MATLAB = MATrix LABoratory

Pregi: rapidità di programmazione, grafica built-in, help in linea
linguaggio standard per l'analisi numerica diffuso ovunque

Difetti: lentezza, instabilità, prezzo (free: Octave e Scilab)

Interfaccia grafica (Desktop layout di default):

1) **Command Window**
Dove si inseriscono tutti i comandi e si ottengono i risultati (a parte, come vedremo, quelli grafici). Più comandi per riga se separati da , o ;
(Il ; sopprime l'output, se non c'è --> output nella variabile d'appoggio ans, ciò che segue il comando % nella riga è interpretato come commento)

```
>> x=1  % assegna il valore 1 alla variabile x e lo scrive in uscita
>> y=2; % assegna il valore 2 alla variabile y senza scriverlo
>> x+y  % calcola la somma delle variabili x e y e scrive il risultato in ans
>> string='ciao';  %assegna la stringa 'ciao' alla variabile string
>> clc  % cancella tutti i comandi della sessione precedente
```

2) **Current Folder**
Mostra il contenuto della directory corrente

```
>> pwd  % mostra il path della directory corrente
>> ls   % mostra tutti i files
>> what  % mostra solo i files matlab (.m)
```

3) **Workspace**
Contiene tutte le variabili in uso. Cliccandoci sopra si apre l'Array Editor che ne mostra il contenuto (utile per gli array). Per richiamare le variabili in uso

```
>> who
>> whos
% Qual'è la differenza?
```

Già che ci siamo provate anche

```
>> why
>> when

>> clear x   % per cancellare una variabile dal workspace
>> clear [all] % per cancellarle tutte
```

4) **Command History**
Contiene giorno per giorno il diario dei comandi inseriti. Per richiamare un comando già usato, nella CW utilizzare la freccia verso l'alto

5) **Editor**
Si attiva dal Menu File\New per scrivere un nuovo script o una function (.m), oppure cliccando 2 volte sul file esistente nel Current Folder

(E) Scrivere uno script (provascript.m) che assegna un valore alle variabili x e y, le somma mostrando il risultato, quindi saluta. Poi eseguirlo:

```
>> provascript
```

Per vedere il contenuto di un file nella CW:

```
>> type nome_file
```

6) **Help**
Per info sui comandi nella CW
>> help nome_comando
Per maggiori dettagli o ricerca dei comandi si può aprire la finestra Help (o i file doc)

```
>> help sqrt
```

Variabili

Nomi e tipo

I nomi non devono iniziare con una cifra e non possono contenere simboli riservati (%,-, ,@). Il tipo non va dichiarato, ma in fase di memorizzazione Matlab distingue tra variabili intere (int8,int16,int32), reali (single o double), complex, character (char, tra apici) o logical (0/1). Per default il tipo è double.

```
%Il range per double e' :
   da -1.79769e+308 a-2.22507e-308 e da 2.22507e-308 a  1.79769e+308

%Il range per single e':
   da -3.40282e+038 a -1.17549e-038 e da  1.17549e-038 a  3.40282e+038

%Il range per int32 e':  da  -2147483648 a 2147483647
```

Provare
```
>> x=2; y=single(x); z=int8(x); w=logical(x);
>> whos
```

Costanti e nomi riservati
realmax, realmin, intmax, intmin, pi (greco), eps(=2.2204e-16), i, j (unità immaginarie), inf(divisione per zero), NaN (Not a Number, 0/0) [Osservazioni]

Provare e commentare:
```
>> a=1; b=1+eps; c=1+eps/2; d=(a==b), e=(a==c), f=0; a/f, d/f

>> z=2+3i, w=3-5j; z+w  % ma i e j possono essere usati come variabili...
>> x=real(z), y=imag(z), c=conj(z), z', ro=abs(z), theta=angle(z)
>> zz=ro*exp(i*theta)   %  formula di Eulero
```

Formati (restano attivi finché non si modificano)
```
format long    % 15 cifre significative
  "    short   % 4 cifre       "      (default)
  "    long e  % come long ma notazione esponenziale
  "    short e % come short ma   "          "
  "    bank    % 2 cifre
  "    rat     % frazione
```

Provare
```
>> a=1/3; a, format long, a, format long e, a, format rat, a
```

Array

Matlab lavora essenzialmente con array, cioè con matrici m x n.
Se m=n=1 --> scalari
Se m=1 --> vettori riga
Se n=1 --> vettori colonna
Ricordarselo nella programmazione aiuta a scrivere programmi più efficienti.

Come assegnare array:

1) <u>esplicitamente</u> (tra parentesi quadre)
righe: elementi separati da spazi o virgole
colonne: elementi separati da ; oppure a capo

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

```
NO:  A=[1,2
     3; 4,5,6]
```

2) <u>tramite un file creato con l'editore</u>
Creare con l'Editor un file B.data contenente i valori

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

Poi caricarlo nel workspace e controllare
```
>> load B.data, B
```

3) <u>tramite funzioni dedicayte di Matlab</u>

Provare:
```
>> eye(m,n)   % matrice identità
>> zeros(m,n) % matrice di zeri
>> ones(m,n)  % matrice di uno
>> rand(m,n)  % matrice di numeri random unif. distr. in (0,1)
>> magic(n)   % quadrato magico nxn
>> hilb(n)    % matrice con elementi a(i,j)=1/(i+j-1)
```

4) <u>tramite : o linspace</u>
```
>> first:increment:last
```
%crea un vettore riga il cui primo elemento è first e i seguenti sono incrementati ogni volta di increment, finché non si supera last.
Se manca l'incremento è 1 per default
```
>> linspace(first,last,n)
```
% distribuzione uniforme di n punti su [first,last] con passo h=(last-first)/(n-1)

Provare
```
>> v=1:10,  w=0:0.1:1, c=1:3:12
>> x=linspace(0,1,20)
```

5) <u>tramite concatenazione</u>
(stando attenti alle dimensioni)

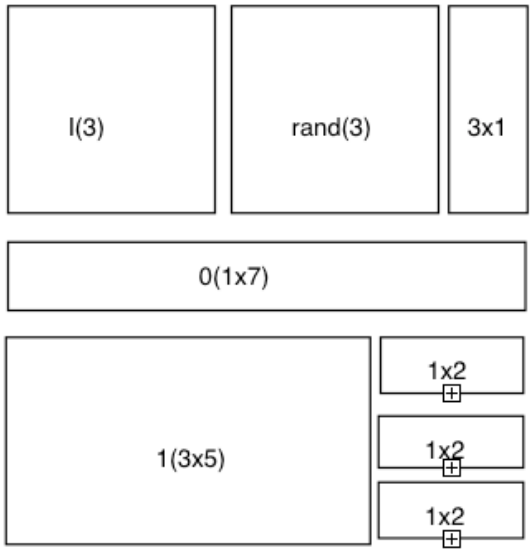
Esempi:
```
>> x=1:5; y=7:11; z=[x y], w=[x; y]
>> v=[z w]   % che succede?
```

6) <u>Tramite uno o più cicli for</u>

```
>> for i=1:n
     v(i)=i^2-1
   end

>> for i=1:n
     for j=1:n
       A(i,j)=1/(i+j-1)
     end
   end    %  lo stesso che hilb(n)
```

Esercizio 1:
Costruire una matrice con la struttura seguente



Alcune funzioni utili per gli array:

```
>> length(v) % restituisce il numero di elementi di un vettore
>> size(A)   % restituisce il vettore [#righe #colonne] di A
>> diag(A)   % matrice diagonale estratta da A
   N.B. Ma vedere anche l'uso diverso di diag(v) se v è un vettore
>> triu(A)   % matrice triangolare superiore estratta da A
>> tril(A)   % matrice triangolare inferiore estratta da A
>> inv(A)    % calcola l'inversa di una matrice quadrata
>> det(A)    % calcola il determinante di una matrice quadrata
>> eig(A)    % calcola gli autovalori di una matrice
>> find(A), spy(A)  % trova e disegna gli elementi diversi da zero di A
>> max, min, sum, prod, mean, sort
   (attenzione: per le matrici agiscono sulle colonne!)
```

Come accedere alle componenti

```
>> v(7)     % il settimo elemento del vettore v
```

Attenzione: se v è una matrice, v(7) è il settimo elemento scorrendo gli elementi colonna per colonna. Provare

```
>> v(2:5)   % è il sottoarray delle componenti 2,3,4 e 5 di v
>> v(1:2:11) % è il sottoarray delle componenti dispari di v

>> A(3,1)   % l'elemento della terza riga prima colonna della matrice A
>> A(3,:)    % è la terza riga di A
>> A(:,2)    % è la seconda colonna di A
```

Operatori: +, -, *, / (divisione a destra), \ (divisione a sinistra)
^ (elevamento a potenza), ' (trasposizione)

Funzionano così tra scalari.
Tra array delle stesse dimensioni + e - funzionano componente per componente,
,/ e ^ lo diventano premettendo un punto: ., ./, .^

Altrimenti:
* indica il prodotto righe per colonne (o il prodotto scalare)
\ , / servono a risolvere sistemi lineari

Per gli array di complessi ' calcola il trasposto coniugato, .' il trasposto non coniugato

Se A(mxn), b(mx1) --> x=A\b è il vettore (nx1) soluzione di Ax=b mediante l'algoritmo di Gauss

Se d(1xn) --> y=d/A è il vettore (1xm) soluzione di yA=d mediante Gauss

<u>Esercizio 2.</u>
Costruire nella modalità a piacere delle matrici A(3x4), B(3x4), C(4x2), D(3x3) e dei vettori v(4x1), w=(1x3), z=(3x1). Poi provare e commentare i seguenti comandi

```
>> A+B
>> 3*A, B/2
>> B*C, D^2
>> B.*A
>> A.^2
>> w*z, z*w, w.*z'
>> x=A\z
>> y=v'/A
```

<u>Esercizio 3</u>. Assegnato un vettore v(5) e una matrice A(3,5), provare i comandi:

```
>> max(v), max(A), max(max(A))
>> sort(v), sort(A)
>> sum(sum(A))
```

<u>Sol. Esercizio 1</u>

```
>> A=[eye(3),rand(3),[1 2 3]';zeros(1,7);ones(3,5),[[1,2];[3,4];[5,6]]]
```

Disegnare il grafico di una funzione

Supponiamo di voler disegnare il grafico della funzione f(x)=sin(pi*x) nell'intervallo [0,1]. Possiamo farlo nella CW attraverso i comandi:

```
>> x=0:0.05:1; plot(x,sin(pi*x))
```

I grafici creati in output appaiono in una finestra separata

```
>> close  % chiude la finestra grafica attiva
>> close all  % le chiude tutte
>> clf   % cancella il contenuto, non la finestra
>> figure  % attiva una diversa finestra grafica
```

Esercizio: provate a generare lo stesso vettore x con linspace

Ma possiamo farlo anche attraverso uno script, cioè un file Matlab con estensione .m scritto con l'editor.

Esempio: provascript.m

```
% Questo programma disegna il grafico della funzione f(x)=sin(pi*x)
% nell'intervallo [0,1]
x=0:0.05:1; plot(x,sin(pi*x))
title('grafico')     % titolo del grafico
legend('sin(pi*x)')  % legenda associata al grafico
axis([0 1 -1.1 1.1])    % estremi della finestra [xmin xmax ymin ymax]
```

Attenzione: axis va dato dopo il plot (per altri usi si veda >> help axis)

Per eseguire il programma:
```
>> provascript
```

Se volessimo confrontarlo con il grafico della funzione sin(2*pi*x), possiamo creare una nuova figura aggiungendo allo script il blocco

```
figure    % crea una nuova finestra grafica (senza sovrascriverebbe)
plot(x,sin(2*pi*x))
title('grafico')     % titolo del grafico
legend('sin(2*pi*x)')  % legenda associata al grafico
axis([0 1 -1.1 1.1])
```

oppure possiamo sovrapporli nella stessa figura modificando così lo script

```
plot(x,sin(pi*x),'b',x,sin(2*pi*x),'r') % I grafico in blu (default), II in red
title('grafico')
axis([0 1 -1.1 1.1])
legend('sin(pi*x)','sin(2*pi*x)')  % legenda associata al grafico
```

Quando uno script ha dei parametri di input e degli output è più propriamente una function, con la struttura del seguente file name_fun.m

```
function [output_args]=name_fun(input_args)
% descrizione dettagliata della funzione, con specifica di input e output
% corpo
....
[output_args]=....
end
```

Nella CW (o in uno script che la utilizzi) va poi chiamata nella forma

```
>> [lista_out]=name_fun(lista_in)
```

dove ovviamente lista_out dovrà corrispondere a output_args, lista_in a input_args.

<u>Esempio</u>
Ecco una funzione che consente di determinare graficamente lo zero di una funzione attraverso il cursore del mouse

```
function xapp=plotapp(func,a,b,h);
%Input
%       func='f', stringa di una funzione di cui si vuole appr. lo
%               zero graficamente (es.: 'sin')
%       a,b = variabili scalari per intervallo [a,b] dove f ammette zero
%       h= passo per tabulazione uniforme di [a,b]
%Output  xapp=radice approssimata di func in [a,b]
X=a:h:b;
Y=feval(func,X);% valuta una funzione data come stringa SENZA argomento, in X
plot(X,Y);
hold on;
title('Posiziona il cursore vicino alla radice e clicca con il mouse');
[xapp,yapp]=ginput(1);
       %ginput vi permette di selezionare punti da una figura
       %usando il mouse per posizionare il cursore.
hold off
fprintf(' La radice approssimata e` tale che: f(%4.6f)=%0.12f\n',xapp,yapp);
```

Provate ad eseguire
```
>> plotapp('sin',0,4,0.1)
```

Funzioni più complicate si possono definire mediante una function dedicata, ad esempio fun.m, e poi eseguire
```
>> plotapp(@fun,a,b,h)
```
(il simbolo @ davanti al nome della funzione opera come un puntatore ad essa).

<u>Esercizio</u>.
Determinare graficamente con un errore dell'ordine di 0.1 gli zeri della funzione f(x)=e^x-x^2(sin(x)+1).

Una lista delle più comuni funzioni matematiche elementari in Matlab:

sin, cos, tan, asin, acos, atan
exp, log, log10, log2
abs, sqrt, sign, rem, mod, factorial
conj, real, imag, angle
round, floor, fix, ceil

Per una lista completa
```
>> help elfun
```

Provare anche il comando

```
>> fplot('fun',[a,b])
```

che produce il grafico 'accurato' di fun.m nell'intervallo [a,b].

Altri comandi grafici di utilità

```
>> xlabel('string')
>> ylabel('string')
>> subplot(m,n,i)   % crea il grafico i di una finestra con mxn grafici
```

Colori e stili:
y (yellow), m (magenta), c (cyan), r (red), g (green), b (blue), k (black)
. (point), o (circle), x (x-mark), + (plus), - (solid), * (star), : (dotted)
-. (dashdot), -- (dashed)

<u>Esercizio</u>. Scrivere un programma che confronti i grafici per punti delle funzioni round, floor, fix e ceil in una stessa finestra, usando colori e simboli diversi. Poi modificarlo in modo che i suddetti grafici appaiano in una matrice 2x2 di quattro sottofinestre contigue mediante il comando subplot.

Cicli e alternative

```
% numero di iterazioni prefissate
for i=1:n
   ... istruzioni ...
end

% numero di iterazioni condizionate
while condizione
   ... istruzioni ...
end

if condizione
   ... istruzioni ...
end

if condizione
   ... istruzioni ...
else
   ... istruzioni ...
end

if condizione1
   ... istruzioni ...
elseif condizione2
   ... istruzioni ...
else
   ... istruzioni ...
end

switch (string)
case 'uno'
  ... istruzioni ...
case 'due'
  ... istruzioni ...
case 'tre'
  ... istruzioni ...
otherwise
  ... istruzioni ...
end
```

Operatori relazionali e logici

<, <=, >, >=
== (uguale), ~= (diverso)
& (and), ~ (not), | (or), xor (or esclusivo)
any, all (esempi...)

Le variabili booleane assumono solo i valori 0 (false) o 1 (true). Ogni altro numero diverso da zero è considerato true.

Osservazione. Per inserire in uno script molte righe di testo si può digitare

if 0
... righe di testo
end

Esercitazione: la successione 3*x+1
Il seguente programma studia la successione per ricorrenza che partendo da un numero intero assegnato, ad ogni passo divide per due se il numero è pari, mentre moltiplica per 3 e somma 1 se il numero è dispari. La successione si arresta se viene raggiunto il numero 1 (*capite perché? o meglio, capite cosa succederebbe continuando?*)

Leggere con attezione il listato, cercando di capire prima di eseguirlo esattamente cosa fa ogni istruzione. Poi eseguirlo e tornare sulle istruzioni non capite.

trexpiuuno.m

```
% I PARTE
u0=input('Inserisci il numero intero iniziale n= ');
cont=0; c=1; x(1)=u0;
while ~cont
    if mod(u0,2)
        u0=3*u0+1;
    else
        u0=u0/2;
    end
    c=c+1; x(c)=u0;
    disp(num2str(u0));
    if (u0==1) cont=cont+1;
    end
end
disp(['Partendo da ',num2str(x(1)),' numero di termini  ',num2str(c)]);
M=max(x(1:c));
plot(1:c,x(1:c),'-')
axis([1 c 0 M])
title(['x0 = ',num2str(x(1)),', numero termini = ',num2str(c),...
    ', valmax = ',num2str(M)])

% II PARTE
N=input('Inserisci la dimensione del vettore N= ');
for i=2:N
    u0=i; x(1)=0;
cont=0; c=0;
while ~cont
```

```
    if mod(u0,2)
        u0=3*u0+1;
    else
        u0=u0/2;
    end
    c=c+1;
   % disp(num2str(u0));
    if (u0==1) cont=cont+1;
    end
end
disp([num2str(i),': numero di iterazioni  ',num2str(c)]);
x(i)=c;
end
[M,ind]=max(x(1:N));
figure
plot(1:N,x(1:N),'*')
title(['max#iteraz = ',num2str(M),' per  x0 =',num2str(ind)])
```

Esercizio: trovare il dato iniziale <= 1000 che genera la sequenza più lunga.

Ancora sugli M - files (.m)
Script files: Tutte le variabili usate sono globali. Ogni script può fare riferimento ad altri scripts, anche ricorsivamente.
Function files: Le variabili sono per default locali (se non espressamente dichiarate: v. >> help global)

Esempio (randint.m):

```
function c=randint(m,n,a,b)
% randint(m,n) crea una matrice mxn con valori random tra 0 e 9
% randint(m,n,a,b) crea una matrice mxn con valori random tra a e b
if nargin<3, a=0; b=9; end
c=floor((b-a+1)*rand(m,n))+a;
```

Scrivere ed eseguire in CW:

```
>> randint(4,5)
>> randint(4,5,1,20)
```

N.B. uso di nargin (# di argomenti input) e nargout (# di argomenti output).

Le prime righe commentate all'inizio di ogni M-file (script o function) vengono associate all'help corrispondente. Provate

```
>> help randint
```

Alcuni comandi di utilità negli scripts.

```
a=input('dammi a')      stampa il commento e sospende l'esecuzione finché non
                        viene inserito da tastiera un valore per a
pause       arresta l'esecuzione finché non si preme un tasto
pause(n)    arresta l'esecuzione per n secondi
break       arresta l'esecuzione del programma, o esce dal ciclo in cui si trova
            (Attenzione: un programma in esecuzione può sempre essere interrotto
            digitando nella CW uno o più volte il comando CTRL-C)
...         permette di spezzare una riga di comando per andare a capo
tic         inizia a contare il tempo
toc         finisce di contare il tempo e lo stampa a video
t=toc;      finisce di contare il tempo e lo memorizza nella variabile t
disp('commento')  stampa il commento contenuto tra apici
error('commento') stampa il commento contenuto tra apici e arresta il programma
```

Esercizio. Confrontiamo in termini di tempi di calcolo due modi di risolvere un sistema lineare Ax=b. Scrivere un programma che assegnata una matrice quadrata A di dimensione n (dato in input) e un vettore b di dimensione nx1 (per esempio generati con la function randint), calcola la soluzione del sistema una volta attraverso la funzione Matlab \ (x=A\b), una volta attraverso l'inversione diretta della matrice (x=inv(A)*b), calcolandone e confrontandone i tempi di esecuzione. Provare con n=10,100,1000,2000,3000.

****

Per finire una panoramica veloce di quel che si può trovare già fatto:

```
>> help elfun     % funzioni matematiche elementari
>> help specfun   % funzioni matematiche speciali
>> help matfun    % funzioni di utilità per le matrici
>> help funfun    % funzioni di utilità per le funzioni
>> help optimfun  % funzioni di ottimizzazione e ricerca degli zeri
```

****

Matrici e sistemi lineari (solo per cominciare)

Norme di vettori e matrici
È spesso utile valutare la norma di vettori o matrici. Matlab fornisce per questo la funzione norm. Vediamone l'uso, richiamando le definizioni.

```
>> norm(X,k)
```

Se X è un vettore:

```
norm(X,p) %  indica la p-norma di X [=SUM(ABS(X).^p)^(1/p)], per ogni p>=1.
norm(X)=norm(X,2)
norm(X,Inf)  % indica la norma infinito di X, cioè il più grande elemento di abs
(X)
norm(X,-Inf)  % fornisce il più piccolo elemento di abs(X)
```

Se X è una matrice mxn :

```
norm(X,1)  % norma matriciale indotta dalla norma 1 (=max somma per colonne dei
valori             % assoluti)
norm(X,Inf) % norma matriciale indotta dalla norma infinito (=max somma per
righe dei           % valori assoluti)
norm(X)=norm(X,2) % norma matriciale indotta dalla norma due (=sqrt(rho(X' X))),
con rho                 % raggio spettrale di X, cioè il più grande valore
singolare di X.
                  % Se X simmetrica, coincide con l'autovalore di X di massimo
modulo
norm(X,'fro')   % norma di Frobenius (=sqrt(trace(X' X))
```

Nella risoluzione dei sistemi lineari risulta fondamentale il buon condizionamento della matrice del sistema (v. richiami di teoria). Matlab fornisce il comando cond:

```
>> cond(X,p)   % = norm(X,p) * norm(inv(X),p)
```

dove p=1,2,Inf oppure 'fro' [e cond(X)=cond(X,2), e coincide col rapporto tra il più grande e il più piccolo valore singolare di X].

Esercizio 1. Presa una matrice quadrata A (nxn), calcolare le norme matriciali 1, 2, infinito e Frobenius di A usando opportunamente i comandi di Matlab max, sum, sqrt, trace, eig e le usuali operazioni tra matrici. Poi confrontare i valori ottenuti con quanto fornito dai comandi norm corrispondenti.

Esercizio 2. Si consideri la matrice di Hilbert, definita da A(i,j)=1/(i+j-1), notoriamente mal condizionata. Verifichiamolo risolvendo un sistema lineare associato e guardando l'errore commesso in funzione del numero di condizionamento. Si esegua lo script seguente:

```
A=hilb(n);
solex=ones(n,1);
b=A*solex;
x=A \ b
err=norm(x-solex)/norm(solex)
cond(A)
```

per n=10, 12, 15, osservando gli effetti del peggioramento del condizionamento di A.

Ricordando altre funzioni Matlab per le matrici, provate ad eseguire anche i seguenti comandi (facendo le considerazioni opportune):

```
>> A=hilb(10);
>> det(A)
>> format long
>> L=eig(A)      % calcola il vettore degli autovalori di A
>> max(L)/min(L)
```

Calcolare anche il condizionamento rispetto alle altre norme matriciali.

Esercizio 3. Provare a generare la matrice di Hilbert di ordine 5 mediante un vostro script, confrontando quanto ottenuto col comando hilb(5). Eseguire poi il comando

```
>> type hilb.m
```

e analizzare il listato cercando di capire come funziona (è un ottimo esempio dell'efficienza della programmazione di Matlab).

Costruzione di matrici sparse

In molte applicazioni compaiono matrici di dimensioni notevoli, ma essenzialmente sparse, cioè con pochi valori diversi da zero, e spesso concentrati lungo delle diagonali. Vediamo come Matlab aiuti a costruire queste matrici risparmiando operazioni inutili (moltiplicazioni per zero) e occupazione di memoria.

<u>Esempio 1</u>.

Supponiamo ad esempio di voler costruire una matrice A di dimensioni 7x7 i cui unici elementi non nulli sono a(2,3)=1, a(3,6)=-2, a(5,5)=3, a(7,1)=1. Potremo scrivere

```
>> i=[2,3,5,7]; j=[3,6,5,1]; v=[1,-2,3,1];
>> A=sparse(i,j,v)
>> T=full(A)
>> nnz(A)
>> find(A)
```

I tre vettori sono rispettivamente il vettore degli indici i, quello degli indici j e quello dei valori. Nella sua forma sparse vengono tenuti in memoria solo gli elementi non nulli insieme ai suoi indici. Per vedere l'intera matrice dovremo 'riempirla con gli zeri' attraverso il comando full. La funzione nnz(A) restituisce il numero di elementi non zero di A. Cosa trova il comando find ?

Esempio 2.
Supponiamo ora di voler costruire una matrice tridiagonale A 9x9, con tutti 2 sulla diagonale principale e -1 nelle due diagonali adiacenti. Possiamo usare il comando spdiags (sparse diagonals):

```
>> e=ones(9,1);
>> A=spdiags([-e 2*e -e], -1:1, 9,9);
>> full(A)
```

In generale spdiags(B,d,m,n) crea una matrice sparsa mxn ponendo le colonne della matrice B lungo le diagonali specificate da d (0=diagonale principale, interi positivi = sopradiagonali, negativi = sottodiagonali).
Ricordiamo poi i comandi tril(A) e triu(A) che estraggono la parte triangolare inferiore o superiore di una matrice A. Nell'esempio seguente estraiamo una matrice tridiagonale da una piena:

```
>> F=floor(10*rand(8)); T=triu(tril(F,1),-1)   % capite come funziona?
```

Risoluzione dei sistemi lineari
Il comando \ (backslash):
>> x=A\b
calcola la soluzione x (vettore colonna nx1) del sistema Ax=b, se A è una matrice quadrata nxn e b un vettore colonna nx1. Risolve il sistema con l'algoritmo più efficiente, dopo aver effettuato test preliminari su A. Ad esempio se A è triangolare inferiore, usa il metodo della sostituzione in avanti; se è triangolare superiore quello della sostituzione all'indietro.

Il comando / (slash):
>> y=b/A
calcola la soluzione del sistema yA=b, se ora y e b sono vettori riga 1xn (corrisponde alla divisione tra array).

Esercizio 4.
Se A e il vettore e sono quelli dell'Esempio 2 , sia B=10*A, b=e/10 e si risolva il sistema lineare Bx=b nei tre modi diversi:

```
>> x=inv(B)*b
>> x=B\b
>> x=(b'/B)'
```