# Introduzione a Matlab

**Corso di Calcolo Intensivo**
**aa 2012-2013**

**Annalisa Massini**



## Introduzione

- MATLAB è un linguaggio di supporto per il calcolo scientifico, cioè per sviluppo, implementazione e analisi di algoritmi numerici e non, che utilizzino appropriati modelli matematici
- MATLAB è l'acronimo di MATrix LABoratory
- La storia di MATLAB comincia negli anni '70 con le librerie matematiche LINPACK e EISPACK (in fortran) per la risoluzione di problemi di algebra lineare
- Negli anni '80 Cleve Moler scrive la prima versione di MATLAB allo scopo di sviluppare un calcolatore grafico interattivo.

# Introduzione

- MATLAB si compone di cinque parti principali:
  - Il linguaggio MATLAB con gestione delle principali strutture di programmazione
  - Ambiente di lavoro MATLAB e sua gestione
  - Ambiente grafico e sua gestione
  - Libreria di funzioni matematiche
  - Libreria per l'interazione con programmi scritti in C o Fortran



# Introduzione

- MATLAB comprende un vasto set di funzioni predefinite e numerose librerie (toolbox) per svariate applicazioni
- Le potenzialità di MATLAB possono essere facilmente estese → è semplice creare nuovi toolbox
- E' possibile convertire un programma MATLAB in codice C e C++ in modo automatico

# Variabili e espressioni

- All'avvio di MATLAB appare il prompt " **>>** "
- Vi sono due tipi di istruzioni:
  - **valutazione** di espressioni "**>>** *espressione*"
  - **assegnamento** "**>>** *variabile* = *espressione*"
- La valutazione di un'*espressione* genera una **matrice** che viene assegnata alla *variabile* indicata.
- Se non si specifica la *variabile* a cui assegnare il risultato, viene assegnata la variabile di sistema "***ans*".**

Esempi:

- **>> 8+2**
  **ans =**
  **10**
- **>> a = 5*ans**
  **a =**
  **50**
- **>> 6.9**
  **ans =**
  **6.9000**



# Variabili e espressioni

- Se un'espressione termina con "**;**" il risultato della valutazione **non** viene mostrato sullo schermo.
- In MATLAB le variabili **non** devono essere dichiarate.
- La dichiarazione coincide con il primo assegnamento.
- MATLAB è case-sensitive

Esempi:

**» b = 6+a;**

**» b**
**b =**
**56**

# Alcune istruzioni di uso comune

- **help** richiama l'help in linea
  - help *comando* visualizza l'help relativo al comando indicato
- **who/whos** elencano le variabili in uso
- **dir** elenca i files contenuti nella directory corrente
- **clear all** elimina tutte le variabili della sessione corrente
  - clear *var1 var2* elimina le variabili *var1* e *var2*
- **lookfor** consente di identificare le funzioni relative ad uno specifico argomento



# Matrici

- In Matlab qualsiasi oggetto è una matrice
- Le variabili scalari sono viste come array di dimensione 1x1 (una riga e una colonna)
- Di default, Matlab lavora con variabili in doppia precisione
- Ogni numero memorizzato in doppia precisione occupa 8 Bytes
- Negli esempi precedenti abbiamo visto i valori reali vengono visualizzati usando 5 cifre, a meno che non venga specificato un altro formato con l'istruzione `format`

# Il comando FORMAT

Il comando **`format`** permette di **visualizzare** un valore in diversi formati.
Consideriamo il x = 4/3

- **`format short`** 1.3333 0.0000 - virgola fissa 5 cifre *default*
- **`format long`** 1.33333333333333 - virgola fissa 15 cifre
- **`format short e`** 1.3333e+000 - virgola mobile 5 cifre
- **`format long e`** 1.333333333333333e+000 - virgola mobile 15 cifre
- **`format short g`** 1.3333 - migliore tra virgola fissa e mobile
- **`format long g`** 1.33333333333333 -migliore tra virgola fissa e mobile
- **`format bank`** 1.33 - formato per denaro (dollari o euro)
- **`format rat`** 4/3 - approssimazione al rapporto
- **`format hex`** 3ff5555555555555 - formato esadecimale



# Osservazioni

- Si può rappresentare solo un numero finito di valori in doppia precisione → c'è sempre una piccola differenza tra un valore e il suo successivo, che si può calcolare con il comando eps:
  - **eps**(x) è la differenza positiva tra abs(x) e il successivo valore in virgola mobile (nella stessa precisione)
- **eps** è la differenza tra 1.0 e il valore maggiore più vicino in virgola mobile, cioè **eps = 2^(-52)**
- **realmin** è il più piccolo valore positivo rappresentato ed è circa **2.2251e-308**, cioè **2^(-1022)**
- **realmax** è il più grande valore positivo rappresentato ed è circa **1.7976e+308** cioè **2^1023**

# Matrici: definizione

- Una matrice può essere definita con la sintassi seguente:

```
» A = [7  8; 8.9  7; 9  8]
  A =
       7.0000   8.0000
       8.9000   7.0000
       9.0000   8.0000
```

```
» B = [1  2  3
  4  5  6]
  B =
       1    2    3
       4    5    6
```

- uno **spazio** o una **virgola** delimitano gli elementi di un stessa riga
- un **punto e virgola** o un **cambio di riga** indicano la fine di una riga



# Matrici: definizione

- Sono presenti funzioni predefinite per la generazione di particolari matrici:
  - **zeros(n,m)** matrice di zeri **nxm**
  - **ones(n,m)** matrice di uno **nxm**
  - **eye(n,m)** matrice identità **nxm**
  - **rand(n,m)** matrice di numeri casuali **nxm**
  - **diag([a11, a22, a33, ..., aNN])** matrice diagonale

# Matrici: accesso

Accesso agli elementi di una matrice:

- » A = [7 8; 8.9 7; 9 8]

```
A =
   7.0000   8.0000
   8.9000   7.0000
   9.0000   8.0000
```

- **A(n,m)** estrae l'**elemento** (n,m) della matrice A

```
» A(1,2)
   ans =
      8
```



# Matrici: accesso

Accesso alle righe/colonne di una matrice:

- **A(n,:)** estrae l'n-esima **riga** della matrice A

```
» A(2,:)
ans =
   8.9000   7.0000
```

**A(:,m)** estrae l'm-esima **colonna** della matrice A

```
» A(:,1)
ans =
   7.0000
   8.9000
   9.0000
```

**N.B.** La **notazione : (due punti)** permette di specificare un **intervallo di indici**. In questo caso si intende tutta una riga o tutta una colonna perchè gli estremi dell'intervallo non sono specificati.

# Estensione di una matrice

- È possibile estendere matrici già esistenti aggiungendo una riga o una colonna (di dimensione opportuna)

  **Colonna**

  - Data A = [ 1  2; 3  4; 5  6 ];
  - Estendiamo a con la colonna di elementi 7 8 9

    **A = [ A [7; 8; 9] ]** (oppure **[ A [7 8 9]' ]** )

```
A =
     1 2 7
     3 4 8
     5 6 9
```



# Istruzioni su matrici

- **diag**: applicata ad una **matrice** ne estrae la diagonale

```
» A=[5 6 ; 7 8]
A =
    5 6
    7 8
» diag(A)
ans =
     5
     8
```

- **diag:** applicata ad un **vettore** crea una matrice diagonale.

```
» diag(ans)
ans =
     5 0
     0 8
```

# Istruzioni su matrici

- **sum**: applicata ad una **matrice** fornisce un vettore che contiene le somme per colonna degli elementi

```
» A=[0 1 2 ;3 4 5 ;6 7 8 ]
A =
     0 1 2
     3 4 5
     6 7 8
```

```
»  B=sum(A)
B =
9 12 15
```

- **sum**: applicata ad un **vettore** fornisce uno scalare dato dalla somma degli elementi

```
»  sum(B)
ans =
      36
```



# Definizione di vettori

- Un vettore può essere creato con la stessa sintassi utilizzata per le matrici oppure usando la **notazione due punti**:
  - » **x = 1:6**

```
x =
   1   2   3   4   5   6
```

- **`a:[step:]b`** crea un vettore riga di estremi a e b

  Il parametro **step** indica l'intervallo tra gli elementi del vettore

  - » **x = 0.5:0.1:0.9**

```
x =
   0.5000   0.6000   0.7000   0.8000   0.9000
```

# Definizione di vettori

- Un vettore può essere creato usando l'istruzione `linspace(a,b,N)` che produce un vettore riga di estremi a e b, costituito da N punti equispaziati
  - **» x = linspace(-1,1,4)**

    **x =**

    **-1.0000 -0.3333 0.3333 1.0000**



# Funzioni elementari per scalari

- Gli operatori aritmetici presenti in MATLAB sono:
  + (somma), - (differenza), * (prodotto), / (quoziente), ^(elevamento a potenza)
- Funzioni trigonometriche
  - sin, cos — seno, coseno
  - tan — tangente
  - asin, acos — arco seno, arco coseno
  - atan — arco tangente

# Funzioni elementari per scalari

- Funzioni matematiche elementari:
  - **abs** valore assoluto di un numero complesso
  - **angle** fase di un numero complesso
  - **conj** complesso coniugato
  - **exp** esponenziale in base e
  - **real**, **imag** parti reale e immaginaria di un numero complesso
  - **log**, **log10** logaritmo naturale ed in base 10
  - **sqrt** radice quadrata

- Le variabili **i** e **j** sono predefinite come **unità immaginarie**, cioè uguali alla radice quadrata di -1



# Funzioni elementari per matrici

Gli operatori elementari sono:

- + **somma** di vettori o matrici (elemento per elemento)
- - **differenza** di vettori o matrici (elemento per elemento)
- * **prodotto** tra vettori e/o matrici (righe per colonne)

Sono **operazioni su matrici** quindi:

- per somma e differenza: gli operandi devono avere le **stesse dimensioni**; se uno dei due operandi è uno **scalare**, esso viene sommato o sottratto a tutti gli elementi della matrice
- per il prodotto: il numero delle colonne della prima matrice deve essere uguale al numero delle colonne della seconda matrice

# Funzioni elementari per matrici

Si hanno anche:

- X = B/A è la soluzione dell'equazione X*A = B
- X = A\ B è la soluzione dell'equazione A*X = B

Le operazioni **puntuali** agiscono su array con le stesse dimensioni:

- **.*** prodotto elemento per elemento
- **./** divisione elemento per elemento
- **.^** potenza elemento per elemento

Le funzioni matematiche trigonometriche applicate alle matrici, si riferiscono ai singoli elementi della matrice



# Funzioni elementari per matrici

- Principali operazioni matriciali:
  - Matrice trasposta **A'**
  - Matrice inversa **inv(A)**
  - Determinante **det(A)**
  - Autovalori **eig(A)**
  - Polinomio caratteristico **poly(A)**
  - Rango **rank(A)**
  - Dimensioni **size(A)**

# Funzioni matematiche

L'elenco di **funzioni matematiche elementari** si ottiene con:

- help **elfun:** le funzioni sono suddivise in trigonometriche, esponenziali, complesse e di arrotondamento

Un elenco più avanzato delle **funzioni matematiche** è:

- help **specfun** funzioni matematiche specializzate, funzioni di teoria dei numeri (fattoriale, primalità, ecc.), trasformazioni di coordinate

Un elenco delle **funzioni su matrici** si ottiene con:

- help **elmat**: funzioni elementari, funzioni di info su matrici, variabili e costanti, funzioni di manipolazione su matrici, funzioni specializzate su matrici



# Istruzioni per la grafica

- La funzione **plot** produce grafici bidimensionali e può essere chiamata con diverse modalità
- **Esempio**

```
» n = 31
» x = linspace(0,2*pi,n)
» y = sin(x)
» plot(x,y)
```

permette di ottenere una rappresentazione grafica raccordando con segmenti di retta nel piano xy i valori (x(i),y(i)) per i che va da 1 a 31

# Istruzioni per la grafica

- La sintassi del comando `plot` è:
  - **plot(vettore1, vettore2, opzioni)**

  dove **vettore1** è per le ascisse

  **vettore2** è per le ordinate

  **opzioni** definisce colore, tipo di linea e di simbolo usati

- Per tracciare quattro funzioni di x, ciascuna con un colore diverso, nello stessa finestra grafica:

```
y2 = sin(x - .4);
y3 = sin(x - .8);
y4 = sin(x - 1.2);
```

- `plot(x,y,x,y2,x,y3,x,y4)` oppure `plot(x,[y;y2;y3;y4])`



# Istruzioni per la grafica

- Un altro modo per rappresentare sullo stesso grafico più curve:
  - dati y1 e y2

```
» x = linspace(0,2*pi)
» y1 = cos(x)
» y2 = sin(x)
```

  - possiamo disegnare i grafici sovrapposti con

```
» plot(x,y1,'-')
» hold on
» plot(x,y2,'--')
» hold off
```

# Istruzioni per la grafica

- E' possibile impostare titolo e etichette per gli assi:

```
» title('titolo del grafico')
» xlabel('asse x')
» ylabel('asse y')
```

- **axis** permette di scegliere dimensione e aspetto degli assi
- **legend** permette di aggiungere una legenda
- **text** permette di porre una scritta su un grafico alle coordinate specificate:

```
» text(x(70)+0.5,r(70),'r = -2x')
```

- Si può specificare la griglia con il comando **» grid on** oppure specificando **grid** nel comando **plot**



# Istruzioni per la grafica

Altre funzioni che consentono la visualizzazione di grafici 2-D, sono:

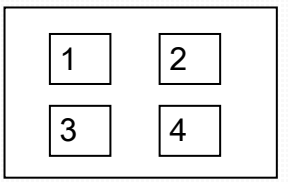
- **loglog** grafico logaritmico in x e y
- **semilogx** grafico logaritmico in x e lineare in y
- **semilogy** grafico logaritmico in y e lineare in x
- **bar** grafico a barre
- **stairs** grafico a scala
- **mesh** grafico 3D

# Istruzioni per la grafica

Il comando **`subplot`** permette di dividere una finestra grafica in una matrice di sottofinestre.

- **`» subplot(m,n,p)`**

  specifica che si usa una matrice di sottofinestre con ***m righe*** e ***n colonne*** e che si disegna nella ***sottofinestra p***, numerando le finestre per righe dall'alto verso il basso.






# Visualizzazione di funzioni

- Il comando **`fplot`** permette di disegnare il grafico di una funzione indicata esplicitamente tra apici (o mediante il nome del m-file)
- Il comando **`fplot(fun, lims)`** visualizza il grafico della funzione indicata nella **stringa fun** sull'**intervallo [lims(1), lims(2)]**
- **Esempio:**

```
» fun='1/(1+x^2)';
» lims=[-5,5];
» fplot(fun,lims);
```

o equivalentemente

```
» fplot('1/(1+x^2)', [-5,5]);
```

# Visualizzazione di funzioni

- Il grafico viene disegnato nell'intervallo definito dalle ascisse **`[xmin,xmax]`**; le ordinate **ymin, ymax** si possono omettere e, in tal caso, vengono determinate automaticamente
- L'opzione **LineSpec** è una **stringa** che specifica il tratto grafico

  `fplot(fun, lims, '- -')` oppure

  `fplot(fun, lims, 'r -')`

- Si può specificare tra apici anche un **array di funzioni**, in tal caso vengono disegnati i grafici di tutte le funzioni
- **Esempio**

```
fplot('[sin(t), sin(t-.25), sin(t-.5)]',[0,2*pi])
```



# Visualizzazione di funzioni

- Il grafico prodotto da Matlab è una rappresentazione approssimata, del grafico della funzione *f,* ottenuto su un insieme non equispaziato di ascisse.
- Per aumentare l'accuratezza della rappresentazione si può richiamare **fplot** con l'opzione **tol** che permette di specificare la tolleranza relativa richiesta e l'opzione **n** che specifica il numero di punti utilizzato per disegnare il grafico:

  **fplot('funzione', [xmin,xmax,ymin,ymax], tol,n)**

- Si può usare anche l'istruzione **ezplot.**

# Visualizzazione di funzioni

Si possono produrre **grafici di funzioni in due variabili**:

- si preparano i valori della griglia, tramite una matrice delle x e una matrice delle y, e i corrispondenti valori di **z = f(x,y)**
- si disegna usando il comando **`mesh`**, oppure i comandi **`surf`**, **`surfl`**, **`contour`**
- **Esempio**

```
» n=7; m=n;
» x=linspace(-2,2,n);
» y=linspace(-2,2,n);
» [X,Y]=meshgrid(x,y); % matrici X e Y, griglia di coordinate
» Z=(1-Y).*cos(X.^2)+(X-1).*cos(Y.^2);
» mesh(X,Y,Z);
```



# Sulle funzioni

- Per **valutare** le funzioni in un punto x o su un insieme di punti memorizzati nel vettore x, si possono utilizzare i comandi **eval** o **feval**.
- Il **calcolo di uno zero** o radice di una funzione **fun** vicino ad un certo valore **x0**, reale o complesso, può essere eseguito con il comando

  **fzero(fun,x0)**

  oltre al valore approssimato della radice, in uscita viene fornito l'intervallo entro il quale il programma ha cercato lo zero

# Polinomi

- Un polinomio è rappresentato da un vettore riga che contiene i coefficienti del polinomio medesimo in ordine decrescente rispetto alle potenze.
- Il polinomio **$3x^3 + 2x + 8$** si rappresenta come:

```
» pol= [3 0 2 8]
```

- Per valutare un polinomio in uno o più punti si usa il comando **polyval** specificando in ingresso i **due vettori pol e x,** che rappresentano i coefficienti e le ascisse:

```
» polyval(pol, 1)
   ans =
     13
```



# Polinomi

- Il comando `roots` serve per calcolare in maniera approssimata gli zeri di un polinomio e richiede in ingresso il solo vettore **pol**.
- Esempio: il polinomio **$x^3 - 6x^2 + 11x - 6$**

```
» p= [1 -6 11 -6]; format long;
» roots(p)
   ans =
    3.00000000000000
    3.00000000000000
    3.00000000000000
```

# Polinomi

- **N.B.** Non sempre si ottengono risultati esatti come nell'esempio, perché i metodi numerici utilizzati incidono sul calcolo, in particolare nel caso di radici multiple.
- Esempio: **$p(x)=(x+1)^7$** i cui coefficienti sono [1 7 21 35 35 21 7 1]

**Operazioni su polinomi**

- **p=conv(p1,p2)** calcola i coefficienti del polinomio prodotto
- **[q,r]=deconv(p1,p2)** calcola il quoziente e il resto della divisione
- **polyfit** calcola gli n+1 coefficienti di un polinomio p di grado n una volta noti i valori di p in n+1 punti distinti



# Programmazione in MATLAB

**Script e Function**

- I files che contengono codice MATLAB sono chiamati M-files
- Possono essere generati usando un qualsiasi editor di testo, l'ambiente Matlab fornisce un editor che può essere richiamato con il comando **edit**

Ci sono due tipi di M-file:

- **Script M-files**
  - possono essere usati come comandi MATLAB dalla finestra comandi
  - operano su dati nel workspace
- **Function M-files**
  - possono essere richiamate da un altro modulo o dalla finestra comandi
  - accettano argomenti in ingresso e producono argomenti in uscita
  - le variabili di una function sono locali

# Programmazione in MATLAB

Il simbolo **%** è usato per indicare un **commento**.

I commenti posizionati all'inizio del file vengono visualizzati tramite help 'nome del programma'

**Function**

Le *funzioni* hanno variabili proprie che vengono cancellate automaticamente dalla memoria al termine della funzione.

- La prima riga di una function deve avere la seguente struttura:

```
Function [parametri in uscita] =
NomeFunzione(parametri in ingresso)
```

- **N.B.** La funzione va salvata come ***NomeFunzione.m***
- Nella funzione devono essere assegnati i valori ai parametri in uscita



# Programmazione in MATLAB

**Esempio**

```
% TrovaMedie calcola la media di un
% vettore di valori casuali la media tra
% il valore massimo e il valore minimo
v=rand(5,1)
media=valmed(v)
medmm=minmax(v)
```

```
function m=valmed(v)
n=length(v)
m=sum(v)/n

function mm=minmax(v)
minimo=min(v)
massimo=max(v)
mm=(minimo+massimo)/2
```

# Programmazione: **ciclo for**

Consideriamo alcuni costrutti fondamentali per la programmazione che permettono l'esecuzione ripetuta di istruzioni e l'esecuzione condizionata di alcune parti.

- Ciclo incondizionato **for… end**
  - **for** Indice = Inizio : Incremento : Fine (*oppure* Indice=vettore)
    blocco istruzioni
    **end**

N.B. E' buona norma scrivere il blocco di istruzioni interne al ciclo for indentato e allineato con l'espressione che governa il ciclo.



# Programmazione: **ciclo for**

- Esempio: media (senza usare sum)

```
n=length(v);
somma=0;
for k=1:n
    somma=somma+v(k)
end
somma=somma/n
```

- Esempio:matrice di Hilbert (doppio ciclo)

```
for i=1:m
  for j=1:n
    H(i,j)=1/(i+j-1);
  end
end
```

# Programmazione: **while**

- Quando si ha la necessità di ripetere istruzioni fino a quando la condizione risulta verificata ***non sapendo a priori il numero di ripetizioni***, si usa il ciclo condizionato **while… end**:
  - `while` ***`Condizione`***
    `blocco istruzioni`
    `end`

  dove **`blocco di istruzioni`** verrà eseguito fintanto che condizione risulta vera

**N.B.** ***`Condizione`*** è un'espressione che Matlab valuta numericamente e interpreta come vera se diversa da 0



# Operatori logici e relazionali

**Operatori relazionali**

- < minore di
- <= minore o uguale di
- > maggiore di
- >= maggiore o uguale di
- == uguale a
- ~= diverso da

**Operatori logici**

- & and
- | or
- ~ not

# Operatori logici e relazionali

- **Esempi di condizioni**

```
>> a=10; b=3; c=25;

>> a==b
ans=
    0
>> a>b
ans=
    1
>> a+b > c
ans=
    0
```



# Programmazione: **while**

Esempi di **while**

- ```
  x = 3.;
  while x < 25
      x = x + 2
  end
  ```
- Successione di Fibonacci

```
a(1)=1; a(2)=1;
n=2;
while a(n) < c
    a(n+1) = a(n) + a(n-1);
    n=n+1;
end
```
```

# Programmazione: if... end

- Può essere necessario eseguire un blocco di istruzioni solo nel caso in cui una particolare condizione risulti verificata:

```
if condizione
    blocco di istruzioni
end
```

- `blocco di istruzioni` sarà eseguito solo se è verificata la *`condizione`*

**Esempio**

```
>> a = 3;
>> if a > 0
>> disp(a);
>> end
     3
```



# Programmazione: if... else... end

- Se si vuole eseguire un blocco di istruzioni nel caso in cui una particolare condizione risulti verificata ed eseguire un altro blocco di istruzioni in caso contrario:

```
if Condizione
    blocco istruzioni 1
else
    blocco istruzioni 2
end
```

- `blocco di istruzioni 1` sarà eseguito solo se è verificata la *`Condizione`* altrimenti verrà eseguito `blocco istruzioni 2`

# Programmazione: if... elseif... end

In generale

```
if  Condizione1
    blocco istruzioni 1
elseif Condizione2
    blocco istruzioni 2
…
else
    blocco istruzioni n
end
```

`blocco istruzioni 1` sarà eseguito solo se *`Condizione1`* risulta verificata, `blocco istruzioni 2` sarà eseguito solo se *`Condizione1`* risulta essere falsa e *`Condizione2`* vera ecc.

`blocco istruzioni n` sarà eseguito soltanto se nessuna delle precedenti condizioni risulterà vera.



# Programmazione: if... else...end

- Esempio

```
if x > 0
   y = sqrt(x);
elseif x == 0
   y = 0;
else
   y = NaN;
  disp('non definito')
end
```

# Programmazione: **switch**

L'istruzione switch effettua una selezione fra più possibilità:

```
switch switch_expr
case case_expr
  statement,...,statement
case {case_expr1,case_expr2,case_expr3,...}
  statement,...,statement
...
otherwise
  statement,...,statement
end
```

# Programmazione: **switch**

**Esempio**:

```
nome='rosa';
% nome='giglio';
% nome='garofano';
switch nome
case 'rosa'
  disp('si tratta di una rosa')
case 'garofano'
  disp('si tratta di un garofano')
case 'giglio'
  disp('si tratta di un giglio')
otherwise
  disp('si tratta di un altro fiore')
end
```

# Ottimizzazione delle prestazioni

Per ottimizzare le prestazioni

- **Vettorizzazione** dei cicli
- **Preallocazione** degli array

L'**aumento di velocità** si ottiene utilizzando

- le funzioni MATLAB
- le notazione "**:**"
- le operazioni su matrici (invece dei cicli)



# Ottimizzazione delle prestazioni

- Esempio ( tavola di logaritmi)

```
x = 0;
for k = 1:1001
     y(k) = log10(x);
     x = x + .01;
end
```

- Una versione vettorizzata dello stesso codice è:

```
x = 0:.01:10;
y = log10(x);
```

Non sempre l'operazione di vettorizzazione è ovvia

# Ottimizzazione delle prestazioni

- **Preallocation**: se non si può vettorizzare un pezzo di codice, si può velocizzare un ciclo FOR preallocando qualsiasi vettore o array nei quali sono immagazzinati i risultati di output.
- Ad esempio, il codice seguente usa la funzione zeros per preallocare il vettore creato nel ciclo FOR.

```
r = zeros(32,1);
for n = 1:32
    r(n) = rank(magic(n));
end
```

- Senza preallocation, l'interprete di MATLAB allarga il vettore r di un elemento ad ogni ciclo.
- La vector preallocation rende l'esecuzione più veloce.



# Salvataggio e richiamo dati

Spesso si hanno dati raccolti con altri programmi che si vorrebbero analizzare mediante Matlab. Il metodo più semplice consiste nell'avere questi dati in un file in formato ASCII.

Consideriamo:

```
1     2              -5
2     0.2500         -9
3     0.0740        -23
4     0.0310        -53
5     0.0160       -105
6     0.0090       -185
7     0.0050       -299
8     0.0030       -453
9     0.0020       -653
10    0.0020       -905
```

# Salvataggio e richiamo dati

Questi dati possono essere salvati su un file (per esempio dati.dat) e caricati in Matlab con il comando load.

In questo esempio, in Matlab viene creata una matrice dati di dimensione 10x3

```
>> load dati.dat
>> whos
Name Size Bytes Class
dati 10x3 240 double array
Grand total is 30 elements using 240 bytes
```

# Salvataggio e richiamo dati

```
>>M = load('dati.dat')
M =
1.0000  2.0000   -5.0000
2.0000  0.2500   -9.0000
3.0000  0.0740  -23.0000
4.0000  0.0310  -53.0000
5.0000  0.0160 -105.0000
6.0000  0.0090 -185.0000
7.0000  0.0050 -299.0000
8.0000  0.0030 -453.0000
9.0000  0.0020 -653.0000
10.0000 0.0020 -905.0000
```

# Salvataggio e richiamo dati

Tutte le variabili definite o calcolate in una sessione di lavoro possono essere salvate mediante il comando **save**

```
>> save      % salva tutte le variabili nel
Saving to: Matlab.mat   % file Matlab.mat

>> save prova % salva nel file prova.mat
```

Per salvare soltanto alcuni valori (per esempio una tabella di dati) e non tutta la sessione di lavoro si utilizza il comando save con la sintassi

***save nomefile variabili formato***

- ***formato*** è un parametro opzionale:
  - -ascii consente di salvare il file in modalità testo
  - se omesso il file viene salvato in formato binario
- ***nomefile*** e privo di estensione, allora il file viene automaticamente salvato con l'estensione .mat

# Salvataggio e richiamo dati

**Esempio:**

```
% salvatabella.m
n=input('fornisci il numero dei valori:');
x=linspace(0,pi,n);
s=sin(x);
c=cos(x);
v=(1:n);
save tabella.dat v x s c -ascii
```

# Salvataggio e richiamo dati

Per visualizzare una tabella di valori precedentemente salvata con il comando save si utilizza il comando **load**

**load *nomefile formato***

*Il formato -ascii è obbligatorio se nomefile è privo di estensione.*

```
% veditabella.m
load tabella.dat
A=tabella;
disp('---------------------------------------');
fprintf('k\t x(k)\t sin(x(k))\t cos(x(k))\n');
disp('---------------------------------------');
fprintf('%d\t %3.2f\t %8.5f\t %8.5f\n',A);
```