

# Giusto un paio di note su Matlab

Federico Ramponi  
<rampo@dei.unipd.it>

## 1 Generare i segnali

### 1.1 Rumori “bianchi”

Generare un “rumore bianco” (numeri pseudocasuali) uniforme in  $[0, 1]$  (vettore riga di 100 elementi):

```
>> WN = rand(1, 100);
```

“Rumore bianco” uniforme in  $[-1, 1]$ :

```
>> WN = 2*rand(1, 100) - 1;
```

“Rumore bianco” gaussiano a media nulla e varianza unitaria:

```
>> WN = randn(1, 100);
```

“Rumore bianco” gaussiano a media  $m$  e deviazione standard  $\sigma$ :

```
>> WN = sigma*randn(1, 100) + m;
```

### 1.2 Filtraggio

Filtraggio con un filtro di funzione di trasferimento  $H(z) = \frac{B(z^{-1})}{A(z^{-1})}$ :

```
>> X = il_mio_segnales;
>> A = [a_0 a_1 a_2 a_3];
>> B = [b_0 b_1 b_2];
>> Y = filter(B, A, X);
```

In Matlab i polinomi si rappresentano tramite i vettori dei loro coefficienti; la funzione `filter` assume che la rappresentazione del filtro sia in  $z^{-1}$ , quindi il codice sopra corrisponde al filtraggio di  $X$  con la funzione di trasferimento

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}$$

Altro modo (più comodo): si possono utilizzare le rappresentazioni di sistemi del Control System Toolbox per costruire una funzione di trasferimento, e poi avviare una “simulazione” del sistema col segnale  $X$  come ingresso. Prima si costruisce il mattoncino di base:

```
>> z = tf('z')
Transfer function:
z
Sampling time: unspecified
```

e poi lo si usa per costruire funzioni di trasferimento complesse:

```
>> H = z^2/(z^2 - 0.5)
Transfer function:
z^2
-----
z^2 - 0.5
Sampling time: unspecified
```

Infine si filtra effettuando una simulazione col sistema appena costruito:

```
>> Y = lsim(H, X);
```

### Esempio: random walk

```
% Random walk:  $y(t) = y(t-1) + x(t)$ ;  $H(z) = 1/(1-z^{-1})$ 
NSAMPLES = 1000;
sigma = sqrt(0.1);
WN = sigma*randn(1, NSAMPLES);

Y1 = filter([1], [1, -1], WN); % Primo modo

z = tf('z');
H = z/(z-1);
Y2 = lsim(H, WN); % Secondo modo
```

## 2 System Identification Toolbox

Il Sysid Toolbox si presenta con un front-end grafico e con una collezione di funzioni. Per avviare l'ambientino grafico usare il comando

```
>> ident
```

e per una demo interattiva il comando

```
>> iddemo
```

(selezionare il demo 1). La dimostrazione interattiva guida l'utente passo passo nella l'identificazione di un modellino ARX a partire da dati reali, e nella relativa validazione.

Il manuale completo ed esaustivo del System Identification Toolbox, a cura di Lennart Ljung, è disponibile in PDF sul sito di Mathworks:

[http://www.mathworks.com/access/helpdesk/help/pdf\\_doc/ident/ident.pdf](http://www.mathworks.com/access/helpdesk/help/pdf_doc/ident/ident.pdf)

È bene fare costante riferimento a questa guida, e all'help in linea di Matlab, per i dettagli delle varie funzioncine.

## 2.1 iddata, arx

Vediamo un esempietto di uso di qualche funzione significativa, seguendo il demo dell'ambiente grafico. Innanzitutto carichiamo i dati:

```
>> load dryer2
```

A questo punto sono visibili (`who`) due vettori di 1000 dati ciascuno, `u2` e `y2` (stando a quel che dice il demo, si tratta di dati reali presi da un asciugacapelli. `u2` è la potenza in ingresso e `y2` è la temperatura dell'aria in uscita).

Eliminiamo dai segnali le rispettive medie:

```
>> u2d = detrend(u2, 0); % 0: rimuove solo la media
>> y2d = detrend(y2, 0);
```

Costruiamo un oggetto `iddata`, che rappresenta un "set di dati" su cui eseguire l'identificazione. La costruzione corretta richiede l'uscita, l'ingresso e *il periodo di campionamento*, che poniamo uguale a 0.08 come specificato nel demo. Per l'identificazione usiamo solo metà del segnale:

```
>> SIZE = size(u2d);
>> SAMPLE_PERIOD = 0.08;
>> data = iddata(y2d(1:SIZE/2), u2d(1:SIZE/2), SAMPLE_PERIOD);
```

Sull'oggetto appena costruito, eseguiamo l'identificazione. La funzione `arx` identifica un modello nella classe ARX con ordini prestabiliti:

$$y(t) + \sum_{k=1}^{n_A} a_k y(t-k) = \sum_{k=0}^{n_B-1} b_k y(t-k-n_K) + e(t)$$
$$\left(1 + a_1 z^{-1} + \dots + a_{n_A} z^{-n_A}\right) y(t) = \left(b_0 + b_1 z^{-1} + \dots + b_{n_B-1} z^{-(n_B-1)}\right) z^{-n_K} u(t) + e(t)$$

`arx` richiede separatamente l'ordine del modello ( $n_A$  e  $n_B$ ) e *il numero di ritardi*  $n_K$  che presupponiamo nel polinomio  $B(z^{-1})$ , inseriti in un vettore del tipo `[n_A n_B n_K]`. Per esempio prendiamo  $n_A = n_B = 4, n_K = 1$ :

```
% identifica
>> n_A = 4;
>> n_B = 4;
>> n_K = 1;
>> arx441 = arx(data, [n_A n_B n_K])
Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + e(t)
```

```

A(q) = 1 - 1.123 q^-1 + 0.09901 q^-2 + 0.2077 q^-3 - 0.0532 q^-4
B(q) = 0.001256 q^-1 + 0.003063 q^-2 + 0.06307 q^-3 + 0.05407 q^-4
Estimated using ARX from data set data
Loss function 0.00160074 and FPE 0.00165279
Sampling interval: 0.08

```

Plottiamo il diagramma di Bode [dell'equivalente a tempo continuo] per farci un'idea di cosa abbiamo trovato:

```
>> plot(arx441)
```

E usiamo la funzione `resid` per plottare la correlazione del residuo:

```
>> resid(data, arx441)
```

Se il modello è stato identificato correttamente (se il “modello vero” sta nella classe che abbiamo scelto), teoricamente il residuo dovrebbe essere bianco. `resid` calcola delle stime della funzione di correlazione del residuo, e plotta un intervallo di confidenza all'interno del quale *tutte le correlazioni eccetto la prima* (la varianza) si dovrebbero trovare. Se così risulta, possiamo ritenerci soddisfatti del modello identificato.

Per essere ancora più soddisfatti, vorremmo che lo stesso test di bianchezza funzionasse non solo sui dati con cui abbiamo identificato il modello (che erano i primi 500), ma anche sui restanti, che abbiamo conservato per la validazione:

```
>> data2 = iddata(y2d(SIZE/2:SIZE), u2d(SIZE/2:SIZE), SAMPLE_PERIOD);
>> resid(data2, arx441)
```

Esempio di classe di modelli troppo restrittiva (alcune correlazioni si trovano significativamente al di sopra della soglia):

```
>> arx111 = arx(data, [1 1 1])
>> resid(data, arx111)
```

## 2.2 armax

La funzione `armax` si usa esattamente allo stesso modo, solo che identifica un modello del tipo:

$$\begin{aligned} \left(1 + a_1 z^{-1} + \dots + a_{n_A} z^{-n_A}\right) y(t) &= \left(b_0 + b_1 z^{-1} + \dots + b_{n_B-1} z^{-(n_B-1)}\right) z^{-n_K} u(t) \\ &+ \left(1 + c_1 z^{-1} + \dots + c_{n_C} z^{-n_C}\right) e(t) \end{aligned}$$

e pertanto richiede l'ordine anche del polinomio  $C(z^{-1})$ :

```
>> n_A = 3;
>> n_B = 3;
>> n_C = 3;
>> n_K = 1;
```

```
>> armax3331 = armax(data, [n_A n_B n_C n_K])
Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + C(q)e(t)
A(q) = 1 - 1.75 q^-1 + 1.071 q^-2 - 0.2473 q^-3
B(q) = 0.001379 q^-1 - 0.0007978 q^-2 + 0.07067 q^-3
C(q) = 1 - 0.7876 q^-1 + 0.2558 q^-2 + 0.1017 q^-3
Estimated using ARMAX from data set data
Loss function 0.00159004 and FPE 0.00164967
Sampling interval: 0.08
```