AA 2011/2012: Tutorato di Matlab – Informazioni utili

Tutor:
Simone Gallerani
Stefano Vitali
e-mail:
simone.gallerani@student.unife.it
stefano.vitali@student.unife.it

- Professori: Paltin Ionescu, Cinzia Bisi
- Le lezioni si tengono il lunedì dalle 14:00 alle 16:30 nel Laboratorio di Informatica Grande
- Studenti suddivisi in 2 gruppi: frequenze a lunedì alterni
- Materiale didattico:
  - dispense fornite di volta in volta, disponibili in copisteria
- Strumenti didattici:
  - postazioni Windows in laboratorio (per accedere → user: Utente _ password: Infonew1)
  - ambiente Matlab R2011b
- Per ottenere Matlab? 
- Per potersi presentare all'esame: obbligo di frequenza per almeno 3 delle 5 lezioni previste → registro presenze da firmare.
- Modalità d'esame: 4 esercizi da risolvere al pc → calcoli matriciali, sistemi a parametri variabili, geometria dello spazio
- Ricevimento/Chiarimenti: alla fine della lezione, oppure via mail

Lezione n° 1

Sommario:

- Matlab… una breve introduzione
- Variabili e matrici
- Operazioni elementari
- Esempi
- M-File e funzioni

Una breve introduzione…

L'ambiente Matlab (acronimo per *Matrix Laboratory*) è un prodotto della Mathworks, integrato per il calcolo numerico-scientifico e per la visualizzazione grafica, nel quale è possibile eseguire interattivamente una istruzione o comando per volta; ogni comando permette di definire variabili, valutare espressioni e funzioni, eseguire grafici,…

Matlab offre ambienti tali da poter studiare metodi statistici, identificare ed ottimizzare modelli dinamici, acquisire dati, sviluppare algoritmi, analizzare segnali, progettare sistemi di controllo, ecc…e quindi è un software molto usato nella ricerca scientifica e nella risoluzione di problemi di ingegneria.

La sua principale caratteristica è quella di non operare semplicemente con numeri ma direttamente con matrici: oltre a semplici funzioni di base, esistono delle estensioni, i Toolbox, che permettono di allargare le conoscenze verso funzioni più specialistiche.

Allo stesso tempo Matlab è un linguaggio di programmazione di tipo interpretativo: è possibile scrivere programmi, ovvero serie di istruzioni o comandi contenuti in un file, detto m-file; ogni istruzione del programma è interpretata e, se priva di errori di sintassi, è eseguita immediatamente; è possibile creare nuove "funzioni", ovvero set di istruzioni contenute in un m-file, precedute da una istruzione speciale (function) che specifica i parametri della nuova funzione (ossia i dati che elabora e i risultati che ottiene).

…l'ambiente Matlab

Attivando Matlab facendo due "click" con il tasto sinistro del mouse sulla rispettiva icona, compare la principale finestra, detta Command Window: si può notare che il cursore è posizionato accanto al prompt di Matlab (>>) in attesa di un comando.

NOTE:

- Ogni comando si manda in esecuzione premendo INVIO. Ogni comando può essere interrotto premendo CTRL-C.

- Se un comando è troppo lungo si può spezzare su più linee, terminando ogni linea che ha una successiva linea di continuazione con tre punti … e premendo INVIO:
  >> prima linea … <INVIO>
  seconda linea … <INVIO>
  ultima linea <INVIO> (per mandare in esecuzione il comando)

- Matlab è **"case-sensitive"**: 'A' e 'a' indicano entità diverse; i nomi della funzioni predefinite sono minuscole, altrimenti non vengono riconosciute.

Il principale scopo di Matlab è valutare espressioni, dalle più semplici alle più complesse: il risultato dell'espressione viene memorizzato in una variabile definita dall'utente o predefinita.
Tutte le variabili presenti in memoria sono racchiuse in una zona che prende il nome di "spazio di lavoro" (Workspace).
Il comando who elenca le variabili del workspace, mentre whos visualizza tutte le variabili con descrizione di struttura più dettagliata; il comando clear, invece, cancella tutte quelle definite fino a quel momento.

Altri comandi di uso generale sono:

- pwd restituisce la directory corrente
- dir elenca i file della directory
- dir *.m elenca i file .m
- cd <newdir> cambia la directory di lavoro
- clc pulisce la finestra

La cronologia dei comandi viene elencata della Command History: questa è utile per richiamare velocemente un comando o un'istruzione digitata durante la sessione di lavoro. È possibile copiare un comando direttamente nella finestra dei comandi semplicemente selezionandolo e trascinandolo con l'aiuto del mouse.
Infine c'è la Current Folder, ovvero la "cartella di lavoro", ove c'è la possibilità di esplorarne il contenuto.

Cominciamo!

In Matlab qualsiasi dato è trattato come una variabile → per introdurre una variabile basta assegnarle un nome e un valore in questo modo:

```
>> a = 3 %Dichiarazione di una variabile

a =

    3
```

La variabile `a` vale 3. Omettendo il ; viene stampato il nome della variabile e il suo valore, altrimenti non c'è *echo* dei comandi. Se si digita soltanto un valore e non lo si assegna a una variabile, Matlab assegna di default tale valore alla variabile **`ans`**.
Non interessa il tipo di dato creato, o la sua precisione (16 o 32 bit per esempio), quanto invece che sia mantenuta in memoria.

```
>> b = 5;
```

N.B.: un singolo dato (scalare) può essere visto come una matrice 1x1.
In Matlab, tutti i dati sono matrici, per cui Matlab alloca dinamicamente lo spazio quando occorre senza dover definire a priori le dimensioni delle matrici.

Attenzione a non sovrascrivere le variabili!

```
>> b = 9; %Se sovrascrivo, perdo il valore precedente
```

Un vettore è una particolare matrice…

```
>> A = [1 2 3]; %Vettore riga
```

oppure, in modo equivalente

```
>> A = [1,2,3]; %Scrittura equivalente

>> B = [4;5;6]; %Vettore colonna
```

Se non è un vettore…

```
>> C = [1,2,3;4,5,6]; %Matrice 2x3 – 2 righe, 3 colonne

>> B = [4 1;5;6] %Questa non è una matrice, quindi Matlab dà errore!!!

>> B = []; %Matrice vuota, ossia priva di elementi
```

L'uso dell'apice…

```
>> B = [4 5 6]'; %Scrittura equivalente di un vettore colonna con l'uso
                 %della trasposta
```

Estrazione di singoli elementi…

```
>> B1 = B(3,1) %Selezione di un elemento di una matrice

B1 =

     6

>> B1 = B(1,3); %Esempio di indici errati!!!
```

In Matlab le variabili non sono dichiarate preventivamente al loro uso dall'utente, ma sono create quando sono necessarie da un gestore della memoria che alloca lo spazio necessario per una matrice dinamicamente, mediante il cosiddetto dimensionamento automatico. Ne seguono 2 esempi:

```
>> B(1,3) = 9 %Scrittura di un singolo elemento di una matrice

B =

     4     0     9

     5     0     0

     6     0     0

>> B(3,3) = 10; %Matlab fa diventare B una matrice 3x3 in modo da creare la
               %posizione (3,3) in cui vogliamo scrivere e riempie le
               %altre nuove celle con degli zeri
```

Abbiamo visto come si dichiara una matrice:

```
>> M = [3 5 6; 7 9 0]; %Dichiarazione di una matrice
```

Ora vogliamo visualizzarne le dimensioni: si usa la funzione predefinita **size** che ritorna un vettore di due interi, la dimensione per righe e quella per colonne.

```
>> size(M); %Dimensioni di una matrice

>> [m n] = size(M); %In questo modo ci salviamo entrambe le dimensioni di M

>> k = size(M,2); %In k verrà salvata solo la seconda dimensione
```

NOTA BENE

Per una panoramica sui comandi di Matlab digitare:

```
>> helpdesk %Help multimediale, attraverso cui si possono scaricare utili
            %manuali: ciò conviene prima di implementare delle funzioni
            %che potrebbero semplificarvi la vita senza perdere la testa
            %in molteplici righe di codice.
```

Digitando **help** dal prompt di comando compare la lista completa dei toolbox presenti: lanciando

```
>> help <nome toolbox>
```

si ha l'elenco completo delle funzioni disponibili per quel toolbox. Digitando

```
>> help <nome comando>
```

si accede alla descrizione di quel comando.

Selezione di una sottomatrice: per accedere a intere righe o colonne di una matrice, si usa **la wildcard** ":" (carattere *jolly*)

```
>> N = M(:,1); %Seleziona e salva in N la prima colonna di M

>> K = M(1:2,2:3); %Sottomatrice di M selezionata prendendo la prima e la
                   %seconda riga, e la seconda e terza colonna
```

Quindi i ":" fanno variare un indice di selezione da un valore iniziale ad uno finale, relativamente per righe e colonne: la sintassi si estende, ovviamente, su matrici di dimensioni arbitrarie

Scrivendo

```
>> L = [N K]; %Esempio di unione fra matrici
```

è possibile verificare l'espressione relazionale di uguaglianza fra matrici:

```
>> L == M %Confronto logico fra matrici

ans =

     1     1     1

     1     1     1
```

ESPRESSIONI RELAZIONALI

All'interno di una espressione relazionale si possono usare i seguenti operatori:

- = = uguale
- ~ = diverso
- < minore
- > maggiore
- <= minore o uguale
- >= maggiore o uguale

Con i ":" si può far generare a Matlab un vettore automaticamente:

```
>> V = [1:6;2:7]; %Dichiarazione di un vettore usando il simbolo ":"

>> Y = [1:2:6]; %Vettore riga con valori che vanno da 1 a 6 con passo 2
```

Il comando **linspace(*a,b,n*)** genera *n* elementi equispaziati tra gli estremi specificati inclusi *a* e *b*: se *n* viene omesso, vengono generati 100 punti.

```
>> J = linspace(1,10,50); %Dichiarazione di un vettore usando il comando
                          %linspace
```

Esistono comandi che generano automaticamente alcune matrici notevoli:

```
>> O = ones(3); %O è una matrice quadrata di ordine 3 i cui elementi sono
               %uguali a 1


>> Z = zeros(3); %Z è una matrice quadrata di ordine 3 i cui elementi sono
                %uguali a 0


>> E = eye(3); %E è la matrice identità di ordine 3
```

ATTENZIONE all'uso del comando **eye** _ esempi

```
>> R = eye(size(M));

>> R = eye(size(M,2));

>> R = eye(min(size(M)));
```

```
      ------------------------------clear--------------------------------
```

OPERAZIONI TRA MATRICI

Siano A e B due matrici mxn

```
>> A = [5 1 2];
>> B = [3 2 7];
```

Abbiamo già visto il cos'è la **trasposta** di una matrice, ovvero:

→ se A è una matrice **mxn**, la sua trasposta è una matrice B **nxm** che si ottiene dalla A scambiando le righe con le colonne. L'operatore di trasposizione è l'apice.

Ora vediamo come usare i semplici operatori aritmetici:

+	somma
-	differenza
*	prodotto
/	quoziente
^	elevamento a potenza

```
>> C = A+B; %Somma di matrici → matrice somma A+B i cui elementi sono la
            %somma degli elementi di ugual indici di A e B rispettivamente

>> D = A-B; %Differenza di matrici → matrice differenza A-B i cui elementi
            %sono la differenza degli elementi di ugual indice di A e B
            %rispettivamente

>> x = [1 2 3]';

>> A = A+x %NON SI PUÒ ESEGUIRE! Matlab dà errore!!!

>> A1 = A+3; %Quando si somma uno scalare ad una matrice si ottiene una
             %matrice i cui elementi sono stati incrementati,
             %singolarmente, per quello scalare

>> B1 = B*2; %Prodotto di matrice per scalare

>> E = A*B'; %Prodotto fra matrici: ATTENZIONE alle dimensioni delle
             %matrici!
```

RIEPILOGO DELLE OPERAZIONI TRA MATRICI

A+B	somma
A-B	differenza
B*C	prodotto
A^p	elevamento a potenza solo con esponenti p interi positivi
A'	trasposizione
A\ B	risoluzione di sistemi: inv(A)*B
A/B	calcolo di A*inv(B)

Se le dimensioni delle matrici sono incompatibili per il tipo di operazione da eseguire viene visualizzato un messaggio di errore.
Le operazioni operano elemento per elemento se uno degli operandi è scalare.

OPERAZIONI PUNTO

Tali operazioni operano elemento per elemento (come per la somma algebrica tra matrici):

A.*B esegue A(i,j)*B(i,j)
A.\B esegue B(i,j)/A(i,j)
A./B esegue A(i,j)/B(i,j)
A.^B esegue A(i,j)^B(i,j)

```
>> L = A*B
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> L = A.*B %Esempio di utilizzo della notazione ".”

L =

    15     2    14

>> A = [2 3 4;6 0 2;1 4 9];
>> B = [1 2 6;3 3 2;6 6 8];

>> C = A./B; %Altro esempio di utilizzo del "." applicato alla divisione


      ------------------------------clear-------------------------------


%Esempio - Decomposizione di matrice

%Definiamo A col comando magic

>> A = magic(3) %Non è una matrice casuale! Il comando magic(n) fornisce una
                %matrice quadrata nxn di qualsiasi dimensione (con elementi
                %da 1 a n^2): sommando gli elementi di ogni riga, ogni
                %colonna e ogni diagonale, in tutti i casi il risultato è lo
                %stesso -> Vedi help

A =

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

```
%Calcoliamo una matrice simmetrica a partire da A
```

La struttura di una matrice simmetrica è la seguente:

```
>> B = (A+A')*1/2; %La matrice B è tale che b_{i,j} = b_{j,i}
```

In altre parole, B è simmetrica se $B = B^T$:

```
>> B == B'; %Verifichiamo che la matrice ottenuta sia simmetrica
```

```
%Calcoliamo una matrice antisimmetrica a partire da A
```

La struttura di una matrice antisimmetrica è la seguente

$$C_{n\times n} = \begin{bmatrix} 0 & c_{12} & \cdots & c_{1n} \\ -c_{12} & 0 & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -c_{1n} & -c_{2n} & \cdots & 0 \end{bmatrix}$$

```
>> C = (A-A')*1/2; %La matrice C è tale da avere al più n(n-1)/2 elementi
                   %indipendenti e non nulli
```

In altre parole, C è antisimmetrica se $C = -C^T$:

```
>> C == -C'; %Verifichiamo che la matrice ottenuta sia antisimmetrica
```

Da qui la decomposizione di matrici → Data una matrice quadrata A, questa è sempre possibile decomporla in una somma di due matrici come segue:

$$\boldsymbol{A = A_{simmetrica} + A_{antisimmetrica}}$$

dove

$$\boldsymbol{A_{simmetrica} = \frac{1}{2}(A + A^T)}$$

$$\boldsymbol{A_{antisimmetrica} = \frac{1}{2}(A - A^T)}$$

Nel nostro caso B=$\boldsymbol{A_{simmetrica}}$ e C=$\boldsymbol{A_{antisimmetrica}}$, quindi:

```
>> A == (B+C); %Infine verifichiamo che la somma delle matrici calcolate sia
               %effettivamente %la matrice A di partenza

      ------------------------------clear-------------------------------
      ------------------------------clc---------------------------------
```

```
%Esempio – Aeroplano
```

Supponiamo di avere i seguenti dati relativi al tempo che ci impiega un aereo ultraleggero per raggiungere diverse destinazioni, in base alla rispettiva velocità media di volo:

```
%Velocità [Km/h]       200     250     400     300
%Temp [h]               2       5       3       4

%Copiamo la tabella in una matrice

>> A = [200 250 400 300;2 5 3 4];

%Mettiamo in un vettore i soli dati relativi alla velocità

>> V = A(1,:);

%Mettiamo in un vettore i soli dati relativi al tempo

>> T = A(2,:);

%Calcoliamo lo spazio totale percorso dall'aeroplano

>> S = V*T';

%Il prodotto fra matrici appena svolto equivale a fare il prodotto scalare
%fra i vettori V e T'. In realtà il comando dot fa il prodotto dei due
%vettori anche se non facciamo la trasposta del secondo.

>> E = dot(V,T);

      ------------------------------clear-------------------------------
      ------------------------------clc---------------------------------

%Norma di un vettore
```

Come sappiamo, il prodotto scalare $\boldsymbol{a} \cdot \boldsymbol{b}$ ($\boldsymbol{a}$ e $\boldsymbol{b}$ vettori $\in \boldsymbol{V}(\mathbb{R})$) è un numero reale che è geometricamente definito come

$$\boldsymbol{a} \cdot \boldsymbol{b} = \| \boldsymbol{a} \| \, \| \boldsymbol{b} \| \, \boldsymbol{cos\theta}$$

dove $\| \boldsymbol{a} \|$ (come $\| \boldsymbol{b} \|$) è la lunghezza del vettore $\boldsymbol{a}$ e $\boldsymbol{\theta}$ ($0° \leq \boldsymbol{\theta} \leq 180°$) è l'angolo compreso tra $\boldsymbol{a}$ e $\boldsymbol{b}$. Questa definizione implica di aver preventivamente definito il concetto di angolo e di lunghezza, mentre nell'approccio analitico la lunghezza, ovvero la **norma** (in inglese *norm*) può essere definita come la grandezza derivata dal prodotto scalare

$$\| \boldsymbol{a} \| = \sqrt{\boldsymbol{a} \cdot \boldsymbol{a}} = \sqrt{\sum_{\boldsymbol{k}} \boldsymbol{a}_{\boldsymbol{k}}^{\boldsymbol{2}}} = \sqrt{\boldsymbol{a}^{\boldsymbol{T}} \boldsymbol{a}}$$

a_k è il k-esimo elemento del vettore $\boldsymbol{a}$. Consideriamo, per esempio, il vettore:

```
>> R = [55,36,25];
```

Per calcolare $\|\mathbf{R}\|$, quindi, occorre sommare tutti i suoi elementi (introducendo l'operazione cumulativa **sum**), ciascuno moltiplicato per se stesso, e porre la somma sotto radice utilizzando la funzione aritmetica **sqrt**:

```
>> N = sqrt(sum((R.*R))); %Norma di un vettore. Attenzione all'uso del "."!

%In modo equivalente si può calcolare la norma di un vettore usando la
%funzione "norm" già inclusa in Matlab

>> M = norm(R);

      ------------------------------clear-------------------------------
      ------------------------------clc---------------------------------

%Esempio - Calcolo velocità

%Distanza [Km]         560    440    490    530    370
%Tempo [h]            10.3   8.2    9.1    10.1   7.5

%Inseriamo i valori della tabella in una matrice

>> W = [560 440 490 530 370;10.3 8.2 9.1 10.1 7.5];
```

Vogliamo determinare la massima velocità sostenuta durante il percorso (che sarà abbinata ad uno dei tratti di strada riportati in tabella).

```
%Prima di tutto calcoliamo la velocità per ciascun tratto come il rapporto
Distanza/Tempo: proviamo a farlo utilizzando un unico comando

>> V = W(1,:)./W(2,:);
```

Vogliamo quindi determinare il valore massimo all'interno di questo vettore: per farlo utilizziamo il comando **max**:

```
>> [velocita_massima strada] = max(V); %l'output è la velocità massima
                                       %cercata e la posizione del dato
                                       %(strada)


      ------------------------------clear-------------------------------
      ------------------------------clc---------------------------------
```

```
%Esempio - Differenza fra elevamento a potenza dei singoli elementi di A e
%calcolo di A*A

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

>> B = A.^2; %Elevamento al quadrato di ciascun elemento

>> C = A*A; %Classico prodotto righe per colonne tra la matrice A per se
            %stessa

>>  B == C %Confronto elemento per elemento tra due matrici

ans =

     0     0     0
     0     0     0
     0     0     0
```

RISOLUZIONE DI SISTEMI

Matlab permette di risolvere numericamente un sistema di equazioni lineari algebriche, come per esempio:

$$\begin{cases} 3x_1 - 3x_2 + x_3 = 1 \\ 2x_1 + x_2 - 3x_3 = 0 \\ -x_1 + x_2 + 2x_3 = 2 \end{cases}$$

Risolviamo il sistema con il metodo di eliminazione di **Gauss-Jordan**: si ha un sistema di tre equazioni algebriche in tre incognite, che si può rappresentare in modo matriciale come $\mathbf{A\ x = b}$, dove $\mathbf{A}$ è la matrice dei coefficienti, $\mathbf{b}$ è il vettore dei termini noti e $\mathbf{w} = (x_1\ x_2\ x_3)'$ è il vettore delle incognite.

```
>> A = [3 -3 1; 2 1 -3; -1 1 2]; %Matrice dei coefficienti

>> B = [1 0 2]'; %Matrice dei termini noti

>> T = [A B]; %Unione fra matrici
```

Numericamente la soluzione del sistema è ottenuta mediante un algoritmo, che tende a rendere il sistema triangolare superiore, in modo che sia facile poi risolverlo a partire dall'ultima equazione per sostituzione all'indietro.

1. passo: si toglie da ogni riga successiva alla prima un opportuno multiplo della prima in modo che la prima incognita rimanga solo nella prima equazione:

$$\begin{aligned} &3x_1 - 3x_2 + x_3 = 1 \\ &2x_1 + x_2 - 3x_3 = 0 \quad \text{tolgo da questa equazione 2/3 della prima equazione} \\ &-x_1 + x_2 + 2x_3 = 2 \quad \text{tolgo da questa equazione -1/3 della prima equazione} \end{aligned}$$

```
>> T(1,:) = T(1,:)/3

T =

    1.0000   -1.0000    0.3333    0.3333
    2.0000    1.0000   -3.0000         0
   -1.0000    1.0000    2.0000    2.0000

>> T(2,:)=T(2,:)-2*T(1,:)

T =

    1.0000   -1.0000    0.3333    0.3333
         0    3.0000   -3.6667   -0.6667
   -1.0000    1.0000    2.0000    2.0000
```

```
>> T(3,:) = T(3,:)+T(1,:)

T =

    1.0000   -1.0000    0.3333    0.3333
         0    3.0000   -3.6667   -0.6667
         0         0    2.3333    2.3333
```

2. passo: si toglie da ogni riga successiva alla seconda un opportuno multiplo della seconda in modo che la seconda incognita non compaia più nelle equazioni successive alla seconda: nel nostro caso la matrice di partenza, da quanto calcolato nel primo passo, è già triangolare superiore. Il sistema è diventato:

$$\begin{cases} 3x_1 - 3x_2 + x_3 = 1 \\ 3x_2 - \frac{11}{3}x_3 = -\frac{2}{3} \\ \frac{7}{3}x_3 = \frac{7}{3} \end{cases}$$

Passo finale: sostituzione all'indietro; si risolvono le equazioni dall'ultima alla prima.

```
>> T(1,:)=[A(1,:) B(1,1)] %è necessario dover rimpiazzare la prima riga di T
                          %con quella di partenza, ovvero con quella che non
                          %risente dei coefficienti moltiplicatori

T =

    3.0000   -3.0000    1.0000    1.0000
         0    3.0000   -3.6667   -0.6667
         0         0    2.3333    2.3333
```

Dalla 3° equazione si deduce banalmente che $x_3 = 1$.

```
>> T(3,:) = T(3,:)*(3/7)

T =

    1.0000   -1.0000    0.3333    0.3333
         0    3.0000   -3.6667   -0.6667
         0         0    1.0000    1.0000
```

Risoluzione per la 2° equazione:

```
>> T(2,:)=T(2,:)/3

T =

    3.0000   -3.0000    1.0000    1.0000
         0    1.0000   -1.2222   -0.2222
         0         0    1.0000    1.0000

>> T(2,:)=[0 1 0 T(2,4)-T(2,3)]

T =

    3.0000   -3.0000    1.0000    1.0000
         0    1.0000         0    1.0000
         0         0    1.0000    1.0000
```

Risoluzione per la 1° equazione:

```
>> T(1,:)=1/3*[3 0 0 T(1,4)-T(1,2)-T(1,3)]

T =

    1.0000         0         0    1.0000
         0    1.0000         0    1.0000
         0         0    1.0000    1.0000
```

Le soluzioni trovate sono: $\boldsymbol{x_1 = 1}$, $\boldsymbol{x_2 = 1}$, $\boldsymbol{x_3 = 1}$

Per fortuna Matlab permette di risparmiare tutti questi passaggi e di risolvere il sistema con una semplice operazione tra matrici, utilizzando Il *backslash* "\".

Riprendendo dal passo finale del nostro esempio:

```
>> T

T =

    3.0000   -3.0000    1.0000    1.0000
         0    3.0000   -3.6667   -0.6667
         0         0    2.3333    2.3333
```

```
>> A1 = T(1:3,1:3);
>> B1 = T(:,4);

>> A1\B1; %Risoluzione di sistemi: inv(A)*B
```

Una forma più elegante offerta da Matlab per calcolare la diagonalizzazione di Gauss e risolvere i sistemi lineari è il comando **`rref`**: date le due matrici A e B di partenza

```
>> rref([A B]);
```

```
      ------------------------------clear---------------------------------
      ------------------------------clc---------------------------------


%Esempio – Uso dell'istruzione for

%Si vuole incrementare il valore di un contatore e stampare il suo valore ad
%ogni iterazione del ciclo

>> for i=0:10, %Ciclo for
i
end


      ------------------------------clear---------------------------------
      ------------------------------clc---------------------------------
```

M-FILE

Le istruzioni in Matlab sono memorizzate in un M-file, la cui estensione è .m.
Per creare un M-file si può selezionare File/New/Script (o M-File) o premere sul pulsante NUOVO ☐ della barra delle applicazioni. Si apre la finestra dell'editor, da salvare poi denominandolo con un nome di fantasia e estensione .m.
Per editare un M-file già esistente, si può usare File/Open o premere il pulsante Apri 🗁 della barra delle applicazioni.

L'editor può essere attivato per modificare un M-file esistente anche mediante il comando

```
>> edit <nome-file.m>
```

```
  %--------------------------------------------------------------------------%

%Esempio di funzione - da scrivere in un file a parte "minor.m"
```

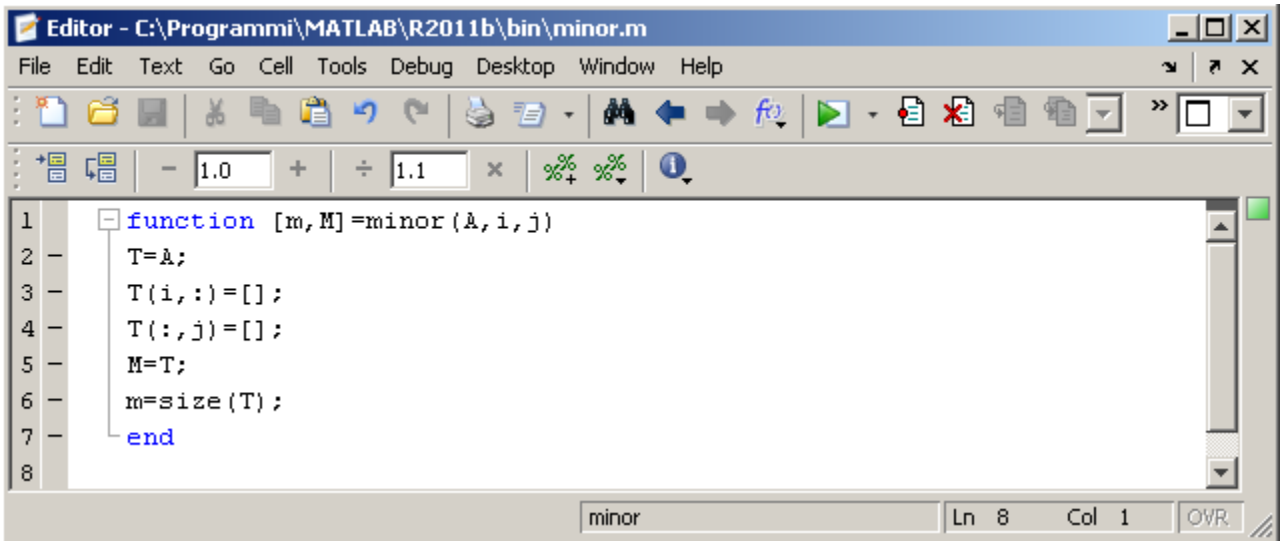
La sintassi di una funzione:

```
function [lista parametri uscita]=nome(lista parametri ingresso)

…

end
```

```
%La funzione minor restituisce una sottomatrice ottenuta eliminando al i-esima
%riga e la j-esima colonna e ne calcola la dimensione.
```



```
%Esempio di esecuzione

>> A = [5 7 4;1 1 3;1 8 6];

>> [dim Matrix] = minor(A,2,3); %Chiamata alla funzione minor
```

OSSERVAZIONE: gli M-file sono di due tipi:

- M-script file
- M-funtion file

M-script file

- Contengono una sequenza di istruzioni Matlab.
- Possono utilizzare variabili che sono già state definite nell'ambiente di lavoro (anche se ciò non è auspicabile). **Le variabili definite in un M-script file restano definite nell'ambiente di lavoro**.
- Una volta definiti (ossia scritti mediante l'editor e salvati) sono **mandati in esecuzione specificando sulla linea di comando il nome del file senza l'estensione .m** oppure selezionando FILE/RUN SCRIPT e specificando il nome del file.
- Si può utilizzare all'interno di un M-file un M-file già definito.

M-function file

- Definiscono una nuova **funzione il cui nome coincide con il nome che si attribuisce al file**.
- Ogni funzione accetta dei dati su cui operare (parametri di ingresso) e restituisce dei risultati (parametri di uscita).
- Il nuovo editor mostra automaticamente la seguente intestazione:

```
function [ output_args ] = Untitled1( input_args )
%UNTITLED1 Summary of this function goes here
%   Detailed explanation goes here


end
```

Se in una lista di parametri compare più di un parametro, essi sono separati da virgola. I parametri specificati nella istruzione function si dicono **parametri formali**.

- Se non ci sono parametri d'uscita si usa:

```
function nome(lista parametri d'ingresso);

function []=nome(lista parametri d'ingresso);
```