Matlab manuale

Calcolo numerico (Politecnico di Torino)

MATLAB

- Strumento per il calcolo scientifico:
  _ calcolatrice tascabile
  _ ambiente grafico
  _ linguaggio di programmazione
- Struttura base: MATRICE (ogni q.tà variabile viene trattata come una matrice), uno scalare reale è una matrice 1x1.
- Non è necessario dichiarare esplicitamente all'inizio del lavoro una variabile (dimensione e tipo di coefficienti).
- Già predefinito un ampio insieme di matrici elementari.
- Sono predefiniti vari operatori algebrici fra matrici (somma, prodotto, elevamento a potenza, calcolo del determinante o del rango).
- Sono predefinite funzioni di uso generale dette BUILT-IN FUNCTIONS -> risolvere problemi complessi: autovettori, autovalori, risoluzione di sistemi lineari, ricerca degli zeri di una funzione.
- TOOLBOXES = raccolte di funzioni dedicate ad uno specifico argomento.

ALL'AVVIO

Prompt >> linea da cui digitare le istruzioni nello spazio di lavoro

Comandi:

Demo -> mostra esempi significativi di possibili applicazioni del software

Doc -> introduce ad aspetti di base di Matlab e mostra quali toolboxes siano installati nella versione in uso

Help (doc) -> permette di ottenere informazioni dettagliate su qualsiasi comando. Il solo comando *help* elenca gli argomenti per i quali è disponibile la guida, suddivisi in grandi aree tematiche.

Trucchi:

- È possibile richiamare "storicamente" i comandi precedentemente digitati usando i tasti ↑,↓
- È possibile spostarsi lungo la linea di comando corrente e modificare la riga con i tasti ←,→
- È possibile completare un'istruzione già digitata in precedenza scrivendo le prime lettere e poi usare il tasto ↑.

SCALARI

Non è necessario definire e dichiarare le variabili perché vengono trattate senza distinzione fra interi, reali e complessi.

Se **a** e **b** sono due variabili scalari abbiamo: somma a+b, sottrazione a-b, prodotto a*b, divisione a/b, potenza a^b. Vale la usuale precedenza tra operazioni.

Variabili predefinite:

pi (pi greco); **i,j** (unità immaginarie), …

Tuttavia ogni variabile può essere sovrascritta. Per cancellare il valore di una variabile o riportarla al valore di default si usa il comando **clear**.

Esempio:

```
>>pi
3.1416
>>pi=5;
>> clear pi
>>pi
3.1416
```

clear all -> cancella il valore di tutte le variabili

whos -> elenca le variabili presenti nello spazio di lavoro

Formati di output

Una variabile intera viene visualizzata generalmente in un formato senza punto decimale.

Una variabile reale viene visualizzata solo con 4 cifre decimali.

```
>> sin(2)
ans =
0.9093
>> log(3)
ans =
1.0986
```

Per modificare il formato di output si può utilizzare:

- **format short** fixed point con 4 cifre decimali
- **format long** fixed point con 14 cifre decimali
- **format short e** floating point con 4 cifre decimali
- **format long e** floating point con 15 cifre decimali
- **format rat** frazione irriducibile

```
>> format long
>> log(3)
ans =
1.09861228866811

>> format short e
>>log(3)
ans =
1.0986e+000

>> format long e
>> log(3)
ans =
1.098612288668110e+000

>> format rat
>> log(3)
ans =
713/649
```

VETTORI

Per introdurre un **vettore riga** è sufficiente inserire tra [ ] i valori delle componenti del vettore stesso separati da spazi bianchi o virgole.

Esempio: per introdurre $w \in \mathbb{R}^{1x3}$:

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

oppure

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

Per introdurre un **vettore colonna** basta inserire tra [ ] i valori delle componenti del vettore stesso separati da un punto e virgola.

Esempio: per introdurre $v \in \mathbb{R}^{3x1}$:

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

Utilità

- **v = [1:10]** -> genera un vettore riga di 10 componenti dato dai valori 1, 2,…, 10.
  v = [1:.5:10] -> genera un vettore riga di 20 componenti dato dai valori 1, 1.5, 2, 2.5, …, 9.5, 10.
  SINTASSI GENERALE: **v = [valore_iniz:passo:valore_finale].** Il passo può anche essere negativo (**v = [10:-.5:1]**)
- **linspace (valore_iniz, valore_finale, N)** genera N valori equispaziati fra valore_iniz e valore_finale (estremi compresi).
  Esempio:
  ```
  >> v = linspace(0, 1, 5)
  0 0.2500 0.5000 0.7500 1.0000
  ```

- Per **accedere alla componente di un vettore** e assegnare alla variabile il valore si scrive: **z = v(3)** {N.B. l'inizializzazione parte da 1 e non da zero!}
- **end** -> parola chiave per accedere all'ultimo elemento di un vettore. (esempio: se **v** ha 10 elementi, **v(end)** = **v(10)**).
- Produce un *messaggio di errore* quando si cerca di accedere ad una componente non definita. (esempio: se **v** ha 10 elementi e vogliamo accedere a **v(11)** o **v(0)** o **v(-2)**)

- **size(v)** -> per controllare la dimensione di una variabile. È utile quando il programma segnala un conflitto di dimensioni fra quantità che si vogliono manipolare.
- **length(v)** -> restituisce la lunghezza del vettore.
- **zeros(n,1)** -> produce un vettore colonna di lunghezza *n* con elementi tutti nulli.

- **zeros(1,n)** -> produce un vettore riga di lunghezza *n* con elementi tutti nulli.
- **ones(n,1)** (**ones(1,n)**) -> genera un vettore colonna (riga) con tutte le componenti pari a 1.

OPERAZIONI SU VETTORI

Dato un vettore *v* di *n* componenti, si può calcolare:

- **v'** -> vettore trasposto (verificare le dimensioni di v'!)
- **norm(v)** -> modulo del vettore $||v|| = \sqrt{\sum_{i=1}^{n} v_i^2}$ (equivalente alla norma 2 del vettore: **norm(v,2)**).

Siano v, w due vettori riga di $\mathbb{R}^n$, con componenti v_i *e* w_i, *i* = 1,…,*n* rispettivamente. Si ha:

- **v+w** -> somma algebrica v+w = ($v_1 + w_1 + \ldots + v_n + w_n$).
- **v*w** (oppure **dot(v,w)**) -> prodotto scalare (v,w) = ($v_1w_1 + v_2w_2 + \ldots + v_nw_n$).

N.B. Attenzione alle dimensioni dei vettori !

Esistono anche delle operazioni su vettori "componente per componente", che in Matlab si eseguono usando la sintassi "punto".

Dati v, w vettori riga di $\mathbb{R}^n$, con componenti v_i *e* w_i, *i* = 1,…,*n*, si ha:

- **v.*w** -> prodotto componente per componente. Esso genera un vettore dato da (v_1w_1, v_2w_2, …, v_nw_n). Se i due vettori non hanno la stessa dimensione si genera un errore.
- **v.^m** -> elevamento a potenza componente per componente. Genera un vettore (v_1^m, v_2^m, …, v_n^m).

ISTRUZIONI DI MANIPOLAZIONE DI SOTTOBLOCCHI DI VETTORI E CONCATENAZIONE

Siano **v = [1 2 3 4 5]** e **w = [100 200]**. Per concatenare due vettori si usa la sintassi:

```
>> z=[v w]

>> z

1 2 3 4 5 100 200
```

Invece per eliminare da **v** delle componenti si usa il *vettore vuoto* **[]**:

```
>> v=[1 2 3 4 5];

>> v(3:4)=[];

>> v
```

```
1  2  5
```

Invece per sostituire alle ultime due componenti di **v** le componenti di **w**, si scrive:

```
>> v=[1  2  3  4  5];
>> w=[100  200];
>> v(end-1:end)=w;
>> v
1  2  3  100  200
```

MATRICI

Per assegnare le matrici:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

si danno rispettivamente i comandi:

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

```
>> B = zeros(2,3);
```

Possiamo calcolare

```
>> C = A+B;
```

```
>> D = A*B';
```

(attenzione alle dimensioni !)

Oppure

```
>> A = eye(5);
```

```
>> B = rand(5);
```

```
>> C = B-A;
```

```
>> s = A(1,2)+c(3,3);
```

Sia **A=eye(4)** e **B=rand(2)**. Per sostituire alle ultime due righe e colonne di **A** la matrice **B**, scriviamo:

```
>> A = eye(4); B = rand(2);
```

```
>> A(3:4,3:4)=B;
```

Per eliminare da **A** la terza colonna usiamo il vettore vuoto **[]**:

```
>> A = rand(4);
```

```
>> A(:,3)=[];
```

Per concatenare due matrici invece:

```
>> A = eye(3,2); B = zeros(3,4);
```

```
>> C = [A,B];
```

GRAFICA

Matlab è un programma che permette di rappresentare graficamente funzioni e vettori o matrici di dati. È anche possibile tracciare grafici di curve bi(tri)dimensionali, superfici e curve di livello.

- **fplot / plot** -> comando per disegnare una funzione.

```
>> fplot('sin(x)', [-pi, pi])
```

Oppure

```
>> x = [-pi:.1:pi];
>> y = sin(x);
>> plot(x,y);
```

- **help plot** -> grafici personalizzati.

Esempi:

```
>> plot(x, sin(x), '-om');
>> hold on
>> plot(x, cos(x), ':*b');
>> xlabel('asse x');
>> ylabel ('asse y'):
>> title ('Funzioni sen(x) e cos(x)');
>> legend ('sin(x)','cos(x)')
```

Grafici tridimensionali:

```
>> x = [0:2*pi/100:2*pi];
>> y = [0:2*pi/100:2*pi];
>> [X,Y] = meshgrid(x,y);
>> Z = sin(X).*cos(Y);
>> mesh (X,Y,Z)
>> surf (X,Y,Z)
>> contour (X,Y,Z)
>> contourf (X,Y,Z)
```

FUNZIONI SIMBOLICHE

Esiste una sintassi che permette di definire una funzione in modo simbolico. Ciò permette di manipolare facilmente funzioni anche molto complesse e dipendenti da più parametri.

- **inline** -> definisce una funzione "in linea", cioè direttamente nello spazio di lavoro, senza ricorrere ad un file esterno.

Esempio: funzione $f(x) = (\sin(x) + x^2)$:

`>> f = inline('(sin(x)+x).^2', 'x')` dove abbiamo indicato esplicitamente che **f** è una funzione di x.

N.B. Attenzione alla sintassi con gli apici e i punti ! Attenzione all'operazione di elevamento a potenza componente per componente.

Ad una funzione così definita non sono associati valori numerici (verificare con **whos f**). Per associare dei valori numerici bisogna scrivere:

```
>> x = 0:0.01:2*pi;
>>y = f(x);
```

La sintassi **f(x)** permette di assegnare a **f** dei valori numerici in corrispondenza degli elementi del vettore x. Tali valori numerici vengono conservati nel vettore **y** (verificare con **whos y**).

Esempio: disegnare grafico di *y = f(x)* con il semplice comando

`>> plot(x,y)` {PERCHE' NON FUNZIONA?}

ISTRUZIONI DI CONTROLLO E ISTRUZIONI CONDIZIONALI

Sintassi generale:

- ```
  if (condizione1==true)
  istruzione1
  ...
  elseif (condizione2==true)
  istruzione2
  ...
  else
  istruzione3
  ...
  end
  ```

- ```
  for contatore = start:passo:end
  istruzione
  ...
  istruzione
  end
  ```

- ```
  while (condizione==true)
  istruzione
  ...
  aggiornamento condizione
  end
  ```

  Operatori logici

  Restituiscono 1 se la condizione è vera, mentre restituiscono 0 se la condizione è falsa.

- **&** -> and
- **||** -> or
- **a==b** -> a è uguale a b?
- **a~=b** -> a è diverso da b?
  ```

Esempi di uso di istruzioni **if, for, while** con operatori logici:

```
>> n=5;
>> for i=1:n
       if (i==1)||(i==3)
            a(i)=1/i;
       else
            a(i)=1/((i-1)*(i-3));
       end
   end


>> n=7;
>> for i=1:n
       for j=1:n
            A(i,j)=1/(i+j-1);
       end
   end


>> n=10; i=1;
>> while (i<=n)
       if(i~=3)
            a(i)=1/(i-3);
       else
            a(i)=1/i;
       end
       i=i+1;
   end
```

Programmare in Matlab: M-FILE

È un file di testo in cui è possibile memorizzare le successioni di comandi, e salvato con l'estensione ".m". un *m-file* è un programma eseguibile.

Per crearne uno occorre aprire un file con l'editor del Matlab, digitare in esso istruzioni e poi salvarlo.

Gli *m-file* possono essere di due tipi:

- **script** -> definiti semplicemente da una sequenza di comandi Matlab
- **function** -> prevedono parametri di input e di output

Script

"grafico_seno.m"

```
x = [-pi:.1:pi];
y = sin(x);
plot(x,y);
```

Digitando poi sul prompt di Matlab

```
>> grafico_seno
```

Function

```
function[y1, y2,…, yn] = nome_function(x1, x2,…, xm)
```

Function "rettangolo.m"

```
function[A,p,d] = rettangolo(a,b)
A = a*b;
p = 2*(a+b);
d = sqrt(a^2 + b^2);
```

Digitando poi sul prompt di Matlab

```
>> [A,p,d] = rettangolo(2,5)
A = 10
p = 14
d = 5.3852
```

ALTRI COMANDI FONDAMENTALI DA CONOSCERE

- **diary mywork.dat** -> apre il file di testo mywork.dat nel quale viene trascritto (a partire da quel momento) il flusso delle istruzioni digitate (è una cronaca del lavoro svolto).
  diary off -> interrompe la scrittura della cronaca e chiude il file mywork.dat.
- **whos** -> elenca le variabili attualmente attive in memoria e dà alcune informazioni importanti sulle loro caratteristiche (tipo di oggetto, dimensioni in memoria,…).
- **save area.mat** -> permette di salvare nel file binario area.mat il contenuto di tutte le variabili attive in memoria in quel momento.
- **save area.mat z x** -> salva le sole variabili z e x.
- **load area.mat** -> ricarica le variabili salvate nel file area.mat e le rende attive in memoria (verificare con **whos**).
- **quit** -> termina la sessione di lavoro e chiude Matlab.

MANUALETTO DI MATLAB

COMANDI D'AVVIO

Per eseguire un comando digitato occorre premere il tato di invio. Per terminare la sessione di lavoro occorre digitare il comando `exit` oppure `quit`.

COMANDI PER GESTIRE UNA SESSIONE DI LAVORO:

`help:` per visualizzare tutti gli argomenti presenti

`help arg:` per visualizzare informazioni su arg

`doc arg:` per visualizzare informazioni dettagliate su arg

`clc:` per cancellare il contenuto della finestra di lavoro

`; :` per non visualizzare il risultato di un'istruzione

`... :` per continuare a scrivere un'istruzione nella riga successiva

`who:` per visualizzare le variabili in memoria

`whos:` per visualizzare informazioni sulle variabili poste in memoria

`clear:` per cancellare tutte le variabili dalla memoria

`clear var1 var2:` per cancellare le var1 e var2 dalla memoria

VARIABILI IN MATLAB

La lunghezza massima dei nomi delle variabili è di 32 caratteri, che comprendono lettere, numeri e "_". Matlab distingue tra lettere maiuscole e minuscole.

```
>> nome_variabile=espressione
```

Se la variabile è di tipo *stringa* occorre racchiudere `espressione` tra una coppia di apici.

16 cifre significative, ma in output generalmente una variabile intera viene visualizzata senza punto decimale, mentre una variabile reale viene visualizzata con 4 cifre decimali.

ALCUNE VARIABILI PREDEFINITE:

ans: variabile temporanea che contiene il risultato più recente

i, j: unità immaginaria

pi: π, 3.14159265

eps: $epsilon$ di macchina

realmax: massimo numero di macchina positivo

realmin: minimo numero di macchina positivo

Inf: ∞, ossia un numero maggiore di `realmax` oppure il risultato di 1/0

NaN: Not a Number (ad esempio il risultato di 0/0)

ALCUNE FUNZIONI PREDEFINITE:

sin

cos

asin

acos

tan

atan

exp

log

log2

log10

sqrt

abs

real: parte reale

imag: parte immaginaria

sign: funzione segno

factorial: fattoriale

round: arrotonda all'intero più vicino

floor: arrotonda per difetto all'interno più vicino

ceil: arrotonda per eccesso all'intero più vicino

chop(x,t): arrotonda `x` a `t` cifre significative

ALCUNI POSSIBILI FORMATI DI OUTPUT:

format: formato di default, equivalente a `format short`

format short: rappresentazione fixed point con 4 cifre decimali

format long: rappresentazione fixed-point con 14 cifre decimali

format short e: rappresentazione floating-point con 4 cifre decimali

format long e: rappresentazione floating-point con 15 cifre decimali

format rat: rappresentazione sottoforma di frazione irriducibile

Gli elementi di un vettore vanno digitati `[]`; gli elementi di un vettore riga vanno separati con uno spazio oppure ',', quelli di un vettore colonna con un ';' oppure premendo il tasto invio dopo l'introduzione di ogni elemento. Non è possibile utilizzare indici nulli o negativi per le componenti di un vettore.

x(i) : individua l'i-esimo componente del vettore `x`

x(end) : individua l'ultimo elemento del vettore `x`

length(x) : determina la lunghezza del vettore `x`

ALCUNI COMANDI PER GENERARE E MANIPOLARE VETTORI:

x' : genera il vettore trasposto di `x`

x=[] : genera il vettore vuoto `x`

sort(x) : riordina in ordine crescente le componenti del vettore `x`

x=[a:h:b] : genera il vettore riga $x = (x_i)_{i=1,\dots,m+1}$ ove $x_i = a + (i-1)h$ e m è la parte intera di $(b-a)/h$

x=linspace(a,b,n) : genera il vettore riga $x = (x_i)_{i=1,\dots,n}$ ove $x_i = a + (i-1)h$ e $h = (b-a)/(n-1)$

x=logspace(a,b,n) : genera il vettore riga $x = (10^{x_i})_{i=1,\dots,n}$ ove $x_i = a + (i-1)h$ e $h = (b-a)/(n-1)$

x(r) : estrae le componenti del vettore `x` i cui indici sono specificati in `r`

x(r)=z : assegna alle componenti del vettore `x` (i cui indici sono specificati in `r`) i valori definiti in `z` rispettivamente

x(r)=[] : rimuove le componenti del vettore `x` (i cui indici sono specificati in `r`)

x([i j])=x([j i]) : scambia le componenti `i` e `j` del vettore `x`

ALCUNE FUNZIONI PREDEFINITE AGENTI SU UN VETTORE `x`:

`a=sum(x)`: genera lo scalare $a = \sum_{i=1}^{n} x_i$

`a=prod(x)`: genera lo scalare $a = \prod_{i=1}^{n} x_i$

`a=max(x)`: genera lo scalare $a = max_i x_i$

`a=min(x)`: genera lo scalare $a = min_i x_i$

`a=norm(x)`: genera lo scalare $a = ||x||_2$

`a=norm(x,1)`: genera lo scalare $a = ||x||_1$

`a=norm(x,inf)`: genera lo scalare $a = ||x||_\infty$

`A=diag(x)`: genera la matrice diagonale $A = (a_{ij})_{i,j=1,\dots,n}$, con $a_{ii} = x_i$

`diag oppure diag(x,k)`: genera una matrice quadrata di dimensione $n+|k|$ con tutti gli elementi uguali a zero tranne quelli della k-esima diagonale sopra (k>0) oppure sotto (k<0) la diagonale principale, che coincidono con gli elementi del vettore x.

Gli elementi di una matrice vanno digitati tra [], procedendo per righe e terminando ciascuna riga con ';' o premendo invio.

`A(i,j)`: individua l'elemento di posto (i,j)

`size(A)`: genera un vettore riga contenente il numero di righe e di colonne della matrice **`A`**

`length(A)`: applicato ad una matrice equivale a calcolare `max(size(A))`

ALCUNI COMANDI PER GENERARE E MANIPOLARE MATRICI:

`A=[]`: genera la matrice vuota A

`A'`: genera la matrice trasposta di A

`A=eye(n)`: genera la matrice identità $A = (a_{ij})_{i,j=1,\dots,n}$, con $a_{ij} = \delta_{ij}$

`A=zeros(n,m)`: genera la matrice $A = (a_{ij})_{i=1,\dots,n,j=1,\dots m}$, con $a_{ij} = 0$

`A=ones(n,m)`: genera la matrice $A = (a_{ij})_{i=1,\dots,n,j=1,\dots m}$, con $a_{ij} = 1$

`A=rand(n,m)`: genera la matrice $A = (a_{ij})_{i=1,\dots,n,j=1,\dots m}$, con $0 < a_{ij} < 1$ pseudo-casuali

`A=hilb(n)`: genera la matrice di Hilbert $A = (a_{ij})_{i,j=1,\dots n}$, con $a_{ij} = 1/(i+j-1)$

A=vander(x) : genera la matrice di Vandermonde $A = (a_{ij})_{i,j=1,\dots n}$, con $a_{ij} = x_i^{n-j}$

A(r,c) : estrae gli elementi di `A` appartenenti all'intersezione delle righe e delle colonne specificate in `r` e in `c` rispettivamente

A(r,c)=C: assegna agli elementi di `A` (i cui indici di riga e colonna sono specificati in `r` e `c`) i valori definiti in `C` rispettivamente

A(r,c)=[] : rimuove gli elementi di `A` (i cui indici di riga e di colonna sono specificati in `r` e `c`)

A([i j],c)=A([j i],c) : scambia gli elementi delle righe `i` e `j` di `A` appartenenti alle colonne specificate in `c`

A(r,[i j])=A(r,[j i]) : scambia gli elementi delle colonne `i` e `j` di `A` appartenenti alle righe specificate in `r`

ALCUNE FUNZIONI PREDEFINITE AGENTI SU UNA MATRICE `A`:

a=norm(A) : genera lo scalare $a = ||A||_2$

a=norm(A,1) : genera lo scalare $a = ||A||_1$

a=norm(A,inf) : genera lo scalare $a = ||A||_\infty$

x=sum(A) : genera il vettore riga $x = (x_j)_{j=1,\dots,n}$, con $x_j = \sum_{i=1}^{n} a_{ij}$

x=max(A) : genera il vettore riga $x = (x_j)_{j=1,\dots,n}$, con $x_j = max_i a_{ij}$

x=min(A) : genera il vettore riga $x = (x_j)_{j=1,\dots,n}$, con $x_j = min_i a_{ij}$

x=diag(A) : genera il vettore colonna $x = (x_i)_{i=1,\dots,n}$, con $x_i = a_{ii}$

B=abs(A) : genera la matrice $B = (b_{ij})_{i,j=1,\dots,n}$, con $b_{ij} = |a_{ij}|$

B=tril(A) : genera la matrice triangolare inferiore $B = (b_{ij})_{i,j=1,\dots,n}$, con $b_{ij} = a_{ij},\ \ i = 1,\dots,n,\ \ 1 \le j \le i$

B=triu(A) : genera la matrice triangolare superiore $B = (b_{ij})_{i,j=1,\dots,n}$, con $b_{ij} = a_{ij},\ \ i = 1,\dots,n,\ \ i \le j \le n$

Queste funzioni si possono applicare anche a matrici rettangolari.

Inoltre le funzioni `sum`, `max` e `min` possono essere utilizzate anche nella forma **sum(A,k)**, **max(A,[],k)** e **min(A,[],k)**, con $k = 1, 2$. Per $k = 1$ agiscono come

scritto sopra. Invece, per $k = 2$ generano un vettore colonna $x = (x_i)_{i=1,\dots,n}$, con $x_i = \sum_{j=1}^{n} a_{ij}$, $x_i = max_j a_{ij}$ e $x_i = min_j a_{ij}$, rispettivamente.

La funzione `diag` può essere utilizzata nella forma **`diag(A,k)`**, con `k` intero positivo o negativo; in questo caso essa genera un vettore colonna coincidente con la k-esima diagonale sopra (k>0) oppure sotto la diagonale principale.

Anche la funzione `tril/triu` può essere utilizzata nella forma **`tril(A,k)/triu(A,k)`**, con k intero positivo o negativo; in questo caso essa estrae la parte triangolare inferiore/superiore a partire dalla k-esima diagonale sopra (k>0) oppure sotto (k<0) la diagonale principale.

L'operazione **`*`** esegue il prodotto righe per colonne.

CASO: matrici quadrate

`A^k`: con k intero positivo -> prodotto della matrice A per se stessa k volte

`A^(-1)`: genera l'inversa della matrice A, ammesso che A sia *non singolare*

Tra le operazioni tra matici bisogna considerare anche le *operazioni puntuali*, che agiscono direttamente sui singoli elementi. Tali operazioni si definiscono premettendo "." al simbolo che identifica l'operazione.

OPERAZIONI PUNTUALI:

`z=x.*y` : genera il vettore riga (colonna) $z = \{z_i\}_{i=1,\dots,n}$, con $z_i = x_i * y_i$

`z=x./y` : genera il vettore riga (colonna) $z = \{z_i\}_{i=1,\dots,n}$, con $z_i = x_i / y_i$

`z=x.^y` : genera il vettore riga (colonna) $z = \{z_i\}_{i=1,\dots,n}$, con $z_i = x_i^{y_i}$

`z=x.^e` : genera il vettore riga (colonna) $z = \{z_i\}_{i=1,\dots,n}$, con $z_i = x_i^{e}$

`C=A.*B` : genera la matrice $C = \{c_{ij}\}_{i,j=1,\dots,n}$, con $c_{ij} = a_{ij} * b_{ij}$

`C=A./B` : genera la matrice $C = \{c_{ij}\}_{i,j=1,\dots,n}$, con $c_{ij} = a_{ij} / b_{ij}$

`C=A.^B` : genera la matrice $C = \{c_{ij}\}_{i,j=1,\dots,n}$, con $c_{ij} = {a_{ij}}^{b_{ij}}$

`C=A.^e` : genera la matrice $C = \{c_{ij}\}_{i,j=1,\dots,n}$, con $c_{ij} = {a_{ij}}^{e}$

LA GRAFICA IN MATLAB

Per disegnare una funzione f della variabile x:

`fplot('f',[xmin xmax])`

dove: **`f`** è l'espressione della funzione che si vuole rappresentare, e **`[xmin xmax]`** è un vettore che ha per componenti gli estremi dell'intervallo del dominio. Se si vuole stabilire

anche l'immagine occorre fornire il vettore **[xmin xmax ymin ymax]** come secondo argomento della funzione `fplot`.

Altrimenti si può usare il comando: **plot(x,y)**, che consiste nel definire un vettore `x` di punti dell'asse delle `x`, generare il vettore `y` contenente le valutazioni della funzione f nei punti precisati in `x`.

ALCUNE POSSIBILI OPZIONI PER I COMANDI `plot e fplot`:

colore:

w: bianco

y: giallo

r: rosso

g: verde

b: blu

k: nero

simbolo:

. : punto

° : circoletto

x : per

+ : più

***** : asterisco

s : quadratino

linea:

- : linea continua

: : linea punteggiata

-. : linea tratto-punto

-- : linea tratteggiata

ALCUNI POSSIBILI COMANDI PER COMMENTARE UN GRAFICO:

title: inserisce un titolo nel grafico

xlabel: inserisce un nome per l'asse x

ylabel: inserisce un nome per l'asse y

grid: inserisce una griglia sugli assi x e y

legend: inserisce una legenda per identificare rappresentazioni diverse

text: inserisce una stringa di testo in una specificata posizione

gtext: inserisce una stringa di testo in una posizione individuata tramite mouse

Si può anche considerare una scala logaritmica sugli assi, utilizzando il comando **semilogx, semilogy, loglog** al posto del comando `plot`.

Per disegnare più grafici nella stessa finestra si usa il comando **hold on** prima di disegnare il grafico da sovrapporre a quello già tracciato, oppure si usa il comando `plot` nella forma **plot(x_1,y_1,'-',x_2,y_2,':')**.

Invece per disegnare grafici diversi in una stessa finestra grafica ma in sottofinestre separate si usa il comando **`subplot(righe,colonne,sottofinestra)`**

dove `righe` e `colonne` indicano la matrice di sottofinestre della finestra grafica principale, e `sottofinestra` indica il numero della sottofinestra che si vuole attivare per disegnarci il grafico. (vengono numerate da sx a dx, dall'alto al basso)

`figure(n)`: comando per attivare la finestra grafica `n`

`close(n)`: comando per chiudere la finestra `n`; **`close all`:** per chiudere tutte le finestre attive

PROGRAMMI MATLAB

Un file contenente istruzioni si chiama ***m-file*** perché deve essere salvato con l'estensione "**`.m`**". Il nome può essere definito mediante lettere, numeri e '_' .

I comandi vanno digitati su righe differenti o su una stessa riga purché separati da ';' o da ','. Per introdurre un commento si deve usare %

2 tipi:

script: definiti da una sequenza di comandi Matlab. Per eseguirlo occorre selezionare la directory in cui l'*m-file* è stato salvato e digitarne il nome (senza estensione) al prompt. Non prevendono un passaggio di parametri di input e output, inoltre le variabili qui definite rimangono nella memoria della sessione di lavoro, come se fossero state definite direttamente al prompt.

function: devono iniziare necessariamente con **`function[y_1,y_2,…,y_n]=nome_function(x_1,x_2,…,x_m)`**

dove `y_1,y_2,…,y_n` sono i parametri di output, e `x_1,x_2,…,x_m` sono quelli di input. `nome_function` deve coincidere con il nome dell'*m-file* in cui è stata salvata la *function*. Per eseguirla da prompt, o all'interno di uno script o di un'altra *function* si scrive:

`[y_1,y_2,…,y_n]=nome_function(x_1,x_2,…,x_m)`

oppure

`nome_function(x_1,x_2,…,x_m)` -> restituisce solo il primo parametro di output che viene salvato in **`ans`**.

Se non si assegna un nome alle variabili di output Matlab restituisce questo messaggio **`??? One or more output arguments not assigned during call to 'nome_function'`**.

Prevedono dei parametri di input e output e le variabili utilizzate vengono trattate come variabili locali e vengono automaticamente cancellate dalla memoria alla fine dell'esecuzione della *function*.

PROGRAMMARE: costrutti sintattici

OPERATORI RELAZIONALI:

< minore

> maggiore

<= minore o uguale

>= maggiore o uguale

== uguale

~= non uguale

È possibile fare confronti tra espressioni, Matlab non disponendo di variabili di tipo logico, assegna un valore numerico al risultato di un confronto.

0 -> valore falso

1(o qualsiasi altro numero) -> valore vero

OPERATORI LOGICI:

`&`: and

`|`: or

`~`: not

`xor`: or esclusivo

AZIONE DEGLI OPERATORI LOGICI SU DUE CONDIZIONI a E `b`:

a	b	a&b	a|b	~a	xor(a,b)
0	0	0	0	1	0
1	0	0	1	0	1
0	1	0	1	1	1
1	1	1	1	0	0

Strutture di programmazione elementari:

- ciclo incondizionato controllato da un contatore

```
for indice=espressione
    blocco di istruzioni
```

```
end
```
dove `indice` è una q.tà che assume i valori definiti da `espressione` a dx dell'uguale.

- ciclo condizionato
```
while condizione
     blocco di istruzioni
end
```
dove condizione è un'espressione che se interpretata come vera assume il valore diverso da 0, come falsa se assume il valore 0.

- strutture condizionali
```
if condizione_1
     blocco di istruzioni
elseif condizione_2
     blocco di istruzioni
     .
     .
     .
else
     blocco di istruzioni
end
```
dove il primo `blocco di istruzioni` verrà eseguito solo se la `condizione_1` risulta vera, il secondo solo se `condizione_1` risulta falsa e `condizione_2` vera e così via.
Il blocco che segue `else` verrà eseguito solo se le condizioni precedenti non risultano vere.

`return:` permette di terminare l'esecuzione del programma prima che si raggiunga l'ultima istruzione

`break:` permette di uscire in maniera forzata da un ciclo, saltando direttamente all'istruzione `end`

`tic/toc:` comando per valutare l'efficienza di un programma in termini di tempo d'esecuzione espresso in secondi.

`tic` -> attiva il timer

```
     calcolo;
```

`toc` -> arresta il timer e restituisce l'"`elapsed_time`"

PRINCIPALI FUNCTION MATLAB PER PROBLEMI DI CALCOLO NUMERICO

ALGEBRA LINEARE:

lu: genera la fattorizzazione di Gauss con pivoting parziale

chol: genera la fattorizzazione di Choleski

qr: genera la fattorizzazione QR

x=A/b: risolve il sistema lineare Ax=b

cond(A): calcola il numero di condizionamento spettrale (in norma 2) di A

cond(A,1): calcola il numero di condizionamento in norma 1 di A

cond(A,inf): calcola il numero di condizionamento in norma ∞ di A

rcond(A): calcola il reciproco del numero di condizionamento in norma 1 di A

rank(A): calcola il rango di A

det(A): calcola il determinante di A

inv(A): calcola l'inversa di A

eig: calcola gli autovalori e autovettori di A

POLINOMI, FUNZIONI E APPROSSIMAZIONE:

polyval: valuta un polinomio

f=inline('espressione', 'x_1',…, 'x_n'): definisce la funzione $f(x_1, \dots, x_n) = espres$.

y=f(x_1, …, x_n): valuta la funzione $y = f(x_1, \dots, x_n)$ definita mediante `inline`

y=feval(f, x_1,…, x_n): valuta la funzione $y = f(x_1, \dots, x_n)$ definita mediante `inline` oppure mediante una *function*

polyfit: calcola i coefficienti del polinomio interpolante o approssimante nel senso dei minimi quadrati

spline: valuta una spline cubica interpolante

EQUAZIONI E SISTEMI DI EQUAZIONI NON LINEARI:

`fzero:` calcola gli zeri di una funzione non lineare

`roots:` calcola gli zeri di un polinomio

`fsolve:` risolve un sistema di equazioni non lineari

CALCOLO DI INTEGRALI:

`quad:` formula di Simpson adattiva

`quadl:` formula di Gauss-Lobatto adattiva

EQUAZIONI E SISTEMI DI EQUAZIONI DIFFERENZIALI ORDINARIE:

`ode45:` Runge-Kutta esplicito di ordine 4 e 5

`ode113:` Adams-Moulton di ordine variabile

`ode23:` Runge-Kutta esplicito di ordine 2 e 3

`ode23t:` trapezi

`ode15s:` multistep lineare implicito di ordine variabile

`ode23s:` Runge-Kutta implicito di ordine 2