATLAB è lasciata al lettore interessato.

Supponiamo di voler graficare la funzione $\sin(x)$ nell'intervallo $[-\pi, \pi]$. Per prima cosa occorre stabilire con quale risoluzione intendiamo graficare la funzione. Supponiamo che siano sufficienti 50 punti per avere un grafico accettabile. A questo punto dobbiamo generare il vettore relativo all'asse x costituito da 50 punti equispaziati da $-\pi$ a π con passo $2\pi/50$:

```
>> npunti=50;
>> x=[-pi:2*pi/npunti:pi];
```

Ricordando che la funzione `sin` come tutte le funzioni matematiche opera su ogni elemento del vettore, possiamo scrivere in modo compatto:

```
>> y=sin(x);
```

Ora, la coppia di vettori x e y di ugual dimensione è in grado di individuare univocamente i punti del piano. Con il comando `plot`:

```
>> plot(x,y);
```

apparirà il grafico voluto. Se vogliamo aggiungere le *label* agli assi (chi leggerà il grafico DEVE SAPERE quali grandezze rappresentano gli assi!!) possiamo usare le seguenti istruzioni che possiamo mettere in uno *script file* per comodità:

```
% GRAFICO
x=[-pi:2*pi/50:pi];
y=sin(x);

plot(x,y,'o');
title('Segnale x(t)');
xlabel('tenpo [ms]');
ylabel('ampiezza [V]');
```

I simboli che si possono usare per le curve sono molteplici. Provare con `-`, `--`, `:`, `-.`, `..`, `+`, `*`, `o`, `x` e poi `s` per *square*, `d` per *diamond*, `v` per *triangle down*, `^` per *triangle up*, oppure ci sono i colori: `y`, `m`, `c`, `r`, `g`, `b`, `w`, `k`.

Una eventuale griglia migliora la leggibilità del grafico, a tal proposito appendere il comando `grid`.

MATLAB sceglie in modo automatico il range degli assi. Se si vogliono forzare gli intervalli `[xmin, xmax]` e `[ymin, ymax]` usare il comando `axis([xmin xmax ymin ymax])`, un esempio di uso:

```
plot(x,y), axis([-0.5 0.5 1e-4 0.1]);
```

Per creare una nuova figura senza eliminare quella precedente è sufficiente anteporre il comando `figure` a `plot`:

```
figure; plot(x,y);
```

mentre per eliminare tutte le figure create usare il comando:

```
close all
```

Per grafici a barre verticali, molto utili quando si devono rappresentare sequenze di campioni, utilizzare il comando

```
stem(x,y)
```

al posto del comando `plot`.

16 Grafici multipli

Partendo dallo script appena creato, possiamo modificarlo in questo modo per ottenere grafici multipli (vedi Fig. 3):

```
% GRAFICO
x=[-pi:2*pi/50:pi];
y=sin(x);
z=exp(-x.^2);

plot(x,y,'o',x,z,'-');
title('Segnale');
xlabel('tempo [ms]');
ylabel('ampiezza [V]');
grid;
legend('sin(x)', 'gauss');
```

Si noti che nella funzione $z = e^{-x^2}$ si è utilizzato l'operatore *element-wise* “.^”, se usassimo il “^” MATLAB cercherebbe di calcolare il quadrato del vettore (che tra l'altro non esiste) e non il quadrato dei singoli elementi.

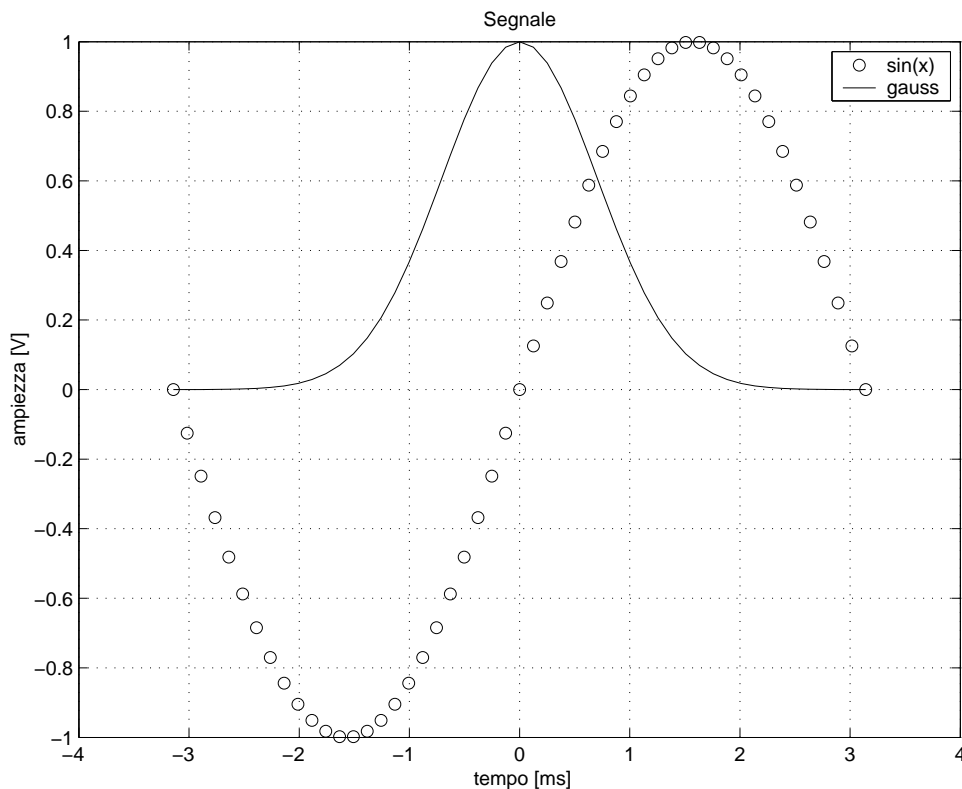


Figura 3: Il grafico multiplo ottenuto.

Un'interessante alternativa al grafico multiplo che abbiamo ottenuto la si può ottenere con una semplice modifica dello script:

```
% GRAFICO
x=[-pi:2*pi/50:pi];
y=sin(x);
z=exp(-x.^2);

subplot(211),plot(x,y,'o');
title('Segnale');
xlabel('tempo [ms]');
ylabel('ampiezza [V]');
grid;
subplot(212),plot(x,z,'-');
title('Segnale');
xlabel('tempo [ms]');
ylabel('ampiezza [V]');
grid;
```

Il risultato è mostrato in figura Fig. 4. Cosa succede se cambiamo i parametri delle funzioni subplot in subplot(121) e subplot(122)?

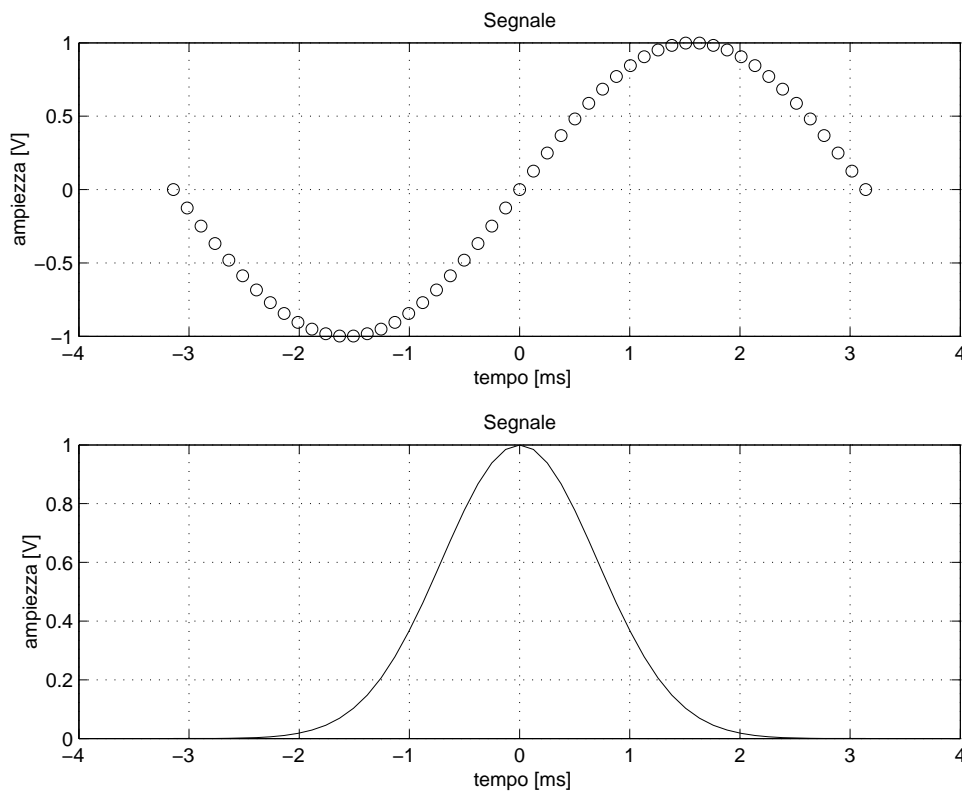


Figura 4: Il grafico multiplo ottenuto con il comando subplot.

17 Grafici in scala logaritmica

Per ottenere un grafico come quello di Fig. 5 con le ordinate in scala logaritmica, occorre sostituire la funzione `plot` con la funzione `semilogy`:

```
snrdB=[0:0.2:10];
snr=10.^(snrdB/10);
Pe=0.5*erfc(sqrt(snr));

semilogy(snrdB,Pe);
title('BEP sistema BPSK');
xlabel('SNR (dB)');
ylabel('Pe');
grid;
```

Analogamente, per l'asse x vedere il comando `semilogx`. Mentre per grafici in doppia scala logaritmica come quello di Fig. 6 occorre fare riferimento al comando `loglog`:

```
tau=0.01; punti=100;
freq=[0:1000/punti:1000];

for i=1:punti+1 fdt(i)=1/sqrt(1+(2*pi*freq(i)*tau)^2); end

loglog(freq,fdt);
title('Modulo fdt filtro RC');
xlabel('freq[Hz]');
ylabel('|H(f)|');
grid;
```

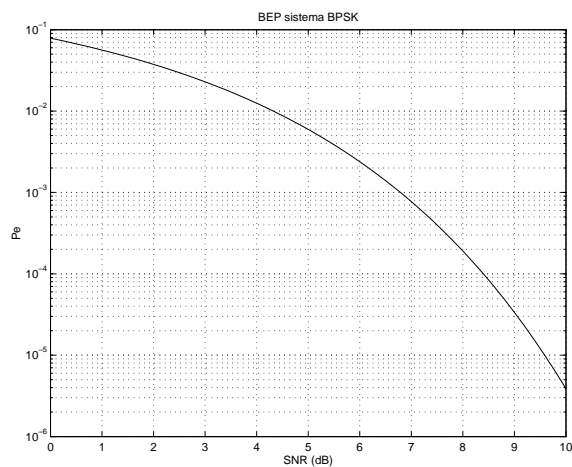


Figura 5: Il grafico con l'asse y in scala logaritmica.

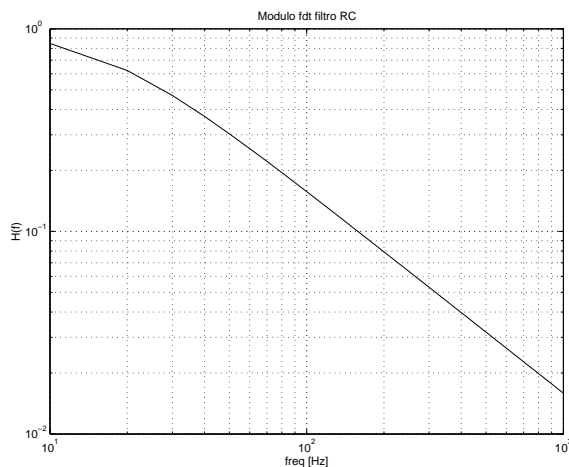


Figura 6: Il grafico con entrambi gli assi in scala logaritmica.

18 Esportare i grafici: .eps o .jpg ?

Una volta prodotto il grafico vi può essere la necessità di salvarlo ed esportarlo in un opportuno formato. L' esportazione di grafici MATLAB può essere effettuata in diversi modi:

- I grafici possono essere salvati direttamente dalla finestra del grafico stesso in formato `.fig`. Questo formato proprietario di MATLAB permette la visualizzazione solo a chi possiede questa piattaforma. Purtroppo, grafici prodotti con diverse versioni di MATLAB non sono sempre compatibili.
- Per chi deve esportare le figure in documenti \LaTeX , conviene salvare il grafico in formato EPS (Encapsulated PostScript). È sufficiente digitare il comando:

```
>> print -deps nomefile.eps
```

- Un'altro formato sicuramente molto diffuso per chi non necessita di alta risoluzione o deve produrre documenti in formato HTML è JPEG (Joint Picture Expert Group):

```
>> print -djpeg nomefile.jpg
```

In ogni caso per un uso avanzato fare riferimento sempre all'*help* in linea:

```
>> help print
```

19 FFT e IFFT

MATLAB implementa routine di calcolo efficiente della trasformata discreta di Fourier (DFT) e della relativa antitrasformata (IDFT), disponibili con i comandi `fft` e `ifft`:

For length N input vector x , the DFT is a length N vector X , with elements

$$X(k) = \sum_{n=1}^N x(n) \exp(-j*2*\pi*(k-1)*(n-1)/N), \quad 1 \leq k \leq N.$$

The inverse DFT (computed by IFFT) is given by

$$x(n) = (1/N) \sum_{k=1}^N X(k) \exp(j*2*\pi*(k-1)*(n-1)/N), \quad 1 \leq n \leq N.$$

Ad esempio:

```
>> x = [1 0 0 0];
>> fft(x)
ans =
     1     1     1     1
```

Se il vettore x ha dimensione pari ad una potenza di 2, MATLAB utilizza algoritmi veloci (FFT) per il calcolo della DFT, altrimenti applica la formula sopra riportata a scapito di un maggior tempo di calcolo!

Per dimostrare come sia possibile realizzare operazioni complesse con un codice molto compatto, si consideri la stima dello spettro di potenza tramite il metodo del *periodogramma*. A tal fine, dato un segnale $x(t)$ a potenza finita, campionato con frequenza f_s , i cui campioni sono contenuti nel vettore $\{x_i\}_{i=1}^N$, lo spettro di potenza può essere stimato come:

$$S_x(q\Delta f) \approx \frac{1}{Nf_s} |DFT[\{x_i\}_{i=1}^N]|^2, \quad q = 1 \dots N$$

che si riduce alla riga di codice:

```
>> S=1/(N*fs)*(abs(fft(x(1:N)))).^2;
```

dove S è il vettore dei campioni dello spettro di potenza alle frequenze $0, \Delta f, 2\Delta f \dots$ con $\Delta f = f_s/N$. A questo punto se si vuole ridurre la varianza dello stimatore tramite una media temporale dei periodogrammi calcolati su blocchi disgiunti del segnale (metodo Bartlett), sarà sufficiente un ciclo `for` ...

Esistono anche comandi per la FFT bi-dimensionale (`fft2`) ed N-dimensionale (`fftn`).

20 Funzione di trasferimento di un filtro: `freqz()`

Questo comando permette di calcolare la f.d.t. di un generico filtro IIR (quindi anche FIR) una volta noti i coefficienti. Ci sono diverse forme di utilizzo di tale comando, tra queste (`help freqz`):

`H = FREQZ(B,A,F,Fs)` returns the complex frequency response at the frequencies designated in vector `F` (in Hz), where `Fs` is the sampling frequency (in Hz).

dove:

- `B` è il vettore dei pesi nel ramo diretto,
- `A` è il vettore dei pesi in retroazione ($a = 1$ per i filtri FIR!!!),
- `F` è il vettore delle frequenze nei quali si desidera calcolare la funzione di trasferimento.
- `Fs` è la frequenza di campionamento, ossia $Fs = 1/T$ con T elemento di ritardo del filtro,
- `H` il vettore complesso contenente la f.d.t. calcolata in F .

21 Filtraggio di segnali con filtri numerici: `filter()`

Mediante questo comando è possibile filtrare un segnale contenuto in un vettore. Per esempio, se si desidera filtrare il segnale i cui campioni sono contenuti nel vettore x mediante un filtro IIR (o FIR):

```
>> y = FILTER(B,A,x)
```

dove:

- `B` è il vettore dei pesi nel ramo diretto,
- `A` è il vettore dei pesi in retroazione ($a = 1$ per i filtri FIR!!!),
- `y` il vettore relativo al segnale filtrato.

Con `filter2` si possono filtrare segnali bi-dimensionali (immagini).

22 Sintesi di filtri numerici: `fir1()`

Questo comando permette di sintetizzare i pesi di filtri FIR a fase lineare (pesi simmetrici) di tipo passa-basso, passa-alto, passa-banda ed elimina-banda tramite il metodo delle finestre. Se si desidera dimensionare un filtro passa-basso con N prese (di ordine $N - 1$) e frequenza di taglio f_t :

```
>> N=5;
>> ft=1000;      % freq. taglio in Hz
>> fs=8000;     % freq. di camp. in Hz
>> b = fir1(N-1, ft/(fs/2));
b =
    0.0246    0.2344    0.4821    0.2344    0.0246
```

si noti che `fir1` accetta solo frequenze normalizzate alla frequenza di Nyquist, ossia $f_s/2$, comprese nell'intervallo $[0, 1]$.

Analogamente per filtri passa-alto:

```
>> b = fir1(N-1, ft/(fs/2), 'high')
```

per un filtro passa-banda con banda passante $[f_1, f_2]$:

```
>> b = fir1(N-1, [f1/(fs/2) f2/(fs/2)], 'bandpass')
```

per un filtro elimina-banda:

```
>> b = fir1(N-1, [f1/(fs/2) f2/(fs/2)], 'stop')
```

e per filtri con bande passanti multiple ...

In tutti i casi sopraelencati, senza esplicito riferimento alla finestra utilizzata, la sintesi si riferisce alla finestra di Hamming. Altri tipi di finestra possono essere specificati:

```
>> b = fir1(N-1, ft/(fs/2), blackman(N));
```

MATLAB implementa svariati tipi di finestre tra cui: `boxcar(N)` (rettangolare), `bartlett(N)`, `hanning(N)`, `hamming(N)`, `kaiser(N,beta)` e `chebwin(N,R)`. Esempio:

```
>> w=kaiser(50,4);
>> stem(w);
```

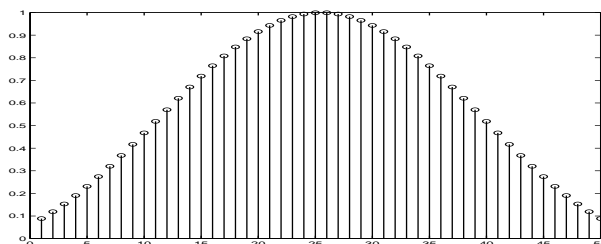


Figura 7: Finestra di Kaiser con $N = 50$ e $\beta = 4$.

23 Sintesi di filtri numerici: fir2()

Questo comando permette di sintetizzare i pesi di filtri FIR a fase lineare (pesi simmetrici) e caratteristica di ampiezza arbitraria tramite il metodo del campionamento nel dominio delle frequenze.

Per esempio si supponga di voler approssimare la seguente f.d.t.:

$$H(f) = \frac{1}{\sqrt[4]{1 - \left(\frac{f}{f_m}\right)^2}} \quad \text{per } |f| < f_m \text{ e } 0 \text{ altrove,}$$

mediante un filtro FIR ad N prese:

```

fm=0.8;           % parametro della f.d.t.
punti=500;       % punti della f.d.t.
F=0:1/punti:1;  % asse delle frequenze normalizzato

%%% campioni della f.d.t. da interpolare %%%
for i=1:punti+1,
    if F(i)<fm A(i)=1/(1-(F(i)/fm)^2)^(1/4); else A(i)=0; end
end

%%% sintesi filtro %%%
N=101;           % num. prese
b=fir2(N-1,F,A);

```

Senza esplicito riferimento alla finestra utilizzata, la risposta impulsiva ricavata tramite IDFT viene moltiplicata per la finestra di Hamming. Altri tipi di finestra possono essere specificati (`help fir2`). A titolo di esempio, in Figura 8 è riportato il confronto tra la f.d.t. desiderata e quella realizzata dal filtro FIR.

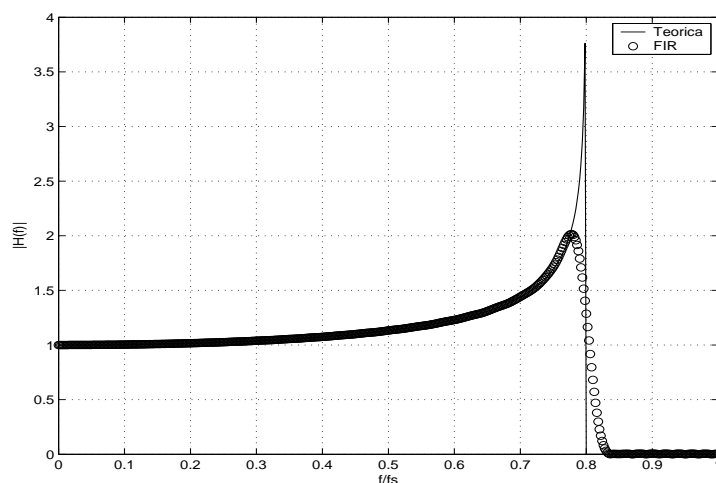


Figura 8: F.d.t. desiderata e relativa f.d.t. del filtro FIR con $N = 101$ prese.

24 Sintesi di filtri numerici: `remez()`

Questo comando permette di sintetizzare i pesi di filtri FIR *equiripple* a fase lineare (pesi simmetrici) di tipo passa-basso, passa-alto, passa-banda, elimina-banda o multibanda tramite l'algoritmo Parks-McClellan:

```
>> b=remez(N,F,A,W)
```

dove:

- **b** è il vettore dei pesi,
- **N** è l'ordine del filtro ($N+1$ pesi),
- **F** è il vettore delle frequenze che definiscono le bande passanti e le bande attenuate della funzione di trasferimento desiderata. Tali frequenze devono essere normalizzate alla frequenza di Nyquist, ossia $f_s/2$ e comprese nell'intervallo $[0, 1]$.
- **A** è il vettore dei valori della f.d.t. alle frequenze definite in **F**,
- **W** il vettore dei pesi da associare ad ogni banda (quindi ha dimensione pari alla metà di **F** o **A**).

Dunque la coppia (**F**, **A**) definisce la maschera di progetto del filtro.

Ad esempio se si desidera sintetizzare un filtro FIR passa-basso nota la sua maschera di progetto:

```
fs = 20000;      % freq. camp = 1/T
ft = 1000;      % freq. taglio
df = 2000;      % banda di transizione
N = 31;         % numero prese
d1 = 0.171;     % massimo ripple in banda passante
d2 = 0.031;     % guadagno (o ripple) in banda attenuata

F = [0 ft/(fs/2) (ft+df)/(fs/2) 1];
A = [1 1 d2 d2];
W = [d1 d1/d2];

b = remez(N-1,F,A,W);
```

Si noti che il vettore dei pesi **W** è stato utilizzato per costringere l'algoritmo a ricercare i pesi ottimi con un ripple (attenuazione) δ_2 (**d2**) in banda attenuata diverso dal ripple in banda passante δ_1 (**d1**). Se omettiamo il vettore dei pesi (ovviamente si può fare) l'algoritmo sintetizza il filtro con $\delta_1 = \delta_2$, quindi con caratteristiche più stringenti, con il rischio di richiedere un numero maggiore di prese per soddisfare le specifiche.

25 Importare ed esportare file Audio

In MATLAB è possibile importare ed esportare file audio nel formato `.wav` al fine di elaborare numericamente suoni o quant'altro. Se si vuole caricare in un vettore x i campioni contenuti nel file audio `suono.dat`:

```
>> [x,fs,nbits,opts]=wavread('telefono-DTMF.wav');
>> fs
fs =
    44100

>> nbits
nbits =
    16

>> opts
opts =
    fmt: [1x1 struct]
    info: [1x1 struct]

>> opts.fmt
ans =
    wFormatTag: 1
    nChannels: 1
    nSamplesPerSec: 44100
    nAvgBytesPerSec: 88200
    nBlockAlign: 2
    nBitsPerSample: 16
```

si noti che `wavread` fornisce anche la frequenza di campionamento, il numero di bit, il numero di canali (mono o stereo...) etc.

Analogamente è possibile esportare in formato `.wav` una qualunque sequenza con valori compresi nell'intervallo $(-1, +1)$. Ad esempio se si vuole registrare un tono con frequenza di 1KHz:

```
>> fs=8000;
>> t=0:1/fs:10000/fs;
>> x=0.5*sin(2*pi*t*1000);
>> wavwrite(x,fs,16,'tono1K.wav');
```

ottenendo così un il file `tono1K.wav` che potrà essere riprodotto da un qualunque applicativo che supporti il formato `.wav`.

Se la scheda audio lo permette si possono campionare segnali con il comando `wavrecord` ed importarli direttamente in MATLAB, oppure si possono riprodurre segnali con il comando `wavplay` (in alternativa utilizzare il comando `sound`).

26 Importare ed esportare Immagini

Elaborazione di segnali significa anche elaborazione di immagini. MATLAB implementa diverse funzioni per importare, elaborare ed esportare immagini. Per importare immagini si può utilizzare il comando:

```
>> x=imread('photo','jpg');
>> size(x)
ans =
    308    228     3
```

che restituisce una matrice x in 3 dimensioni, ovvero costituita da 3 matrici bidimensionali che rappresentano il livello dei tre colori R, G e B in formato `uint8` cioè interi (da 8 bit) da 0 a 255. Se vogliamo convertire la matrice x in `double` occorre fare un casting esplicito con:

```
>> y=double(x);
```

A questo punto l'immagine può essere elaborata numericamente. Per esempio, se vogliamo ridurre la luminosità dell'immagine, è sufficiente per esempio moltiplicare i campioni per un fattore minore di 1, ad esempio 0.5:

```
>> y=y*0.5;
```

Se vogliamo visualizzare il risultato, possiamo utilizzare il comando `image` dopo un eventuale recasting:

```
>> z=uint8(y);
>> image(z);
```

Infine, per salvare l'immagine elaborata:

```
>> imwrite(z,'photo_new.jpg');
```

Ulteriori dettagli sulla elaborazione delle immagini e sui formati supportati, fare riferimento all'`help` delle varie funzioni e alla manualistica.

27 Generare variabili aleatorie

La generazione di variabili aleatorie avviene tramite funzioni predefinite MATLAB. In particolare:

```
>> rand
ans =
    0.9501
```

restituisce un numero reale compreso nell'intervallo (0.0, 1.0), mentre:

```
>> rand(1,4)
ans =
    0.3529    0.8132    0.0099    0.1389
```

genera una matrice 1×4 di numeri casuali. Per la generazione di variabili aleatorie Gaussiane fare riferimento al comando:

```
>> randn
ans =
   -0.4326
```

che genera numeri casuali con distribuzione Normale, ossia Gaussiana con valor medio nullo e varianza unitaria. L'estensione al caso generale è semplice:

```
>> medio=10;
>> varianza=3;
>> medio + randn(1,4)*sqrt(varianza)
ans =
    7.1151    10.2171    10.4983     8.0143
```