18/11/2013

Informatica per la Finanza
Informatica per le Scienze Economiche
Informatica e Programmazione per il Turismo

Parte 2

Matlab e la programmazione vettoriale

Programma

- Introduzione all'uso di MatLab:
  - il tipo matrice e suoi manipolatori,
  - funzioni predefinite,
  - programmare con MatLab,
  - *creazione di grafici*.

Bibliografia

- Esistono numerosi siti che forniscono esempi e guide.
  - G. Ciaburro, MATLAB Guida all'uso, edizioni FAG
    - Una sintetica ed economica guida.
  - W. J. Palm III, MATLAB introduzione per gli Ingegneri, McGraw Hill
  - G. Naldi, L. Pareschi, Matlab Concetti e progetti, Apogeo
  - La documentazione di MatLab è gratuita:
    www.mathworks.com/access/helpdesk/help/techdoc/index.html
    - **Ben fatto è anche l'help in linea**

Matlab/Octave

- Un clone (fa molte cose come Matlab) è Octave:

  http://www.gnu.org/software/octave

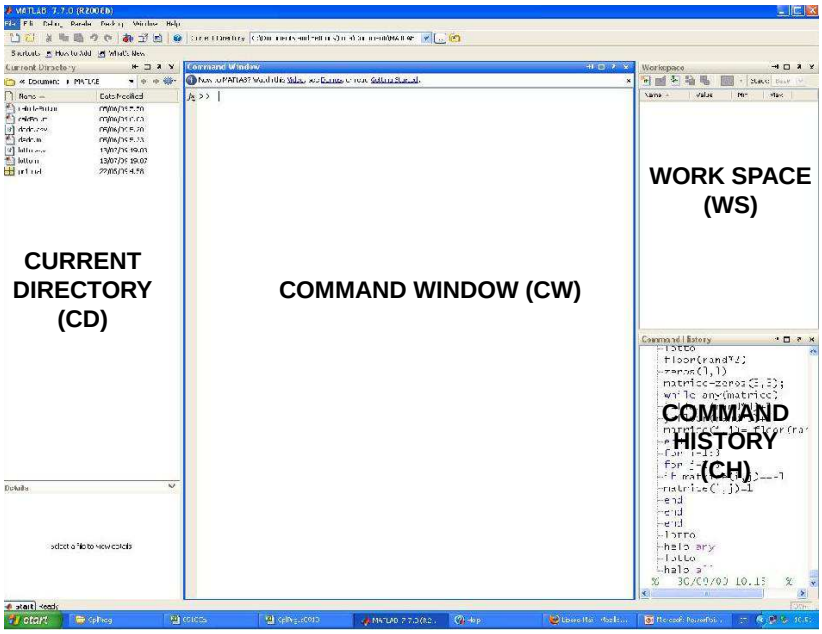
  – Per un interfaccia grafica simile a quella di Matlab far riferimento a:
    - QOcTerm
    - QtOctave
  – Attenzione però che tutte le interfacce hanno dei problemi

MATLAB: introduzione

- MATLAB è un ambiente interattivo programmabile per la manipolazione di quantità scalari e vettoriali
- MATLAB offre la possibilità di
  – salvare e recuperare sessioni di lavoro (attraverso `m-file`)
  – creare grafici a partire dai dati manipolati
- Toolbox (librerie) specifici ne estendono le potenzialità

MATLAB: introduzione

- MATLAB presenta una interfaccia grafica per il suo utilizzo che comprende diverse parti (finestre):
  - **`Command Window (CW)`**: dove si inseriscono i comandi
  - **`Workspace (WS)`**: contiene le variabili definite nella sessione di lavoro con le loro proprietà
  - **`Current Directory (CD)`**: indica la cartella di lavoro corrente
  - **`Command History (CH)`**: contiene l'elenco dei comandi digitati



Comandi utili

- Per eliminare una variabile dal WS:
  - **`clear`** ***`nomeVariabile`***
    - *`clear`* non seguito da *`nomeVariabile`* elimina tutte le variabili!
- Per pulire la zona del WS (senza cancellare le variabili)
  - **`clc`**
- Per avere un elenco delle variabili definite:
  - **`who`**
    - Con più dettagli **`whos`**
- Per avere una spiegazione su di una funzione:
  - **`Help`** ***`nomeFunzione`***

MATLAB: introduzione

- MATLAB lavora fondamentalmente con le matrici (ovvero lavora su quantità vettoriali)
  - Le matrici 1 x 1 rappresentano le quantità scalari (i numeri, i caratteri)
  - Le matrici 1 x n i vettori riga
  - Le matrici n x 1 i vettori colonna
- Il linguaggio distingue fra maiuscole e minuscole
- Se non è indicato il nome della variabile su cui poggiare il risultato usa la variabile **`ans`**

Tipi di dati

- Esistono diversi tipi di dati numerici (tipo `NUMERIC`):
  - Interi con segno o senza ad 8, 16, 32, 64 bit
  - Virgola mobile a singola (`single`) e doppia (`double`) precisione
    - Se non esplicitamente convertiti con apposite funzioni tutti i dati numerici sono rappresentati in virgola mobile a doppia precisione
- Esiste anche la possibilità di trattare i numeri complessi

Definire una matrice

- Nella finestra CW dopo il prompt (**>>**) si scrive ad esempio:

```
>> X = [1 2]
```

  Si è definita in questo modo:
  - Una matrice 1X2 di nome X

```
>> Y = [9;10]
```

  Si è definita in questo modo:
  - Una matrice 2 X1 di nome Y
    - Si poteva ottenere lo stesso risultato utilizzando l'operatore di trasposizione su di una matrice riga
- Per riferirsi ad una posizione della matrice si scrive il nome della matrice mettendo fra parentesi tonde, separate da virgola la coordinata di riga e colonna:

```
>> X(1,1)
```

Operazioni

- In MatLab esistono due concetti spesso usati come sinonimi in altri contesti
  - **Matrici** (Matrix) intese nel senso dell'algebra lineare come trasformazioni lineari su cui si applicano gli operatori dell'algebra lineare
  - **Array** (Vettori) intesi come insiemi ordinati numerici a più dimensioni su cui si applicano operatori elemento per elemento

Operazioni sulle matrici

Trasposizione

- Scambia le righe con le colonne
- Operatore MATLAB: '

Nota: se le componenti sono numeri complessi ne calcola il coniugato

```
>> x=[1 2; 7 8]
x =

     1     2
     7     8

>> x'
ans =

     1     7
     2     8
```

Operazioni sulle matrici

Addizione

- Le matrici debbono avere le stesse dimensioni
- Si esegue l'operazione elemento per elemento
- Operatore MATLAB: **+**

Date le matrici:

```
y=[2 3; 6 -10.2]
x=[1 2; 7 8]

x + y

ans =

    3.0000    5.0000
   13.0000   -2.2000
```


Operazioni sulle matrici

Sottrazione

- Le matrici debbono avere le stesse dimensioni
- Si esegue l'operazione elemento per elemento
- Operatore MATLAB: **-**

Date le matrici:

```
y=[2 3; 6 -10.2]
x=[1 2; 7 8]

y - x

ans =

    1.0000    1.0000
   -1.0000  -18.2000
```

Operazioni sulle matrici

Moltiplicazione

- Data una matrice **A** *NxM* ed una matrice **B** *MxK* il prodotto righe per colonne fra A e B produce una matrice **C** *NxK*
- Operatore MATLAB: *****
- In questo modo si ottiene anche il prodotto scalare di due vettori
- Si ricorda che il prodotto NON è commutativo

Date le matrici:

```
a = [1 2 3; 4 5 6]
b = [8 9;10 11; 12 13]

>> a * b
ans =

    64    70
   154   169

>> b * a
ans =

    44    61    78
    54    75    96
    64    89   114
```


Esempi

```
M=[ 1 2 ;          N=[10  20 30;
    3 4]                40 50 60]
```

- **M * N** 2x3

```
[1*10+2*40   1*20+2*50   1*30+2*60
 3*10+4*40   3*20+4*50   3*30+4*60]
```

- **N * M NON SONO CONFORMI**
  - **Non si può moltiplicare una matrice 2x3 con una 2X2**

Operazioni sulle matrici

Divisione destra

- Data una matrice **A** ed una matrice **B** la divisione destra fra **A** e **B** indicata con **/** equivale a

  $((B^{-1})^T*A^T)^T$

- e produce una matrice **C**
- Operatore MATLAB: **/**
- Si ricorda che l'operatore NON è commutativo

Date le matrici:

```
y=[2 3; 6 -10.2]
x=[1 2; 7 8]

>> x/y
ans =

    0.5781   -0.0260
    3.1094    0.1302

>> y/x
ans =

    0.8333    0.1667
  -19.9000    3.7000
```


Operazioni sulle matrici

Divisione sinistra

- Data una matrice **A** ed una matrice **B** la divisione sinistra fra **A** e **B** indicata con \ equivale a

  $A^{-1}*B$

  e produce una matrice **C**
- Operatore MATLAB: \
- Si ricorda che l'operatore NON è commutativo

Date le matrici:

```
y=[2 3; 6 -10.2]
x=[1 2; 7 8]

>> x\y
ans =

   -0.6667   -7.4000
    1.3333    5.2000

>> y\x
ans =

    0.8125    1.1563
   -0.2083   -0.1042
```

Operazioni sulle matrici

- I sistemi di equazioni lineari hanno forma:

  A x = b

  - Dove A indica la matrice dei coefficienti delle incognite e b il vettore con i termini noti
  - L'operatore \ permette di trovare il vettore **x** soluzione del sistema:

    >> x = A \ b

- Dato il sistema:

$$\begin{cases} 2x - 5y + 4z = -3 \\ x - 2y + z = 5 \\ x - 4y + 6z = 6 \end{cases}$$

- Si traduce in:

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

- **x = A \ b** calcola :

```
x =
  112.0000
   67.0000
   27.0000
```


Operazioni sulle matrici

Potenza

- Data una matrice **A** *MxM* ed un intero n la potenza equivale ad n volte il prodotto della matrice A con sé stessa.
- Se n è negativo moltiplica n volte la matrice inversa di A con sé stessa
- Operatore MATLAB: **^**

NOTA: la matrice identità viene definita mediante la funzione:

eye()

- Specificando due dimensioni:
  - **eye(2,3)** definisce una matrice identità 2x3
- Specificando una sola dimensione si ottiene una matrice quadrata
  - **eye(3)** definisce una matrice identità 3x3

Date le matrici:

```
y=[2 3; 6 -10.2]
x=[1 2; 7 8]

>> x^3
ans =
   141   174
   609   750

>> y^-3
ans =
    0.0246    0.0056
    0.0112    0.0018

>> y^0
ans =
     1     0
     0     1
```

Operazioni sugli array

Addizione, Sottrazione

- Gli array debbono avere le stesse dimensioni
- Si esegue l'operazione elemento per elemento
- Operatori MATLAB: **+ -**

Nota: sono le stesse operazioni che si compiono sulle matrici

Dati due array:

```
xV = [1 2 3; -4 -5 -6]
yV = [7 8 9; 10 11 12]

>> xV + yV

ans =
     8    10    12
     6     6     6

>> xV - yV

ans =
    -6    -6    -6
   -14   -16   -18
```


Operazioni sugli array

Moltiplicazione

- Le matrici debbono avere le stesse dimensioni
- Si esegue l'operazione elemento per elemento
- Operatore MATLAB: **.***

Dati due array:

```
xV = [1 2 3; -4 -5 -6]
yV = [7 8 9; 10 11 12]

>> xV .* yV

ans =

     7    16    27
   -40   -55   -72
```

Operazioni sugli array

Divisione

- Le matrici debbono avere le stesse dimensioni
- Si esegue l'operazione elemento per elemento
- Operatori MATLAB:
  - `./`
    - `A ./ B` esegue la divisone di ogni elemento di A con il corrispondente elemento di B
  - `.\`
    - `A .\ B` esegue la divisone di ogni elemento di B con il corrispondente elemento di A

Dati due array:

```
xV = [1 2 3; -4 -5 -6]
yV = [7 8 9; 10 11 12]

>> xV ./ yV

ans =

    0.1429    0.2500    0.3333
   -0.4000   -0.4545   -0.5000

>> xV .\ yV

ans =

    7.0000    4.0000    3.0000
   -2.5000   -2.2000   -2.0000
```


Operazioni sugli array

Potenza

- Data una matrice A ed un numero n eleva ad n ogni elemento di A
- Operatore MATLAB: .^

Dati due array:

```
xV = [1 2 3; -4 -5 -6]
yV = [7 8 9; 10 11 12]

>> xV .^ 3

ans =

     1     8    27
   -64  -125  -216

>> xV .^ 2.5

ans =

 1.0000     5.6569    15.5885
0.0+32.0000i   0.0+55.9017i
   0.0+88.1816i
```

Operazioni sugli array

Trasposizione

- Scambia le righe con le colonne
- Operatore MATLAB: **.'**

Nota: se le componenti sono numeri complessi le traspone senza calcolare il coniugato. Per gli altri tipi di numeri non si notano differenze rispetto all'operatore applicato alle matrici **a'**

Dati due array:

```
xV = [1 2 3; -4 -5 -6]
yV = [7 8 9; 10 11 12]
>> xV.'

ans =

     1    -4
     2    -5
     3    -6
```


Operazioni con scalari

- Sulle matrici ed array si può operare
  - sommando,
  - sottraendo,
  - moltiplicando,
  - dividendo

  tutti gli elementi per uno scalare
- Gli operatori per le matrici e gli array hanno lo stesso comportamento.

Data la matrice **x=[1 2; 7 8]**

```
>> x * 3
ans =
     3     6
    21    24

>> x .* 3
ans =
     3     6
    21    24

>> 3 * x
ans =
     3     6
    21    24
```

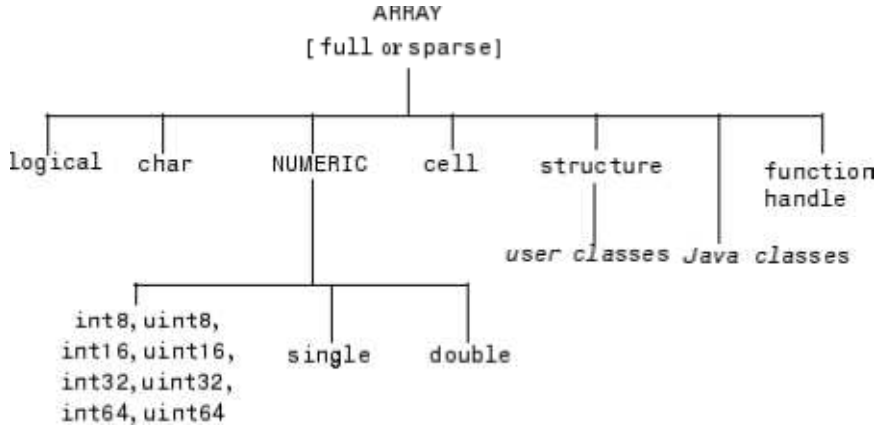
Tipi di dati e variabili in Matlab

- Il tipo di dato fondamentale è la matrice (**`array`**)
- Le variabili
  - non necessitano di dichiarazione di tipo
  - iniziano con una lettera e possono contenere lettere, cifre o il carattere _
  - si distingue fra minuscole e maiuscole
  - hanno 31 caratteri significativi

Tipi di dati e variabili in Matlab

- Le variabili sono automaticamente dichiarate come matrici composte da elementi di tipo **`double`**
  - formato IEEE 754 a 64 bit (virgola mobile in doppia precisione)
- Le matrici possono contenere altri tipi di dati
  - Occorre indicare esplicitamente nel momento del primo uso di una variabile se non è di tipo **`double`**

Tipi di dati e variabili



Tipi di dati e variabili

- la famiglia ***Numeric*** comprende diversi sottotipi riducibili a:

  interi con segno e senza segno in complemento a due ad 8, 16, 32, 64 bit
  - si esplicitano con le funzioni
    - **`int8(), int16(), int32(), int64()`**
      - » per creare/usare variabili/costanti in complemento a due con valori compresi fra -2^{dim-1} e 2^{dim-1} -1 (dim è il numero di bit che compongono la variabile, rispettivamente 8, 16, 32 e 64)
    - **`uint8(), uint16(), uint32(), uint64()`**
      - » per creare/usare variabili/costanti intere positive con valori compresi fra 0 e 2^{dim} -1 (dim è il numero di bit che compongono la variabile, rispettivamente 8, 16, 32 e 64)
  - I valori decimali sono arrotondati
    - Altre funzioni per trasformare da numero decimale in intero sono:
      - » **`round(), floor(), ceil(), fix()`**
  - Per conoscere l'intervallo in cui son compresi i valori di un tipo si usano le funzioni **`intmin()`** ed **`intmax()`** con parametro la classe di interi (es. **`intmax('int8')`**)

Tipi di dati e variabili

- 00001001 = 9

- 11111111 = -1 complemento a 2 a 8 bit
- 11111111 = 255 numero binario puro (senza segno: **`unsigned`**)
  - i numeri `unsigned` sono sempre positivi
    - se ho n bit 0, ..., $2^n - 1$

Tipi di dati e variabili

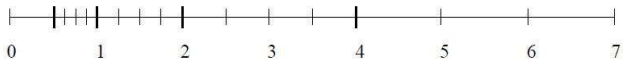
virgola mobile (**`single`**) a precisione singola formato IEEE 754 a 32 bit
- si esplicitano con la funzione
  - **`single()`**

virgola mobile a doppia precisione (**`double`**) formato IEEE 754 a 64 bit
- tutte le variabili sono dichiarate automaticamente di questo tipo
  - Per convertire un valore in questo tipo si usa la funzione **`double()`**

– Funzioni utili per:
- conoscere l'intervallo di definizione del tipo
  - **`realmax()`** e **`realmin()`** per **`double`**
  - **`realmax('single')`** e **`realmin('single')`** per **`single`**
- conoscere la precisione del valore
  - **`eps()`** con argomento una variabile/valore
- verificare se un valore è in virgola mobile
  - **`isfloat()`** con argomento una variabile/valore (restituisce 1 o 0)

Tipi di dati e variabili

- I numeri in virgola mobile sono rappresentati come
  - Numeri razionali con un numero finito di cifre
  - I numeri in virgola mobile NON sono numeri reali, né un sottointervallo completo dei razionali
  - I numeri in virgola mobile si addensano intorno allo zero per diradarsi con i valori più grandi
    - Si cerca di ridurre l'errore relativo
    - Il valore restituito da **`eps(val)`** indica l'intervallo fra il valore **`val`** ed il successivo rappresentabile in virgola mobile



Tipi di dati e variabili

- Per i numeri in virgola mobile sono definiti tre valori speciali **`Inf`**, **`-Inf`** e **`NaN`**
  - **`Inf`** e **`–Inf`** sono restituiti da operazioni che generano overflow e divisioni per 0
    - » La funzione **`inf()`** restituisce la costante **`Inf`**
  - **`NaN`** indica un valore che non è un numero reale o complesso, anche le espressioni che restituiscono un valore indefinito (**`0/0`**, **`Inf/Inf`**, …) restituiscono **`NaN`**.
    - » La funzione **`isnan()`** verifica se l'argomento è un **`NaN`**
    - » La funzione **`NaN()`** restituisce la costante **`NaN`**

Tipi di dati e variabili

I numeri possono essere visualizzati come:

`short` (quello automatico)

`long`

`rational` (come numero razionale)

`hex` (esadecimale)

– per impostare il formato si usa la funzione
`format` *`tipoFormato`*

Tipi di dati e variabili

`logic`

- è il tipo booleano, può assumere solo due valori
  - **`1`** (vero), **`0`** (falso)
- La funzione **`logical()`** converte gli argomenti di tipo **`numeric`** in valori di tipo **`logic`**
- Gli operatori sono i seguenti:
  - Operatori logici: **`& | ~ xor any all`**
    - » Funzionano come operatori su array
    - » Esistono anche funzioni equivalenti, `and()`, `or()`, `not()` per I primi tre operatori
  - Operatori corto-circuito: **`&& ||`**
    - » Non completano la valutazione se quello che hanno valutato consente la valutazione complessiva
    - » **Si applicano solo ad operandi scalari**
  - Operatori relazionali: **`== ~= < > <= >=`**
  - Tutte le funzioni che iniziano con **`is`** restituiscono un valore **`logic`**

Tipi di dati e variabili

`char`

- è il tipo carattere
  - un carattere è un valore compreso fra apici singoli
- le stringhe sono vettori di caratteri si scrivono fra apici singoli

`structures`

- sono come il tipo struct in C

`cells`

- sono una sorta di vettore di strutture

Espressioni

- Le espressioni si costruiscono a partire da:
  - Variabili
  - Costanti
  - Operatori
  - Funzioni

Operatori: precedenze

```
()
.'  .^  '  ^
+ - ~                (operatori unari)
.* ./ .\ * / \
+ -                  (operatori binari)
:
<   <=   >   >=   ==   ~=
&
|
&&
||
```

Più importante

Valutazione da sinistra verso destra

Meno importante

Espressioni numeriche

- Se un'espressione contiene operandi dello stesso tipo l'espressione è del tipo degli operandi coinvolti
- Se un'espressione contiene operandi di tipo diverso si applicano le seguenti regole
  - Interi o vettori di interi di categoria diversa:
    - NON si può fare
  - Interi o vettori di interi con uno **scalare** **`double`**:
    - risultato intero (operazioni eseguite in doppia precisione e poi arrotondate)
    - Gli interi devono essere tutti della stessa categoria
  - ***`single`*** *con altri tipi:*
    - *risultato* ***`single`***
    - *non si possono scrivere espressioni con tipi di classe intera*
  - **`double`** con altri tipi eccetto nei casi precedenti
    - risultato **`double`**

Espressioni numeriche

Operatori

- sono quelli dell'aritmetica con le usuali regole di precedenza
- Esistono inoltre
  - gli operatori `\ ^ '`
  - gli operatori sugli array (quelli in cui l'operatore è preceduto dal `.`)
- **Attenzione**
  - *Sui tipi numerici* ***`int64`*** *e* ***`uint64`*** *non si applicano gli operatori aritmetici tranne* **+** *e* **–** *unario*
  - *Sulle matrici di tipi interi non si possono applicare gli operatori* ***`* \ /`***

Espressioni logiche

- Sono tutte quelle espressioni che restituiscono un valore vero (diverso da 0) o falso (uguale a 0)
  - Operandi matriciali
    - Gli operandi relazionali restituiscono una matrice di valori con 0 ed 1 a seconda se il confronto elemento per elemento è risultato falso o vero
      - Gli operatori logici (`& | ~`) producono un vettore che risulta vero se tutti gli elementi che lo compongono sono veri
    - Per avere un risultato scalare (vero o falso) si usano altri operatori
      - `all()`, `any()` ad esempio.

Espressioni logiche (matrici)

- `a = [ 5  6  7];`
- `b = [ 8  -3  10];`

```
a > b
ans =
      0     1     0
a & b
ans =
      1     1     1
```

(i valori diversi da 0 indicano un valore vero, **`true`**, che viene visualizzato con 1)

Espressioni logiche (matrici)

```
>> C=[2 3 4; 5 6 7]
C =  2     3     4
     5     6     7

>> D=[10 11 0; 9 0 7]
D = 10    11     0
     9     0     7
```

```
>> C > D
ans =
     0     0     1
     0     1     0

>> C & D
ans =
     1     1     0
     1     0     1
```

Espressioni logiche

- Funzioni su matrici
  - **`isequal(mat1, mat2, ...)`**
    - » Restituisce 1 se le matrici sono dello stesso tipo e dimensione ed hanno gli stessi valori; 0 altrimenti
  - **`isempty(mat)`**
    - » Restituisce 1 se la matrice è vuota o ha almeno una dimensione uguale a zero; 0 altrimenti
  - **`all(mat)`**
    - » Restituisce 1 per ogni colonna di **`mat`** i cui elementi sono tutti diversi da zero; 0 altrimenti.
  - **`any(mat)`**
    - » Restituisce 1 per ogni colonna di **`mat`** in cui vi è almeno un elemento diverso da zero; 0 altrimenti.

Esempio

```
>> C=[2 3 4; 5 6 7]
C =  2     3     4
     5     6     7

>> D=[10 11 0; 9 0 7]
D = 10    11     0
     9     0     7
```

```
>> isequal(C,D)
  ans = 0

>> isequal(C,C)
  ans = 1

>> any(D)
  ans =
     1     1     1

>> all(D)
  ans =
     1     0     0
```

Esempio

- Data una matrice X verificare se tutti i suoi elementi sono maggiori 0

```
all(all(X>0))
```

```
X=[1 2; 0 5]
```

Valutiamo `X > 0` → `[1 1; 0 1]`

Valutiamo `all([1 1; 0 1])` → `[0 1]`

Valutiamo `all([0 1])` → `0`

Esempio

- Verificare se nella matrice X c'è qualche elemento maggiore di 0

```
any(any(X>0))
```

```
X=[1 2; 0 5]
any(any(X>0))
```

`X>0` → `[1 1; 0 1]`

`any([1 1; 0 1])` → `[1 1]`

`any([1 1])` → `1`

Operatore :

Permette la definizione di intervalli di valori

- creazione di intervalli
  - ***inizioIntervallo*:*fineIntervallo***
  - ***inizioIntervallo*:*passo*:*fineIntervallo***

  (se non indicato ***passo*** vale 1)
- Esempi
  - Creare un insieme di valori fra 1.5 e 5 con passo .5

    `1.5:.5:5`
  - Creare un insieme di valori fra 8.5 e 10.5 con passo 1

    `8.5:10.5`

Operatore :

- indicare una parte di una matrice

  Esempi

  Data la matrice **`A=[1 2 3; 4 5 6; 7 8 9]`**
  - **`A(:,2)`** restituisce gli elementi della seconda colonna
  - **`A(1:2,2)`** restituisce i primi due elementi della seconda colonna
  - **`A(3,:)`** restituisce gli elementi della terza riga
  - **`A(:)`** restituisce un vettore colonna con il contenuto di tutte le colonne della matrice
  - **`A(:) = 1:9`** sostituisce tutti gli elementi della matrice con i valori da 1 a 9. La matrice mantiene la stessa dimensione. E' un errore fornire più/meno elementi di quelli che può contenere la matrice

Esempio

```
>> C=[2 3 4; 5 6 7]
C =  2     3     4
     5     6     7

>> D=[10 11 0; 9 0 7]
D = 10    11     0
     9     0     7
```

```
>> T=1:4:12
 T =  1     5     9

>> Q=C(1:2,1)
 Q =  2
      5

>> D(:) = 1:6
D =  1     3     5
     2     4     6
```


Operatore …

– Serve per far proseguire un'istruzione sulla riga successiva
  - Non può essere usato all'interno di stringhe (sequenze di caratteri chiuse fra apici singoli ')

Espressioni

Funzioni

- possiede un gran numero di funzioni predefinite
- fornisce la possibilità di creare funzioni
- prevede il passaggio di funzioni come parametro

Alcune funzioni predefinite

```
pi()
realmin()
realmax()
inf()
NaN()
eps()
```


Funzioni su matrici

- Matlab possiede numerose funzioni per la manipolazione diretta delle matrici per
  - Creare nuove matrici
  - Manipolare il contenuto delle stesse
  - ...

  Riepiloghiamo e completiamo quanto già visto

Funzioni sulle matrici

`length(matrice)`:

- restituisce il valore della dimensione più grande di matrice;
  - » per i vettori riga il numero delle righe,
  - » per i vettori colonna il numero di colonne,
  - » negli altri casi restituisce solo il valore della dimensione più grande.

`size(matrice)`:

- restituisce un vettore riga con tutte le dimensioni di matrice.
- `size(matrice,dim)`: restituisce il valore della dimensione `dim` di matrice.

Esempio

- Data una generica matrice X creare la matrice Y formata dalle righe dispari di X e dalle colonne pari di X

```
Y = X(1:2:size(X,1), …
      2:2:size(X,2))
```

```
>> X=[7 8 9 12; -3 4 5 9; 8
   9 12 34; -2 0 3 -4]

X =  7     8     9    12
    -3     4     5     9
     8     9    12    34
    -2     0     3    -4

>> Y = X(1:2:size(X,1), ...
         2:2:size(X,2))
Y =  8    12
     9    34
```

Funzioni su matrici

- **eye(n,m)**
  - Crea una matrice n x m di 0 con 1 solo sulla diagonale principale (quella dove la coordinata di riga è uguale quella della colonna)

```
>> H=eye(2,3)
H =  1     0     0
     0     1     0
>> K=eye(3,2)
K =  1     0
     0     1
     0     0
```

  - eye(n) crea la matrice identità n x n
- **zeros(n,m)**
  - Crea una matrice n x m di 0
- **ones(n,m)**
  - Crea una matrice n x m di 1

Funzioni su matrici

- **isequal(mat1, mat2, ...)**
  - Restituisce 1 se le matrici sono dello stesso tipo e dimensione ed hanno gli stessi valori
- **isempty(mat)**
  - Restituisce 1 se la matrice è vuota ha almeno una dimensione uguale a zero
- **linspace(X1,X2,N)**
  - Restituisce un vettore di **N** punti equispaziati fra **X1** ed **X2**
    - » Se **N** è omesso genera 100 punti
    - » Se **N** vale 1 genera **X2**

Funzioni su matrici

`all(A, dim)` verifica se tutti gli elementi di un array sono non nulli;

Restituisce:

» Se A è un vettore: 1 se tutti gli elementi sono diversi da 0, altrimenti 0

» Se A è una matrice: tratta le colonne di A come se fossero un vettore e restituisce un vettore riga

» Se è indicato il valore per il parametro ***dim*** verifica sulla dimensione specificata dim=1 verifica sulle colonne ; dim=2 verifica sulle righe

La funzione duale è ***any()***: vale 1 se qualche elemento è diverso da 0

Esempio

`A=[1 2 3; 4 0 6]`

`all(A)` e **`all(A,1)`** restituiscono 1 0 1

`all(A,2)` restituisce 1
0

Funzioni su matrici

`repmat(M,n,m)`: crea una matrice contenente n volte sulle righe ed m volte sulle colonne la matrice M

– Se m non è specificato replica n volte anche sulle colonne

Esempio

```
M=[1 2 3;
   4 5 6]

repmat(M,3,2)
ans =
     1     2     3     1     2     3
     4     5     6     4     5     6
     1     2     3     1     2     3
     4     5     6     4     5     6
     1     2     3     1     2     3
     4     5     6     4     5     6
```

Esempi

```
Matrice=[ 1 2; 3 4]
repmat(Matrice, 2, 3)
   1 2   1 2   1 2
   3 4   3 4   3 4
   1 2   1 2   1 2
   3 4   3 4   3 4
```

```
M=[1 2; 3 4]
repmat(M*2,2,2)
```

- `M*2 = [2 4; 6 8]`
- `repmat([2 4; 6 8],2,2)`

```
2 4 2 4
6 8 6 8
2 4 2 4
6 8 6 8
```


Funzioni su matrici

Le dimensioni delle matrici debbono essere consistenti!

`cat(dim,Mat1,Mat2, ...)`:
- Concatena M1, M2 alla dimensione indicata da dim

`horzcat(Mat1,Mat2, ...)`:
- Concatena M1, M2 orizzontalmente
- Equivale a **`[Mat1, Mat2, …]`**

`vertcat(Mat1,Mat2, ...)`:
- Concatena M1, M2 verticalmente
- Equivale a **`[Mat1; Mat2; …]`**

`blkdiag(mat1,mat2, mat3, ...)`:
- crea una matrice diagonale a blocchi inserendo sulla diagonale principale, una dopo l'altra le matrici.

Esempi

```
M1 = 1     2     3
     4     5     6
M2 =-1    -2
    -3    -4
    -5    -6

>> cat(1,M1,M2)
   Errore: non
     compatibili
```

```
>> cat(1,M1,M2')
ans =1     2     3
     4     5     6
    -1    -3    -5
    -2    -4    -6

>> [M1;M2']
ans =1     2     3
     4     5     6
    -1    -3    -5
    -2    -4    -6

>> vertcat(M1,M2')
ans =1     2     3
     4     5     6
    -1    -3    -5
    -2    -4    -6
```


Esempi

```
M1 = 1     2     3
     4     5     6
M2 =-1    -2
    -3    -4
    -5    -6

>> cat(2,M1,M2)
   Errore: non
     compatibili
```

```
>> cat(2,M1',M2)
ans =
  1     4    -1    -2
  2     5    -3    -4
  3     6    -5    -6

>> [M1',M2]
ans =
  1     4    -1    -2
  2     5    -3    -4
  3     6    -5    -6

>> horzcat(M1',M2)
ans =
  1     4    -1    -2
  2     5    -3    -4
  3     6    -5    -6
```

Esempi

```
M1 = 1     2     3
     4     5     6

M2 = -1    -2
     -3    -4
     -5    -6
```

```
>> blkdiag(M1,M2)

ans =
     1     2     3     0     0
     4     5     6     0     0
     0     0     0    -1    -2
     0     0     0    -3    -4
     0     0     0    -5    -6

>> blkdiag(M2,M1)

ans =
    -1    -2     0     0     0
    -3    -4     0     0     0
    -5    -6     0     0     0
     0     0     1     2     3
     0     0     4     5     6
```


Funzioni su matrici

sum(X, *dim*) somma gli elementi della dimensione indicata da ***dim*** (dim deve essere minore o uguale alla dimensione della matrice). Se ***dim*** non è indicato somma gli elementi della prima dimensione con un numero di elementi maggiore di 1 (dim=1 somma sulle colonne, dim=2 somma sulle righe).

Analogamente si comporta la funzione ***prod()***

Esempio

```
X=[1 4 6; 14 5 8]
sum(X)= sum(X,1) = [15 9 14]
sum(X,2) = [11; 27]
```

Funzioni su matrici

diff(X,*n*,*dim*) calcola le differenze fra elementi della prima dimensione con un numero di elementi maggiore di 1. Se ***n*** è indicato applica ricorsivamente n volte la funzione. Se viene specificato ***dim*** è applicato alla dimensione dim.

Calcola la differenza Rispetto alle colonne

Calcola le differenze fra Gli elementi adiacenti di X

Esempio

```
X=[1 3 5;
   8 9 10]
diff(X)=diff(X,1)=diff(X,1,1)=
       =[X(2:2,:)-X(1:2-1,:)]=[7 6 5]
diff(X,2) = diff(diff(X)) = [-1 -1]
diff(X,1,2) = [2     2;   1     1]
```


Funzioni su matrici

sort(M,*dim*,*modo*) ordina gli elementi di M. Ha un comportamento diverso a seconda del tipo di M quando ***dim*** non è specificato

- Vettore riga: ordina gli elementi di A
- Matrice 2-D: ordina ogni colonna
- Matrice N-D: ordina rispetto alla prima dimensione con più di 1 elemento
- Ordina in maniera crescente (**'ascend'**), specificando **'descend'** come valore di ***modo*** in maniera decrescente; il valore di ***dim*** può essere omesso
- Esempio

```
Y=[1 4 -4; 8 12 -3; -4 5 6; 1 -4 -7]
sort(Y) =sort(Y,1)= [-4 -4 -7;1 4 -4; 1 5 -3;8 12 6]
sort(Y,2) = [-4  1  4; -3  8 12; -4  5  6; -7 -4  1]
```

Esempio

- PB: dato un vettore di numeri decimali di 20 elementi caricare una matrice inserendo nella prima riga il vettore dato, nella seconda il doppio dei valori nella prima riga; quindi visualizzare un grafico nel quale i valori della prima riga fungono da ascissa, quella della seconda da ordinata

```
tic;
vet = rand(1,20);
vet= sort(vet);
matr=[vet;vet*2];
plot((matr(1,:))',(matr(2,:).^2)');
tT=toc;
```

Al termine delle operazioni nella variabile tT vi è il tempo in ms impiegato ad per eseguirle

Polinomi

- In Matlab si può definire un polinomio mediante il vettore ad esso equivalente:

$$X^5 + 10x^4 + 0.6x^2 + x + 6$$

Si può scrivere come

```
pol =[1 10 0 0.6 1 6]
```

Esempio

```
X^2+x+1
P2=[1 1 1]

 [1    -2   -27     0   108]
108 +0x -27 x^2 -2 x^3 + x^4
```


Polinomi

- Per valutare un polinomio in un punto o in un vettore di punti si usa
  polyval(*polinomio,punto*)
- Per determinarne le radici si usa
  roots(*polinomio*)
- Per creare un polinomio (monico) a partire dalle sue radici si usa
  poly(*rad1, rad2, ..., radn*)
- Per moltiplicare due polinomi
  conv(*polinomio1, polinomio2*)
- Per dividere due polinomi (restituisce quoziente e resto)
  deconv(*polinomio1, polinomio2*)

Esempi

- Date le radici del polinomio:
  3
  7
  -23
  14

ricavare il polinomio:

```
>> poly([3 7 -23 14])

ans =
  1 -1 -391 3409 -6762
```

- Dato il polinomio
  $x^4 - x^3 - 391x^2 + 3409x - 6762$

determinarne le radici:

```
>>roots([1,-1,-391,3409,-6762])

ans =
  -23.0000
   14.0000
    7.0000
    3.0000
```


Esempi

$x = [2\ 3\ 4] = 2x^2+3x+4$
$y = [1\ 2] = x+2$

$x*y=$
$2x^3+3x^2+4x+4x^2+6x+8=$
$=2x^3+7x^2+10x+8=$
$[2\ 7\ 10\ 8]$

$2x^2+3x+4 : x+2 = 2x-1$
$\underline{-2x^2-4x}$
$0\ -x+4$
$\underline{x+2}$
6

```
>> x=[2 3 4]
     x =  2     3     4

>> y=[1 2]
     y =    1     2

>> conv(x,y)
  ans =
     2     7    10     8

>> [q r] = deconv(x,y)
    q =  2    -1

    r = 0     0     6
```

Esempi

```
pol1 = [1 1 1]
```
$x^2 + x + 1$

```
pol2 = [1 1]
```
$x + 1$

```
>> h=conv(pol1,pol2)
```
$(x^2 + x + 1)(x+1)$

```
h=[1   2   2   1]
```
$(x^3 + 2x^2 + 2x + 1)$

```
>>[q r]=deconv(pol1,pol2)
```
$(x^2 + x + 1)/(x+1)$

```
q = [1 0]
```
x

```
r = [0 0 1]
```
1

Esempio

- Disegnare la funzione $f(x)=x^4+2x+1$
  - per x che varia fra -3 e 3 con passo 0.1

```
X=-3:0.1:3
Y=polyval([1 0 0 2 1],X)
plot(X,Y)
```

  - per 5000 punti equispaziati fra -3 e 3

```
X=linspace(-3,3,5000)
Y=polyval([1 0 0 2 1],X)
plot(X,Y)
```

Matrici multivariate

- MATLAB utilizza un approccio per colonne nell'analisi dei dati per statistiche multivariate.
  - Ogni colonna in un insieme di dati rappresenta una variabile ed ogni riga una osservazione
  - L'elemento (i,j) rappresenta la i-ma osservazione della j-ma variabile

```
D = [ 72 134 3.2;
      81 201 3.5;
      69 156 7.1;
      82 148 2.4;
      75 170 1.2 ]
```

- Si interpreta come 5 osservazioni di tre variabili
- Applicando a **D** le funzioni **mean()** e **std()** si calcolano la media e la deviazione standard delle tre variabili