INTRODUZIONE A MATLAB

Versione 1.2 (X-1998)
a cura di Michele Taragna
(c/o DAUIN-Politecnico di Torino)

Il programma MATLAB: indice

Introduzione

Il programma MATLAB è nato principalmente come programma destinato alla gestione di matrici. Le versioni successive sono state completate con serie di funzioni che permettono le più complesse analisi numeriche, adatte ad esempio all'analisi e alla soluzione di problemi di controllo.

Linea di comando di MATLAB

La linea di comando di MATLAB è indicata da un prompt come in DOS: >> . Accetta dichiarazioni di variabili, espressioni e chiamate a tutte le funzioni disponibili nel programma. Tutte le funzioni di MATLAB non sono altro che files di testo, simili a quelli che l'utente può generare con un text editor, e vengono eseguite semplicemente digitandone il nome sulla linea di comando. MATLAB permette inoltre di richiamare le ultime righe di comandi inseriti usando le frecce in alto e in basso.

Help di MATLAB

MATLAB presenta un help in linea con informazioni sulla sintassi di tutte le funzioni disponibili.

Per accedere a queste informazioni, basta digitare:

```
help nome_funzione
```

È anche possibile avere un help di tutte le funzioni di una certa categoria; ad esempio per sapere quali sono le funzioni specifiche per l'analisi ed il controllo di sistemi dinamici, basta digitare:

```
help control
```

Per sapere quali sono le varie categorie di funzioni disponibili (i cosiddetti *toolbox*), basta digitare:

```
help
```

Files di MATLAB

I files interpretati dal programma sono file di testo ASCII con estensione .m ; sono generati con un text editor e sono eseguiti in MATLAB semplicemente digitandone il nome sulla linea di comando (senza estensione!).

È possibile inserire dei commenti al loro interno precedendo ogni linea di commento col percento %

Attenzione! **Può essere molto utile andare nelle directories dove si trova il programma ed analizzare come le varie funzioni sono state implementate. Ciò è possibile poiché ogni funzione ed ogni comando MATLAB richiama un file .m**

Punteggiatura e variabili

- **Le istruzioni (siano esse contenute in un file .m lanciato da MATLAB, oppure digitate direttamente dalla linea di comando) vanno sempre terminate con un punto e virgola, altrimenti è visualizzato il risultato dell'applicazione dell'istruzione.**

Es.: `var1=6;`

Es.: `var2=linspace(-10,10,10000);`

- **Alcuni costrutti in MATLAB permettono la concatenazione di diverse istruzioni; queste vanno separate con delle virgole.**

```
Es.: if flag==0,
       istruzioni separate da virgole;
     end;
```

- **Le variabili seguono le regole dei linguaggi di programmazione come il Pascal o il C. MATLAB è *case-sensitive* e accetta nomi di variabili lunghi fino ad un massimo di 19 caratteri alfanumerici, con il primo obbligatoriamente alfabetico.**

- **Per visualizzare il contenuto di una variabile è sufficiente digitarne il nome senza punto e virgola sulla linea di comando.**

- **Tutti i calcoli effettuati in MATLAB sono eseguiti in doppia precisione, ma si possono visualizzare in un formato diverso usando i comandi:**

`format short`	**Virgola fissa con 4 decimali**
`format long`	**Virgola fissa con 15 decimali**
`format short e`	**Notazione scientifica 4 dec.**
`format long e`	**Notazione scientifica 15 dec.**

- **Il risultato dell'ultima operazione è memorizzato nella variabile `ans`.**

Comandi di uso generale

- **`who`: elenco delle variabili definite in memoria**
- **`whos`: informazioni su tutte le variabili in memoria**
- **`clear`: cancella tutte le variabili in memoria o una in particolare se specificata**
- **`save`: salva tutte le variabili in memoria sul file specificato, in vari formati**
- **`load`: richiama in memoria le variabili salvate sul file specificato**
- **`diary`: salva sul file di testo ASCII `diary` quanto da quel momento in poi appare sullo schermo**
- **`what`: elenco di tutte le funzioni MATLAB nell'area di lavoro (estensione .m) e dei file di dati che sono stati salvati (estensione .mat)**

Operatori scalari

Gli operatori disponibili sono:

- **+, -, *, /, ^,**
- **sin, cos, tan,**
- **asin, acos, atan,**
- **exp, log (naturale), log10 (in base 10),**
- **abs, sqrt, sign**

Numeri complessi

- L'unità complessa è `i` o `j` ed è predefinita =>
  NON usare `i` o `j` come variabili o indici nei cicli
- Un numero complesso si scrive nella forma `a+j*b`
  Es.: `z=2+j*3`
- Operatori applicabili a numeri complessi:
  `abs` : modulo, es. `abs(z)`
  `angle` : fase, es. `angle(z)`
  `real` : parte reale, es. `real(z)`
  `imag` : parte immaginaria, es. `imag(z)`

Matrici e loro operatori

- **L'inserimento di un vettore o di una matrice in generale viene effettuato tra parentesi quadre, separando gli elementi delle righe con spazi o virgole, e le diverse righe con punti e virgola (oppure andando a capo ad ogni nuova riga).**

  Es. di vettore riga: `x = [1, 2, 3];`

  Es. di vettore colonna: `y = [1; 4; 7];`

  Es. di matrice:
  ```
  A = [1 2 3; 4 5 6; 7 8 9];
  ```
  oppure:
  ```
  A = [1 2 3
       4 5 6
       7 8 9 ];
  ```

Per far riferimento agli elementi di una matrice A:

- **l'elemento a_{mn} è indirizzato come `A(m,n)`;**

  es. `A(2,3)` dà `6`

- **la riga m-esima è indirizzata come `A(m,:)`, dove tutte le colonne sono indicate con due punti;**

  es. `A(2,:)` dà `[4 5 6]`

- **la colonna n-esima è indirizzata come `A(:,n)`, dove tutte le righe sono indicate con due punti;**

  es. `A(:,3)` dà `[3; 6; 9]`

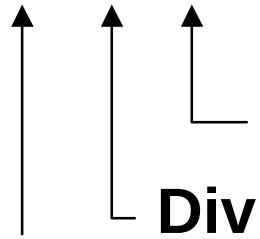
- **la sottomatrice avente elementi a_{mn}, con $m1 \le m \le m2$ e $n1 \le n \le n2$, è indirizzata come `A(m1:m2,n1:n2)`;**

  es. `A(1:2,2:3)` dà `[2, 3; 5, 6]`

Nota: in MATLAB gli indici partono sempre da 1

- **Gli operatori applicabili a matrici sono:**

  `+ - * ^ / \ '`



Trasposta

Divisione a sinistra: `A\B=inv(A)*B`

Divisione a destra: `B/A=B*inv(A)`

***Attenzione*: ricordarsi sempre che quando si fanno operazioni tra matrici è importante l'ordine dei fattori:**

`A*B ≠ B*A`

- **Altre funzioni operanti essenzialmente su vettori (riga o colonna) sono:**

  max, min, median,

  sort,

  sum, prod,

  length

- **Esistono poi particolari operatori (`.*`, `./`, `.^`) che permettono di effettuare operazioni su vettori *elemento per elemento*, senza ricorrere a cicli. Ad esempio, se `x` è un vettore, per moltiplicare elemento per elemento i due vettori `sin(x)` e `cos(x)` basta fare:**

```
y = sin(x) .* cos(x);
```

- **Altre funzioni che operano invece essenzialmente su matrici sono:**

inv	←	**inversa della matrice**
det	←	**determinante della matrice**
size	←	**dimensioni della matrice**
rank	←	**rango della matrice**
eig	←	***vedi slide seguente***

Attenzione: **tutte le funzioni che operano su matrici hanno dei vincoli sugli operandi introdotti. Ad esempio non si può invertire una matrice non quadrata. Per ulteriori spiegazioni sulla sintassi della funzione utilizzare il comando help.**

- **La funzione eig opera su matrici quadrate nel modo seguente:**

  `y=eig(A);` produce un vettore y contenente gli autovalori della matrice `A`.

  `[U,D]=eig(A);` produce una matrice `U` avente per colonne gli autovettori della matrice `A`, ed una matrice `D` diagonale avente sulla stessa gli autovalori della matrice `A`.

Esistono poi varie funzioni predefinite per la creazione di matrici:

`eye(n)` : **matrice identità n righe n colonne**

`zeros(m,n)`: **matrice di 0 con m righe e n colonne**

`ones(m,n)` : **matrice di 1 con m righe e n colonne**

`rand(m,n)` : **matrice casuale di valori tra 0 e 1**

`diag(X)` : **se `X` è un vettore con n elementi, produce una matrice quadrata diagonale di dimensione n per n con gli elementi di `X` sulla diagonale. Se invece `X` è una matrice quadrata di dimensione n per n, produce un vettore di n elementi pari a quelli sulla diagonale di `X`.**

Il comando : è usato per generare vettori riga:

- **senza specificare incremento**

  es. `t=1:5 => t=[1 2 3 4 5]`

- **con incremento positivo specificato**

  es. `t=0:0.2:1 => t=[0 0.2 0.4 0.6 0.8 1]`

- **con incremento negativo specificato**

  es. `t=2:-0.2:1 => t=[2 1.8 1.6 1.4 1.2 1]`

Polinomi

Alcune funzioni che verranno elencate in seguito necessitano di polinomi come parametri d'ingresso.

MATLAB tratta i polinomi come particolari vettori riga, i cui elementi sono i coefficienti dei monomi del polinomio in ordine di potenza decrescente. Es. il polinomio

$$S^4+111S^3+1100S^2+1000S^1+4$$

viene rappresentato come:

```
num=[1 111 1100 1000 4];
```

Per sommare o sottrarre due polinomi, si usano +,- con l'accortezza di rappresentare ambedue i due polinomi come polinomi dello stesso ordine:

La funzione conv moltiplica due vettori e quindi due polinomi. Es. il prodotto tra polinomi :

$(S^2+S^1+1)*(S^2+111S^1+1000)$

viene effettuato con:

```
prod = conv([1 1 1],[1 111 1000]);
```

che dà come risultato il vettore

```
[1 112 1112 1111 1000])
```

Per moltiplicare più di 2 polinomi, occorre utilizzare più conv in forma annidata: ad esempio il prodotto

$(S^2+S^1+1)*(S^2+111S^1+1000)*(S^2+3)$

viene effettuato mediante

```
conv([1,1,1],conv([1,111,1000],[1,0,3]));
```

- La funzione `roots` calcola le radici del polinomio.
- Es.:

```
p = [1 3 5 2];
r = roots(p);
```

In `r` sono memorizzate le radici del polinomio `p`.

- La funzione inversa è la funzione `poly`:

```
pp = poly(r);
```

In `pp` viene ripristinato il polinomio originale `p`.

- La funzione **`residue`** calcola residui e poli di una funzione di trasferimento. Es.:

```
num = [1 30 1000];
den = conv([1 1 1],[1 111 1000]);
[r,p,k] = residue(num,den);
```

In `r` sono memorizzati i residui della funzione, in `p` i poli ed in `k` l'eventuale termine aggiuntivo nel caso in cui il numero di zeri sia maggiore o uguale al numero di poli della funzione.

Istruzione IF-THEN-ELSE

- La forma generale del costrutto IF-THEN-ELSE è la stessa di un qualsiasi linguaggio di programmazione:

```
if      condizione1,
          operazioni1;
elseif  condizione2,
          operazioni2;
else
          operazioni3;
end;
```

`Condizione1,2` **devono essere condizioni che restituiscono come risultato VERO (1) o FALSO (0). Gli operatori disponibili per tali confronti sono:**

`<`, `>`

`<=`, `>=`

`==` ← **uguale**

`~=` ← **diverso**

`&` ← **and logico**

`|` ← **or logico**

`~` ← **not logico**

Operazioni1,2,3 sono le operazioni da compiere se la condizione corrispondente risulta vera. Le varie istruzioni sono separate da virgole e l'ultima è seguita da un punto e virgola.

```
if      n==10,
          a=b*c,
          d=e/f;
elseif  n~=20,
          a=e*f,
          d=a/b;
else
          disp ('Errore !!!');
end;
```

Istruzioni per cicli

I cicli si possono fare con due diversi costrutti:

```
for k = 1:step:n,
        operazioni,
end;
```

Il ciclo esegue le operazioni (separate da virgole) incrementando la variabile `k` da `1` a `n` con il passo indicato in `step`.

Oppure ...

```
while condizione,
        operazioni;
end;
```

Il ciclo esegue le operazioni (separate da virgole) fino a che la condizione è verificata. La condizione viene costruita con le stesse regole (vincoli ed operatori) di quella dell'IF-THEN-ELSE.

***Attenzione*: prevedere una inizializzazione prima del ciclo che verifichi la `condizione` per far sì che il programma entri nel ciclo, ed inoltre inserire nelle `operazioni` qualcosa che possa interagire e quindi modificare la `condizione`, altrimenti il ciclo sarà ripetuto all'infinito.**

Stringhe di testo, input e output

Il testo in MATLAB è sempre inserito tra apici (').

Es.: `string='Ciao';`

Per visualizzare stringhe o messaggi si adopera la funzione disp.

Es.: `disp('Premere un tasto');`

La funzione error mostra un messaggio di errore ed interrompe l'esecuzione di un file .m

Es.:
```
error('A deve essere simmetrica');
```

La funzione input mostra un messaggio e permette l'inserimento di dati.

Es.:
```
num_di_iter=input('Inserire il numero
                   di iterazioni: ');
```

Grafici

La funzione `plot` crea grafici bidimensionali: riceve in ingresso due vettori della stessa lunghezza e stampa i punti corrispondenti alle coordinate fornite dai due vettori. Ad esempio se si hanno due vettori x e y, il grafico corrispondente si ottiene come:

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

Per tracciare il grafico di una qualsiasi funzione, è perciò necessario crearsi un opportuno vettore da usare come ascisse, passarlo alla funzione per ricavare un vettore contenente le ordinate, ed usare la funzione `plot` sui due vettori così ottenuti. Ad esempio per tracciare la funzione sin(x) tra -4 e 4 si può usare la serie di comandi:

```
x=-4:0.01:4;
y=sin(x);
plot(x,y);
```

Se si usa la funzione `plot` con un solo parametro complesso, il grafico rappresenterà la parte reale e la parte immaginaria degli elementi del vettore: ad esempio

```
plot(y);
```

con `y` complesso, equivale a:

```
plot(real(y),imag(y));
```

Per creare grafici di colori diversi o usando caratteri diversi dal punto si può specificare dopo le coordinate una stringa di 2 elementi. Il primo è il colore del grafico, il secondo il simbolo usato per contrassegnare i punti. Ad esempio

```
plot(x,y,'g+');
```

crea un grafico in verde usando dei + al posto dei punti. Questa opzione può essere usata nei casi di grafici sovrapposti da stampare (se la stampante a disposizione non è a colori e se non si cambia il tipo di simbolo, non si capisce più nulla ...).

- **L'insieme delle scelte possibili è il seguente:**

r	red	.	point
g	green	o	circle
b	blue	x	x-mark
w	white	+	plus
m	magenta	*	star
c	cyan	-	solid
y	yellow	:	dotted
k	black	--	dashed
		-.	dash-dot

- **Altri comandi sono:**

  `grid` : sovrappone al grafico un grigliato

  `title` : aggiunge un titolo del disegno

  `xlabel` : aggiunge una legenda per l'asse x

  `ylabel` : aggiunge una legenda per l'asse y

  `axis` : riscala gli assi del grafico, es.:

  ```
  axis([xmin,xmax,ymin,ymax]);
  ```

  `clf` : cancella il grafico corrente

  `close all` : chiude tutte le finestre grafiche

- **Il comando `figure` crea una nuova finestra grafica in cui far comparire il disegno; per spostarsi sulla n-esima finestra grafica, basta digitare `figure(n)`**

Per visualizzare più grafici sulla stessa schermata si può usare la funzione `subplot`. La funzione vuole 3 parametri: il primo indica in quante parti verticali dividere lo schermo, il secondo in quante parti orizzontali, e il terzo in quale parte eseguire il plot successivo: ad esempio

```
subplot(211), plot(funz1);
subplot(212), plot(funz2);
```

crea due finestre divise da una linea orizzontale, e visualizza in quella alta il grafico di funz1, e in quella bassa quello di funz2.

Le due funzioni che possono essere utilizzate per creare vettori per le ascisse sono:

```
x = linspace(0.01,100,1000);
x = logspace(-2,2,1000);
```

La `linspace` crea un vettore `x` di 1000 elementi compreso tra 0.01 e 100 separati linearmente.

La `logspace` crea lo stesso vettore, con elementi separati logaritmicamente. Si osservi che i primi due parametri sono gli esponenti degli estremi dell'intervallo espressi in base 10.

La funzione usata per creare grafici tridimensionali è la funzione mesh: riceve in ingresso una matrice, utilizza come ascissa la coordinata di riga, come ordinata la coordinata di colonna, e come quota il valore dell'elemento individuato dalle due coordinate. Ad esempio si hanno due vettori xx e yy, si definisce il dominio della funzione mediante il comando meshdom dopodiché si può calcolare la funzione e disegnarla mediante mesh. Ad esempio per disegnare un paraboloide:

```
xx = -2:0.1:2;
yy = xx;
[x,y] = meshdom(xx,yy);
z = x .* exp(-x.^2 - y.^2);
mesh(z);
```

Curve di livello

Il grafico delle curve di livello di una funzione tridimensionale si ottiene con la funzione contour.

Riprendendo l'esempio precedente, per tracciare il grafico delle curve di livello del paraboloide, basta eseguire l'istruzione:

```
xx = -2:0.1:2;
yy = xx;
[x,y] = meshdom(xx,yy);
z = x .* exp(-x.^2 - y.^2);
h=[-.2 -.1 0 .1 .5 1];
cs=contour(z,h,xx,yy);clabel(cs);
```

Gradiente

Il comando gradient genera il grafico dei vettori indicanti il gradiente di una funzione.

Riprendendo l'esempio precedente, per tracciare il grafico del gradiente del paraboloide basta eseguire l'istruzione:

```
xx = -2:0.1:2;
yy = xx;
[x,y] = meshdom(xx,yy);
z = x .* exp(-x.^2 - y.^2);
[px,py]=gradient(z,1,1);
quiver(px,py);
```

Grafici sovrapposti

- Il comando **hold on** permette di sovrapporre due o più grafici
- Il comando **hold off** elimina tale possibilità di sovrapposizione di grafici (situazione standard)
- Ad esempio per avere sullo stesso grafico le curve di livello e i vettori gradiente basta fare (vedi slide seguente):

Esempio di sovrapposizione di grafici:

```
xx = -2:0.1:2;
yy = xx;
[x,y] = meshdom(xx,yy);
z = x .* exp(-x.^2 - y.^2);
[px,py]=gradient(z,1,1);
h=[-.2 -.1 0 .1 .5 1];
cs=contour(z,h,xx,yy); clabel(cs),
hold on,
quiver(px,py),
hold off;
```

Grafici semilogaritmici e logaritmici

- La funzione **semilogx** genera grafici con scala delle ascisse logaritmica in base 10. La sintassi è identica a quella della funzione `plot`.
- La funzione **semilogy** genera grafici con scala delle ordinate logaritmica in base 10.
- La funzione **loglog** genera grafici con entrambe le scale logaritmiche in base 10.

Definizione di nuove funzioni

- **In MATLAB è possibile creare nuove funzioni. Basta creare un file con estensione .m e nome del file uguale a quella della funzione desiderata.**
- **La prima riga del file deve contenere il nome della funzione e gli argomenti di ingresso e di uscita. Ad esempio, in**

```
function z = fun1(a,b)
```

oppure in

```
function [x,y] = fun2(a,b)
```

risulta che `fun1` e `fun2` sono nomi di funzioni; `a` e `b` sono argomenti d'ingresso; `x`, `y` e `z` sono argomenti d'uscita .

- **Il blocco di linee di commento consecutive che eventualmente segue la prima linea del file viene visualizzato digitando il comando `help` seguito dal nome della funzione creata.**
- **Le variabili utilizzate in una funzione sono *locali* e quindi indipendenti da quelle dell'ambiente chiamante.**
- **È possibile utilizzare anche variabili *globali*, a patto che vengano definite come tali sia nell'ambiente chiamante sia nella funzione, utilizzando il comando `global` seguito dai nomi delle variabili, separati da spazi. Es.:**

```
global F G H
```

Funzioni predefinite: ode23

Integra numericamente un sistema di equazioni differenziali ordinarie usando il metodo di Runge-Kutta di ordine 2 e 3. Es.:

```
[t,X]=ode23('xprimo',t0,tfinale,x0,tol);
```

integra il sistema di equazioni differenziali definito nel file `xprimo.m` sull'intervallo temporale da `t0` a `tfinale`, con condizione iniziale `x0`, garantendo l'accuratezza `tol` per la soluzione ottenuta.

La variabile `t` è un vettore colonna contenente gli istanti di integrazione, mentre `X` è una matrice la cui m-esima riga contiene i valori delle variabili del sistema di equazioni nel m-esimo istante temporale precisato in `t` (si veda anche il comando `odedemo`).

Es.: si integri numericamente il sistema di equazioni differenziali ordinarie

$$dx_1/dt = x_2$$

$$dx_2/dt = x_2*(1-(x_1)^2)-x_1$$

sull'intervallo temporale [0,20] con condizioni iniziali nulle.

Si può integrare il sistema nel modo seguente:

```
t0 = 0;          % Istante iniziale
tfinale = 20;    % Istante finale
x0 = [0,0];      % Condizione iniziale
tol = 1e-3;      % Tolleranza
[t,X] = ode23('xprimo',t0,tfinale,x0);
```

dove il file `xprimo.m` contiene la seguente definizione del sistema di equazioni differenziali:

```
function xdot = xprimo(t,x)
xdot(1) = x(2);
xdot(2) = x(2)*(1-x(1)^2)-x(1);
```

Si osservi che, nel file `xprimo.m`, la variabile `t` costituisce il generico istante d'integrazione ed è uno scalare il cui valore è determinato dalla ode23 in modo da garantire l'accuratezza `tol` desiderata.

La variabile `t` restituita dal comando ode23 contiene invece tutti gli istanti d'integrazione utilizzati ed è quindi un vettore la cui dimensione è pari al numero di volte in cui il comando ode23 ha richiamato internamente il file `xprimo.m` per calcolare la soluzione.

Funzioni predefinite: impulse

Calcola la risposta all'impulso di un sistema. Es. per tracciare il grafico della risposta all'impulso del sistema con funzione di trasferimento:

$$H(S)=(2S^2+5S+1)/(S^3+2S^2+3s)$$

basta fare:

```
num = [2 5 1];
den = [1 2 3 0];
t = linspace(0,10,1000);
plot(t,impulse(num,den,t));
```

Funzioni predefinite: step

Calcola la risposta al gradino di un sistema. Es.:

```
step(A,B,C,D,IU,t);
```

stampa la risposta al gradino di un sistema del tipo:

$$\dot{x} = A\,x + B\,u$$

$$y = C\,x + D\,u$$

cui sia stato applicato in ingresso un gradino.

Funzioni predefinite: ss2tf e tf2ss

- La funzione `ss2tf` permette il passaggio dalla rappresentazione in variabili di stato (matriciale) alla rappresentazione in funzione di trasferimento (zeri e poli).
- La funzione `tf2ss` opera il passaggio inverso.

Funzioni predefinite: bode

La funzione bode calcola e disegna il diagramma di Bode a partire sia da un sistema di equazioni di stato (usando le matrici A,B,C ...), sia dalla funzione di trasferimento del sistema. Ad esempio:

```
num = [2 5 1];
den = [1 2 3];
w = logspace(1,5,100); % Frequenze
bode(num,den,w);
```

Per le diverse varianti si utilizzi l'help di MATLAB.

Funzioni predefinite: nyquist

Permette di calcolare il diagramma di Nyquist sia a partire da un sistema di equazioni di stato (e quindi usando le matrici A,B,C ...), sia a partire dalla funzione di trasferimento del sistema.

Il funzionamento è perfettamente identico alla funzione bode della slide precedente. Per le diverse varianti si utilizzi l'help di MATLAB.

Funzioni predefinite: nichols

Permette di disegnare il diagramma di Nichols sia a partire da un sistema di equazioni di stato (e quindi usando le matrici A,B,C ...), sia a partire dalla funzione di trasferimento del sistema.

Il funzionamento è perfettamente identico a quello delle funzioni bode e nyquist. Per le diverse varianti si utilizzi l'help di MATLAB.

Con `ngrid` si sovrappone al grafico l'opportuno grigliato.

Funzioni predefinite: margin

- La funzione `margin` calcola margine di guadagno, di fase e relative frequenze a partire da un sistema di equazioni di stato (usando le matrici di stato A,B,C,D) o dalla funzione di trasferimento. Es.:

```
A = [1,3,0;2,1,0;0,2,4]; B = [1;2;5];
C = [3,0,0]; D = [1];
[Gm,Pm,Wcg,Wcp] = margin(A,B,C,D);
```

- In `Gm` c'è il margine di guadagno, in `Pm` il margine di fase, in `Wcg` e `Wcp` le frequenze corrispondenti.
- Se richiamata senza parametri di uscita, `margin` disegna il diagramma di Bode della funzione di trasferimento del sistema, mostrando il margine di guadagno, di fase e le relative frequenze.

Funzioni predefinite: rlocus

Calcola il luogo delle radici di una funzione di trasferimento. Ad esempio, per calcolare il luogo delle radici di $H(S)=(2S^2+5S+1)/(S^2+2S+3)$, si fa:

```
num = [2 5 1];
den = [1 2 3];
t = linspace(0,20,100);
r = rlocus(num,den,t);
plot(r,'.'),
xlabel('Real'),
ylabel('Imag');
```

Funzioni predefinite: cloop

- La funzione `cloop` calcola la funzione di trasferimento del sistema in catena chiusa, con retroazione unitaria negativa (se non specificato diversamente) o positiva, a partire dalla funzione di trasferimento del sistema in catena aperta. Es.:
- nel caso di retroazione unitaria negativa:

```
[numc, denc] = cloop(num,den);
```

- nel caso di retroazione unitaria positiva:

```
[numc, denc] = cloop(num,den,+1);
```

Bibliografia

- MATLAB User's Guide (Manuale)
- MATLAB Primer
- MATLAB Help
- The student edition of MATLAB (BCI 92.756)
- Matrices and MATLAB : a tutorial (BCI 93.879)
- Cavallo,Setola,Vasca, “Guida operativa a MATLAB, SIMULINK e Control Toolbox”, Liguori Editore, Napoli, 1994 (BCI 96.582)
- Tibaldi, “Note introduttive a MATLAB e Control System Toolbox”, Società editrice Esculapio, Progetto Leonardo, Bologna, 1993
- Programmi MATLAB per esercitazioni di elementi di automatica (BCI 95.602)