Una guida introduttiva all'uso del Matlab [1]

Barbara Del Prete, Felice Iavernaro

25/11/2003

[1] Il presente documento è parte del lavoro della tesi di laurea della studentessa Barbara Del Prete, e ne riporta alcuni capitoli relativi alle nozioni di base sull'uso del Matlab. Esso non è pertanto da intendersi come definitivo ed è distribuito unicamente per uso personale degli studenti del corso di Calcolo Numerico I di Matematica, a.a. 2003/04. Eventuali segnalazioni e commenti saranno graditi (e-mail: felix@dm.uniba.it).

Indice

Introduzione

MATLAB è uno strumento interattivo il cui elemento base è un array che non richiede dimensionamento. Questo consente di risolvere molti problemi tecnici in un intervallo di tempo che bisognerebbe spendere per dichiarare, ad esempio, matrici e vettori in un linguaggio non interattivo, come C o Fortran.

In ambiente universitario MATLAB è lo strumento standard per i corsi di base e avanzati di Matematica, Ingegneria e Scienze. Nell'industria MATLAB viene scelto per l'alta produttività nella ricerca, nello sviluppo e nell'analisi.

MATLAB fu scritto originariamente in Fortran da Cleve Moler, in un processo evolutivo durato diversi anni, con l'intento di fornire un facile accesso ai software basati sull'uso di matrici. Gli algoritmi alla base del calcolo matriciale furono scritti da molte altre persone che collaborarono ai progetti LINPACK e EISPACK.
L'attuale MATLAB è stato scritto in C dalla The MathWorks. La prima versione è di Steve Bangert, che scrisse l'interprete grammaticale, Steve Kleiman, che implementò la grafica, e John Little e Cleve Moler, che scrissero le routines analitiche, la guida dell'utente e la maggior parte degli M-file.
Negli anni MATLAB si è sviluppato grazie agli input di molti utenti, imponendosi in ambito ingegneristico mondiale come strumento per la simulazione di sistemi lineari e non lineari e, più in generale, per l'analisi numerica. Oggi MATLAB comprende strumenti per l'analisi dei dati, l'esplorazione e la visualizzazione, l'elaborazione numerica e simbolica, la grafica scientifica ed ingegneristica, la modellizzazione, la simulazione, la programmazione, lo sviluppo delle applicazioni e la conversione automatica di programmi MATLAB nei codici C e C++. MATLAB comprende strumenti per l'algebra lineare e per le operazioni con matrici, funzioni di Fourier, funzioni statistiche, matematiche e trigonometriche, funzioni per la risoluzione di equazioni differenziali, supporti per le matrici sparse, funzioni interattive per la rappresentazione grafica 2-D, 3-D e 4-D. MATLAB comprende anche famiglie di applicazioni dedicate alla

risoluzione di problemi specifici, chiamate *toolbox*, che consentono di conoscere e di applicare tecnologie specializzate per particolari classi di problemi, come sistemi di controllo, reti neurali, elaborazione dei segnali, simulazioni, ricerche mediche e molti altri.
Con più di 500 funzioni matematiche, statistiche ed ingegneristiche, MATLAB consente un accesso immediato al calcolo numerico. Le routine sono veloci e accurate: questi algoritmi sono stati sviluppati da esperti matematici in maniera tale da ottimizzare le operazioni con le matrici. Probabilmente la caratteristica più importante di MATLAB è proprio la sua semplice estensibilità: questo consente di contribuire come autori di MATLAB, creando le proprie applicazioni. Negli anni in cui MATLAB è stato disponibile, molti scienziati, matematici e ingegneri hanno sviluppato nuove ed interessanti applicazioni, tutte senza la necessità di scrivere neppure una linea di Fortran o di altri codici a basso livello.

Il presente lavoro cerca di fornire uno strumento interattivo per l'apprendimento dell'uso di MATLAB: vengono illustrati i comandi e le funzioni principali del linguaggio di programmazione, con l'ausilio di numerosi esempi che permettono un'immediata interazione con quanto viene affermato teoricamente. Gli esercizi proposti, inoltre, offrono uno strumento di autoverifica e di incentivo dell'uso personalizzato del programma da parte del lettore.

Nel Capitolo 1 sono state fornite alcune nozioni basilari per consultare questa trattazione (come leggere le note a piè pagina e come interpretare i colori con cui è scritto il testo) e per usare MATLAB (come avviare il programma, come usare l'*help*, come ottimizzare i tempi di immissione dei dati, come utilizzare alcuni comandi del sistema operativo, come lavorare con l'editor di testo, come chiudere il programma).

Nel Capitolo 2 sono stati illustrati i metodi per lavorare con i numeri complessi (dei quali si sono trattati tutti i tipi di formato), con le variabili, le variabili predefinite, le variabili Inf e NaN della matematica estesa, le funzioni e gli operatori elementari, esponenziali e logaritmici di MATLAB.

Nel Capitolo 3 sono stati trattati gli aspetti basilari dell'argomento delle matrici, elemento caratterizzante MATLAB: sono stati esposti i possibili metodi di inserimento di una matrice (da tastiera, da file esterni e da M-file), i comandi per operare sulle variabili presenti nell'area di lavoro (per memorizzarle, cancellarle, caricarle o per leggerne le proprietà), le operazioni su matrici (elementari, elemento per elemento, di estrazione di elementi, di estrazione di sottomatrici, di concatenazione), i comandi per operare su una singola matrice, le funzioni esponenziali su matrici, l'elenco di alcune matrici particolari predefinite in MATLAB, l'uso dell'operatore : e la costruzione di intervalli MATLAB.

Segue una serie di Appendici: l'Appendice A contiene la soluzione di tutti gli esercizi proposti nei vari Capitoli; l'Appendice B elenca tutte le variabili utilizzate negli esempi e negli esercizi proposti durante la trattazione; l'Appendice C rappresenta l'indice alfabetico di tutti i comandi MATLAB presenti nella trattazione, con link che rimandano alla parte di testo in cui sono stati esposti.

Capitolo 1

Nozioni preliminari

Il presente documento è scritto in formato PDF (Portable Document Format) ed interfacciato con MATLAB (versione 5.3).
Il programma di lettura cui si fa riferimento è Acrobat Reader (versione 4.0).
Nel seguito sono elencate alcune note e suggerimenti per un uso ottimale delle funzioni di lettura.

1.1 Note a piè pagina

Nel testo si troveranno delle note a piè pagina, che si potranno leggere cliccando una volta con il pulsante sinistro del mouse sul numerino che le segnala. Letta la nota, si può ritornare al punto in cui ci si trovava tramite la freccia rivolta verso l'alto della tastiera, tramite lo slider [1] a destra della finestra, oppure tramite la freccia rivolta verso sinistra della barra degli strumenti [2].

Esempio 1 *Vediamo come si legge una nota a piè pagina: accanto alla parola slider nelle righe precedenti c'è un numero* 1*. Avvicinandoci al numerino con il puntatore del mouse, che ha la forma di una mano, vediamo che il puntatore cambia forma, diventando un indice rivolto verso l'alto. Questo segnala che cliccando su quella parte del documento, verremo rimandati ad un'altra parte di testo.*
Cliccando una volta con il pulsante sinistro del mouse sul numero 1 *veniamo*

[1] Lo slider è la parte più a destra della finestra, dove ci sono le freccette per muoversi verso l'alto o verso il basso e la barra di scorrimento veloce. Sia le freccette che la barra si usano andando su esse con il puntatore del mouse, premendo il pulsante sinistro del mouse e tenendolo premuto finché non viene raggiunta la linea del testo desiderata.

[2] La barra degli strumenti è la riga in alto alla finestra sulla quale ci sono i disegni delle varie attività che si possono fare tramite il programma.

rimandati alla fine della pagina, dove possiamo leggere la spiegazione della parola slider.
Letta la nota, torniamo al punto in cui ci trovavamo tramite la freccia rivolta verso l'alto della tastiera, tramite lo slider oppure, più velocemente, tramite la freccia rivolta verso sinistra della barra degli strumenti.

1.2 Colori del testo

Nel testo (scritto in nero) si troveranno alcune parole di colore diverso. Cliccando una volta con il pulsante sinistro del mouse sui termini di colore diverso da quello in cui è scritto il testo, il programma rimanda ad altre parti della trattazione cui si riferisce il termine su cui si è cliccato.
Nel presente lavoro si sono adottate alcune convenzioni:

- il nome dei comandi MATLAB è stato scritto in blu;
- il nome delle variabili utilizzate è stato scritto in celeste;
- le altre parti del testo cui la trattazione fa riferimento sono state segnalate in verde;
- la parola Soluzione, tramite la quale si può leggere la soluzione degli esercizi proposti, è stata scritta in rosso;
- i link mediante i quali si può raggiungere una figura da visualizzare sono di color magenta.

Per ritornare al punto in cui si è interrotta la lettura basta cliccare una volta sulla freccia rivolta verso sinistra della barra degli strumenti.

Esempio 2 *In alcune parti del testo si troverà scritto: consideriamo la matrice A. Cliccando sulla lettera A (scritta in celeste) con il mouse, si viene rimandati alla sezione di testo in cui sono elencate tutte le variabili, tra cui la matrice A.*
Una volta presa visione della matrice, si può tornare alla sezione che si stava leggendo, cliccando sulla freccia rivolta verso sinistra della barra degli strumenti.

1.3 Come avviare MATLAB

Supponiamo di trovarci su sistema operativo [3] UNIX o LINUX. Per lanciare MATLAB si scrive il comando **matlab**.
Se invece ci troviamo sotto WINDOWS, bisogna fare clic con il pulsante sinistro del mouse su **Avvio**, scegliere **Programmi**, quindi fare clic sulla cartella **Matlab** e infine su **MATLAB 5.3**. Se sul desktop [4] compare un'icona di MATLAB si può avviare il programma più velocemente, cliccando due volte consecutive sull'icona con il pulsante sinistro del mouse.
Queste operazioni avviano MATLAB: esce una videata, alla cui sinistra appare il simbolo $\gg$ che si chiama prompt di MATLAB [5].
A questo punto siamo pronti per iniziare a lavorare.

1.4 Come usare l'help

Esistono diversi modi per accedere ad informazioni online circa le funzioni di MATLAB: in questo paragrafo li analizziamo tutti.

1.4.1 Il comando *help*

MATLAB dispone di un manuale in linea in lingua inglese, che può essere richiamato mediante i comandi di help.
Tutte le funzioni MATLAB sono organizzate in gruppi logici e la struttura delle directory di MATLAB è basata su questa suddivisione. Per esempio, tutte le funzioni dell'algebra lineare sono comprese nella directory *matfun*. Per avere una lista e una breve descrizione di tutte le funzioni comprese in questa directory si scrive $\gg help\ matfun$ e si preme il tasto INVIO [6].
Il comando $\gg help$ senza altri parametri restituisce l'elenco di tutti gli argomenti di base (cioè di tutte le directory di cui sopra) con una descrizione delle categorie cui le funzioni appartengono.

---

[3] Il sistema operativo è un insieme di programmi e procedure che gestiscono le risorse hardware e software del computer e consentono all'utente di interagire con quest'ultimo.

[4] Il desktop è la videata iniziale, che compare dopo aver avviato WINDOWS.

[5] Il prompt di un programma è il simbolo che esce sulla sinistra della finestra, alla riga in cui il programma si aspetta l'istruzione successiva.
Nel seguito di questa trattazione il simbolo $\gg$ denoterà il prompt di MATLAB.

[6] In tutta la trattazione si ometterà di precisare che bisogna premere il tasto INVIO alla fine di un'istruzione, perché questa venga eseguita da MATLAB.

Eseguendo $\gg$ *help NomeArgomento* compare un elenco di funzioni relative all'argomento cui siamo interessati. NomeArgomento può essere il nome di una funzione (in tal caso il comando *help* fornisce informazioni sulla funzione) oppure il nome di una directory (caso in cui il comando *help* mostra il contenuto della directory specificata).

Esempio 3 *Un esempio di funzione MATLAB è la radice quadrata, che si ottiene con la funzione sqrt. Se vogliamo sapere come si usa questa funzione, lanciamo $\gg$ help sqrt. Il comando restituisce:*

SQRT   Square root.

SQRT(X) is the square root of the elements of X. Complex results are produced if X is not positive.

See also SQRTM.

Overloaded methods

help sym/sqrt.m [7]

Osservazione 1 Come si può vedere dall'esempio, MATLAB, dopo aver fornito la spiegazione del comando, rimanda anche ad altri argomenti correlati ad esso. Nel caso dell'esempio viene suggerito di leggere anche l'help relativo al comando *SQRTM*, che serve per calcolare la radice quadrata di matrici.

Osservazione 2 Dall'esempio si può anche osservare che MATLAB usa le lettere maiuscole per scrivere il nome della funzione cui siamo interessati. Questo è fatto solo per mettere in risalto il termine che denota la funzione rispetto al resto del testo in cui viene descritto come usarla.
Comunque, quando si scrive un comando bisogna usare le lettere minuscole in quanto MATLAB fa distinzione tra lettere maiuscole e lettere minuscole negli input: se un'istruzione viene scritta con le lettere maiuscole MATLAB segnala l'errore.

[7] Traduzione:

SQRT   Radice quadrata.

SQRT è la radice quadrata degli elementi di X. Se X è negativo vengono forniti risultati complessi.

Si consulti anche *SQRTM*.

Gli ultimi due righi indicano che è possibile calcolare, mediante la funzione *sqrt*, anche la radice simbolica, se l'argomento è simbolico, cioè se la sua classe è sym.

Esempio 4 *Supponiamo di lanciare l'istruzione* $\gg SQRT(2)$*. MATLAB segnala l'errore con il messaggio:*
??? Undefined variable or capitalized internal function SQRT; Caps Lock may be on. [8]

Questa proprietà si esprime dicendo che MATLAB è **case sensitive**.

1.4.2 La finestra dell'help

La finestra dell'help è disponibile su PC e Mac selezionando dal menù [9] **Help** l'opzione **Help Windows**, oppure cliccando sul punto interrogativo della barra degli strumenti. La stessa opzione è disponibile su tutti i tipi di computer, digitando il comando $\gg$ *helpwin*.
Per leggere uno degli argomenti compresi nella finestra dell'help, basta digitare $\gg$ *helpwin NomeArgomento*, oppure cliccare due volte con il pulsante sinistro del mouse sulla riga che contiene l'argomento desiderato.
Per chiudere la finestra si clicca con il pulsante sinistro del mouse sulla crocetta più in alto a destra della finestra oppure sul pulsante Close.

1.4.3 Il comando $lookfor$

Il comando $lookfor$ seguito da una parola chiave consente di cercare tutte le funzioni il cui nome inizia con quella parola: $lookfor$ effettua la ricerca attraverso la prima linea dell'help (detta linea $H1$) relativo ai vari comandi e restituisce le prime linee relative ai comandi il cui nome contiene la parola chiave assegnata.

Esempio 5 *Digitando* $\gg lookfor\ sqrt$ *appare scritto:*
SQRT Square root.
SQRTM Matrix square root.
SQRT Symbolic matrix element-wise square root. [10]

[8] Traduzione:
??? *SQRT* è una variabile non definita oppure una funzione interna scritta con lettere maiuscole; il tasto delle lettere maiuscole potrebbe essere acceso.

[9] Guardando in alto alla finestra si trova una prima riga sulla quale è scritto il nome del programma che si sta utilizzando, che nel nostro caso è MATLAB. Nella riga immediatamente sotto a questa ci sono varie parole (File, Edit, ...) che indicano dei menù. Cliccando con il pulsante sinistro del mouse su un menù appare una tendina (cioè una piccola finestra) contenente varie righe, che rappresentano le opzioni che si possono scegliere (lo si fa andando sull'opzione desiderata con il puntatore del mouse e cliccando con il pulsante sinistro).

[10] Traduzione:
SQRT Radice quadrata.

Se la parola chiave immessa non è compresa tra i comandi MATLAB, viene lanciato un messaggio che segnala che non è stato trovato nessun argomento con il nome assegnato.

Esempio 6 *MATLAB non contiene nessuna funzione chiamata $inversa$, quindi se scriviamo*
$\gg lookfor\ \ inversa$
MATLAB restituisce:
inversa not found. [11]

1.4.4 L'Help Desk

MATLAB consente di avere sul proprio sistema locale l'accesso ad un grande numero di informazioni scritte in HTML (HyperText Markup Language), alle quali si può accedere con un browser di Internet come Netscape o Microsoft Explorer.
L'Help Desk può essere attivato su PC e Mac selezionando dal menù **Help** l'opzione **Help Desk (HTML)**, oppure, su tutti i tipi di computer, digitando il comando $\gg$ *helpdesk*.
Tutti gli operatori e le funzioni MATLAB hanno pagine di riferimento online nel formato HTML, che possono essere lette attraverso l'Help Desk. Queste pagine contengono più dettagli ed esempi rispetto agli altri tipi di help finora trattati.
Pagine di riferimento online di MATLAB sono disponibili anche nel formato PDF attraverso l'Help Desk. Esse sono accessibili mediante Adobe Acrobat Reader. Queste pagine riproducono lo stile di una pagina stampata, completa di caratteri, grafici e immagini e rappresentano la via migliore per ottenere copie stampate di materiale di riferimento.

1.4.5 Link alla MathWorks

Se si sta lavorando su un computer collegato ad Internet, l'Help Desk realizza una connessione alla MathWorks, la casa produttrice di MATLAB. Si può utilizzare la posta elettronica per spedire domande, inviare suggerimenti e

---

$SQRTM$ Radice quadrata di matrici.

$SQRT$ Radice quadrata simbolica elemento per elemento.

[11] Traduzione:
inversa non trovato.

segnalare possibili virus. Si può anche usare il motore di ricerca del sito della MathWorks per consultare data base di supporto tecnico.

1.5 Esempi e demo

Per vedere esempi e demo di MATLAB su PC e Mac bisogna selezionare dal menù **Help** l'opzione **Examples and Demos**, oppure, su un generico computer, bisogna lanciare l'istruzione $\gg$ *demo*.
In questo modo viene visualizzata una finestra, all'interno della quale si trovano tre riquadri.
Nel riquadro più a sinistra appare scritto: $+MATLAB$ oppure $-MATLAB$. Nel primo caso, cliccando due volte con il pulsante sinistro del mouse su questa parola, viene visualizzato un elenco di argomenti generali sui quali è possibile vedere esempi (matrici, numeri, visualizzazione...). Nel secondo caso l'elenco è già visualizzato.
È possibile leggere informazioni circa questi argomenti cliccando una volta con il pulsante sinistro del mouse sul nome dell'argomento cui si è interessati. Nel secondo riquadro in alto appare una spiegazione dell'argomento selezionato. Si può leggere questa spiegazione, facendo scorrere il testo con lo slider a destra del riquadro.
Nel riquadro in basso appaiono gli esempi che è possibile visualizzare inerenti all'argomento selezionato. Cliccando una volta con il pulsante sinistro del mouse sull'esempio prescelto e poi sul pulsante che appare in basso a destra (su cui c'è scritto Run... e il nome dell'esempio prescelto), viene visualizzata la finestra contenente il demo, nella quale è possibile selezionare le varie opzioni.
Cliccando una volta con il pulsante sinistro del mouse sul tasto Close, si esce dalla finestra di demo. La stessa procedura si usa per uscire dal demo.

1.6 L'uso delle frecce della tastiera

Le frecce della tastiera consentono di richiamare e riutilizzare comandi scritti in precedenza, permettendo di ridurre notevolmente i tempi di immissione dei dati.
Utilizzando ripetutamente il tasto $\uparrow$ vengono visualizzate le linee di comando precedentemente scritte.
Utilizzando il comando $\uparrow$ dopo una o più lettere, vengono visualizzate solo le linee di comando precedenti che iniziano con quella lettera o con quel gruppo

di lettere.
Per tornare ad un'istruzione sorpassata basta premere il tasto $\downarrow$.
Con i tasti $\leftarrow$ e $\rightarrow$ ci si sposta verso sinistra oppure verso destra sulla riga di comando su cui ci si trova.

Esempio 7 *Supponiamo di voler calcolare la radice quadrata del numero* 5. *Abbiamo visto che l'operatore che ci consente di calcolare la radice quadrata è sqrt. Supponiamo di lanciare l'istruzione* $\gg sqt(5)$*, cioè di sbagliare a scrivere il comando sqrt.*
MATLAB segnala l'errore con il seguente messaggio:
??? Undefined function or variable 'sqt'. [12]
Invece di riscrivere tutta l'istruzione, si può semplicemente premere il tasto $\uparrow$ *della tastiera. Il comando scritto precedentemente riappare affianco al prompt. Usando il tasto* $\leftarrow$ *si sposta il cursore (che appare affianco all'ultima lettera dell'istruzione) fino al punto in cui si deve inserire la r mancante. Premendo INVIO dopo aver inserito la lettera mancante, questa volta non sarà visualizzato alcun messaggio di errore e il programma calcolerà il valore richiesto.*

Esempio 8 *Supponiamo di aver inserito le seguenti istruzioni:*
$\gg a = 3$
$\gg b = 1$
$\gg c = -3$
e di voler cambiare il valore di a *da* 3 *a* -3.
Premendo il tasto $\uparrow$ *ricompare l'istruzione* $\gg c = -3$*, premendolo nuovamente appare l'istruzione* $\gg b = 1$ *e quindi l'istruzione* $\gg a = 3$*. Spostandoci verso sinistra con il tasto* $\leftarrow$*, aggiungiamo il segno* $-$ *prima del* 3*. Premendo INVIO viene visualizzato:*
$a =$
$\quad -3$
Alternativamente, si può scrivere $\gg a$ *e premere* $\uparrow$*. In questo modo ricompare la riga* $\gg a = 3$*, che può essere modificata.*

1.7 Comandi DOS/UNIX/LINUX

MATLAB rende disponibili alcuni comandi importati dal DOS, da UNIX e da LINUX:

[12] Traduzione:
??? Funzione o variabile '*sqt*' non definita.

$\gg$ **cd**: permette di esaminare o di cambiare la directory corrente; la sintassi possibile è:

- *cd* (o equivalentemente **pwd**): visualizza la directory nella quale ci si trova.

  Esempio 9 *Supponiamo di aver appena aperto MATLAB su PC; lanciando $\gg$ cd viene visualizzato:*
  $C:$[13] $\backslash MATLABR11\backslash work$
  Questo significa che ci troviamo nella sottodirectory work della directory $MATLABR11$ contenuta nell'unità di memoria C.
  Analogamente, lanciando $\gg$ pwd si ottiene:
  ans [14]=
  $C:\backslash MATLABR11\backslash work$

- $cd_{\sqcup}$ [15] $..$: sposta nella directory superiore a quella corrente.

  Esempio 10 *Supponiamo di aver appena aperto MATLAB su PC e di voler uscire dalla directory work; lanciando $\gg$ $cd_{\sqcup}..$ viene visualizzato il prompt. Adesso ci troviamo in $C:\backslash MATLABR11$, cioè nella directory $MATLABR11$ contenuta nell'unità di memoria C (per verificarlo è sufficiente lanciare $\gg$ cd).*

- *cd NomeDir*: fa passare dalla directory corrente alla directory NomeDir (che deve essere contenuta nella directory corrente, altrimenti viene visualizzato un messaggio di errore).

[13] Un programma e i dati sui quali esso opera devono essere presenti nella memoria centrale del computer per poter essere eseguiti. Generalmente la memoria di un computer non è tuttavia così vasta da contenere certi tipi di dati (come matrici molto grandi o elenchi molto lunghi) e un ampliamento della memoria centrale è in genere un'operazione molto costosa. Si sono studiate allora alternative più economiche per memorizzare grandi volumi di dati: le unità periferiche di memorizzazione secondaria (disco rigido, floppy disk, CD-ROM...). Queste vengono denotate con le lettere A, B, C etc. Usualmente con C si denota il disco rigido.
In tutta la trattazione ipotizziamo di aver installato il programma MATLAB nella locazione di memoria $C:\backslash MATLABR11$, ma potrebbe anche non essere così: durante la fase di installazione è infatti possibile specificare un path differente. Tralasciamo questa eventualità.

[14] Per la spiegazione del significato della variabile *ans* si legga il paragrafo La variabile *ans*.

[15] In tutta la trattazione quando si trova questo simbolo significa che bisogna lasciare uno spazio (premendo una volta la barra sulla tastiera) tra ciò che è scritto a sinistra e ciò che e scritto a destra del simbolo.

Esempio 11 *Supponiamo di aver appena aperto MATLAB su PC e di volerci spostare nella sottodirectory help di* $MATLABR11$*: dopo aver lanciato* $\gg$ *cd␣.. per uscire dalla directory work, si lancia* $\gg$ *cd help. Premendo INVIO, ricompare il prompt.*
Ora ci troviamo in $C:\backslash MATLABR11\backslash help$.

Esempio 12 *Supponiamo di trovarci in* $C:\backslash MATLABR11\backslash help$ *(directory nella quale non esiste nessuna sottodirectory chiamata help) e di lanciare l'istruzione* $\gg$ *cd help. Viene visualizzato il seguente messaggio d'errore:*
??? Name is nonexistent or not a directory [16]

$\gg$ **dir**: permette di esaminare il contenuto della directory corrente; la sintassi possibile è:

- *dir* (o equivalentemente **ls**): elenca i file della directory corrente.

  Esempio 13 *Supponiamo di esserci spostati dalla sottodirectory work alla directory* $C:\backslash MATLABR11\backslash help$*. Lanciando* $\gg$ *dir appare la lista dei file presenti nella directory help:*

. [17]	*helpdesk.html*	*nofunc.html*	*support*
.. [18]	*join_d.gif*	*nohelp.html*	*techdoc*
about_d.gif	*join_up.gif*	*pdf_doc*	*toolbox*
about_up.gif	*jse_base*	*pdficonsmall.gif*	*tscope.gif*
benefits.html	*mapfiles*	*search1.html*	*www.gif*
docroadmap.html	*mathlib.html*	*search2.html*	
full_doc_set.gif	*mla_about.jpg*	*search3.html*	
fulldocset.html	*mla_bullet.gif*	*search4.html*	
help_desk.gif	*mla_join.jpg*	*subscribe.html*	

- *dir NomeDir*: elenca tutti i file della directory NomeDir (che deve essere contenuta nella directory corrente).

  Esempio 14 *Supponiamo di trovarci in* $C:\backslash MATLABR11$.
  Lanciando $\gg$ *dir help appare la lista dei file presenti nella directory help (vista nell'esempio precedente).*

[16] Traduzione:
??? Il nome è inesistente oppure non è una directory
[17] Il simbolo . denota la directory corrente.
[18] Il simbolo .. denota la directory madre.

$\gg$ **what**: è analogo a *dir*, ma visualizza solo i file con estensione [19] .m, .mat e i file MEX [20].

Esempio 15 *Supponiamo di trovarci in* $C:\backslash MATLABR11\backslash bin$ *e di lanciare* $\gg$ *what. Appare:*
MEX-files in the current directory $C:\backslash MATLABR11\backslash bin$ [21]

*clbs*110	*gui*	*libmi*	*msctof*	*perl*300
comp_ja	*gx*5050*r*	*libmx*	*msvcirt*	*rnimatlab*
compiler	*hardcopy*	*libut*	*msvcrt*	*simulink*
feng	*hg*	*lmgr*325*c*	*mt*7*s*110	*uiw*
fmat	*libeng*	*mapleoem*	*mwoles*05	*w*32*ssi*
fmex	*libmat*	*mfc*42	*nativejava*	
fmx	*libmatlbmx*	*mipcole*	*numerics*	
glren	*libmccmx*	*mpath*	*ot*5050*r*	

che sono i file MEX presenti nella directory (nel caso esaminato non erano presenti file con estensione .m o .mat nella directory, altrimenti sarebbero stati elencati anche questi).

$\gg$ **which** $NomeFunzione$: visualizza la directory in cui è localizzata la funzione $NomeFunzione$.

Esempio 16 *Supponiamo di trovarci in* $C:\backslash MATLABR11\backslash bin$ *e di lanciare il comando* $\gg$ *which libut. Otteniamo:*
$C:\backslash MATLABR11\backslash bin\backslash libut.dll$*, che ci dice che il file libut.dll si trova nella sottodirectory bin della directory* $MATLABR11$ *che si trova nell'unità di memoria C.*

$\gg$ **type** $NomeFile$: stampa su video il file (ASCII) NomeFile.
Se NomeFile viene fatto seguire dall'estensione del file, MATLAB visualizza il file con quel nome e con quella estensione; se invece il file non viene fatto seguire da nessuna estensione, MATLAB usa l'estensione .m per default [22];

[19] L'estensione di un file è un insieme di lettere preceduto da un punto che segue il nome del file e ne identifica il tipo. L'estensione provvede ad informare il sistema operativo su quale applicazione usare per aprire il file.

[20] Un file MEX è una procedura scritta in C oppure in Fortran e opportunamente compilata in modo da presentarsi come una funzione MATLAB.

[21] Traduzione:
File MEX presenti nella directory corrente $C:\backslash MATLABR11\backslash bin$

[22] La parola default traduce l'espressione in maniera automatica, automaticamente.

$\gg$ **more on**: serve quando la parte di testo da visualizzare sullo schermo non è contenuta in un'unica videata: questo comando impedisce lo scorrimento delle righe oltre la finestra. Se c'è una parte di testo che ancora non è stata letta, sul fondo della finestra appare scritto:
$- -$ *more* $- -$
Per proseguire nella lettura, basta premere un tasto qualsiasi della tastiera: viene visualizzato il rigo successivo. Se invece si preme la barra sulla tastiera, viene visualizzata la schermata successiva.
Se il testo è già stato visualizzato tutto, sul fondo della finestra ricompare il prompt.
Per default *more* è disattivato;

$\gg$ **more**(n): serve per visualizzare solo n righe per volta sullo schermo;

$\gg$ **more off**: serve per disattivare il comando *more on*.

Esercizio 1 *Si apra MATLAB e si legga la directory in cui ci si trova.*
Ci si sposti in $C:\backslash MATLABR11\backslash extern$ *e si leggano le sottodirectory presenti in questa directory.*
Ci si sposti nella directory examples e si leggano i file presenti nella sottodirectory mex. Si controlli se in questa directory ci sono file con estensione .m, .mat oppure file MEX.
Si visualizzi sullo schermo il file yprime.m, leggendone 15 *righe per volta.*
Dopo aver letto il file, si disattivi l'opzione more.
Si legga dove si trova il file yprimef.f.
Soluzione.

1.8 Altri comandi del sistema operativo

È possibile utilizzare comandi del sistema operativo, senza dover necessariamente uscire da MATLAB: il carattere **!** indica che le linee di input successive sono assegnate come comandi del sistema operativo.

Esempio 17 *L'istruzione* $\gg$! *del* [23] *NomeFile.m cancella il file NomeFile che ha estensione .m dall'area di lavoro di MATLAB (detta workspace), cioè dalla directory* $C:\backslash MATLABR11\backslash work$.

[23] Il comando DOS **del** *NomeFile* serve per cancellare il file NomeFile. Se ci si trova su UNIX o LINUX, per cancellare il file NomeFile si usa **rm** *NomeFile*.

Se NomeFile.m non è presente nella directory in cui ci troviamo, viene visualizzato il messaggio d'errore:
File non trovato

1.9 Come scrivere un nuovo file...

1.9.1 ...sotto WINDOWS

Per accedere da MATLAB ad un editor di testo su PC per scrivere un nuovo file ci sono le seguenti possibilità:

- si sceglie **New** e poi **M-file** dal menù **File**;
- si lancia l'istruzione $\gg$ *edit* dal prompt di MATLAB;
- si clicca una volta con il tasto sinistro del mouse sull'apposita icona della barra degli

Appare una nuova videata bianca, nella quale è possibile scrivere il testo del file.

Terminata la digitazione, si salva il file seguendo la procedura richiesta dal sistema operativo su cui si trova l'editor di testo che si sta utilizzando.

Osserviamo che sulla prima riga della finestra compare il nome dell'editor che si sta utilizzando e, affianco a questo, il path [24] del file. Modificando il file, affianco al suo nome compare un asterisco: questo ci ricorda che il file è stato modificato e, quindi, deve essere salvato nuovamente. Ad esempio, modificando il file già salvato ciao.m, il path nella prima riga appare così: $C:\backslash MATLABR11\backslash bin\backslash ciao.m^*$. Salvando il file, non compare più l'asterisco accanto al suo nome nel path.

Per chiudere l'editor, si può scegliere **Exit Editor/Debugger** dal menù **File** oppure si può cliccare una volta con il pulsante sinistro del mouse sulla crocetta più in alto a desta della finestra.

[24] Il path è il percorso che individua il punto della locazione di memoria in cui il file risiede. Ad esempio $C:\backslash MATLABR11\backslash work\backslash ciao.m$ è il path del file ciao.m, memorizzato nella sottocartella work della cartella $MATLABR11$ presente nell'unità di memoria C.

1.9.2 ...sotto UNIX o LINUX

Per accedere ad un editor di testo sotto UNIX o LINUX si usa il simbolo ! seguito dal generico comando che si userebbe se si stesse usando normalmente il prompt del sistema operativo.
Al momento non ci sono in UNIX o in LINUX editor predefiniti, come nel caso di WINDOWS: occorre aprire un editor installato sulla macchina, ad esempio:

$\gg$!*emacs*

$\gg$!*vi*

$\gg$!*gedit*

1.10 Come aprire un file salvato...

1.10.1 ...sotto WINDOWS

Per aprire con l'editor di MATLAB un file già salvato ci sono le seguenti possibilità:

- dopo aver caricato l'editor con una delle procedure analizzata in precedenza, si clicca una volta con il pulsante sinistro del mouse sull'icona che serve per aprire un file che è già stato salvato con un nome, riportata in figura Appare la finestra Open, nella quale è possibile scegliere il file da aprire, oppure la sottocartella nella quale il file è stato salvato;

- dal prompt di MATLAB si lancia l'istruzione $\gg$ *edit* seguita dal path del file che si intende aprire. Se il file è stato salvato nella sottocartella work di MATLAB è sufficiente lanciare $\gg$ *edit* $NomeFile.Estensione$;

- si sceglie **Open** dal menù **File** in MATLAB: appare la finestra Open. Si apre il file seguendo il suo path;

- si preme l'apposita icona sulla barra degli strumenti di MATLAB: appare la finestra Open. Si apre il file seguendo il suo path.

1.10.2 ...sotto UNIX o LINUX

Dall'editor che è caricato sulla macchina che si sta usando, si lancia il comando che permette di aprire un file; ad esempio:

$\gg$!$emacs\ NomeFile$

$\gg$!$vi\ NomeFile$

$\gg$!$gedit\ NomeFile$

1.11 Come chiudere MATLAB...

1.11.1 ...sotto WINDOWS

Per chiudere MATLAB su PC e Mac si può:

- lanciare il comando $\gg$ *quit*;
- lanciare il comando $\gg$ *exit*;
- fare clic con il pulsante sinistro del mouse sulla crocetta più in alto a destra della finestra;
- selezionare con il pulsante sinistro del mouse **Exit MATLAB** dal menù **File**;
- premere contemporaneamente il tasto Ctrl e il tasto Q della tastiera.

1.11.2 ...sotto UNIX o LINUX

Per chiudere MATLAB su UNIX o LINUX si può:

- lanciare il comando $\gg$ *quit*;
- lanciare il comando $\gg$ *exit*;
- fare clic con il pulsante sinistro del mouse sulla crocetta più in alto a destra della finestra.

Capitolo 2

Espressioni

2.1 Introduzione

Come la maggior parte degli altri linguaggi di programmazione, MATLAB è dotato di espressioni matematiche, ma a differenza degli altri linguaggi di programmazione queste espressioni coinvolgono unicamente matrici. Gli elementi principali per la costruzione di queste espressioni sono:

- i numeri;
- le variabili;
- gli operatori elementari;
- le funzioni.

2.2 Numeri

MATLAB usa la notazione decimale convenzionale, con un punto per i numeri decimali e facendo precedere i numeri dal segno $+$ (che può essere omesso) oppure dal segno $-$.
Un numero si definisce in modo ovvio premendo il tasto che lo indica sulla tastiera e quindi premendo INVIO. Ad esempio, scrivendo $\gg 2$ e premendo INVIO, si ottiene l'assegnazione desiderata, che verrà visualizzata sullo schermo in questo modo:
ans [1]=

[1]Per la spiegazione del significato della variabile *ans* si legga il paragrafo La variabile *ans*.

2

Esercizio 2 *Si inseriscano i seguenti numeri:* $+3 \quad +3.52 \quad -4.02$.
Soluzione.

Per specificare una potenza di 10 viene utilizzata la lettera e.

Esempio 18 *Il numero* -3×10^{18} *si può ottenere lanciando*
$\gg -3e18$
oppure esplicitamente
$\gg -3 * 10\hat{}18$
e premendo INVIO; MATLAB lo visualizza in questo modo:
$ans =$
$-3.0000e + 018$

Esempio 19 *Il numero* 4×10^{-5} *si può ottenere lanciando*
$\gg 4e - 5$
oppure esplicitamente
$\gg 4 * 10\hat{} - 5$
e premendo INVIO; MATLAB lo visualizza in questo modo:
$ans =$
$4.0000e - 005$

Esercizio 3 *Si inseriscano i numeri* 2000000 *e* 0.03 *come potenze di* 10.
Soluzione.

L'unità immaginaria viene denotata con la lettera i o, equivalentemente, con la lettera j. Essa è ottenuta come radice quadrata di -1.
I simboli i e j vengono utilizzati anche per inserire tutti gli altri numeri complessi. In caso di moltiplicazione di un numero per l'unità immaginaria si può omettere il segno $*$. Questo vuol dire che il numero complesso che ha parte reale uguale a 2 e parte immaginaria uguale a 3 si può rappresentare nei seguenti modi:

$$\begin{array}{ll} 2+3*i & 2+3*j \\ 2+3i & 2+3j \\ 2+i*3 & 2+j*3 \end{array}$$

Qualsiasi sia la scelta fatta, MATLAB visualizza il numero in questo modo:
$ans =$
$2.0000 + \ 3.0000i$

Osservazione 3 Quanto detto vale soltanto per le costanti numeriche: se considerassimo $\gg a = 3$ e scrivessimo $\gg z = 2 + ai$, MATLAB lancerebbe il messaggio d'errore:
??? Undefined function or variable 'ai'. [2].
Questo perché MATLAB non leggerebbe ai come il prodotto della variabile a per l'unità immaginaria, ma come una nuova variabile, che non è stata definita.

Esercizio 4 *Si inserisca il numero complesso con parte reale uguale a* 4 *e parte immaginaria uguale a* -6.
Soluzione.

Tutti i numeri vengono memorizzati usando il formato lungo secondo lo standard IEEE. I numeri in formato virgola mobile hanno una precisione finita di 16 cifre decimali significative e un range da 10^{-308} a 10^{+308} .

2.3 Formato

Operando con MATLAB abbiamo visto che i risultati vengono rappresentati sullo schermo in maniere diverse: ad esempio abbiamo visto (in un esercizio trattato precedentemente) che, scrivendo $\gg -4.02$, MATLAB visualizza il numero in questa maniera :
$ans =$
    -4.0200
Normalmente di un numero vengono visualizzate solo le prime 5 cifre significative. I comandi di formato (che si usano scrivendo $\gg format$ seguito dall'opzione scelta) controllano il formato dei valori visualizzati da MATLAB sullo schermo e permettono di alterarlo. Essi riguardano soltanto il modo in cui i numeri vengono visualizzati sullo schermo e non influenzano la maniera in cui MATLAB calcola o memorizza i valori numerici: in ogni caso, indipendentemente dal formato con cui i dati sono rappresentati sullo schermo, MATLAB opera in doppia precisione.
Analizziamo i vari tipi di formato disponibili:

- **format short**: 5 cifre, notazione a virgola fissa.

  Esempio 20 $\gg format\ short$ *(o, equivalentemente,* $\gg format$*, oppure omettendo proprio l'istruzione se non si è mai modificato il* $format$

[2]Traduzione:
??? Funzione o variabile 'ai' non definita.

dall'avvio di MATLAB, essendo format short il formato di default di MATLAB)

$\gg -3.456$

$ans =$

-3.4560

- **format long**: 15 cifre, notazione a virgola fissa.

  Esempio 21 $\gg format\ long$

  $\gg -3.456$

  $ans =$

  -3.45600000000000

- **format short e**: 5 cifre, notazione esponenziale.

  Esempio 22 $\gg format\ short\ e$

  $\gg -3.456$

  $ans =$

  $-3.4560e+000$

- **format long e**: 15 cifre, notazione esponenziale.

  Esempio 23 $\gg format\ long\ e$

  $\gg -3.456$

  $ans =$

  $-3.456000000000000e+000$

- **format short g**: 5 cifre, notazione migliore tra quella a virgola fissa e quella esponenziale.

  Esempio 24 $\gg format\ short\ g$

  $\gg -3.456$

  $ans =$

  -3.456

  Esempio 25 $\gg format\ short\ g$

  $\gg 0.00000000000000000002$

  $ans =$

  $2e-020$

- **format long g**: 15 cifre, notazione migliore tra quella a virgola fissa e quella esponenziale.

  Esempio 26 $\gg format\ long\ g$
  $\gg -3.456$
  $ans =$
  $$-3.456$$

  Esempio 27 $\gg format\ long\ g$
  $\gg$ 0.00000000000000000002
  $ans =$
  $$2e-020$$

- **format hex**: notazione esadecimale.

  Esempio 28 $\gg format\ hex$
  $\gg -3.456$
  $ans =$
  $c00ba5e353f7ced9$

- **format bank**: 2 cifre decimali.

  Esempio 29 $\gg format\ bank$
  $\gg -3.456$
  $ans =$
  -3.46

- **format rat**: approssimato ad una frazione.

  Esempio 30 $\gg format\ rat$
  $\gg -3.456$
  $ans =$
  $-432/125$

- **format +**: solo +, $-$ e spazi bianchi.

  Esempio 31 $\gg format\ +$
  $\gg -3.456$
  $ans =$
  $-$

Esempio 32 $\gg format\ +$
$\gg 3.456$
$ans =$
$+$

Esistono inoltre le due seguenti opzioni:

- **format compact**: visualizza in maniera più compatta, riducendo al minimo i caratteri di fine linea;
- **format loose**: torna alla maniera standard di rappresentazione.

2.4 Variabili

MATLAB non richiede nessun tipo di dichiarazione o di dimensionamento delle variabili. Quando MATLAB incontra il nome di una nuova variabile, automaticamente crea la variabile e alloca la quantità di memoria necessaria. Se una variabile esiste già, MATLAB cambia il suo contenuto e, eventualmente, alloca la nuova memoria (se, ad esempio, a quella variabile è assegnata una matrice di dimensioni maggiori rispetto a quelle iniziali).
Una variabile che abbia valore scalare si definisce in modo ovvio mediante l'istruzione di assegnazione. Ad esempio scrivendo:
$\gg b = 2$
si ottiene l'assegnazione desiderata, che verrà visualizzata sullo schermo in questo modo:
$b =$
    2
Questo vuol dire che alla variabile chiamata b viene assegnato il valore numerico 2.
I nomi delle variabili sono composti da lettere. Il numero massimo di caratteri per la definizione di una variabile è 31; se il nome della variabile eccede questo limite MATLAB usa solo i primi 31 caratteri per identificare la variabile.

Esempio 33 *Supponiamo di eseguire la seguente assegnazione:*
$\gg abcdefghilmnopqrstuvzabcdefghilmnopqrstuvz = 1$
MATLAB visualizza:
$abcdefghilmnopqrstuvzabcdefghil =$
    1
cioè si ferma alla 31^a *lettera del nome della variabile.*

Essendo case sensitive, MATLAB fa distinzione tra lettere minuscole e maiuscole nei nomi delle variabili, quindi, ad esempio, A ed a rappresentano due variabili diverse.

2.5 La variabile *ans*

Se si esegue un'operazione oppure si valuta una funzione [3] senza assegnare il risultato ad una variabile, MATLAB assegna, per default il risultato alla variabile *ans* (diminutivo di answer).

Esempio 34 *Scrivendo* $\gg 2+3$ *si ottiene:*
$ans\ =$
$\quad 5$

Se viene lanciata una seconda istruzione che non contiene assegnazione, la variabile *ans* viene aggiornata al nuovo valore e il vecchio valore viene perso.

Esempio 35 *Supponiamo di aver eseguito l'operazione dell'esempio precedente,che ha assegnato alla variabile ans il valore* 5 *ed eseguiamo un'altra operazione:* $\gg 6-8$. *Si ottiene:*
$ans\ =$
$\quad -2$
Da questo punto in poi (e fino a quando non c'è una nuova operazione senza assegnazione) la variabile ans vale -2 *e non più* 5.

2.6 Altre variabili predefinite

Oltre i e j, MATLAB comprende altre variabili predefinite:

- **pi**: 3.4159265... (pi greco)
  Il valore *pi* rappresenta il rapporto tra una circonferenza e il suo diametro. Il valore di *pi* può essere calcolato mediante l'espressione $4*atan(1)$ oppure mediante l'espressione $imag(log(-1))$;
- **eps**: precisione di macchina in virgola mobile, 2^{-52};
- **realmin**: il più piccolo numero in virgola mobile che MATLAB può rappresentare; qualsiasi valore ad esso inferiore dà underflow.

[3] Per l'uso delle funzioni in MATLAB si legga il paragrafo Funzioni.

Secondo lo standard IEEE il valore di *realmin* in doppia precisione corrisponde a 2^{-1022}, cioè circa a $2.2251e-308$;

- **realmax**: il più grande numero in virgola mobile che MATLAB può rappresentare; qualsiasi valore ad esso superiore dà overflow.
  Secondo lo standard IEEE il valore di *realmax* in doppia precisione corrisponde ad un bit meno di 2^{1024}, cioè circa a $1.7977e+308$;

- **flops**: numero di operazioni in virgola mobile eseguite nella sessione di lavoro di MATLAB.
  Non è possibile contare assolutamente tutte le operazioni in virgola mobile eseguite: vengono conteggiate solo quelle più importanti. Le *addizioni* e le *sottrazioni* vengono conteggiate come una operazione se eseguite su numeri reali, come due operazioni se eseguite su numeri complessi. Le *moltiplicazioni* e le *divisioni* vengono conteggiate come una operazione se eseguite su numeri reali, come sei operazioni se eseguite su numeri complessi. Le *funzioni elementari* vengono conteggiate come una operazione se eseguite su numeri reali, come più operazioni se eseguite su numeri complessi.
  Mediante $\gg flops(0)$ si azzera il contatore delle operazioni effettuate nella sessione di lavoro.

  Esempio 36 *Se A e B sono matrici reali di dimensione $n \times n$, allora eseguire $A+B$ richiede n^2 operazioni, eseguire $A*B$ richiede $2*n^3$ operazioni.*

Osservazione 4 Il nome delle variabili non è riservato: è possibile cambiare il valore di una variabile il cui nome è predefinito.
Per esempio si può fissare $eps = 1e-6$ e quindi usare questo valore nei calcoli. Per ripristinare il valore predefinito della variabile si usa l'istruzione $\gg$ *clear eps*.
Analogamente, con le lettere i e j potrebbero essere denotate delle altre variabili che non siano l'unità immaginaria; ad esempio esse potrebbero essere utilizzate all'interno di cicli for [4]: in questo modo il valore prestabilito viene annullato. Per ripristinarlo si può, anche in questo caso, scrivere $\gg$ *clear* i (oppure $\gg$ *clear* j), ma si può anche imporre $\gg i = sqrt(-1)$ (oppure $\gg j = sqrt(-1)$).

[4] Si consulti il paragrafo Iterazione del capitolo Programmazione.

2.7 Operatori elementari

Nelle espressioni in MATLAB vengono usati gli operatori matematici che ci sono familiari, rispettando le regole algebriche:

$+$	addizione;
$-$	sottrazione;
$*$	moltiplicazione;
$/$	divisione a destra;
$\backslash$	divisione a sinistra;
^	elevamento a potenza;
$'$	complesso coniugato trasposto;
( )	ordine nelle operazioni.

Gli operatori elencati vengono usati in maniera ovvia quando si opera su numeri (array di dimensione 1×1).
Analizzeremo nei paragrafi Operazioni su matrici e Operazioni elemento per elemento del capitolo successivo come utilizzare questi operatori su matrici.

2.8 Inf e NaN

MATLAB comprende la matematica estesa: oltre a tutti i numeri naturali, razionali, reali e immaginari si possono considerare due valori non numerici, che fanno parte dell'analisi. Essi sono:

- **Inf**: rappresentazione dell'aritmetica IEEE per infinito, che si ottiene effettuando operazioni come la divisione per zero, oppure valutando un'espressione matematica ben definita che dia overflow, cioè che restituisca un valore superiore a *realmax*;

- **NaN** [5]: forma indeterminata, come 0/0 oppure $inf - inf$.

Con Inf e NaN si possono effettuare le operazioni dell'aritmetica; valgono le seguenti regole:

1. **Addizione e sottrazione**:

[5] Sta per Not-a-Number, cioè valore non numerico.

$$+Inf + num = \begin{cases} +Inf & \text{se } num \; \epsilon \; \mathbb{C} \\ +Inf & \text{se } num = +Inf \\ NaN & \text{se } num = -Inf \\ NaN & \text{se } num = \pm NaN \end{cases}$$

Osservazione 5 Osserviamo che il risultato $+Inf$ viene visualizzato da MATLAB semplicemente con Inf, cioè senza l'indicazione del segno. Anche in input si può omettere il segno, cioè si può scrivere soltanto $\gg Inf$ anziché $\gg +Inf$.

$$-Inf + num = \begin{cases} -Inf & \text{se } num \; \epsilon \; \mathbb{C} \\ NaN & \text{se } num = +Inf \\ -Inf & \text{se } num = -Inf \\ NaN & \text{se } num = \pm NaN \end{cases}$$

$$NaN \pm num = \begin{cases} & \text{se } num \; \epsilon \; \mathbb{C}, \\ NaN & \text{se } num = \pm Inf, \\ & \text{se } num = NaN \end{cases}$$

Osservazione 6 Le operazioni appena considerate godono della proprietà commutativa.

Osservazione 7 NaN non ha segno. Con il simbolo $\pm$ si è voluto denotare (per brevità) la somma o, indifferentemente, la differenza con il valore NaN.

2. **Moltiplicazione**:

$$\pm Inf * num = \begin{cases} \pm Inf & \text{se } num > 0 \\ \mp Inf & \text{se } num < 0 \\ NaN & \text{se } num = 0 \\ \pm Inf & \text{se } num = +Inf \\ \mp Inf & \text{se } num = -Inf \\ NaN & \text{se } num = NaN \end{cases}$$

$$NaN * num = \begin{cases} & \text{se } num \; \epsilon \; \mathbb{C}, \\ NaN & \text{se } num = \pm Inf, \\ & \text{se } num = NaN \end{cases}$$

3. **Divisione a destra**:

$$\pm Inf/num = \begin{cases} \pm Inf & \text{se } num > 0 \\ \mp Inf & \text{se } num < 0 \\ \pm Inf & \text{se } num = 0 \\ NaN & \text{se } num = \pm Inf \\ NaN & \text{se } num = NaN \end{cases}$$

$$NaN/num = \begin{cases} & \text{se } num \in \mathbb{C}, \\ NaN & \text{se } num = \pm Inf, \\ & \text{se } num = NaN \end{cases}$$

Osservazione 8 Nel caso di divisione a desta di $\pm Inf$, di NaN, o di un qualsiasi numero complesso per 0 (oppure per un numero molto prossimo a 0), MATLAB esegue l'operazione richiesta, ma lancia anche un messaggio di warning.

Esempio 37
$\gg 5/0$

Warning: Divide by zero. [6]
$ans =$

Inf

$\gg 5/1e-400$

Warning: Divide by zero.
$ans =$

Inf

4. **Divisione a sinistra**:

$$\pm Inf \backslash num = \begin{cases} 0 & \text{se } num \in \mathbb{C} \\ NaN & \text{se } num = \pm Inf \\ NaN & \text{se } num = NaN \end{cases}$$

[6] Traduzione:
Attenzione: divisione per zero.

$$NaN \backslash num = \begin{cases} & \text{se } num \in \mathbb{C}, \\ NaN & \text{se } num = \pm Inf, \\ & \text{se } num = NaN \end{cases}$$

5. **Elevamento a potenza**:

$$\pm Inf\hat{\ }num = \begin{cases} \pm Inf & \text{se } num > 0 \\ 0 & \text{se } num < 0 \\ \pm 1 & \text{se } num = 0 \\ \pm Inf & \text{se } num = +Inf \\ 0 & \text{se } num = -Inf \\ NaN & \text{se } num = NaN \end{cases}$$

$$NaN\hat{\ }num = \begin{cases} NaN & \text{se } num \in \mathbb{C} \setminus \{0\}, \\ 1 & \text{se } num = 0 \\ NaN & \text{se } num = \pm Inf, \\ NaN & \text{se } num = NaN \end{cases}$$

Esercizio 5 *Si verifichino con degli esempi numerici tutte le proprietà di Inf e di NaN elencate nel paragrafo.*
Soluzione.

2.9 Funzioni

MATLAB include la possibilità di utilizzare un gran numero di funzioni matematiche standard, come *abs*, *sqrt*, *exp*, *sin* e altre.
Eseguire la radice quadrata o il logaritmo di un numero negativo non è un errore: MATLAB fornisce automaticamente l'appropriato risultato complesso.
Come si evincerà dagli esempi addotti, le funzioni MATLAB elencate in questo paragrafo possono essere utilizzate non solo per eseguire operazioni su scalari, ma anche su matrici: queste funzioni applicate ad una matrice, operano elemento per elemento, restituendo una matrice delle stesse dimensioni di quella di partenza, i cui elementi sono i valori che la funzione assume nei singoli elementi della matrice; questo consente un notevole risparmio in termini di tempi di esecuzione e numero di istruzioni.
MATLAB comprende anche molte funzioni matematiche avanzate, come le funzioni di Bessel e le funzioni Gamma. La maggior parte di queste funzioni accettano argomenti complessi.

Per avere una lista delle funzioni matematiche, dalle più elementari alle più avanzate, si può ricorrere all'*help*: $\gg$ *help elfun* restituisce una lista delle funzioni elementari di MATLAB; $\gg$ *help specfun* oppure $\gg$ *help elmat* restituiscono una lista delle funzioni matematiche più avanzate.

2.9.1 Funzioni elementari

MATLAB dispone delle seguenti funzioni elementari:

- *Arrotondamento*
  - **round** arrotonda all'intero più vicino;
  - **fix** arrotonda all'intero più vicino dalla parte dello zero;
  - **floor** arrotonda per difetto all'intero più vicino;
  - **ceil** arrotonda per eccesso all'intero più vicino.

  Osservazione 9 Le istruzioni precedenti sono tali che, applicate ad un valore immaginario, la parte reale e la parte immaginaria sono trattate separatamente.

Esempio 38 *Analizziamo l'uso del comando* ***round*** *applicato ai seguenti casi:*

$\gg round(3.1)$
$ans =$
3
$\gg round(3.8)$
$ans =$
4
$\gg round(-3.1 + 4.3i)$
$ans =$
$-3.0000 + \ 4.0000i$

$\gg round(-3.1)$
$ans =$
-3
$\gg round(-3.8)$
$ans =$
-4
$\gg round(l)$
$ans =$
$$\begin{matrix} 1 & 1 \\ 3 & 1 \end{matrix}$$

Esempio 39 *Analizziamo l'uso del comando* ***fix*** *applicato ai seguenti casi:*

$\gg fix(3.1)$
$ans =$

$\gg fix(-3.1)$
$ans =$

3 | -3
$\gg fix(3.8)$ | $\gg fix(-3.8)$
$ans =$ | $ans =$
3 | -3
$\gg fix(-3.1+4.3i)$ | $\gg fix(l)$
$ans =$ | $ans =$

$-3.0000+\ 4.0000i$

$$\begin{matrix} 0 & 1 \\ 3 & 0 \end{matrix}$$

Esempio 40 *Analizziamo l'uso del comando* ***floor*** *applicato ai seguenti casi:*

$\gg floor(3.1)$ | $\gg floor(-3.1)$
$ans =$ | $ans =$
3 | -4
$\gg floor(3.8)$ | $\gg floor(-3.8)$
$ans =$ | $ans =$
3 | -4
$\gg floor(-3.1+4.3i)$ | $\gg floor(l)$
$ans =$ | $ans =$

$-4.0000+\ 4.0000i$

$$\begin{matrix} 0 & 1 \\ 3 & 0 \end{matrix}$$

Esempio 41 *Analizziamo l'uso del comando* ***ceil*** *applicato ai seguenti casi:*

$\gg ceil(3.1)$ | $\gg ceil(-3.1)$
$ans =$ | $ans =$
4 | -3
$\gg ceil(3.8)$ | $\gg ceil(-3.8)$
$ans =$ | $ans =$
4 | -3
$\gg ceil(-3.1+4.3i)$ | $\gg ceil(l)$
$ans =$ | $ans =$

$-3.0000+\ 5.0000i$

$$\begin{matrix} 1 & 2 \\ 4 & 1 \end{matrix}$$

- *Approssimazioni razionali*

- **rem** resto di una divisione intera.
  Con l'istruzione $\gg rem(X, Y)$ si ottiene $X-$ fix $(X./$ [7]Y)$.*Y$, dove la quantità $fix(X./Y)$ è la parte intera del quoziente $X./Y$. La funzione rem restituisce un risultato che è compreso tra 0 e $sign(X)*abs(Y)$. Se Y è 0, rem restituisce NaN. Gli argomenti X e Y della funzione rem devono essere array reali della stessa dimensione, oppure scalari reali;

- **rat** espansione razionale.
  Sebbene tutti i numeri in virgola mobile siano numeri razionali, a volte può essere necessario approssimarli con dei numeri razionali più semplici, che siano frazioni il cui numeratore e il cui denominatore siano numeri interi abbastanza piccoli. La funzione rat provvede a questo. Le approssimazioni razionali sono generate mediante l'espansione troncata $x=\dfrac{n}{d}\approx d_1+\dfrac{1}{d_2+\frac{1}{d_3+\ldots+\frac{1}{d_k}}}$, cioè mediante lo sviluppo in fratti. L'accuratezza dell'approssimazione cresce esponenzialmente con il numero dei termini.
  La sintassi completa della funzione rat è $[n, d] = rat(x, tol)$. Il risultato che si ottiene con questa istruzione è costituito da due numeri n e d tali che $\left|x-\dfrac{n}{d}\right|\leq tol\,|x|$. Il valore di default per la tolleranza è 10^{-6};

- **rats** approssimazione razionale.
  La funzione $rats(X, strlen)$ richiama la funzione rat, restituendo stringhe di lunghezza $strlen$ che rappresentano l'approssimazione razionale di X. Il valore di default per $strlen$ è 13: utilizzare l'istruzione $rats(X)$ equivale ad utilizzare $rats(X, 13)$.

Esempio 42 *Analizziamo l'uso del comando **rem** applicato ai seguenti casi:*
$\gg rem(7,5)$
$ans =$

  2

$\gg rem(7,0)$
Warning: Divide by zero. [8]

[7]Per quanto riguarda le operazioni precedute da un punto presenti in questo paragrafo, si consulti il paragrafo Operazioni elemento per elemento.

[8]Traduzione:
Attenzione: divisione per zero.

$ans =$

NaN

$\gg rem(C, G)$

$ans =$

1	0
0	4

Esempio 43 *Analizziamo l'uso del comando* ***rat*** *applicato ai seguenti casi:*

$\gg rat(5.5)$

$ans =$

$6 + 1/(-2)$

$\gg rat(5.56)$

$ans =$

$6 + 1/(-2 + 1/(-4 + 1/(3)))$

$\gg rat(pi)$

$ans =$

$3 + 1/(7 + 1/(16))$

$\gg rat(l)$

$ans =$

$1 + 1/(-4 + 1/(3 + 1/(3 + 1/(-3))))$

$3 + 1/(3 + 1/(-2))$

$1 + 1/(5)$

$1 + 1/(-5)$

Esempio 44 *Analizziamo l'uso del comando* ***rats*** *applicato ai seguenti casi:*

$\gg rats(5.5)$

$ans =$

$11/2$

$\gg rats(5.56)$

$ans =$

$139/25$

$\gg rats(pi)$

$ans =$

$355/113$

$\gg rats(l)$

$ans =$

$73/100$	$6/5$
$17/5$	$4/5$

- *Fattorizzazione intera*
  - **gdc** massimo comune divisore.
    L'istruzione $G = gcd(A, B)$ restituisce un array contenente il M.C.D. degli elementi corrispondenti degli array A e B (che, ovviamente devono avere le stesse dimensioni), i cui elementi devono essere numeri interi. Per convenzione, $gcd(0, 0)$ restituisce il valore 0; tutti gli altri input restituiscono interi positivi in G.
    L'istruzione $[G, C, D] = gcd(A, B)$, dove A e B sono array di lunghezza m, oltre all'array G contenente i M.C.D. degli elementi di A e B, restituisce gli array C e D tali che
    $A_i * C_i + B_i * D_i = G_i \quad \forall i = 1, \ldots, m;$
  - **lcm** minimo comune multiplo.
    L'istruzione $L = lcm(A, B)$ restituisce un array contenente il m.c.m. degli elementi corrispondenti degli array A e B, che devono avere le stesse dimensioni e che possono contenere soltanto valori interi e strettamente positivi.

Esempio 45 *Analizziamo l'uso del comando **gcd** applicato ai seguenti casi:*

$\gg gcd(5, 10)$
$ans =$

$$5$$

$\gg gcd(5, 7)$
$ans =$

$$1$$

$\gg gcd(5, 0)$

$ans =$

$$5$$

$\gg gcd(0, 0)$

$\gg gcd(C, G)$
$ans =$

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

$\gg [R, S, T] = gcd(C, G)$
$R =$

$$\begin{matrix} 1 & 2 \\ 3 & 4 \end{matrix}$$

$S =$

$$\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

$T =$

$$\begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix}$$

$ans =$

$\quad 0$

Esempio 46 *Analizziamo l'uso del comando **lcm** applicato ai seguenti casi:*

$\gg lcm(5,10)$

$ans =$

$\quad 10$

$\gg lcm(5,7)$

$ans =$

$\quad 35$

$\gg lcm(C,H)$

$ans =$

$$\begin{matrix} 3 & 2 \\ 12 & 20 \end{matrix}$$

Esercizio 6 *Si calcoli il m.c.m. e il M.C.D. tra i numeri* 15 *e* 20.
Soluzione.

- *Aritmetica complessa*
  - **real** parte reale;
  - **imag** coefficiente dell'immaginario;
  - **conj** complesso coniugato;
  - **abs** valore assoluto oppure modulo complesso;
  - **angle** angolo di fase espresso in radianti.

Esempio 47 *Analizziamo l'uso del comando **real** applicato ai seguenti casi:*

$\gg real(5)$

$ans =$

$\quad 5$

$\gg real(5+3i)$

$ans =$

$\quad 5$

$\gg real(3i)$

$ans =$

$\quad 0$

$\gg real(E)$

$ans =$

$$\begin{matrix} 10 & 3 & 0 & -7 \\ 0 & 8 & -6 & -6 \end{matrix}$$

Esempio 48 *Analizziamo l'uso del comando* ***imag*** *applicato ai seguenti casi:*

$\gg imag(5)$
$ans =$

$\quad 0$

$\gg imag(5+3i)$
$ans =$
$\quad 3$
$\gg imag(3i)$
$ans =$
$\quad 3$

$\gg imag(E)$
$ans =$

$$\begin{matrix} 0 & 4 & 12 & 1 \\ 2 & 0 & 0 & -1 \end{matrix}$$

Esempio 49 *Analizziamo l'uso del comando* ***conj*** *applicato ai seguenti casi:*
$\gg conj(5)$
$ans =$
$\quad 5$
$\gg conj(5+3i)$
$ans =$
$\quad 5.0000 - \ 3.0000i$
$\gg conj(3i)$
$ans =$
$\quad 0 - \ 3.0000i$
$\gg conj(E)$
$ans =$

$$\begin{matrix} 10.0000 & 3.0000 - \ 4.0000i & 0 - \ 12.0000i & -7.0000 - \ 1.0000i \\ 0 - \ 2.0000i & 8.0000 & -6.0000 & -6.0000 + \ 1.0000i \end{matrix}$$

Esempio 50 *Analizziamo l'uso del comando* ***abs*** *applicato ai seguenti casi:*

$\gg abs(4)$
$ans =$
$\quad 4$
$\gg abs(-4)$
$ans =$

$\quad 4$

$\gg abs(4+3i)$
$ans =$
$\quad 5$
$\gg abs(E)$
$ans =$

$$\begin{matrix} 10.0000 & 5.0000 & 12.0000 & 7.0711 \\ 2.0000 & 8.0000 & 6.0000 & 6.0828 \end{matrix}$$

Esempio 51 *Analizziamo l'uso del comando* ***angle*** *applicato ai seguenti casi:*

$\gg angle(3)$
$ans =$
0
$\gg angle(4i)$
$ans =$
1.5708

$\gg angle(3+4i)$
$ans =$
0.9273
$\gg angle(E)$
$ans =$

$$\begin{matrix} 0 & 0.9273 & 1.5708 & 2.9997 \\ 1.5708 & 0 & 3.1416 & -2.9764 \end{matrix}$$

- *Segno*
  - **sign** funzione signum.
    L'istruzione $Y = sign(X)$ restituisce un array Y delle stesse dimensioni di X nel quale ogni elemento è:
    * 1 se il corrispondente elemento di X è maggiore di 0;
    * 0 se il corrispondente elemento di X è uguale a 0;
    * -1 se il corrispondente elemento di X è minore di 0.

    Per X complesso e non nullo, $sign(X) = X./abs(X)$.

Esempio 52 *Analizziamo l'uso del comando* ***sign*** *applicato ai seguenti casi:*
$\gg sign(5)$
$ans =$
1
$\gg sign(-5)$
$ans =$
-1
$\gg sign(0)$
$ans =$
0
$\gg sign(D)$
$ans =$

$$\begin{matrix} 0 & 1 & 1 \\ -1 & 1 & 0 \end{matrix}$$

$\gg sign(E)$
$ans =$

$$\begin{matrix} 1.0000 & 0.6000+\ 0.8000i & 0+\ 1.0000i & -0.9899+\ 0.1414i \\ 0+\ 1.0000i & 1.0000 & -1.0000 & -0.9864-\ 0.1644i \end{matrix}$$

Esercizio 7 *Si inserisca il numero complesso n con parte reale uguale a 2 e parte immaginaria uguale a 3 e se ne calcoli il modulo e l'angolo di fase.*
Si moltiplichi n per π e si memorizzi il risultato nella variabile eps. Si calcoli il complesso coniugato di eps e si arrotondi il risultato per eccesso all'intero più vicino.
Si ripristini il valore di eps predefinito in MATLAB.
Soluzione.

2.9.2 Funzioni trigonometriche

Le funzioni trigonometriche di MATLAB sono:

sin	seno	**sinh**	seno iperbolico
cos	coseno	**cosh**	coseno iperbolico
tan	tangente	**tanh**	tangente iperbolica
cot	cotangente	**coth**	cotangente iperbolica
sec	secante	**sech**	secante iperbolica
csc	cosecante	**csch**	cosecante iperbolica

Esempio 53 $\gg sin(pi/4)$
$ans =$
0.7071

Osservazione 10 Tutte le precedenti funzioni operano sugli array cui vengono applicate elemento per elemento. Il loro dominio e il loro range comprende anche valori complessi.
Tutti gli angoli vengono espressi in radianti.

Esempio 54 $\gg sin(l)$
$ans =$

$$\begin{matrix} 0.6669 & 0.9320 \\ -0.2555 & 0.7174 \end{matrix}$$

Esempio 55 $\gg sin(4 * i * pi)$
$ans =$
$0 \ + 1.4338e + 005i$

Osservazione 11 Le espressioni $sin(pi)$ e $cos(pi/2)$ non valgono esattamente 0 perché pi non vale esattamente π: viene fornito un valore che approssima 0 nei margini della precisione di macchina, eps.

Per la stessa ragione, cioè essendo *pi* un'approssimazione del valore esatto π, le espressioni $tan(pi/2)$ e $sec(pi/2)$ non vengono calcolate come $infinito$, ma come l'inverso della precisione di macchina, *eps*.

Osservazione 12 Può essere utile ricordare le seguenti relazioni:

1. $sin(x + i * y) = sin(x) * cosh(y) + i * cos(x) * sinh(y)$
2. $sin(z) = \dfrac{e^{i*z} - e^{-i*z}}{2 * i}$
3. $sinh(z) = \dfrac{e^{z} - e^{-z}}{2}$
4. $cos(x + i * y) = cos(x) * cosh(y) - i * sin(x) * sinh(y)$
5. $cos(z) = \dfrac{e^{i*z} + e^{-i*z}}{2}$
6. $cosh(z) = \dfrac{e^{z} + e^{-z}}{2}$
7. $tan(z) = \dfrac{sin(z)}{cos(z)}$
8. $tanh(z) = \dfrac{sinh(z)}{cosh(z)}$
9. $cot(z) = \dfrac{1}{tan(z)}$
10. $coth(z) = \dfrac{1}{tanh(z)}$
11. $sec(z) = \dfrac{1}{cos(z)}$
12. $sech(z) = \dfrac{1}{cosh(z)}$
13. $csc(z) = \dfrac{1}{sin(z)}$
14. $csch(z) = \dfrac{1}{sinh(z)}$

Esercizio 8 *Mediante valori numerici si verifichino le* 14 *uguaglianze elencate nell'osservazione precedente.*
Soluzione.

asin	arcoseno	**asinh**	arcoseno iperbolico
acos	arcocoseno	**acosh**	arcocoseno iperbolico
atan	arcotangente	**atanh**	arcotangente iperbolica
acot	arcocotangente	**acoth**	arcocotangente iperbolica
asec	arcosecante	**asech**	arcosecante iperbolica
acsc	arcocosecante	**acsch**	arcocosecante iperbolica

Osservazione 13 Tutte le precedenti funzioni operano sugli array cui vengono applicate elemento per elemento. Il loro dominio e il loro range comprende anche valori complessi.
Per array reali il dominio di queste funzioni è l'intervallo $[-1, 1]$.
Sempre per array reali risulta:
$[-\frac{\pi}{2}, \frac{\pi}{2}]$ è il range della funzione arcoseno;
$[0, \pi]$ è il range della funzione arcocoseno;
$[-\frac{\pi}{2}, \frac{\pi}{2}]$ è il range della funzione arcotangente.
Per elementi esterni all'intervallo $[-1, 1]$ queste funzioni assumono valori complessi.
Tutti gli angoli vengono espressi in radianti.

Osservazione 14 Può essere utile ricordare le seguenti relazioni:

1. $asin(z) = -i * log[i * z + (1 - z^2)^{\frac{1}{2}}]$
2. $asinh(z) = log[z + (1 + z^2)^{\frac{1}{2}}]$
3. $acos(z) = -i * log[z + i * (1 - z^2)^{\frac{1}{2}}]$
4. $acosh(z) = log[z + (z^2 - 1)^{\frac{1}{2}}]$
5. $atan(z) = \frac{i}{2} * log(\frac{i + z}{i - z})$
6. $atanh(z) = \frac{1}{2} * log(\frac{1 + z}{1 - z})$
7. $acot(z) = atan(\frac{1}{z})$
8. $acoth(z) = atanh(\frac{1}{z})$
9. $asec(z) = acos(\frac{1}{z})$
10. $asech(z) = acosh(\frac{1}{z})$

11. $acsc(z) = asin(\frac{1}{z})$

12. $acsch(z) = asinh(\frac{1}{z})$

Esercizio 9 *Mediante valori numerici si verifichino le* 12 *uguaglianze elencate nell'osservazione precedente.*
Soluzione.

2.10 Operatori esponenziali e logaritmici

Gli operatori esponenziali di MATLAB sono:

- **pow2** esponenziale in base 2.
  $X = pow2(Y)$ restituisce un array X i cui elementi X_i sono uguali a 2 elevato alla potenza Y_i .
  Se E ed F sono rispettivamente un array di numeri interi ed un array di numeri reali, $X = pow2(F, E)$ calcola il vettore $X = F. * (2.\hat{}E)$.

  Esempio 56 $\gg pow2(H)$
  $ans =$

  $$\begin{matrix} 8 & 4 \\ 16 & 32 \end{matrix}$$

  Esempio 57 $\gg pow2(G, H)$
  $ans =$

  $$\begin{matrix} 16 & -8 \\ 48 & 256 \end{matrix}$$

- **exp** esponenziale in base e.
  La funzione exp è una funzione elementare che opera sugli elementi di un array, il cui dominio comprende anche i numeri complessi.
  $Y = exp(X)$ restituisce l'esponenziale di ogni elemento di X, cioè e^X.
  Per valori complessi del tipo $z = x + i * y$ questa funzione è tale che $e^z = e^x * [cos(y) + i * sin(y)]$.

  Esempio 58 *Lanciando* $\gg exp(1)$ *si ottiene la base dei logaritmi naturali, cioè:*
  $ans =$
  2.7183

Esempio 59 $\gg exp(H)$

$ans =$

20.0855	7.3891
54.5982	148.4132

Gli operatori logaritmici di MATLAB sono:

- **log** logaritmo naturale.
  La funzione log è una funzione elementare che opera sugli elementi di un array, il cui dominio comprende anche i numeri negativi.
  $Y = log(X)$ restituisce l'array Y i cui elementi rappresentano il logaritmo naturale degli elementi di X. Per elementi z complessi oppure negativi, con $z = x + i * y$, il risultato complesso che fornisce MATLAB è dato da $log(z) = log(abs(z)) + i * atan(y/x)$.
  Il valore $abs(log(-1))$ fornisce una buona approssimazione di π.

  Esempio 60 $\gg log(G)$

  $ans =$

0.6931	0.6931 + 3.1416i
1.0986	2.0794

- **log2** logaritmo in base 2.
  $Y = log2(X)$ calcola il logaritmo in base 2 degli elementi di X e lo memorizza nell'array Y.
  $[F, E] = log2(X)$ restituisce gli array F ed E tali che F è un array di elementi reali (il cui range di solito è dato da $0.5 \leq abs(F) < 1$), mentre E è un array di numeri interi. Per X ad elementi reali, E ed F soddisfano l'equazione $X = F. * 2.\hat{}E$.

  Esempio 61 $\gg log2(G)$

  $ans =$

1.0000	1.0000 + 4.5324i
1.5850	3.0000

- **log10** logaritmo in base 10.
  La funzione $log10$ opera elemento per elemento sugli array cui viene applicata. Il suo dominio comprende anche i numeri negativi, per i quali MATLAB restituisce risultati complessi.
  $Y = log10(X)$ calcola il logaritmo in base 10 degli elementi di X e lo memorizza nell'array Y.

Esempio 62 $\gg log10(G)$

$ans =$

0.3010	$0.3010 + \ 1.3644i$
0.4771	0.9031

Esercizio 10 *Si valuti il numero* e*. Si memorizzi il suo logaritmo in base* 2 *nella variabile* a *e il suo logaritmo in base* 10 *nella variabile* b*.*
Si calcoli il resto della divisione intera tra a *e* b *e si esprima l'approssimazione razionale e l'espansione razionale del risultato.*
Soluzione.

Esercizio 11 *Si moltiplichi il più piccolo numero in virgola mobile per* 10^{308} *e si memorizzi il risultato nella variabile* a*.*
Si estragga la radice di indice 1000 *del più grande numero in virgola mobile e si memorizzi il risultato nella variabile* b*.*
Si calcoli la somma delle due variabili a *e* b *e si memorizzi l'esponenziale in base* 2 *del risultato nella variabile* d*. Si arrotondi* d *all'intero più vicino e si memorizzi il risultato nella variabile* l*.*
Si calcoli la radice cubica di l *e si arrotondi il risultato verso lo* 0*. Si calcoli parte reale e parte immaginaria di quest ultimo valore.*
Soluzione.

Capitolo 3

Matrici

3.1 Introduzione

Il nome MATLAB deriva da MATrix LABoratory. Questo mette in evidenza che la componente caratteristica di MATLAB è la matrice; ciò rende MATLAB uno strumento adatto a tutte le attività che abbiano come substrato matematico l'algebra lineare.

In MATLAB una matrice è un array [1] rettangolare di numeri. Gli scalari possono essere intesi come particolari matrici di dimensione 1×1, mentre le matrici con una sola riga o con una sola colonna (cioè le matrici $1 \times n$ oppure $n \times 1$) rappresentano i vettori. MATLAB consente anche di utilizzare array a n indici (con $n \geq 3$), che rappresentano i tensori di ordine n. Ad esempio, un array tridimensionale può rappresentare dati fisici, come la temperatura in una stanza campionata su una griglia rettangolare, oppure può rappresentare un campione di matrici dipendenti dal tempo, A(t).

In MATLAB ci sono anche altre vie per memorizzare dati numerici e non numerici, ma, inizialmente, è bene pensare ad ogni variabile come una matrice: dove gli altri linguaggi di programmazione lavorano con un numero per volta, MATLAB consente di lavorare in maniera facile e veloce con le matrici.
In MATLAB le matrici possono essere fornite in cinque modi diversi:

1. da tastiera, inserendo una lista di elementi;

2. caricate in memoria da file di dati esterni;

[1] In maniera informale i termini *matrice* e *array* vengono usati intercambiabilmente. Più precisamente, un array è una struttura di dati cui si accede mediante indici; un array con un solo indice si dice *vettore*; una *matrice* è un array numerico bidimensionale che rappresenta una trasformazione lineare.

3. generate mediante M-file;

4. generate mediante function interne;

5. generate come il risultato di un file MEX.

Esaminiamo i primi tre punti, lasciando ai capitoli successivi la descrizione degli altri.

3.2 Ingresso da tastiera

Per inserire una matrice da tastiera bisogna tener presenti alcune convenzioni:

- gli elementi sulla stessa riga vanno separati da spazi bianchi o da virgole;
- per indicare la fine di una riga si usa un punto e virgola oppure si preme il tasto INVIO e si scrive su una nuova riga;
- l'intera lista di elementi deve essere racchiusa in parentesi quadre.

Esempio 63 *Il vettore riga a può essere inserito mediante l'istruzione*
$\gg a =[\ 1\ 2\ 3\ 4\ ]$ *dalla quale si ottiene:*
$a =$

1 2 3 4

Esempio 64 *Il vettore colonna c può essere inserito mediante l'istruzione*
$\gg c = [\ 1$ [2]
2
3
4]
oppure mediante l'istruzione:
$\gg c =[\ 1\ ;\ 2\ ;\ 3\ ;\ 4\ ]$

In entrambi i casi si ottiene:
$c =$
1
2
3
4

[2]In tutta la trattazione si ometterà di precisare che bisogna premere il tasto INVIO ogni volta che si va a capo.

Esempio 65 *La matrice* A *può essere memorizzata mediante le seguenti istruzioni:*
$\gg A = [\ 16, 3, 2, 13; 5, 10, 11, 8; 9, 6, 7, 12; 4, 15, 14, 1\ ]$

$\gg A = [\ 16\ 3\ 2\ 13\ ;\ 5\ 10\ 11\ 8\ ;\ 9\ 6\ 7\ 12\ ;\ 4\ 15\ 14\ 1\ ]$

$\gg A = [\ 16, 3, 2, 13$
$5, 10, 11, 8$
$9, 6, 7, 12$
$4, 15, 14, 1\ ]$

$\gg A = [\ 16\ 3\ 2\ 13$
$5\ 10\ 11\ 8$
$9\ 6\ 7\ 12$
$4\ 15\ 14\ 1\ ]$

Qualsiasi sia la scelta fatta, MATLAB visualizza la matrice in questo modo:

$A =$

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Una volta che è stata inserita, la matrice resta memorizzata nell'area di lavoro di MATLAB e può essere richiamata ogni volta che occorre mediante il suo nome: con il comando $\gg A$ viene visualizzata nuovamente la matrice.
La matrice A (come ogni altra variabile) viene conservata in memoria finché non è esplicitamente cancellata con il comando $\gg$ *clear* A oppure finché non viene chiusa la sessione di MATLAB.

Per conoscere quali sono tutte le variabili presenti in memoria si usa il comando $\gg$ *who* che ne fornisce l'elenco. Il comando $\gg$ *whos* oltre all'elenco fornisce anche la dimensione delle variabili in memoria ed altre informazioni.

Osservazione 15 Quando un'istruzione viene fatta seguire da un **punto e virgola**, MATLAB esegue l'operazione, senza visualizzarne il risultato sullo schermo. Questo è particolarmente utile quando si lavora con matrici di dimensioni molto grandi: il flusso sullo schermo della lista degli elementi della matrice potrebbe rivelarsi solo una perdita di tempo.

Ad esempio l'istruzione
$\gg a = [\,1\ \ 2\ \ 3\ \ 4\,];$
effettua l'assegnazione della variabile a, ma non visualizza i suoi elementi sullo schermo.

Osservazione 16 Per continuare un'istruzione molto lunga su una linea successiva di schermo si usano **tre punti**. Ad esempio le istruzioni:
$\gg\ d = [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ ]$
e
$\gg\ d = [\ 1\ 2\ 3\ 4\ 5\,\sqcup\ \ldots$
6 7 8 9 10]
restituiscono lo stesso risultato, cioè
$d =$

1 2 3 4 5 6 7 8 9 10

Esercizio 12 *Si memorizzino i vettori*

$$vett1 = \begin{pmatrix} -2 & 12 & 0 & -14 \end{pmatrix} \text{ e } vett2 = \begin{pmatrix} -2 \\ 12 \\ 0 \\ -14 \end{pmatrix}.$$

Soluzione.

Esercizio 13 *Si memorizzi la matrice identità di ordine* 5 *nella variabile* I.
Soluzione.

3.3 Ingresso da file esterni

Supponiamo che il file IN.DAT contenga i seguenti dati, scritti in ASCII, come output dell'esecuzione di un programma scritto per esempio in Pascal, Fortran, Basic, oppure C:

1 2 3 4 5 6 7 8 9 10

Il comando $\gg$ *load* $IN.DAT$ provoca la lettura del file e la sua memorizzazione in una variabile con lo stesso nome del file. Nel nostro caso avremo il vettore $IN = [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ ]$.

Osservazione 17 Mediante il comando $\gg$ *load* $IN.DAT$ il vettore IN viene soltanto caricato in memoria e non visualizzato. Per leggere gli elementi del vettore è sufficiente lanciare
$\gg IN$
L'istruzione visualizzerà il vettore.

Duale del comando load è il comando *save*.

3.4 Ingresso da M-file

Le matrici e tutte le variabili in uso possono essere salvate in **M-file**, che sono file di testo contenenti codici MATLAB.
Dall'editor si crea un file contenente i dati che si vogliono memorizzare, seguendo la procedura spiegata nel paragrafo Come scrivere un nuovo file.... Si salva il file con un nome avente estensione .m. Lanciando il nome del file (senza estensione) MATLAB legge il file e crea la variabile memorizzata.

Esempio 66 *Consideriamo la matrice A e supponiamo di volerla salvare in un M-file.*
Da MATLAB lanciamo il comando $\gg$ *edit, che apre l'editor di testo. Scriviamo le seguenti righe:*
A = [16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1]
Salviamo il file con il nome A.m. Lanciando $\gg$ *A da MATLAB, si ottiene proprio la matrice memorizzata.*

3.5 *Clear*, *who* e *whos*

Il comando **clear** da solo cancella tutte le variabili e le function in memoria. Ci sono varie opzioni:

$\gg$ *clear variables*: cancella tutte le variabili e le function dal workspace, cioè dall'area di lavoro [3] (come il comando $\gg$ *clear* da solo);

$\gg$ *clear global*: cancella tutte le variabili globali dall'area di lavoro;

$\gg$ *clear functions*: cancella tutte le M-function dall'area di lavoro;

$\gg$ *clear mex*: cancella tutti i file MEX dall'area di lavoro;

[3] MATLAB carica le funzioni che vengono eseguite durante una sessione di lavoro dal disco rigido nella memoria RAM. Ciò permette di velocizzare sensibilmente i tempi di esecuzione le volte successive che dette funzioni vengono utilizzate.
Il comando *clear*, ovviamente, cancella le funzioni dalla RAM e non dal disco rigido.

$\gg$ *clear all*: cancella tutte le variabili, le variabili globali, le function e i link Mex dall'area di lavoro;

$\gg$ *clear var1 var2 ...* : cancella solo le variabili $var1$, $var2$ ... specificate. Scrivendo $\gg$ *clear X∗* vengono cancellate tutte le variabili il cui nome inizia per X. Quando il nome di una variabile è memorizzato in una stringa [4] si usa la sintassi $\gg$ *clear('name')* (dove name è la stringa in cui è memorizzato il nome della variabile).

Il comando **who** fornisce la lista di tutte le variabili correnti. Ci sono varie opzioni:

$\gg$ *who global*: elenca tutte le variabili globali presenti nell'area di lavoro;

$\gg$ *who −file filename*: elenca le variabili specificate nel file filename.mat;

$\gg$ *who var1 var2 ...* : restringe l'elenco delle variabili a quelle specificate. Può essere usato il carattere * per ottenere l'elenco delle variabili il cui nome inizia nel modo precisato: ad esempio $\gg$ *who A∗* elenca tutte le variabili il cui nome inizia per A;

$\gg$ $s = who(\ldots)$ restituisce il cell array [5] s che contiene i nomi delle variabili nel workspace. Fra le parentesi tonde deve essere scritto il nome delle variabili in stringhe di caratteri.

Esempio 67 *Supponiamo di avere nell'area di lavoro la matrice A e la costante b; digitando l'istruzione*
$\gg$ *who*
si ottiene:
Your variables are [6]*:*

A *b*

[4] Si consulti il paragrafo Stringhe del capitolo Stringhe, cell array e strutture.
[5] Si consulti il paragrafo Cell array del capitolo Stringhe, cell array e strutture.
[6] Traduzione:
Le tue variabili sono

Esempio 68 *Supponiamo di avere nell'area di lavoro la matrice A e la costante b; digitando l'istruzione*
$\gg s = who(\ 'A\ ',\ 'b\ ')$
si ottiene:
$s =$

'A'
'b'

Il comando **whos** è la forma lunga del comando *who*. Esso fornisce la lista di tutte le variabili presenti nell'area di lavoro, insieme ad altre informazioni relative alla loro dimensione, alla loro classe, ai bytes occupati, etc. Ci sono varie opzioni:

$\gg$ *whos global*: elenca tutte le variabili globali presenti nell'area di lavoro;

$\gg whos\ -file\ filename$: elenca le variabili specificate nel file filename.mat;

$\gg whos\ var1\ var2\ \ldots$: restringe l'elenco delle variabili a quelle specificate. Può essere usato il carattere * per ottenere l'elenco delle variabili il cui nome inizia nel modo precisato: ad esempio $\gg whos\ A*$ elenca tutte le variabili il cui nome inizia per A;

$\gg s = whos(\ldots)$ restituisce una struttura con i seguenti campi:
name: nome della variabile;
bytes: numero di bytes allocati per l'array;
class: classe a cui appartiene la variabile.

Esempio 69 *Supponiamo di avere nell'area di lavoro la matrice A e la costante b; digitando l'istruzione*
$\gg$ *whos*
si ottiene:

Name	*Size*	*Bytes*	*Class* [7]
A	4×4	128	*double array* [8]

[7] Traduzione:

Nome	Dimensione	Bytes	Classe

[8] Traduzione:
array in doppia precisione

$b \qquad 1 \times 1 \qquad 8$ *double array*

Grand total is 17 *elements using* 136 *bytes* [9]

Osservazione 18 In tutta la trattazione ogni volta che viene lanciato il comando $\gg$ *whos* si suppone di avere in memoria soltanto le variabili elencate nell'output. In questa ipotesi, l'indicazione del numero degli elementi e dei bytes occupati dalle variabili nel workspace (che si trova dopo la frase Grand total is ...) coinciderà con quella della trattazione, altrimenti si avranno risultati diversi. Se nel workspace ci sono variabili non elencate dall'output di *whos* è sufficiente cancellarle mediante l'istruzione $\gg$ *clear* $NomeVar1 \;\; NomeVar2 \;\; \ldots$ (dove $NomeVar1, \;\; NomeVar2 \;\; , \ldots$ sono i nomi delle variabili che si intendono cancellare). In alternativa è sufficiente lanciare anziché $\gg$ *whos*, l'istruzione $\gg$ *whos* $Var1 \;\; Var2 \;\; \ldots$ (dove $Var1, \;\; Var2 \;\; , \ldots$ sono le variabili cui si è interessati).

Esempio 70 *Supponiamo di avere nell'area di lavoro la matrice A; digitando l'istruzione*
$\gg s = whos(\ 'A\ '\ )$
si ottiene:
$s =$

$$\begin{aligned} name &: \ 'A' \\ size &: \ [4 \quad 4] \\ bytes &: \ 128 \\ class &: \ \text{'double'} \end{aligned}$$

3.6 *Load* e *save*

Il comando **load** carica nell'area di lavoro le variabili presenti sul disco. Le opzioni più importanti sono:

$\gg$ *load*: carica tutte le variabili dal file matlab.mat;

$\gg$ *load fname*: carica tutte le variabili dal file fname.mat;

$\gg$ *load* $fname \;\; X \;\; Y \;\; Z \;\; \ldots$: carica dal file fname.mat solo le variabili specificate X, Y, Z ...

[9] Traduzione:
In tutto ci sono 17 elementi che occupano 136 bytes

Il comando **save** memorizza le variabili presenti nell'area di lavoro in un file. Si potrebbe pensare che ogni variabile abbia bisogno di un file in cui essere memorizzata, ma questo renderebbe proibitivo lo spazio richiesto sul disco: il comando *save* memorizza tutte le variabili in un unico file. Le opzioni possibili sono:

$\gg$ *save*: memorizza tutte le variabili presenti nell'area di lavoro nel file matlab.mat;

$\gg$ *save* $NomeFile$: memorizza tutte le variabili presenti nell'area di lavoro nel file NomeFile.mat;

$\gg$ *save* $NomeFile\ NomeVar$: memorizza la variabile $NomeVar$ nel file NomeFile.mat;

$\gg$ *save* $NomeFile\ NomeVar1\ NomeVar2\ NomeVar3\ \ldots$: salva le variabili $NomeVar1$, $NomeVar2$, $NomeVar3\ \ldots$ presenti nell'area di lavoro nel file NomeFile.

Osservazione 19 Il comando *save* memorizza le variabili in formato MATLAB (.mat), quindi esse, dopo essere state salvate su un file, non sono leggibili che da MATLAB.
Salvato il file in questo formato, l'istruzione $\gg$ *load* $NomeFile$ ricarica in memoria tutte le variabili memorizzate in NomeFile mantenendo inalterato il nome con cui erano state memorizzate.
Volendo invece salvare dei dati in formato ASCII, ovvero leggibili poi dall'esterno con un normale word processor o con altri compilatori, come per esempio il Fortran, si deve usare l'opzione -ascii, cioè si deve usare l'istruzione:
$\gg$ *save* $NomeFile.Estensione\ NomeVar\ -ascii$
che permette di salvare la variabile $NomeVar$ nel file NomeFile.Estensione (dove Estensione è l'estensione del file) in formato ASCII.

Esercizio 14 *Si memorizzi il vettore colonna* v *contenente cinque* 0 *e un* 1*, poi lo si salvi sul file ASCII V.DAT e dall'esterno, con un word processor, si modifichi l'*1 *finale in* 2*. Infine da MATLAB si ricarichi il file.*
Soluzione.

3.7 Operazioni su matrici

In MATLAB si possono eseguire tutte le operazioni dell'algebra lineare.
Una caratteristica di MATLAB è il consentire operazioni tra una matrice ed uno scalare: lo scalare viene interpretato come una matrice di dimensioni congruenti i cui elementi sono tutti pari allo scalare dato.

- $+$ **addizione**.
  $A+B$ somma A a B. A e B devono avere le stesse dimensioni, tranne nel caso in cui una delle due variabili sia uno scalare, che può essere sommato ad una matrice di qualsiasi dimensione.

  Esempio 71 $\gg b+b$
  $ans =$
  4

  Esempio 72 $\gg b+C$
  $ans =$

3	4
5	6

  Esempio 73 $\gg C+F$
  $ans =$

-1.0000	$2.0000+\ 1.0000i$
$6.0000-\ 1.0000i$	4.0000

  Esempio 74 $\gg B+C$
  ??? Error using $==>\ +$
  Matrix dimensions must agree. [10]

- $-$ **sottrazione**.
  $A-B$ sottrae B ad A. A e B devono avere le stesse dimensioni, tranne nel caso in cui una delle due variabili sia uno scalare, che può essere sottratto ad una matrice di qualsiasi dimensione.

[10] Traduzione:
??? Errore nell'uso di $==>\ +$
Le dimensioni delle matrici devono coincidere.

Esempio 75 $\gg C - F$

$ans =$

$$\begin{matrix} 3.0000 & 2.0000 - 1.0000i \\ 0 + 1.0000i & 4.0000 \end{matrix}$$

- $*$ **moltiplicazione**.

Se A e B sono scalari, $A * B$ restituisce il prodotto dei due numeri.
Se A (oppure B) è uno scalare e B (oppure A) è una matrice, $A * B$ restituisce la matrice i cui elementi sono il prodotto degli elementi della matrice assegnata per lo scalare.
Se A e B sono matrici $\left(A = \{a_{ij}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,k}} \text{ e } B = \{b_{ij}\}_{\substack{i=1,\ldots,k \\ j=1,\ldots,m}}\right)$, il risultato del prodotto $A * B$ è dato dalla matrice C=$\{c_{ij}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}}$ i cui elementi sono $c_{ij} = \sum_{t=1}^{k} a_{it} * b_{tj} \quad \forall \begin{array}{c} i=1,\ldots,n \\ j=1,\ldots,m \end{array}$, come è noto dall'algebra lineare.

Esempio 76 $\gg 2 * 3$

$ans =$

$$6$$

Esempio 77 $\gg C * 2$

$ans =$

$$\begin{matrix} 2 & 4 \\ 6 & 8 \end{matrix}$$

Esempio 78 $\gg C * F$

$ans =$

$$\begin{matrix} 4.0000 - 2.0000i & 0 + 1.0000i \\ 6.0000 - 4.0000i & 0 + 3.0000i \end{matrix}$$

- $/$ **divisione a destra**.

Se A e B sono scalari, A/B restituisce il quoziente dei due numeri.
Se A è una matrice e B è uno scalare, A/B restituisce la matrice i cui elementi sono il quoziente degli elementi della matrice assegnata per lo scalare.
Se A e B sono matrici quadrate delle stesse dimensioni, eseguire il rapporto A/B equivale ad eseguire il prodotto $A * inv(B)$ (dove $inv(B)$ è l'inversa della matrice B).

Esempio 79 $\gg 10/5$

$ans =$

$$2$$

Esempio 80 $\gg C/2$

$ans =$

$$\begin{matrix} 0.5000 & 1.0000 \\ 1.5000 & 2.0000 \end{matrix}$$

Esempio 81 $\gg C/F$

$ans =$

$$\begin{matrix} 0- & 2.0000i & 0.7000- & 1.1000i \\ 0- & 4.0000i & 1.7000- & 2.1000i \end{matrix}$$

- $\backslash$ **divisione a sinistra**.
  Il simbolo precedente denota l'operatore della divisione a destra a cui siamo abituati: eseguire A/B significa dividere A per B.
  Il simbolo $\backslash$, invece, denota la divisione a sinistra: scrivere $A\backslash B$ equivale a scrivere B/A, cioè a dividere B per A.
  Se A e B sono scalari, $A\backslash B$ restituisce il quoziente di B per A.
  Se A è uno scalare e B una matrice $A\backslash B$ restituisce la matrice i cui elementi si ottengono dividendo gli elementi di B per lo scalare A.
  Se A e B sono matrici quadrate delle stesse dimensioni, eseguire $A\backslash B$ equivale ad effettuare il prodotto $inv(A) * B$ (dove $inv(A)$ è l'inversa della matrice A).
  Se A è una matrice $n \times n$ e B è un vettore colonna con n componenti, oppure una matrice con lo stesso numero di colonne, allora $X = A\backslash B$ è la soluzione dell'equazione $AX = B$, calcolata con il metodo di eliminazione di Gauss. Se A è mal condizionata [11], oppure è molto vicina ad essere singolare [12], MATLAB lancia un messaggio di warning.
  Se A è una matrice $m \times n$ con $m > n$ e B è un vettore colonna con m componenti, o una matrice con lo stesso numero di colonne, allora $X = A\backslash B$ è la soluzione nel senso dei minimi quadrati del sistema di equazioni $AX = B$ con numero di equazioni maggiore del numero delle incognite.

[11] Parleremo del numero di condizionamento nel paragrafo ... . Per ora accenniamo che una matrice è tanto meglio condizionata quanto più il suo numero di condizionamento si avvicina ad 1.

[12] Si dice *singolare* una matrice il cui determinante è nullo o molto prossimo a 0.

Il rango effettivo di A è determinato con il metodo di fattorizzazione QR con pivoting.

Esempio 82 $\gg 10\backslash 5$

$ans =$

$$0.5000$$

Esempio 83 $\gg 2\backslash C$

$ans =$

$$\begin{matrix} 0.5000 & 1.0000 \\ 1.5000 & 2.0000 \end{matrix}$$

Esempio 84 $\gg C\backslash F$

$ans =$

$$\begin{matrix} 7.0000 - \ 1.0000i & 0 - \ 2.0000i \\ -4.5000 + \ 0.5000i & 0 + \ 1.5000i \end{matrix}$$

- ^ **elevamento a potenza**.

X^p è X elevato alla potenza p, se p è uno scalare. Se p è un intero, la potenza viene calcolata mediante moltiplicazioni ripetute p volte dell'elemento X. Se l'intero è negativo, X viene prima invertito, poi moltiplicato p volte. Per altri valori di p, il calcolo coinvolge autovalori ed autovettori: se X è diagonalizzabile e $[V, D] = eig(X)$, allora X^$p = V * D$^p/V.

Se X è uno scalare e P una matrice, X^P è X elevato alla matrice P usando gli autovalori e gli autovettori.

X^P, con X e P matrici, è un errore.

Esempio 85 $\gg C$^2

$ans =$

$$\begin{matrix} 7 & 10 \\ 15 & 22 \end{matrix}$$

Esempio 86 $\gg C$^-3

$ans =$

$$\begin{matrix} -14.7500 & 6.7500 \\ 10.1250 & -4.6250 \end{matrix}$$

Esempio 87 $\gg C\hat{}1.7$

$ans =$

$$\begin{array}{rr} 4.2469 - 0.1148i & 6.0298 + 0.0525i \\ 9.0447 + 0.0788i & 13.2916 - 0.0360i \end{array}$$

Esempio 88 $\gg 3\hat{}C$

$ans =$

$$\begin{array}{rr} 87.8864 & 127.1198 \\ 190.6797 & 278.5661 \end{array}$$

Esempio 89 $\gg C\hat{}C$

??? Error using $==>$ ^

Matrix dimensions must agree.

- *sqrt* **radice quadrata**.

  $sqrt(A)$ restituisce la radice quadrata di tutti gli elementi dell'array A. Per gli elementi dell'array che sono negativi o complessi, $sqrt(A)$ restituisce risultati complessi.

Esempio 90 $\gg sqrt(F)$

$ans =$

$$\begin{array}{rr} 0 + 1.4142i & 0.7071 + 0.7071i \\ 1.7553 - 0.2848i & 0 \end{array}$$

- $'$ **complesso coniugato trasposto**.

  Utilizzando la scrittura A' si ottiene la matrice trasposta coniugata di A.

  Se A è una variabile complessa, per eseguire una semplice trasposizione senza coniugazione bisogna usare l'operatore $.'$: $A.'$ restituisce la matrice trasposta di A.

Esempio 91 *Utilizzando la scrittura* $\gg A'$ *si ottiene la matrice trasposta coniugata di* A*, cioè:*

$ans =$

$$\begin{array}{rrrr} 16 & 5 & 9 & 4 \\ 3 & 10 & 6 & 15 \\ 2 & 11 & 7 & 14 \\ 13 & 8 & 12 & 1 \end{array}$$

Utilizzando l'istruzione $\gg A.'$ *si ottiene lo stesso output, essendo* A *una matrice reale.*

Esempio 92 *Utilizzando la scrittura* $\gg E'$ *si ottiene la matrice trasposta coniugata di* E*, cioè:*

$ans =$

$$\begin{array}{rr} 10.0000 & 0 - 2.0000i \\ 3.0000 - 4.0000i & 8.0000 \\ 0 - 12.0000i & -6.0000 \\ -7.0000 - 1.0000i & -6.0000 + 1.0000i \end{array}$$

invece utilizzando la scrittura $\gg E.'$ *si ottiene la matrice trasposta di* E*, cioè:*

$ans =$

$$\begin{array}{rr} 10.0000 & 0 + 2.0000i \\ 3.0000 + 4.0000i & 8.0000 \\ 0 + 12.0000i & -6.0000 \\ -7.0000 + 1.0000i & -6.0000 - 1.0000i \end{array}$$

- ( ) **priorità nelle operazioni**.
  Le parentesi tonde specificano l'ordine secondo cui un'espressione deve essere valutata (a tal fine non si usa nessun altro tipo di parentesi).

  Esempio 93 *L'istruzione* $\gg (x{+}y){*}z$ *è ovviamente diversa dall'istruzione* $\gg x + (y * z)$.

Esercizio 15 *Verificare con esempi che sommando una matrice e la sua trasposta, si ottiene una matrice simmetrica.*
Soluzione.

Esercizio 16 *Verificare con esempi che moltiplicando una matrice e la sua trasposta, si ottiene una matrice simmetrica.*
Soluzione.

3.8 Operazioni elemento per elemento

Gli operatori analizzati nel paragrafo precedente sono quelli che obbediscono alle leggi dell'algebra lineare, quindi, ad esempio, l'operatore $*$ esegue il prodotto scalare tra vettori: considerati i vettori $x = (x_1 \ \ x_2 \ \ x_3)$ e $y = (y_1 \ \ y_2 \ \ y_3)^T$, x*y restituisce il prodotto scalare $x_1y_1 + x_2y_2 + x_3y_3$.
MATLAB dispone anche di un altro tipo di operazioni aritmetiche: le operazioni elemento per elemento. Il carattere **.** distingue i due tipi di operazioni.

Volendo ottenere, ad esempio, un vettore i cui elementi siano i prodotti degli elementi omologhi di due vettori, si deve premettere il carattere **.** all'operatore $*$ che si sta utilizzando. Questo vuol dire che, considerando i due vettori x ed y precedenti, $x.*y$ restituisce il vettore $(x_1y_1\ , x_2y_2\ , x_3y_3)$.
In generale, il prodotto $A.*B$ restituisce la matrice i cui elementi sono dati dai prodotti $A_{ij}*B_{ij}$. A e B devono avere le stesse dimensioni, tranne nel caso in cui una delle due variabili sia uno scalare.

Esempio 94 $\gg C.*3$

$ans =$

3	6
9	12

Osservazione 20 Se M è una matrice ed s uno scalare (come nell'esempio precedente) le operazioni $M*s$ ed $M.*s$ restituiscono lo stesso risultato.

Esempio 95 $\gg C.*C$

$ans =$

1	4
9	16

Analogamente sono definiti gli operatori ./ e .\ che eseguono rispettivamente la divisione a destra e la divisione a sinistra fra gli elementi delle variabili A e B (che devono avere le stesse dimensioni, tranne nel caso in cui una delle due variabili sia uno scalare), cioè costruiscono le matrici i cui elementi sono rispettivamente A_{ij}/B_{ij} e $A_{ij}\backslash B_{ij}$.

Esempio 96 $\gg C./2$

$ans =$

0.5000	1.0000
1.5000	2.0000

Osservazione 21 Se M è una matrice ed s uno scalare (come nell'esempio precedente) le operazioni M/s ed $M./s$ restituiscono lo stesso risultato.
Per quel che riguarda l'operatore \ si deve osservare che non è possibile eseguire l'operazione $M\backslash s$, in quanto MATLAB restituisce il seguente messaggio d'errore:
??? Error using ==> \
Matrix dimensions must agree. [13]

[13] Traduzione:
??? Errore nell'uso di == > \
Le dimensioni delle matrici devono essere compatibili.

Esempio 97 $\gg C./G$

$ans =$

$$\begin{matrix} 0.5000 & -1.0000 \\ 1.0000 & 0.5000 \end{matrix}$$

Esempio 98 $\gg C.\backslash G$

$ans =$

$$\begin{matrix} 2 & -1 \\ 1 & 2 \end{matrix}$$

L'operazione $A.\hat{}B$ (con A e B aventi le stesse dimensioni, tranne nel caso in cui una delle due variabili sia uno scalare) eleva gli elementi A_{ij} ai corrispondenti B_{ij} .

Esempio 99 $\gg C.\hat{}3$

$ans =$

$$\begin{matrix} 1 & 8 \\ 27 & 64 \end{matrix}$$

Osservazione 22 Se M è una matrice ed s uno scalare (come nell'esempio precedente) le operazioni $M\hat{}s$ ed $M.\hat{}s$ non restituiscono lo stesso risultato: $M\hat{}s$ rappresenta il prodotto $M * M * \ldots * M$ (cioè M moltiplicata p volte per se stessa), mentre $M.\hat{}s$ restituisce la matrice i cui elementi sono $(M_{ij})^{\wedge s}$, cioè gli elementi di M elevati alla potenza s.

Esempio 100 $\gg G.\hat{}C$

$ans =$

$$\begin{matrix} 2 & 4 \\ 27 & 4096 \end{matrix}$$

Poichè le operazioni di addizione e sottrazione dell'algebra lineare coincidono con quelle eseguite elemento per elemento, i simboli $.+$ e $.-$ hanno lo stesso significato di $+$ e di $-$ e quindi non ha senso usarli.

3.9 Comandi su una matrice

In questo paragrafo sono elencati alcuni dei comandi più utilizzati in MATLAB per operare su una matrice e vengono introdotti vari esempi che mostrano il modo per adoperare questi comandi.
Molti comandi, che normalmente operano su vettori, quando vengono applicati a matrici, operano sulle colonne di queste.

- **size**.
  L'istruzione $size(A)$ applicata alla matrice A di dimensioni $m \times n$ restituisce il vettore riga di due elementi $[m, n]$ contenente il numero m di righe e il numero n di colonne della matrice A.
  L'istruzione $[m, n] = size(A)$ restituisce il numero di righe e di colonne della matrice A nel vettore assegnato $[m, n]$.
  L'istruzione $size(A, dim)$ restituisce la lunghezza della dimensione specificata dallo scalare dim dell'array A.
  Applicando l'istruzione $[M_1, M_2, \ldots, M_N] = size(A)$ all'array A con N indici si ottiene in output il vettore delle grandezze delle varie dimensioni M_1, M_2, $\ldots$, M_N di A.

  Esempio 101 $\gg size(A)$
  $ans =$

  4 4

  Esempio 102 $\gg size(B)$
  $ans =$

  2 5

  Esempio 103 $\gg size(B, 1)$
  $ans =$

  2

  Esempio 104 $\gg size(B, 2)$
  $ans =$

  5

  Esempio 105 *Vedremo nel paragrafo Alcune matrici particolari che il comando* $zeros(m, n, p, \ldots)$ *costruisce array di dimensione* $m \times n \times p \times \ldots$ *con elementi tutti uguali a* 0*; costruiamo l'array* z *di dimensione* $2 \times 3 \times 4$ *con elementi tutti uguali a* 0*:*
  $\gg z = zeros(2, 3, 4)$
  $z(:, :, 1) =$
  0 0 0
  0 0 0

$z(:,:,2) =$

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

$z(:,:,3) =$

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

$z(:,:,4) =$

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Lanciando $\gg size(z)$ *si ottiene:*

$ans =$

$$\begin{matrix} 2 & 3 & 4 \end{matrix}$$

Esempio 106 $\gg size(x)$

$ans =$

$$\begin{matrix} 1 & 5 \end{matrix}$$

Esempio 107 $\gg size(c)$

$ans =$

$$\begin{matrix} 4 & 1 \end{matrix}$$

- **length**.
  L'istruzione $length(X)$ fornisce la lunghezza del vettore X, ossia il numero dei suoi elementi. Come si vede dagli ultimi due esempi precedenti, è inutile calcolare il numero degli elementi di un vettore con l'istruzione $size$: mediante l'istruzione $length$ si ha un risultato più immediato.

  Esempio 108 $\gg length(x)$

  $ans =$

  $$5$$

  Esempio 109 $\gg length(c)$

  $ans =$

  $$4$$

- **max** e **min**.

  L'istruzione $max(X)$ $[min(X)]$ dà il massimo [minimo] elemento del vettore X; se X è una matrice $max(X)$ $[min(X)]$ restituisce un vettore che contiene i valori massimi [minimi] per ogni colonna della matrice X.
  Utilizzando l'istruzione con la sintassi completa $[ymax, imax] = max(X)$ si ottiene in $ymax$ il massimo del vettore X (oppure il vettore dei massimi delle colonne nel caso in cui X sia una matrice) e in $imax$ l'indice (oppure il vettore degli indici nel caso matriciale) in cui viene raggiunto il massimo.
  Analogamente $[ymin, imin] = min(X)$ è la sintassi completa relativamente alla funzione min.

  Esempio 110 *Consideriamo il vettore y:*

  $\gg ymax = max(y)$
  $ymax =$
  $\quad 10$
  $\gg [ymax, imax] = max(y)$
  $ymax =$
  $\quad 10$
  $imax =$
  $\quad 2$

  $\gg ymin = min(y)$
  $ymin =$
  $\quad -5$
  $\gg [ymin, imin] = min(y)$
  $ymin =$
  $\quad -5$
  $imin =$
  $\quad 3$

  Esempio 111 *Consideriamo la matrice A:*

  $\gg ymax = max(A)$
  $ymax =$
  $\quad 16 \quad 15 \quad 14 \quad 13$
  $\gg [ymax, imax] = max(A)$
  $ymax =$
  $\quad 16 \quad 15 \quad 14 \quad 13$
  $imax =$
  $\quad 1 \quad 4 \quad 4 \quad 1$

  $\gg ymin = min(A)$
  $ymin =$
  $\quad 4 \quad 3 \quad 2 \quad 1$
  $\gg [ymin, imin] = min(A)$
  $ymin =$
  $\quad 4 \quad 3 \quad 2 \quad 1$
  $imin =$
  $\quad 4 \quad 1 \quad 1 \quad 4$

- **sort**.

  L'istruzione $sort(x)$ ordina in maniera crescente gli elementi del vettore x. Nel caso in cui x sia una matrice l'istruzione $sort(x)$ opera similmente sulle singole colonne della matrice x.

La sintassi completa è $[y, i] = sort(x)$ che consente di ottenere nel vettore y gli elementi di x ordinati in maniera crescente e nel vettore i gli indici del vettore x tali che $x(i)$ fornisca il vettore ordinato y.
Analogamente su matrici $[y, i] = sort(x)$ ordina per colonne la matrice x e restituisce la matrice di permutazioni i.
Sono consentiti elementi reali, complessi e stringhe.
Quando ci sono due elementi uguali, la locazione nell'array di input determina la locazione nell'array ordinato.
Quando x ha elementi complessi, l'ordine viene stabilito in base al modulo e, se ci sono moduli uguali, gli elementi vengono ordinati rispetto all'angolo di fase sull'intervallo $[-\pi, \pi]$.
Se x comprende elementi uguali a NaN, il comando *sort* li posiziona alla fine dell'array di output.

Esempio 112 *Consideriamo il vettore* y*:*
$\gg yord = sort(y)$
$yord =$

$$\begin{matrix} -5 & 1 & 6 & 7 & 10 \end{matrix}$$

$\gg [yord, i] = sort(y)$
$yord =$

$$\begin{matrix} -5 & 1 & 6 & 7 & 10 \end{matrix}$$

$i =$

$$\begin{matrix} 3 & 1 & 4 & 5 & 2 \end{matrix}$$

Esempio 113 *Consideriamo la matrice* A*:*
$\gg yord = sort(A)$
$yord =$

$$\begin{matrix} 4 & 3 & 2 & 1 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 16 & 15 & 14 & 13 \end{matrix}$$

$\gg [yord, i] = sort(A)$
$yord =$

$$\begin{matrix} 4 & 3 & 2 & 1 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 16 & 15 & 14 & 13 \end{matrix}$$

$i =$

4	1	1	4
2	3	3	2
3	2	2	3
1	4	4	1

- **sum**.
  L'istruzione *sum* fornisce la somma degli elementi dell'oggetto che si sta considerando: per i vettori il comando $sum(X)$ calcola la somma degli elementi del vettore X; per le matrici $sum(X)$ è un vettore riga i cui elementi rappresentano la somma di ognuna delle colonne della matrice X; per gli array X di dimensione n l'istruzione $sum(X, i)$ (con $1 \leq i \leq n$) effettua la somma lungo la dimensione i-esima dell'array X.
  Per default $sum(X) = sum(X, 1)$.

Esempio 114 $\gg sum(x)$

$ans =$

15

Esempio 115 $\gg sum(A)$

$ans =$

34 34 34 34

Osservazione 23 MATLAB preferisce lavorare con le colonne delle matrici, anziché con le righe, quindi il modo più semplice per ottenere la somma degli elementi delle righe della matrice A è quello di lanciare l'istruzione $\gg sum(A, 2)$, che effettua la somma degli elementi di A sulla seconda dimensione, cioè sulle colonne.

Osservazione 24 Un'alternativa al procedimento precedente è quella di considerare la somma degli elementi delle colonne della matrice trasposta di A e quindi fare il trasposto del risultato: con l'istruzione $\gg sum(A')$ si ottiene:

$ans =$

34 34 34 34

Quindi la somma degli elementi delle righe di A si ottiene mediante l'istruzione $\gg ans\,'$ che restituisce:

$ans =$

34

34

34

34

Osservazione 25 La matrice A dicesi **quadrato magico**.
Si definisce *quadrato magico* una matrice $n \times n$ che ha come elementi numeri interi compresi tra 1 e n^2 e che ha le somme degli elementi di ogni riga, di ogni colonna, della diagonale principale e dell'antidiagonale uguali ad un unico valore.
La funzione che permette di costruire quadrati magici è *magic*.

- **mean**.
L'istruzione *mean* fornisce il valore medio degli elementi dell'oggetto che si sta considerando: per i vettori il comando $mean(X)$ calcola il valor medio degli elementi del vettore X; per le matrici $mean(X)$ è un vettore riga i cui elementi rappresentano il valor medio di ognuna delle colonne della matrice X.

Esempio 116 $\gg mean(y)$
$ans =$

3.8000

Esempio 117 $\gg mean(A)$
$ans =$

8.5000 8.5000 8.5000 8.5000

Esempio 118 $\gg mean(B)$
$ans =$

3.5000 4.5000 5.5000 6.5000 7.5000

Esempio 119 $\gg mean(E)$
$ans =$

$5.0000 + 1.0000i \quad 5.5000 + 2.0000i \quad -3.0000 + 6.0000i \quad -6.5000$

- **diag**.

  L'istruzione *diag* restituisce matrici diagonali oppure la diagonale di una matrice a seconda che si stiano considerando vettori oppure matrici:

  - definito il vettore v, l'istruzione $diag(v)$ restituisce una matrice diagonale i cui elementi sono gli elementi del vettore v;
  - se v è un vettore di lunghezza n, l'istruzione $diag(v,k)$ restituisce una matrice quadrata di ordine $n+|k|$ con gli elementi di v sulla k-esima diagonale. Se $k=0$ si torna al caso precedente (cioè gli elementi del vettore vengono memorizzati nella diagonale principale della matrice di output), se $k>0$ gli elementi del vettore vengono memorizzati al di sopra della diagonale principale della matrice di output, se $k<0$ gli elementi del vettore vengono memorizzati al di sotto della diagonale principale della matrice di output;
  - l'istruzione $diag(A)$, dove A è una matrice, restituisce il vettore colonna i cui elementi sono gli elementi della diagonale principale di A;
  - l'istruzione $diag(A,k)$ restituisce il vettore colonna i cui elementi sono gli elementi della k-esima diagonale di A.

Esempio 120 $\gg diag(x)$

$ans =$

$$\begin{array}{ccccc}
1 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 0 & 5
\end{array}$$

Esempio 121 $\gg diag(x,2)$

$ans =$

$$\begin{array}{ccccccc}
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Esempio 122 $\gg diag(A)$
$ans =$

16

10

7

1

Esempio 123 $\gg diag(A, 2)$
$ans =$

2

8

- **rank**.
  L'istruzione *rank* calcola il rango di una matrice, ovvero il massimo numero di righe o di colonne linearmente indipendenti. Ci sono vari modi per calcolare il rango di una matrice; MATLAB usa il metodo basato sulla decomposizione in valori singolari, che, pur essendo l'algoritmo che impiega più tempo, è anche l'algoritmo più stabile.

  Esempio 124 $\gg rank(A)$
  $ans =$

  3

  Esempio 125 $\gg rank(B)$
  $ans =$

  2

  Esempio 126 $\gg rank(E)$
  $ans =$

  2

- **det**.
  L'istruzione $det(A)$ fornisce il determinante della matrice A. Il determinante viene calcolato mediante la fattorizzazione in matrici triangolari che si ottiene con il metodo di eliminazione di Gauss ed è nullo se la matrice è singolare.

Esempio 127 $\gg det(A)$

$ans =$

$$0$$

Osservazione 26 Ovviamente la matrice della quale si intende calcolare il determinante deve essere quadrata, altrimenti MATLAB segnala l'errore.

Esempio 128 $\gg det(B)$

??? Error using $==>$ *det*

Matrix must be square. [14]

- **inv**.

  Con l'istruzione $inv(X)$ si ottiene l'inversa della matrice quadrata X, cioè la matrice Y tale che $X * Y = I = Y * X$ (dove I rappresenta l'identità di ordine pari alle dimensioni di X).

  Esempio 129 $\gg Y = inv(C)$

  $Y =$

  $$\begin{matrix} -2.0000 & 1.0000 \\ 1.5000 & -0.5000 \end{matrix}$$

  Risulta che:

  $\gg C * Y$

  $ans =$

  $$\begin{matrix} 1.0000 & 0 \\ 0.0000 & 1.0000 \end{matrix}$$

  e

  $\gg Y * C$

  $ans =$

  $$\begin{matrix} 1.0000 & 0 \\ 0.0000 & 1.0000 \end{matrix}$$

  cioè il prodotto di C per la sua inversa e dell'inversa di C per C restituisce la matrice identica di ordine 2.

  Osservazione 27 Lo stesso risultato che si ottiene con $\gg inv(X)$ si può ottenere mediante l'istruzione $\gg X\hat{}(-1)$.

[14]Traduzione:
??? Errore nell'uso di $==>$ *det*
La matrice deve essere quadrata.

Se la matrice della quale si vuole calcolare l'inversa è singolare, MATLAB lancia il seguente messaggio d'errore:

Warning: Matrix is singular to working precision. [15]

Su macchine con aritmetica IEEE, questo è solo un messaggio di warning, ma l'inversa viene calcolata: si ottiene una matrice con tutti gli elementi uguali a Inf.
Su macchine prive dell'aritmetica IEEE, come il VAX, questa operazione viene trattata come un vero e proprio errore.

Esempio 130 $\gg inv(I)$
Warning: Matrix is singular to working precision.
$ans =$

$$\begin{matrix} Inf & Inf & Inf \\ Inf & Inf & Inf \\ Inf & Inf & Inf \end{matrix}$$

Se l'inversa viene calcolata, ma non è accurata, viene visualizzato questo messaggio, prima della matrice:

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. $RCOND = xxx$. [16]

L'errore di arrotondamento ha impedito all'algoritmo che inverte la matrice di individuare l'esatta singolarità, ma il valore $rcond$, che rappresenta una stima del reciproco del numero di condizionamento, è dell'ordine di eps, la precisione di macchina, cosicché l'inversa calcolata non è molto utile.

Esempio 131 *La matrice* A *è singolare, infatti:*
$\gg det(A)$
$ans =$

[15] Traduzione:
Attenzione: la matrice è singolare per la precisione di macchina.

[16] Traduzione:
Attenzione: la matrice è prossima ad essere singolare o mal condizionata.
I risultati potrebbero essere imprecisi. $RCOND = xxx$.

0

quindi, calcolando l'inversa di A non si ottengono risultati accurati:

$\gg inv(A)$

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. $RCOND = 1.175530e-017$.

$ans =$

$1.0e+015\ *$

$$\begin{array}{rrrr} 0.1251 & 0.3753 & -0.3753 & -0.1251 \\ -0.3753 & -1.1259 & 1.1259 & 0.3753 \\ 0.3753 & 1.1259 & -1.1259 & -0.3753 \\ -0.1251 & -0.3753 & 0.3753 & 0.1251 \end{array}$$

Esercizio 17 *Si verifichi che, moltiplicando una matrice singolare per la sua inversa, non si ottengono risultati attendibili e si osservi come il risultato si discosta dalla matrice identica (si utilizzi la matrice A).*
Soluzione.

Esercizio 18 *Risolvere il sistema lineare di equazioni:*

$$\begin{cases} 2x_1 + 5x_2 - 4x_3 + 5x_4 = 3 \\ 5x_2 - x_3 + 4x_4 = 2 \\ 7x_1 + 3x_2 + 5x_3 - 2x_4 = -7 \\ -x_1 + 3x_3 + 6x_4 = 4 \end{cases}$$

Soluzione.

- **poly**.
  Se A è una matrice $n \times n$, il comando $poly(A)$ restituisce un vettore riga di lunghezza $n+1$, i cui elementi rappresentano i coefficienti del polinomio caratteristico della matrice quadrata A, $p(\lambda) = det(\lambda I - A)$ (dove I rappresenta la matrice identica delle stesse dimensioni di A).
  I coefficienti sono ordinati secondo le potenze decrescenti; questo significa che il polinomio caratteristico della matrice A è dato da
  $\lambda^n + p_{n-1} * \lambda^{n-1} + \ldots + p_1 * \lambda + p_0$.
  Vedremo in seguito che la stessa istruzione applicata ad un vettore ha un significato completamente diverso.

Esempio 132 $\gg poly(C)$

$ans =$

$$\begin{array}{rrr} 1.0000 & -5.0000 & -2.0000 \end{array}$$

Esercizio 19 *Si calcolino i coefficienti di $p(\lambda)$, polinomio caratteristico della matrice G, e si dica qual è la sua espressione.*
Soluzione.

- **trace**.
L'istruzione $trace(A)$ fornisce la traccia (cioè la somma degli elementi della diagonale principale) della matrice A.

Esempio 133 $\gg trace(A)$
$ans =$

34

Esempio 134 $\gg trace(B)$
$ans =$

8

Esempio 135 $\gg trace(E)$
$ans =$

18

- **norm**.
Il comando *norm* calcola la norma di un vettore o di una matrice. La sintassi completa è $norm(X, Argomento)$, dove $Argomento$ può assumere i seguenti valori:
  - **1** per calcolare la norma 1 di X, cioè $||X||_1 = \sum_{i=1}^{n} |x_i|$ nel caso in cui X è un vettore ($X = \{x_i\}_{i=1,\ldots,n}$) o $||X||_1 = \max_{j=1,\ldots,m} \sum_{i=1}^{n} |x_{ij}|$ nel caso in cui X è una matrice $\left(X = \{x_{ij}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}}\right)$;
  - **2** per calcolare la norma 2 di X, cioè $||X||_2 = \sqrt{X^H X} = \sqrt{\sum_{i=1}^{n} |x_i|^2}$ se X è un vettore ($X = \{x_i\}_{i=1,\ldots,n}$) oppure $||X||_2 = \sqrt{\rho^{17}(X^H X)}$ se X è una matrice $\left(X = \{x_{ij}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}}\right)$;

[17]Con $\rho(A)$ si denota il raggio spettrale della matrice A, cioè l'autovalore di modulo massimo di A.

– **p** (con p generico numero complesso) per calcolare la norma p di X, cioè $||X||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$ nel caso in cui X è un vettore ($X = \{x_i\}_{i=1,\dots,n}$); se X è una matrice gli unici valori di p disponibili sono 1 e 2, che abbiamo già analizzato;

– **inf** (o equivalentemente **′inf′**) per calcolare la norma infinito di X, cioè $||X||_\infty = \max_{i=1,\dots,n} |x_i|$ se X è un vettore ($X = \{x_i\}_{\ i=1,\dots,n}$) oppure $||X||_\infty = \max_{i=1,\dots,n} \sum_{j=1}^{m} |x_{ij}|$ nel caso in cui X è una matrice $\left(X = \{x_{ij}\}_{\substack{i = 1, \dots, n \\ j = 1, \dots, m}} \right)$;

– **′fro′** per calcolare la norma di Frobenius di $X = \{x_{ij}\}_{\substack{i = 1, \dots, n \\ j = 1, \dots, m}}$ cioè $||X||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |x_{ij}|^2} = \sqrt{tr^{18}(A^H A)}.$

Osservazione 28 Se *Argomento* non viene specificato MATLAB calcola, per default, la norma euclidea, quindi scrivere $norm(x)$ equivale a scrivere $norm(x, 2)$.

Esempio 136 *Consideriamo il vettore* x*:*

$\gg norm(x, 1)$

$ans =$

15

$\gg norm(x, 2)$ *oppure, equivalentemente,* $\gg norm(x)$

$ans =$

7.4162

$\gg norm(x, 3)$

$ans =$

6.0822

$\gg norm(x, 3.6)$

$ans =$

5.7431

[18] Con $tr(A)$ si denota la traccia della matrice A, cioè la somma degli elementi della diagonale principale di A.

$\gg norm(x, 3.6 + 2 * i)$
$ans =$

5.7431

$\gg norm(x, inf)$ *oppure, equivalentemente,* $\gg norm(x, \ 'inf\ ' \ )$
$ans =$

5

$\gg norm(x, \ 'fro\ ' \ )$
$ans =$

7.4162

Esempio 137 *Consideriamo la matrice* A*:*
$\gg norm(A, 1)$
$ans =$

34

$\gg norm(A, 2)$ *oppure, equivalentemente,* $\gg norm(A)$
$ans =$

34.0000

$\gg norm(A, inf)$ *oppure, equivalentemente,* $\gg norm(A, \ 'inf\ ' \ )$
$ans =$

34

$\gg norm(A, \ 'fro\ ' \ )$
$ans =$

38.6782

- **kron**.
  Con l'istruzione $kron(A, B)$ si calcola il prodotto tensoriale di Kronecker di A e B (che possono essere matrici oppure vettori), ovvero la matrice il cui blocco ij-esimo è pari al prodotto dell'elemento di posto ij di A per B, cioè $(A \otimes B)_{ij} = A_{ij} * B$.

Esempio 138 $\gg kron(C, D)$
$ans =$

$$\begin{matrix} 0 & 1 & 1 & 0 & 2 & 2 \\ -1 & 2 & 0 & -2 & 4 & 0 \\ 0 & 3 & 3 & 0 & 4 & 4 \\ -3 & 6 & 0 & -4 & 8 & 0 \end{matrix}$$

- **fliplr** e **flipud**.

  L'istruzione $fliplr$, creata inizialmente per uso grafico, serve per invertire la direzione della matrice cui viene applicata da sinistra a destra. Analogamente, $flipud$ serve per invertire la direzione della matrice cui viene applicata dall'alto verso il basso.

  Esempio 139 *Consideriamo la matrice* B*: l'istruzione* $\gg fliplr(B)$ *restituisce:*

  $ans =$

5	4	3	2	1
10	9	8	7	6

  mentre l'istruzione $\gg flipud(B)$ *restituisce:*

  $ans =$

6	7	8	9	10
1	2	3	4	5

  Esercizio 20 *Si verifichi che, se* A *è una matrice quadrata, considerando la matrice di permutazione*

  $$P = \begin{pmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 1 & \dots & 0 & 0 \\ 1 & 0 & \dots & 0 & 0 \end{pmatrix}_{n \times n}$$

  dove $[n,n] = size(A)$*, risulta* $fliplr(A) = A * P$ *e* $flipud(A) = P * A$*.*
  Soluzione.

- **rot90**.

  L'istruzione $rot90(A)$ ruota la matrice A di $90°$. Volendo, si può ruotare la matrice di $(90k)°$ con il comando $rot90(A,k)$.

  Esempio 140 $\gg rot90(A)$

  $ans =$

13	8	12	1
2	11	7	14
3	10	6	15
16	5	9	4

Esempio 141 $\gg rot90(A, 3)$

$ans =$

4	9	5	16
15	6	10	3
14	7	11	2
1	12	8	13

- **magic**.
  Un quadrato magico di dimensione n si ottiene con l'istruzione $magic(n)$.

Esempio 142 $\gg magic(3)$

$ans =$

8	1	6
3	5	7
4	9	2

Esempio 143 $\gg magic(4)$

$ans =$

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

- **compan**.
  Il comando $compan(p)$, con p vettore ($p = \{p_i\}_{i=1,\ldots,n}$), genera la matrice di Frobenius associata a p, cioè la matrice di dimensioni $n \times n$ che ha questa forma:

$$\begin{pmatrix} -\frac{p_2}{p_1} & -\frac{p_3}{p_1} & -\frac{p_4}{p_1} & \ldots & -\frac{p_{n-1}}{p_1} & -\frac{p_n}{p_1} \\ 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{pmatrix}$$

Osservazione 29 Il polinomio caratteristico della matrice $compan(p)$ è il vettore $\frac{1}{p_1} * p$.

Esempio 144 $\gg compan(c)$

$ans =$

-2	-3	-4
1	0	0
0	1	0

- **triu** e **tril**.

  L'istruzione $triu(X, k)$ preleva la parte triangolare superiore della matrice X a partire dalla k-esima diagonale (come nel caso dell'istruzione $diag$, se $k > 0$ viene considerata una sovradiagonale, se $k < 0$ viene considerata una sottodiagonale).

  Analogamente opera il comando $tril(X, k)$, che considera la parte inferiore della matrice X a partire dalla k-esima diagonale.

Esempio 145 $\gg triu(A, 2)$

$ans =$

0	0	2	13
0	0	0	8
0	0	0	0
0	0	0	0

Esempio 146 $\gg triu(A, -2)$

$ans =$

16	3	2	13
5	10	11	8
9	6	7	12
0	15	14	1

Esempio 147 $\gg tril(A, 2)$

$ans =$

16	3	2	0
5	10	11	8
9	6	7	12
4	15	14	1

Esempio 148 $\gg tril(A, -2)$

$ans =$

0	0	0	0
0	0	0	0
9	0	0	0
4	15	0	0

- **reshape**.
  L'istruzione *reshape* permette di alterare la forma di un vettore oppure di una matrice: $reshape(A, m, n)$ genera una matrice $m \times n$ i cui elementi sono presi per colonne da A.
  Se il numero $m * n$ è diverso dal numero degli elementi di A, MATLAB visualizza un messaggio di errore.

  Esempio 149 $\gg reshape(c, 2, 2)$

  $ans =$

  $$\begin{matrix} 1 & 3 \\ 2 & 4 \end{matrix}$$

  Esempio 150 $\gg reshape(A, 8, 2)$

  $ans =$

  $$\begin{matrix} 16 & 2 \\ 5 & 11 \\ 9 & 7 \\ 4 & 14 \\ 3 & 13 \\ 10 & 8 \\ 6 & 12 \\ 15 & 1 \end{matrix}$$

  Esempio 151 $\gg reshape(A, 12, 8)$

  ??? Error using ==> reshape
  To RESHAPE the number of elements must not change. [19]

3.10 Funzioni esponenziali su matrici

Come abbiamo già visto, l'elemento caratterizzante MATLAB è la matrice: anche uno scalare è in realtà una matrice di dimensioni 1×1. Ciò spiega perché le operazioni scalari definite nei paragrafi precedenti sono estendibili a matrici. Ad esempio, considerata la matrice

[19] Traduzione:
??? Errore nell'uso di == > *reshape*
Per RIDIMENSIONARE il numero degli elementi deve restare immutato.

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ : & : & : \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$$

l'istruzione $\gg C = exp(A)$ restituisce la matrice

$$C = \begin{pmatrix} e^{a_{11}} & \dots & e^{a_{1n}} \\ : & : & : \\ e^{a_{m1}} & \dots & e^{a_{mn}} \end{pmatrix}$$

Osservazione 30 Questa matrice non è l'esponenziale della matrice A, che si definisce, invece, come $e^A = \sum_{i=0}^{\infty} \frac{A^n}{n!}$.

In questo paragrafo vengono elencate le funzioni esponenziali definite in senso matriciale proprio.

- **expm**.
  La funzione $expm(A)$ calcola e^A, cioè eleva la costante e alla matrice A. La funzione $expm$ è predefinita e usa un approssimante di Padé come indicato nel file expm1.m.
  Un secondo metodo per calcolare la matrice esponenziale consiste nell'utilizzare l'approssimazione in serie di Taylor. Questo metodo è dimostrato nel file expm2.m. L'approssimazione mediante la serie di Taylor non è un metodo consigliabile, in quanto essa è spesso lenta e poco precisa.
  Una terza via per calcolare la matrice esponenziale, che si trova nel file expm3.m, consiste nel diagonalizzare la matrice A ($[\ T, V\ ] = eig(A)$), applicare la funzione ai singoli autovalori e quindi trasformare nuovamente la matrice. Questo metodo fallisce se la matrice in input non ha autovettori linearmente indipendenti o se $cond(T)$ è molto grande.

  Esempio 152 $\gg expm(C)$

  $ans =$

51.9690	74.7366
112.1048	164.0738

  Osservazione 31 Osserviamo che l'istruzione $\gg expm(L)$ restituisce la matrice

  $ans =$

2.7183	1.7183	1.0862
0	1.0000	1.2642
0	0	0.3679

mentre l'istruzione $\gg exp(L)$ restituisce:

$ans =$

2.7183	2.7183	1.0000
1.0000	1.0000	7.3891
1.0000	1.0000	0.3679

Queste due matrici hanno gli elementi diagonali uguali; questo sarebbe stato verificato per una qualsiasi matrice triangolare. Tutti gli altri elementi, compresi quelli al di sotto della diagonale principale, sono differenti.

Osservazione 32 Ovviamente la matrice cui si applica *expm* deve essere quadrata, altrimenti MATLAB segnala l'errore.

Esempio 153 $\gg expm(B)$
??? Error using ==> expm
Matrix must be square. [20]

- **logm**.
  La funzione $logm(A)$ è l'inversa della funzione $expm(A)$: essa calcola il logaritmo della matrice cui viene applicata. Se la matrice A ha autovalori negativi i risultati sono complessi.
  Lanciata l'istruzione $B = logm(A)$, viene visualizzato un messaggio di warning nel caso in cui la matrice $expm(B)$ non è sufficientemente vicina ad A.
  Utilizzando l'istruzione $[B, esterr] = logm(A)$ non viene visualizzato nessun messaggio di warning, ma viene restituita una stima del residuo relativo, $\dfrac{norm(expm(B) - A)}{norm(A)}$.

  Osservazione 33 Se la matrice A è reale e simmetrica, oppure complessa e hermitiana [21], lo è anche la matrice $logm(A)$.

[20] Traduzione:
??? Errore nell'uso di == > *expm*
La matrice deve essere quadrata.

[21] Una matrice A si dice *hermitiana* se $A = A^H$, dove con A^H si denota la matrice trasposta coniugata di A.

Esempio 154 $\gg logm(C)$

$ans =$

$$\begin{matrix} -0.3504 + 2.3911i & 0.9294 - 1.0938i \\ 1.3940 - 1.6406i & 1.0436 + 0.7505i \end{matrix}$$

Osservazione 34 Ovviamente la matrice cui si applica *logm* deve essere quadrata, altrimenti MATLAB segnala l'errore.

Esempio 155 $\gg logm(B)$

??? Error using ==> schur

Matrix must be square. [22]

Error in ==> $C : \backslash MATLABR11\backslash toolbox\backslash matlab\backslash matfun\backslash funm.m$

On line 37 *==>* $[Q, T] = schur(A)$;

Error in ==> $C : \backslash MATLABR11\backslash toolbox\backslash matlab\backslash matfun\backslash logm.m$

On line 22 *==>* $[L, esterr] = funm(A,\ 'log\ ')$; [23]

Osservazione 35 Alcune matrici non hanno alcun logaritmo, reale o complesso, quindi *logm* non può restituire alcun risultato.

Osservazione 36 Per la maggior parte delle matrici risulta:
$logm(expm(X)) = X = expm(logm(X))$.
Questa identità non è valida per alcune X. Per esempio se fra gli autovalori di X è compreso uno zero esatto, allora $logm(X)$ genera infinito. Oppure, se gli elementi di X sono troppo grandi, $expm(X)$ può dare overflow.

- **sqrtm**.
  L'istruzione $sqrtm(A)$ applicata alla matrice definita positiva A, ne calcola la radice quadrata, ovvero la matrice B tale che $A = B*B$. Vengono restituiti risultati complessi se la matrice A ha autovalori negativi.
  Lanciata l'istruzione $B = sqrtm(A)$, viene visualizzato un messaggio di

[22] Traduzione:
??? Errore nell'uso di == > *schur*
La matrice deve essere quadrata.

[23] Queste linee indicano il path dei file in cui è stato commesso l'errore.

warning nel caso in cui $B * B$ non è sufficientemente vicina ad A.
Utilizzando l'istruzione $[B, esterr] = sqrtm(A)$ non viene visualizzato nessun messaggio di warning, ma viene restituita una stima del residuo relativo, $\dfrac{norm(B * B - A)}{norm(A)}$.
L'algoritmo usato da *sqrtm* è basato sulla decomposizione di Shur; esso può fallire in alcune situazioni in cui la matrice alla quale viene applicata l'istruzione *sqrtm* ha autovalori ripetuti.

Osservazione 37 Se la matrice A è reale, simmetrica e definita positiva, oppure complessa, hermitiana e definita positiva, lo è anche la matrice $sqrtm(A)$.

Esempio 156 $\gg sqrtm(C)$

$ans =$

$$\begin{matrix} 0.5537 + 0.4644i & 0.8070 - 0.2124i \\ 1.2104 - 0.3186i & 1.7641 + 0.1458i \end{matrix}$$

Osservazione 38 Ovviamente la matrice cui si applica *sqrtm* deve essere quadrata, altrimenti MATLAB segnala l'errore.

Esempio 157 $\gg sqrtm(B)$

??? Error using ==> schur

Matrix must be square. [24]

Error in ==> $C:\backslash MATLABR11\backslash toolbox\backslash matlab\backslash matfun\backslash sqrtm.m$

On line 31 *==>* $[Q, T] = schur(A)$; [25] *% T is real/complex according to A.* [26]

Osservazione 39 Alcune matrici non hanno alcuna radice quadrata, reale o complessa, quindi *sqrtm* non può restituire alcun risultato.

[24] Traduzione:

??? Errore nell'uso di == > *schur*

La matrice deve essere quadrata.

[25] Questa linea indica il path del file in cui è stato commesso l'errore.

[26] Traduzione:

T è reale/complessa a seconda della matrice A.

Esempio 158 *Considerando* $K = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$*, l'istruzione* $\gg sqrtm(K)$ *restituisce:*

Warning: Matrix is singular and may not have a square root. [27]

$> In\ C : \backslash MATLABR11\backslash toolbox\backslash matlab\backslash matfun\backslash sqrtm.m\ at\ line\ 62$

$ans =$

$$\begin{matrix} NaN & Inf \\ NaN & NaN \end{matrix}$$

- **funm**.

  Con l'istruzione $Y = funm(X, \ 'function\ ')$ viene valutata la funzione $function$ mediante l'algoritmo di Parlett. X deve essere una matrice quadrata e $function$ una funzione element-wise, cioè per cui le operazioni siano quelle elemento per elemento. L'algoritmo utilizza la fattorizzazione di Shur della matrice e può dare risultati non soddisfacenti oppure interrompersi completamente nel caso in cui la matrice ha autovalori ripetuti.

  Nel caso in cui i risultati non sono accurati viene visualizzato un messaggio di warning.

  I comandi $funm(X, \ 'log\ ')$ e $funm(X, \ 'sqrt\ ')$ sono equivalenti ai comandi $logm(X)$ e $sqrtm(X)$. I comandi $funm(X, \ 'exp\ ')$ e $expm(X)$ valutano la stessa funzione, ma utilizzano algoritmi differenti. È preferibile utilizzare $expm(X)$.

  Con l'istruzione $[Y, esterr] = funm(X, \ 'function\ ')$ non viene visualizzato nessun messaggio, ma è restituita una stima molto approssimata dell'errore relativo nel risultato fornito.

  Se X è simmetrica oppure hermitiana, allora la sua forma di Shur è diagonale, quindi $funm$ riesce a fornire un risultato abbastanza accurato.

  Nel caso in cui siano richieste altre operazioni su matrici, si può usare la funzione $funm$, con sintassi $funm(X, \ 'funzione\ ')$, che esegue la funzione specificata sulla matrice X.

Esempio 159 *Vedremo in seguito che l'istruzione* $qr(A)$ *restituisce la fattorizzazione QR della matrice A. Lanciando* $\gg funm(C, \ 'qr\ ')$ *si ottiene:*

[27] Traduzione:
Attenzione: la matrice è singolare e potrebbe non avere una radice quadrata.

$ans =$

$$\begin{matrix} 3.8604 & -2.2222 \\ -3.3333 & 0.5271 \end{matrix}$$

3.11 Alcune matrici particolari

MATLAB dispone di alcuni comandi per la costruzione di matrici predefinite, particolarmente usate nell'ambito matematico. In questo paragrafo vengono elencati i comandi per costruire le matrici più utilizzate.

- **eye**.
  Il comando *eye* serve per costruire la matrice identità: $eye(n)$ costruisce la matrice identità $n \times n$; $eye(m, n)$ costruisce una matrice $m \times n$ con elementi uguali ad 1 sulla diagonale principale, uguali a 0 altrove.

  Esempio 160 $\gg eye(3)$

  $ans =$

  $$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

  Esempio 161 $\gg eye(3, 5)$

  $ans =$

  $$\begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{matrix}$$

- **zeros**.
  Il comando *zeros* serve per costruire un array di 0: $zeros(n)$ costruisce una matrice $n \times n$ i cui elementi sono tutti uguali a 0; $zeros(m, n)$ costruisce una matrice $m \times n$ di 0; $zeros(m, n, p, \ldots)$ costruisce un array di dimensione $m \times n \times p \times \ldots$ con tutti gli elementi uguali a 0.

  Esempio 162 $\gg zeros(3)$

  $ans =$

  $$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Esempio 163 $\gg zeros(3,5)$
$ans =$

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Esempio 164 $\gg zeros(2,3,4)$
$ans(:,:,1) =$

0	0	0
0	0	0

$ans(:,:,2) =$

0	0	0
0	0	0

$ans(:,:,3) =$

0	0	0
0	0	0

$ans(:,:,4) =$

0	0	0
0	0	0

- **ones**.
  Un'altra matrice particolare è quella i cui elementi sono tutti uguali a 1: essa si ottiene con il comando *ones*. Il comando $ones(n)$ costruisce una matrice $n \times n$ i cui elementi sono tutti uguali a 1; $ones(m,n)$ costruisce una matrice $m \times n$ di 1; $ones(m,n,p,\ldots)$ costruisce un array di dimensione $m \times n \times p \times \ldots$ con tutti gli elementi uguali a 1.

Esempio 165 $\gg ones(3)$
$ans =$

1	1	1
1	1	1
1	1	1

Esempio 166 $\gg ones(3,5)$
$ans =$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

- **rand** e **randn**.

  Il comando *rand* serve per costruire matrici di numeri casuali, uniformemente distribuiti nell'intervallo $[0, 1]$: $rand(n)$ restituisce una matrice $n \times n$ con elementi casuali; $rand(m, n)$ restituisce una matrice $m \times n$ di elementi casuali; $rand(m, n, p, \ldots)$genera un array di elementi casuali di dimensione $m \times n \times p \times \ldots$

  Il comando *randn* opera come il comando *rand*, solo che sceglie elementi casuali, con distribuzione normale a media nulla e varianza unitaria.

  Esempio 167 *Con l'istruzione* $\gg rand(3)$ *si è ottenuta la matrice:*
  $\gg rand(3)$
  $ans =$

  $$\begin{matrix} 0.4447 & 0.9218 & 0.4057 \\ 0.6154 & 0.7382 & 0.9355 \\ 0.7919 & 0.1763 & 0.9169 \end{matrix}$$

  Esempio 168 *Con l'istruzione* $\gg randn(2, 3)$ *si è ottenuta la matrice:*
  $ans =$

  $$\begin{matrix} -0.4326 & 0.1253 & -1.1465 \\ -1.6656 & 0.2877 & 1.1909 \end{matrix}$$

  Osservazione 40 Lanciando nuovamente l'istruzione $\gg rand(3)$ oppure $\gg randn(2, 3)$ non si ottiene la stessa matrice ottenuta in precedenza.

- **hilb** e **invhilb**.

  Il comando $hilb(n)$ costruisce la matrice di Hilbert $H = \{h_{ij}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,m}}$ che ha elementi $h_{ij} = \dfrac{1}{i+j-1} \quad \forall \begin{matrix} i = 1, \ldots, n \\ j = 1, \ldots, m \end{matrix}$. Questa matrice è un famoso esempio di matrice mal condizionata. Essa è anche un buon esempio per capire quanto sia efficiente lo stile di programmazione di MATLAB: i tradizionali cicli *FOR* oppure *DO* sono sostituiti da dichiarazioni vettoriali; questo approccio è più veloce, anche se occupa più memoria. Per calcolare la matrice inversa di una matrice di Hilbert di ordine n si usa il comando $invhilb(n)$: esso genera l'esatta matrice inversa della matrice di Hilbert se $n < 15$, altrimenti esso genera un'approssimazione della matrice inversa di Hilbert. In quest'ultimo caso, confrontando la matrice che si ottiene con il comando $invhilb(n)$ con quella che si ottiene con il comando $inv(hilb(n))$ si può notare che ci sono degli errori sulle cifre decimali, cioè le due matrici non coincidono: l'errore è dovuto a tre cause:

1. l'errore causato dalla rappresentazione di $hilb(n)$ (che viene di molto amplificato a causa del cattivo condizionamento della matrice);
2. l'errore di macchina dovuto al calcolo dell'inversa della matrice $hilb(n)$;
3. l'errore, eventuale, che si compie nella rappresentazione di $invhilb(n)$.

Esempio 169 $\gg hilb(5)$

$ans =$

1.0000	0.5000	0.3333	0.2500	0.2000
0.5000	0.3333	0.2500	0.2000	0.1667
0.3333	0.2500	0.2000	0.1667	0.1429
0.2500	0.2000	0.1667	0.1429	0.1250
0.2000	0.1667	0.1429	0.1250	0.1111

Esempio 170 *L'inversa della matrice di Hilbert esaminata nell'esempio precedente si può costruire con l'istruzione* $\gg invhilb(5)$*, dalla quale si ottiene:*

$ans =$

25	−300	1050	−1400	630
−300	4800	−18900	26880	−12600
1050	−18900	79380	−117600	56700
−1400	26880	−117600	179200	−88200
630	−12600	56700	−88200	44100

Osservazione 41 Si può verificare che la matrice inversa di una matrice di Hilbert ottenuta mediante *invhilb* è diversa dalla stessa matrice ottenuta mediante *inv*: utilizzando $format\ long$ si possono notare le differenze tra le cifre decimali ottenute, ad esempio, mediante $\gg invhilb(5)$ e $\gg inv(hilb(5))$.

- **vander**.

  La matrice di Vandermonde $V = \{v_{ij}\}_{\substack{i=1,\dots,n \\ j=1,\dots,m}}$ ha elementi $v_{ij} = (a_i)^{j-1}$ $\forall \begin{array}{l} i=1,\dots,n \\ j=1,\dots,m \end{array}$ dove $a_1, a_2, \dots, a_n$ sono n numeri complessi. Questa matrice si ottiene con il comando $vander(c)$, dove c è il vettore che contiene i numeri $a_1, a_2, \ldots, a_n$

Esempio 171 $\gg$ *vander*(c)

$ans =$

1	1	1	1
8	4	2	1
27	9	3	1
64	16	4	1

- **pascal**.
  L'istruzione $pascal(n)$ restituisce la matrice di Pascal di ordine n, cioè una matrice simmetrica, definita positiva con elementi interi ed estratti dal triangolo di Pascal. La matrice inversa di una matrice di Pascal ha ancora elementi interi.

Esempio 172 *L'istruzione* $\gg pascal(5)$ *restituisce:*

$ans =$

1	1	1	1	1
1	2	3	4	5
1	3	6	10	15
1	4	10	20	35
1	5	15	35	70

e l'inversa di questa matrice si ottiene con l'istruzione $\gg inv(ans)$, *che restituisce:*

$ans =$

5	-10	10	-5	1
-10	30	-35	19	-4
10	-35	46	-27	6
-5	19	-27	17	-4
1	-4	6	-4	1

Esercizio 21 *Si costruisca M, matrice di Hilbert di dimensione 3, ed N, matrice di Pascal di dimensione 3. Si moltiplichino le due matrici elemento per elemento e si ruoti di 270° la matrice così ottenuta. Della nuova matrice si calcoli la traccia, la norma 1, la norma 2, la norma infinito e la norma di Frobenius.*
Soluzione.

Esercizio 22 *Si costruisca la matrice H, inversa della matrice di Hilbert di dimensione 3, e si calcoli il numero delle operazioni necessarie per calcolare:*

1. $expm(H)$

2. $funm(H, \, 'exp\,' \,)$

Soluzione.

3.12 Operatore :

L'operatore : è uno dei più importanti di MATLAB. Analizziamo in questo paragrafo alcuni dei suoi molteplici usi.

- Se $n1 \in \mathbb{N}$ ed $n2 \in \mathbb{N}$, con $n1 < n2$, mediante l'espressione $n1 : n2$ si ottiene un vettore riga che contiene tutti i numeri interi compresi tra $n1$ e $n2$.
  Se $n1 \equiv n2$, in output si ottiene il vettore ridotto al solo elemento $n1 \equiv n2$.
  Se $n1 > n2$, in output si ottiene un vettore vuoto.

  Esempio 173 $\gg 1 : 10$

  $ans =$

  1 2 3 4 5 6 7 8 9 10

  Esempio 174 $\gg 1 : 1$

  $ans =$

  1

  Esempio 175 $\gg 10 : 1$

  $ans =$

  Empty matrix: 1-*by*-0 [28]

- Se $a \in \mathbb{R}$ e $b \in \mathbb{R}$, con $a < b$, l'istruzione $a : b$ restituisce un vettore riga i cui elementi sono $a,\ a+1,\ a+2,\ \ldots\ , a+m$ dove m è un numero intero tale che $a+m \leq b$ e $a+(m+1) > b$. Questo significa che gli elementi del vettore di output vanno da a a b con passo 1, arrestandosi al numero che non supera b.
  Nel caso in cui $a \equiv b$, in output si ottiene il vettore ridotto al solo elemento $a \equiv b$.
  Nel caso in cui $a > b$, in output si ottiene un vettore vuoto.

[28] Traduzione:
Matrice vuota: 1×0

Esempio 176 $\gg 1.2 : 5.5$

$ans =$

1.2000 2.2000 3.2000 4.2000 5.2000

Esempio 177 $\gg 1.2 : 1.2$

$ans =$

1.2000

Esempio 178 $\gg 5.5 : 1.2$

$ans =$

Empty matrix: 1-*by*-0 [29]

- Se $a \in \mathbb{R}$, $b \in \mathbb{R}$ e $h \in \mathbb{R}$, l'istruzione $a : h : b$ restituisce un vettore riga i cui elementi sono $a,\ a+h,\ a+2h,\ \ldots\ , a+mh$ dove m è un numero intero tale che $a + mh \leq b$ e $a + (m+1)h > b$. Questo significa che gli elementi del vettore di output vanno da a a b con incremento h, arrestandosi al numero che non supera b.
  L'incremento h può essere un numero intero oppure reale e non deve essere necessariamente sottomultiplo di $|b - a|$.
  Se l'incremento è positivo si parte da $a < b$. L'incremento può anche essere negativo; in tal caso si parte da $a > b$.
  Nel caso in cui l'incremento è positivo (rispettivamente negativo) e $a > b$ (rispettivamente $a < b$) si ottiene la matrice vuota.
  Nel caso in cui $a \equiv b$, in output si ha il vettore ridotto al solo elemento $a \equiv b$.

Esempio 179 $\gg 10 : 2 : 20$

$ans =$

10 12 14 16 18 20

Esempio 180 $\gg 10 : 3 : 21$

$ans =$

10 13 16 19

[29] Traduzione:
Matrice vuota: 1×0

Esempio 181 $\gg 10 : 3 : 10$

$ans =$

10

Esempio 182 $\gg 100 : 5 : 78$

$ans =$

Empty matrix: 1-*by*-0 [30]

Esempio 183 $\gg 100 : -5 : 78$

$ans =$

100 95 90 85 80

Esempio 184 $\gg 100 : -5 : 100$

$ans =$

100

Esempio 185 $\gg 78 : -5 : 100$

$ans =$

Empty matrix: 1-*by*-0 [31]

3.13 Costruzione di intervalli MATLAB...

3.13.1 ...mediante l'operatore :

L'operatore : si rivela molto utile per la rappresentazione di intervalli numerici. Un intervallo MATLAB è un insieme di numeri in cui l'informazione significativa è contenuta solo nel valore iniziale, nel valore finale ed eventualmente nel passo di tabulazione usato. In MATLAB un intervallo è dunque rappresentato da un numero finito di punti. Essendo permessi passi non interi, l'operatore : consente di rappresentare tutti i tipi di intervalli reali con punti equidistanti.

[30] Traduzione:
Matrice vuota: 1×0

[31] Traduzione:
Matrice vuota: 1×0

Esempio 186 *L'intervallo reale* $[\ 1\ ;\ 1,5\ ]$ *può essere costruito numericamente mediante l'operatore* : .
Bisogna costruire un vettore il cui elemento iniziale sia 1*, il cui elemento finale sia* $1,5$ *e che sia costituito da punti equidistanti. Questo problema si risolve con il metodo esaminato nel paragrafo precedente: si costruisce il vettore* $1 : h : 1.5$*, dove il passo* h *deve essere scelto tra i sottomultipli di* $b - a = 1.5 - 1 = 0.5$*, cioè* $h = \dfrac{b-a}{N} = \dfrac{1.5-1}{N} = \dfrac{0.5}{N}$ *. Si può scegliere, ad esempio,* $h = 0.1$*: l'intervallo richiesto si ottiene mediante l'istruzione* $\gg [\ 1 : 0.1 : 1.5\ ]$*, che restituisce:*
ans =

1.0000 1.1000 1.2000 1.3000 1.4000 1.5000

Osservazione 42 Volendo considerare un numero maggiore di punti nell'intervallo, basta assumere un passo di discretizzazione h più piccolo.

3.13.2 ...mediante *linspace* e *logspace*

Altri due metodi per costruire intervalli consistono nell'utilizzo delle funzioni *linspace* e *logspace*: esse permettono di ottenere lo stesso risultato raggiunto con l'operatore : , prefissando però il numero di punti anziché il passo.

La funzione **linspace** serve per costruire un vettore di punti equidistanti: mediante $linspace(x1, x2)$ si ottiene un vettore riga di 100 punti equidistanti compresi tra $x1$ e $x2$, mentre con $linspace(x1, x2, N)$ si ottiene un vettore riga di N elementi equidistanti compresi tra $x1$ e $x2$.

La funzione **logspace** genera vettori di punti spaziati logaritmicamente: mediante $logspace(d1, d2)$ si ottiene un vettore riga di 50 punti logaritmicamente spaziati e compresi tra 10^{d1} e 10^{d2}, mentre mediante $logspace(d1, d2, N)$ si ottiene un vettore riga di N punti logaritmicamente spaziati e tutti compresi tra 10^{d1} e 10^{d2}.

Esempio 187 *L'istruzione* $\gg linspace(1, 1.5, 6)$ *restituisce*
ans =

1.0000 1.1000 1.2000 1.3000 1.4000 1.5000

e quindi è equivalente a $\gg [\ 1 : 0.1 : 1.5\ ]$.

Esempio 188 $\gg logspace(1, 2, 5)$
ans =

10.0000 17.7828 31.6228 56.2341 100.0000

3.14 Elementi di una matrice

Sia A una matrice con m righe ed n colonne. L'accesso agli elementi di A si ottiene tramite indici: denotiamo con $A(i, j)$ l'elemento della matrice A sulla i-esima riga e sulla j-esima colonna (con $i = 1, \ldots, m$ e $j = 1, \ldots, n$).
L'elemento i-esimo di un vettore riga X può essere equivalentemente individuato mediante il comando $X(i)$ oppure mediante il comando $X(1, i)$. Analogamente l'elemento j-esimo di un vettore colonna Y può essere equivalentemente individuato mediante il comando $Y(j)$ oppure mediante il comando $Y(j, 1)$.
Se proviamo ad usare un indice di riga o di colonna superiore a quelli consentiti viene segnalato un messaggio di errore: consideriamo, ad esempio, la matrice A e supponiamo di usare l'istruzione $\gg A(1, 6)$: viene visualizzato il messaggio:

??? Index exceeds matrix dimensions. [32]

Se invece assegnamo un valore ad un elemento non compreso nella matrice, le dimensioni crescono in modo da comprendere anche il nuovo elemento.

Esempio 189 *L'istruzione* $\gg x(3)$ *restituisce il terzo elemento di* x*, cioè:*
$ans =$

3

invece l'istruzione $\gg x(8) = 2$ *crea il nuovo vettore:*
$x =$

1 2 3 4 5 0 0 2

Osservazione 43 Osserviamo che, da questo punto in poi il vettore x non è più quello definito nel capitolo Variabili utilizzate, ma questo nuovo vettore. Per gli esempi successivi deve essere ricaricato il vettore x di partenza (è sufficiente lanciare $\gg$ *load filename* x, dove filename è il nome del file nel quale sono state salvate tutte le variabili in uso).

Osservazione 44 In assenza di informazioni su un elemento, MATLAB usa per default il valore 0. Quindi nell'esempio precedente il sesto ed il settimo elemento del vettore (su cui non si hanno informazioni) vengono posti uguali a 0.

[32] Traduzione:
??? Un indice è superiore alle dimensioni della matrice.

Esempio 190 *L'istruzione* $\gg A(3,4)$ *restituisce l'elemento di* A *sulla terza riga e sulla quarta colonna, cioè:*

$ans =$

12

invece l'istruzione $\gg A(1,6) = 1$ *crea la nuova matrice:*

$A =$

16	3	2	13	0	1
5	10	11	8	0	0
9	6	7	12	0	0
4	15	14	1	0	0

Osservazione 45 Anche in questo caso la matrice A non è più quella definita nel capitolo Variabili utilizzate, ma quest'ultima. Per gli esempi successivi deve essere ricaricata la matrice A di partenza.

Scrivendo $x(i) = [\ ]$ (con $1 \leq i \leq length(x)$) si elimina l'elemento x_i dal vettore $x = \{x_k\}_{k=1,\ldots,length(x)}$.
La stessa procedura non può essere utilizzata per le matrici, in quanto tutte le righe e tutte le colonne devono avere sempre lo stesso numero di elementi. Si può, però, in maniera analoga, eliminare un'intera riga oppure un'intera colonna o addirittura tutti gli elementi di una matrice: considerata una generica matrice A

$\gg A(i,:) = [\ ]$ elimina la riga i-esima della matrice;

$\gg A(:,j) = [\ ]$ elimina la colonna j-esima della matrice;

$\gg A(:,:) = [\ ]$ elimina tutte le righe e tutte le colonne della matrice, cioè restituisce la matrice vuota (quindi l'istruzione equivale a $\gg A = [\ ]$).

Esempio 191 $\gg x(4) = [\ ]$

$x =$

1 2 3 5

Osservazione 46 Si ricarichi il vettore x di partenza.

Esempio 192 *Consideriamo la matrice* B*:*

$\gg B(2,4) = [\ ]$

??? Indexed empty matrix assignment is not allowed. [33]

[33]Traduzione:
??? Non è consentita l'assegnazione di elementi vuoti in una matrice.

$\gg B(:,4) = [\ ]$
$B =$

1	2	3	5
6	7	8	10

Osservazione 47 Si ricarichi la matrice B di partenza.

$\gg B(2,:) = [\ ]$
$B =$

1	2	3	4	5

Osservazione 48 Si ricarichi la matrice B di partenza.

$\gg B(:,:) = [\ ]$
$B =$
Empty matrix: 0-by-5 [34]

Osservazione 49 Si ricarichi la matrice B di partenza.

Volendo modificare un elemento di un vettore oppure di una matrice basta individuare l'elemento mediante il suo indice di riga e di colonna e assegnare il nuovo valore.

Esempio 193 $\gg x(4) = 9$
$x =$

1	2	3	9	5

Osservazione 50 Se x fosse stato un vettore colonna anziché riga, l'istruzione $\gg x(4) = 9$ avrebbe comunque cambiato il quarto elemento di x in 9.

Osservazione 51 Si ricarichi il vettore x di partenza.

Esempio 194 $\gg B(2,4) = 1$
$B =$

1	2	3	4	5
6	7	8	1	10

Osservazione 52 Si ricarichi la matrice B di partenza.

Esercizio 23 *Definito il vettore* $v = [\ 5, -1, 3, 2, 1, 4, -7\ ]$

[34] Traduzione:
Matrice vuota: 0×5

1. *si calcoli la sua lunghezza mediante il comando length;*
2. *si calcolino le sue dimensioni mediante il comando size;*
3. *si ponga come decimo elemento del vettore l'opposto del suo quinto elemento;*
4. *si calcoli il massimo elemento del vettore e l'indice cui esso corrisponde;*
5. *si calcoli il minimo elemento del vettore e l'indice cui esso corrisponde;*
6. *si ordinino in maniera decrescente gli elementi del vettore, mostrando gli indici cui corrispondono nel vettore di partenza gli elementi ordinati;*
7. *si calcoli la somma degli elementi del vettore;*
8. *si calcoli il valor medio degli elementi del vettore.*

Soluzione.

Esercizio 24 *Dopo aver memorizzato il vettore* $v = [\ 3, 6, 9, 12, 15\ ]$ *si costruisca a partire da esso il vettore* $z = [\ 3, 9, 12, 15, 8\ ]$.
Si costruisca la matrice che ha questo vettore sulla prima sovradiagonale e se ne calcoli determinante e rango.
Soluzione.

3.15 Estrazione di sottomatrici...

Una potente capacità di MATLAB consiste nella possibilità di prelevare sottomatrici da matrici assegnate; ciò si realizza usando come indici dei vettori oppure mediante l'operatore :.

3.15.1 ...mediante vettori

Se A è una matrice di dimensioni $m \times n$, l'istruzione $\gg A(i,j)$ può essere utilizzata nei seguenti casi:

- caso $\begin{cases} i \text{ scalare} & \text{con } 1 \leq i \leq m \\ j \text{ scalare} & \text{con } 1 \leq j \leq n \end{cases}$

  $\gg A(i,j)$ rappresenta l'elemento sulla i-esima riga e sulla j-esima colonna della matrice A, come si è visto nel paragrafo precedente;

- caso $\begin{cases} i \text{ vettore} & i = \{i_k\}_{k=1,\ldots,s} \quad \text{con } 1 \leq i_k \leq m \quad \forall k = 1, \ldots, s \\ j \text{ scalare} & \text{con } 1 \leq j \leq n \end{cases}$

$\gg A(i, j)$ restituisce il vettore colonna i cui elementi sono gli elementi $A(i_1, j)$, $A(i_2, j), \ldots, A(i_s, j)$ della matrice A.

Esempio 195 *Sia A una matrice di dimensioni $m \times n$ e siano i e k due interi tali che $1 \leq i \leq m$, $1 \leq k \leq m$ e $i < k$ e j un intero tale che $1 \leq j \leq n$.*
La scrittura $\gg A(i : k, j)$ è molto utilizzata in MATLAB: come abbiamo già visto, $i : k$ rappresenta il vettore riga i cui elementi sono i, $i + 1$, $i + 2$, $\ldots$, k. Allora $\gg A(i : k, j)$ ha come output il vettore colonna i cui elementi sono gli elementi che vanno dall'i-esimo al k-esimo della j-esima colonna della matrice A.

Esempio 196 $\gg A(2 : 4, 1)$
ans =

    5
    9
    4

- caso $\begin{cases} i \text{ scalare} & \text{con } 1 \leq i \leq m \\ j \text{ vettore} & j = \{j_z\}_{z=1,\ldots,r} \quad \text{con } 1 \leq j_z \leq n \quad \forall z = 1, \ldots, r \end{cases}$

$A(i, j)$ restituisce il vettore riga i cui elementi sono gli elementi $A(i, j_1)$, $A(i, j_2), \ldots, A(i, j_r)$della matrice A.

Esempio 197 *Sia A una matrice di dimensioni $m \times n$ e siano i un intero tale che $1 \leq i \leq m$, j e k due interi tali che $1 \leq j \leq n$, $1 \leq k \leq n$ e $j < k$.*
La scrittura $\gg A(i, j : k)$ è molto utilizzata in MATLAB: come abbiamo già visto, $j : k$ rappresenta il vettore riga i cui elementi sono j, $j + 1$, $j + 2$, $\ldots$, k. Allora $\gg A(i, j : k)$ ha come output il vettore riga i cui elementi sono gli elementi che vanno dallo j-esimo al k-esimo della i-esima riga della matrice A.

Esempio 198 $\gg A(1, 2 : 4)$
ans =

    3   2   13

- caso $\begin{cases} i \text{ vettore} & i = \{i_k\}_{k=1,\dots,s} & \text{con } 1 \le i_k \le m \quad \forall k = 1,\dots,s \\ j \text{ vettore} & j = \{j_z\}_{z=1,\dots,r} & \text{con } 1 \le j_z \le n \quad \forall z = 1,\dots,r \end{cases}$

  $\gg A(i,j)$ restituisce la sottomatrice di A di dimensione $length(i) \times length(j)$:

$$\begin{pmatrix} A(i_1,j_1) & A(i_1,j_2) & \dots & A(i_1,j_r) \\ A(i_2,j_1) & A(i_2,j_2) & \dots & A(i_2,j_r) \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ A(i_s,j_1) & A(i_s,j_2) & \dots & A(i_s,j_r) \end{pmatrix}$$

Esempio 199 $\gg A(1:3,2:4)$

$ans =$

3	2	13
10	11	8
6	7	12

Esempio 200 $\gg A(4:-1:2,2:4)$

$ans =$

15	14	1
6	7	12
10	11	8

Esempio 201 *Consideriamo i vettori:*

$\gg v = [\ 4\ 1\ 1\ 3\ ];$

$\gg w = [\ 2\ 2\ 4\ ];$

Allora mediante l'istruzione $\gg A(v,w)$ *si ottiene la matrice:*

$ans =$

15	15	1
3	3	13
3	3	13
6	6	12

Osservazione 53 Ovviamente, si possono considerare anche indici di riga o di colonna superiori al numero delle righe o al numero delle colonne della matrice che si sta considerando: MATLAB segnala l'errore.

Esempio 202 $\gg A(5, 1:3)$
??? Index exceeds matrix dimensions. [35]

Osservazione 54 Il vettore i, o il vettore j, o entrambi i vettori i e j possono essere, eventualmente, vuoti: in questi casi si ottiene una matrice vuota.

Esempio 203
$\gg A(1, [\ ])$
$ans =$
    Empty matrix: 1-*by*-0 [36]
$\gg A([\ ], 1)$
$ans =$
    Empty matrix: 0-*by*-1 [37]
$\gg A([\ ], [\ ])$
$ans =$
    [ ]

3.15.2 ...mediante l'operatore :

Molto utile per l'estrazione di sottomatrici da matrici assegnate risulta l'operatore :. Esso al posto dell'indice di riga di una matrice indica che bisogna considerare tutti gli elementi della colonna (o delle colonne) in esame della matrice; al posto dell'indice di colonna di una matrice indica che bisogna considerare tutti gli elementi della riga (o delle righe) in esame della matrice.
Questo vuol dire che, considerata la matrice A di dimensioni $m \times n$:

- $\gg A(:, j)$ ha come output tutti gli elementi della j-esima colonna della matrice A, se j è uno scalare ($1 \leq j \leq n$).

  Esempio 204 *Consideriamo la matrice A: l'istruzione $\gg A(:, 2)$ restituisce il vettore colonna:*

[35] Traduzione:
??? Un indice è superiore alle dimensioni della matrice.
[36] Traduzione:
Matrice vuota 1×0
[37] Traduzione:
Matrice vuota 0×1

$ans =$

3
10
6
15

- $\gg A(:, j)$ ha come output tutti gli elementi delle colonne $j_1, j_2, \ldots, j_s$ della matrice A, se $j = \{j_k\}_{k=1,\ldots,s}$ (con $1 \leq j_k \leq n \quad \forall k = 1, \ldots, s$).

  Esempio 205 $\gg A(:, [\ 2, 2, 4, 1\ ])$

  $ans =$

3	3	13	16
10	10	8	5
6	6	12	9
15	15	1	4

- $\gg A(i, :)$ ha come output tutti gli elementi della i-esima riga della matrice A, se i è uno scalare ($1 \leq i \leq m$).

  Esempio 206 *Consideriamo la matrice A: l'istruzione $\gg A(2, :)$ restituisce il vettore riga:*

  $ans =$

5	10	11	8

- $\gg A(i, :)$ ha come output tutti gli elementi delle righe $i_1, i_2, \ldots, i_s$ della matrice A, se $i = \{i_k\}_{k=1,\ldots,s}$ (con $1 \leq i_k \leq m \quad \forall k = 1, \ldots, s$).

  Esempio 207 $\gg A([\ 2, 2, 4, 1\ ], :)$

  $ans =$

5	10	11	8
5	10	11	8
4	15	14	1
16	3	2	13

- la scrittura $\gg x(:)$, con x vettore (riga oppure colonna), serve per leggere tutti gli elementi di x riguardati come elementi di un vettore colonna.
  La scrittura $\gg A(:)$, con A matrice, serve per leggere tutti gli elementi della matrice A riguardati come elementi di un unico vettore colonna (gli elementi di A vengono letti per colonne).

Esempio 208 $\gg c(:)$

$ans =$

1

2

3

4

Esempio 209 $\gg x(:)$

$ans =$

1

2

3

4

5

Esempio 210 $\gg C(:)$

$ans =$

1

3

2

4

Esercizio 25 *Si costruisca, senza visualizzarlo sullo schermo, il vettore riga*

$$v = [\ -1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1\ ]$$

- *inserendo direttamente i valori numerici da tastiera;*
- *con l'istruzione linspace;*
- *utilizzando l'operatore* : .

Si costruisca il vettore z i cui elementi corrispondono ai primi 3 elementi di v. Si calcoli la matrice di Frobenius associata a z, memorizzandola nella matrice C e si approssimino per difetto gli elementi di questa nuova matrice, memorizzandoli nella matrice D.
Si calcoli la radice quadrata della matrice D e la radice quadrata di tutti gli elementi di D.
Soluzione.

Esercizio 26 *Si costruisca il vettore x contenente 31 valori ottenuti con spaziatura logaritmica in base 2 dell'intervallo $[\ \frac{1}{2}\ ,\ 2^5\ ]$, senza visualizzarlo sullo schermo.*
Si costruisca Z, matrice di Vandermonde ottenuta con i primi 5 elementi del vettore x. Si elevino al quadrato tutti gli elementi della terza e della quarta colonna di Z e si memorizzi il risultato nella matrice U, visualizzandolo in $format\ long$.
Si estragga la sottomatrice u composta dagli elementi della seconda e della terza riga di U e si calcoli e^u.
Si ritorni al formato predefinito in MATLAB.
Si elevi la matrice u precedente alla potenza -0.1 e si memorizzino gli elementi della prima colonna della matrice risultante nel vettore v. Si calcoli la norma 5.4 del vettore v.
Soluzione.

Esercizio 27 *Si costruisca il vettore $x = [\ 1\ 2\ 3\ 4\ 5\ 6\ 20\ ]$ e, utilizzando questo risultato, il vettore $y = [\ 1\ 2\ 3\ 4\ 5\ 6\ 20\ 20\ 6\ 5\ 4\ 3\ 2\ 1\ ]$. Si eliminino gli ultimi due elementi di questo vettore e lo si trasformi in una matrice 3×4, memorizzandola in Y.*
Si elimini dalla matrice Y l'ultima colonna e si moltiplichi la nuova matrice ottenuta per il quadrato magico di dimensione 3. Si memorizzi il risultato in Z.
Si prelevi la parte superiore della matrice Z a partire dalla prima diagonale e si memorizzi il risultato nella matrice S.
Si estragga la sottomatrice di S costituita dagli elementi sulla prima e seconda riga e sulla seconda e terza colonna e la si memorizzi in R.
Si modifichi l'elemento di posto $(2, 1)$ della matrice R in 1 e si effettui il prodotto di Kronecker della nuova matrice R per la matrice logaritmo della matrice R.
Soluzione.

3.16 Concatenazione di matrici

In MATLAB le matrici e i vettori possono essere affiancati in modo da costruire matrici più grandi; questa operazione prende il nome di **concatenazione** e può essere realizzata in due modi:

- mediante le stesse istruzioni con cui si costruiscono matrici e vettori di numeri, cioè con l'operatore [ ].

  Esempio 211 $\gg [\ C\ \ D\ ]$

  $ans =$

  $$\begin{matrix} 1 & 2 & 0 & 1 & 1 \\ 3 & 4 & -1 & 2 & 0 \end{matrix}$$

  Esempio 212 $\gg [\ C\ ,\ C\ ]$

  $ans =$

  $$\begin{matrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{matrix}$$

  Esempio 213 $\gg [\ C\ ;\ C\ ]$

  $ans =$

  $$\begin{matrix} 1 & 2 \\ 3 & 4 \\ 1 & 2 \\ 3 & 4 \end{matrix}$$

- mediante il comando **cat**: mediante $cat(dim, a1, a2, a3, \ldots)$ le matrici $a1$, $a2$, $a3$, ... vengono concatenate lungo la direzione dim. Ovviamente $cat(1, a, b)$ equivale a $[\ a\ ;\ b\ ]$ e $cat(2, a, b)$ equivale a $[\ a\ ,\ b\ ]$ (o, equivalentemente, $[\ a\ \ b\ ]$).

  Esempio 214 $\gg cat(1, C, G)$

  $ans =$

  $$\begin{matrix} 1 & 2 \\ 3 & 4 \\ 2 & -2 \\ 3 & 8 \end{matrix}$$

  Esempio 215 $\gg cat(2, C, G)$

  $ans =$

  $$\begin{matrix} 1 & 2 & 2 & -2 \\ 3 & 4 & 3 & 8 \end{matrix}$$

Esercizio 28 *Si consideri la matrice* A *e si valutino le sottomatrici* $A1 = A(2:3, 2:4)$ *e* $A2 = A(\ [\ 1\ \ 2\ ]\ , 1:3)$. *Si concatenino le matrici così* *ottenute nella matrice* $G = \begin{pmatrix} A1 & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} & A2 \end{pmatrix}$.

Soluzione.

Esercizio 29 *Costruiti i vettori* $a = [\ 1 \quad 3 \quad 5 \quad 7\ ]$ *e* $b = [\ 2 \quad 4 \quad 6 \quad 8\ ]$, *si costruisca, a partire da essi, il vettore* $c = [\ 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8\ ]$.
Si aggiunga tra il terzo e il quarto elemento del vettore c *l'elemento* 7 *e si trasformi il vettore così ottenuto in* S, *matrice* 3×3 *con elementi uguali agli elementi di* c. *Si estragga la sottomatrice costituita dalle prime due righe e dalle ultime due colonne di* S.
Soluzione.

Esercizio 30 *Si costruisca la matrice* T *la cui prima colonna rappresenta la discretizzazione dell'intervallo* $[\ 0, 1\ ]$ *in* 5 *parti uguali e la cui seconda colonna rappresenta il valore che la funzione coseno assume nei punti della discretizzazione ottenuti.*
Soluzione.

Capitolo 4

Complementi sulle matrici

4.1 Conversione fra sistemi di coordinate

Gli operatori MATLAB che permettono la conversione tra sistemi di coordinate sono:

- **cart2pol**
  Consente il passaggio da coordinate cartesiane a coordinate cilindriche o polari: nei casi tridimensionali mediante l'istruzione $[teta, ro, z] = cart2pol(x, y, z)$ le coordinate cartesiane (x_i, y_i, z_i) memorizzate nei vettori x, y e z vengono trasformate in coordinate cilindriche e memorizzate nei corrispondenti elementi $teta_i$, ro_i e z_i dei vettori $teta$, ro e z; nei casi bidimensionali mediante l'istruzione $[teta, ro] = cart2pol(x, y)$ le coppie (x_i, y_i) in coordinate cartesiane vengono convertite nelle coppie $(teta_i, ro_i)$ in coordinate polari.
  Ricordiamo che l'elemento $teta_i$ indica lo spostamento in senso antiorario rispetto all'asse x del punto di coordinate cartesiane (x_i, y_i, z_i) che si sta considerando, ro_i è la distanza dall'origine della proiezione nel piano xy del punto, mentre z_i rappresenta la distanza del punto dal piano xy. L'anomalia $teta$ è espressa in radianti.
  Ovviamente i vettori x, y e z in input devono avere la stessa lunghezza.

  Esempio 216 *Consideriamo i punti $P_1(1, 1, 0)$ e $P_2(2, 1, 1)$ in coordinate cartesiane e trasformiamoli in coordinate cilindriche:*
  $\gg x = [\ 1\ \ 2\ ]$
  $\gg y = [\ 1\ \ 1\ ]$
  $\gg z = [\ 0\ \ 1\ ]$

$\gg [teta, ro, zeta] = cart2pol(x, y, z)$
$teta =$

0.7854 0.4636

$ro =$

1.4142 2.2361

$zeta =$

0 1

Esempio 217 *Consideriamo i punti $P_1(1,1)$ e $P_2(2,1)$ in coordinate cartesiane e trasformiamoli in coordinate polari:*
$\gg x = [\ 1\ \ 2\ ]$
$\gg y = [\ 1\ \ 1\ ]$
$\gg [teta, ro] = cart2pol(x, y)$
$teta =$

0.7854 0.4636

$ro =$

1.4142 2.2361

- **pol2cart**
  Consente il passaggio da coordinate polari o cilindriche a coordinate cartesiane.

  Esempio 218 *Consideriamo i punti $S_1(0.7854, 1.4142)$ e $S_2(0.4636, 2.2361)$ in coordinate polari e trasformiamoli in coordinate cartesiane:*
  $\gg teta = [\ 0.7854\ \ 0.4636\ ]$
  $\gg ro = [\ 1.4142\ \ 2.2361\ ]$
  $\gg [x, y] = pol2cart(teta, ro)$
  $x =$

  1.0000 2.0001

  $y =$

  1.0000 0.9999

  Come si può notare, trascurando gli errori dovuti all'approssimazione sui dati di input, si ottengono proprio i punti di partenza dell'esempio precedente.

- **cart2sph**

  Consente il passaggio da coordinate cartesiane a coordinate sferiche. Anche in questo caso la trasformazione si ottiene mediante un'istruzione del tipo $[fi, teta, ro] = cart2sph(x, y, z)$, che restituisce i valori di ϕ e di θ espressi in radianti.

  Esempio 219 *Consideriamo i punti P_1 e P_2 in coordinate cartesiane e trasformiamoli in coordinate sferiche:*
  $\gg [fi, teta, ro] = cart2sph(x, y, z)$
  $fi =$

  0.7854 0.4636

  $teta =$

  0 0.4205

  $ro =$

  1.4142 2.4495

- **sph2cart**

  Consente il passaggio da coordinate sferiche a coordinate cartesiane.

  Esempio 220 *Considerando i valori di fi, $teta$ e ro calcolati nell'esempio precedente lanciamo:*
  $\gg [x, y, z] = sph2cart(fi, teta, ro)$
  si ottiene:
  $x =$

  1.0000 2.0000

  $y =$

  1 1

  $z =$

  0 1.0000

  che sono proprio le coordinate cartesiane dei punti dai quali siamo partiti nell'esempio precedente.

Esercizio 31 *Si risolva il sistema di equazioni:*

$$\begin{cases} x^2 + y^2 = a \\ \dfrac{x}{y} = b \end{cases}$$

per $a = (1, 4, 3)$ *e* $b = (1, \sqrt{3}, 0.5)$

Suggerimento: si risolva il sistema in coordinate polari.
Soluzione.

Soluzione esercizio 1 Poichè in coordinate polari risulta $\begin{cases} x = r * cos(\theta) \\ y = r * sin(\theta) \end{cases}$ allora si può scrivere: $\begin{cases} x^2 + y^2 = r^2 * (cos^2\theta + sin^2\theta) = r^2 \\ \dfrac{x}{y} = \dfrac{cos(\theta)}{sin(\theta)} = ctg(\theta) \end{cases} \Rightarrow \begin{cases} r^2 = a \\ ctg(\theta) = b \end{cases}$

Quindi: $\begin{cases} r = \sqrt{a} \\ \theta = arctg(b) \end{cases}$

Da tutto questo si deduce che, lanciando le seguenti istruzioni, si ottiene la soluzione del sistema in coordinate cartesiane:

$\gg r = sqrt(a);$
$\gg teta = acot(b);$
$\gg x = r. * cos(teta);$
$\gg y = r. * sin(teta);$

La soluzione è data dalle terne
$x = (0.7071, 1.7321, 0.7746)$
$y = (0.7071, 1.0000, 1.5492).$

4.2 Operatori

Oltre agli operatori elementari, negli M-file è possibile utilizzare altri due tipi di operatori MATLAB: gli **operatori relazionali** e gli **operatori logici**.

4.2.1 Operatori relazionali

Gli operatori relazionali confrontano gli elementi corrispondenti di array aventi le stesse dimensioni oppure di un array con uno scalare, operando elemento per elemento.
In output si ottiene un array delle stesse dimensioni degli array di partenza, che ha elementi uguali ad 1 (vero) in corrispondenza degli elementi analoghi per i quali è verificata la relazione richiesta, 0 (falso) altrove.
Nel caso in cui si confronta uno scalare con un array, MATLAB confronta lo scalare con ognuno degli elementi dell'array.
MATLAB dispone dei seguenti operatori relazionali:

$<$	minore
$<=$	minore o uguale

$>$	maggiore
$>=$	maggiore o uguale
$==$	uguale
$\sim=$	diverso

Osservazione 55 Mentre gli operatori $<$, $<=$, $>$ e $>=$ confrontano soltanto la parte reale degli elementi cui vengono applicati, $==$ e $\sim=$ controllano sia la parte reale che la parte immaginaria degli elementi cui vengono applicati.

Esempio 221

Esempio 222

Esempio 223

4.2.2 Operatori logici

MATLAB dispone dei seguenti operatori logici:

&	and
$\vert$	or
xor	or esclusivo
$\sim$	not

Ognuno di questi operatori rispetta un set di regole, che determina il risultato di un'operazione logica:

- un'operazione logica che utilizza l'operatore & è vera se tutti gli operandi sono logicamente veri, è falsa nel caso in cui anche uno solo degli operandi è falso;
- un'operazione logica che utilizza l'operatore $|$ è vera se almeno uno degli operandi è logicamente vero, è falsa se tutti gli operandi sono falsi;
- un'operazione logica che utilizza l'operatore $\sim$ nega la proposizione cui viene applicata, quindi è vera se l'operando è logicamente falso, è falsa se esso è logicamente vero.

4.2.3 Precedenze nell'uso degli operatori

MATLAB consente di costruire espressioni che utilizzano operatori di diverso tipo e valuta siffatte espressioni eseguendo le operazioni nell'ordine in cui sono scritte, da sinistra a destra e tenendo conto delle seguenti precedenze (scritte dal grado di precedenza maggiore a quello minore):

1. ( )
2. $\sim$
3. $.'$ $\quad .\wedge$ $\quad '$ $\quad \wedge$
4. $.*$ $\quad ./$ $\quad .\backslash$ $\quad *$ $\quad /$ $\quad \backslash$
5. $+$ $\quad -$
6. $:$ $\quad <$ $\quad <=$ $\quad >$ $\quad >=$ $\quad =$ $\quad \sim=$
7. $\&$ $\quad |$

4.2.4 Operazioni logiche

Esercizio 32 *Dato il vettore y, si visualizzino soltanto le sue componenti maggiori di 2.*
Soluzione.

Soluzione esercizio 2 *La richiesta può essere soddisfatta lanciando le istruzioni:*
$\gg k = y > 6$
$k =$
$\quad 0 \quad 1 \quad 0 \quad 0 \quad 1$
$\gg y(k)$
$ans =$
$\quad 10 \quad 7$

Capitolo 5

Programmazione

Finora abbiamo utilizzato MATLAB in modalità interattiva, cioè digitando le istruzioni da tastiera una per volta: questo ci consente di utilizzare il computer come una calcolatrice estesa. Ma ci sono dei limiti a questo tipo di approccio: supponiamo di avere bisogno di dover ripetere più volte lo stesso comando o lo stesso gruppo di comandi, oppure di dover ripetere più volte le stesse istruzioni ma con parametri diversi, oppure di dover effettuare dei controlli sui parametri che stiamo utilizzando. Come fare per non riscrivere ogni volta le stesse istruzioni? La soluzione consiste nello scrivere un programma, mediante le istruzioni del linguaggio MATLAB.

5.1 M-file

I file che contengono i codici del linguaggio MATLAB sono chiamati M-file. Il procedimento per utilizzare un M-file è il seguente:

- si scrive il file mediante un editor di testo;
- si salva il file con l'estensione .m [1] per far si che MATLAB lo riconosca come un M-file;
- si richiama il file dal prompt di MATLAB oppure dall'interno di un altro M-file in modo da mandarlo in esecuzione.

Per costruire un M-file in MATLAB è sufficiente seguire alcune semplici regole:

[1] Se si sta usando l'editor di MATLAB, l'estensione viene automaticamente inserita, per cui il .m può essere omesso.

- le istruzioni possono essere scritte l'una sotto l'altra ognuna su una riga differente, oppure l'una affianco all'altra, separate da una virgola o da un punto e virgola;
- il punto e virgola alla fine di un'istruzione comporta il fatto che il risultato dell'istruzione non viene visualizzato in fase di esecuzione;
- si possono aggiungere righe di commento (che saranno ignorate durante l'esecuzione del programma) facendole precedere dal simbolo **%**.

Gli M-file possono essere di due tipi: **script file** oppure **function file**. Una prima differenza sta nel fatto che una function ha una riga di intestazione che comincia con la parola function, nella quale vengono dichiarati il nome della function, il nome delle variabili di input e gli (eventuali) argomenti in output (non è detto che ve ne siano necessariamente: per esempio, la function potrebbe realizzare un grafico come output), mentre uno script file non ha riga di intestazione.
Entrambi i tipi di file sono costituiti dai seguenti elementi:

- la **linea H**1 [2]: è la prima riga di commento del file, successiva alla riga di intestazione [3];

  Osservazione 56 Lanciando $\gg$ *lookfor* $NomeFile$ (dove NomeFile è il nome con il quale è stato salvato l'M-file) MATLAB visualizza la linea H1 [4].

- eventuali **righe di commento** che precedono il programma;

  Osservazione 57 Lanciando $\gg$ *help* $NomeFile$ (dove NomeFile è il nome con il quale è stato salvato l'M-file) MATLAB visualizza la linea H1 e tutte le righe di commento che sono scritte all'inizio del file e che precedono la prima riga bianca oppure la prima riga di istruzioni

- il **corpo del file** che contiene le istruzioni del programma;
- eventuali **righe di commento** inserite tra le istruzioni del programma.

[2] L'abbreviazione H1 sta per Help 1.
[3] Tali righe non sono obbligatorie e possono, quindi, anche essere omesse.
[4] Ovviamente bisogna trovarsi nella cartella in cui è memorizzato il file.

5.1.1 Script file

Uno script file è un M-file scritto con un editor di testo e contenente istruzioni da eseguire quando il file è chiamato da MATLAB: in pratica è come scrivere un blocco di istruzioni e poi farle eseguire l'una di seguito all'altra, ottimizzando i tempi che richiederebbe l'inserimento di una istruzione per volta direttamente dal prompt di MATLAB.
Uno script file lavora direttamente sulle variabili presenti in memoria e non consente l'uso di variabili locali; inoltre non accetta variabili in input, ma opera sui dati presenti nel workspace.

Esempio 224 *Possiamo costruire uno script file molto semplice che calcoli la somma di due variabili assegnate.*
Dopo aver aperto l'editor di testo scriviamo le seguenti righe di programma:

% Calcola la somma di due scalari assegnati
%
% Sintassi: $\quad$ *somma*1

$s = 4$;
$t = 3$;
$r = s + t$

Salviamo il file con il nome somma$1.m$ *ed usciamo dall'editor. Lanciando dal prompt di MATLAB il nome del file senza estensione (cioè* $\gg somma1$*) si ottiene in output:*
$r =$
$\quad 7$
Questo vuol dire che lo script file ha eseguito la somma delle variabili in esso contenute.

Osservazione 58 Sebbene uno script file non restituisca dati di output, alcune delle variabili in esso utilizzate possono rimanere memorizzate nel workspace ed essere usate in operazioni successive.

Esempio 225 *Nell'esempio precedente la variabile* $r = 7$ *(come anche le variabili* s *e* t*) resta memorizzata nel workspace, quindi lanciando, ad esempio,* $\gg r + 2$ *si ottiene:*
$ans =$
$\quad 9$

5.1.2 Function file

Un function file (o, più semplicemente, una function) è un M-file che può accettare dati di input e restituire variabili in output. Per costruire una function si procede come nel caso in cui si voglia scrivere uno script file, con l'unica differenza che la function deve avere una riga di intestazione del tipo:
$function\ [variabili\ di\ uscita] = NomeFunction(variabili\ di\ ingresso)$.
Il nome della function ($NomeFunction$) può coincidere o meno con quello del file con estensione .m in cui essa viene memorizzata. In ogni caso, per eseguire la function, bisogna lanciare la seguente istruzione:
$\gg [nomi\ variabili\ di\ uscita] = NomeFile(dati\ di\ ingresso)$
dove NomeFile è il nome del file nel quale è stata salvata la function.

Esempio 226 *Possiamo costruire una function molto semplice che calcoli la somma di due variabili a e b che possono essere assegnate da tastiera, quando viene mandata in esecuzione la function.*
Dopo aver aperto l'editor di testo, scriviamo le seguenti istruzioni:

$function\ s = somma2(a, b)$

% Calcola la somma di due scalari assegnati
%
% Sintassi: $s = somma2(a, b)$
%
% Dati di input
% a e b: scalari che si intendono sommare
% Dati di output
% s: somma di a e b

$s = a + b$;

e salviamo il file con il nome somma2.m. Lanciando dal prompt di MATLAB $\gg s = somma2(4, 3)$ *si ottiene:*
$s =$
7

Osservazione 59 Dai due precedenti esempi di evince come sia diverso operare con una function oppure con uno script file: volendo sommare due variabili diverse da 4 e 3 mediante la function basta lanciare il comando iniziale con i nuovi valori da assegnare alle variabili; mediante lo script file è invece necessario andare a modificare le righe del programma in cui si è effettuata l'assegnazione delle due variabili.

Non è detto che la function abbia necessariamente degli argomenti in output; nel caso in cui non vi siano dati di output la riga di intestazione può essere scritta così:
$function\ [\ ] = NomeFunction(variabili\ di\ ingresso)$
o, più semplicemente, così:
$function\ NomeFunction(variabili\ di\ ingresso)$.

Esempio 227 *Possiamo scrivere una function che ci consenta di rappresentare la funzione seno all'interno di un assegnato intervallo: non si avranno dati di output in questa function, ma come output si otterrà il grafico della funzione richiesta.*

$function\ seno(a, b, n)$

% Rappresenta la funzione seno nell'intervallo $[a, b]$ considerando
% n punti equidistanti.
%
% Sintassi: seno(a, b, n)
%
% Dati di input
%   a: primo estremo dell'intervallo di definizione
%   b: secondo estremo dell'intervallo di definizio-
%   ne
%   n: numero di punti equidistanti da considerare
%   nell'intervallo $[a, b]$ nei quali valutare la
%   funzione

% valutiamo le ascisse:
$x = linspace(a, b, n);$

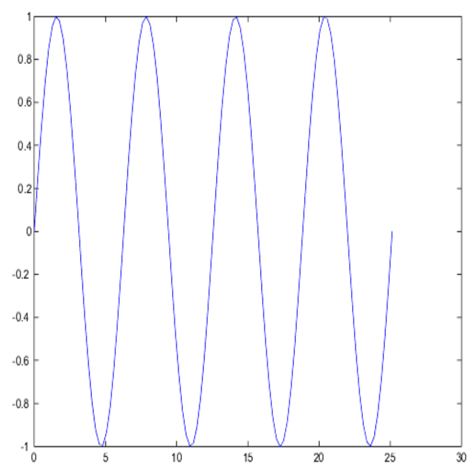
% valutiamo le ordinate:
$y = sin(x);$

% rappresentiamo la funzione:

$plot(x, y)$

Salviamo il file con il nome grafsen.m.
Lanciando dal prompt di MATLAB $\gg grafsen(0, 8*pi, 100)$ *si ottiene il grafico della funzione seno valutata in* 100 *punti equidistanti dell'intervallo* $[0, 8\pi]$*, rappresentato nella figura 5.1.*

Figura 5.1: Rappresentazione della funzione seno nell'intervallo $[0, 8\pi]$.



A differenza di uno script file, una function può accettare dati in input e utilizza al suo interno variabili che sono, per default, locali.
Questo vuol dire che le variabili definite all'interno di una function non modificano il workspace.

Esempio 228 *Supponiamo di aver eseguito la function dell'esempio precedente, dopo aver lanciato il comando* $\gg$ *clear. Lanciando* $\gg$ *whos si ottiene:*
$\gg$
Questo significa che non vi sono nuove variabili nel workspace dopo aver eseguito la function.

Analizziamo come vengono gestiti i parametri in input e in output in una function. Supponiamo di avere scritto una function che ammette come dati di output le variabili a_1, a_2, $\ldots$, a_m e come dati di input le variabili b_1, b_2, $\ldots$, b_n; la riga di intestazione nel function file sarà
$function\ [a1, a2, \ldots, am] = NomeFunction(b1, b2, \ldots, bn)$

Esempio 229 *Consideriamo il file operazioni.m:*

$function\ [s, d, p, q] = operazioni(a, b)$

% Consente di effettuare le quattro operazioni (addizione,
% sottrazione, moltiplicazione e divisione) tra due numeri
%
% Sintassi: $[s, d, p, q] = operazioni(a, b)$
%
% Dati di input
% a e b: numeri tra i quali eseguire le operazioni
% Dati di output
% s: risultato della somma tra a e b
% d: risultato della differenza tra a e b
% p: risultato del prodotto tra a e b
% q: risultato del quoziente tra a e b

$s = a + b;$
$d = a - b;$
$p = a * b;$
$q = a/b;$

Lanciando dal prompt di MATLAB
$\gg [c1, c2, \ldots, cm] = NomeFile(d1, d2, \ldots, dn)$
si ottiene quanto segue: i valori delle variabili in input d_1, d_2, …, d_n vengono scaricati nelle variabili locali b_1, b_2, …, b_n della function. Attraverso queste vengono generate le variabili (sempre locali) a_1, a_2, …, a_m i cui valori vengono riversati, alla fine dell'esecuzione, nelle variabili di output c_1, c_2, …, c_m del workspace.
Simbolicamente si può scrivere:

$function\ [a1, a2, \ldots, am] = NomeFunction(b1, b2, \ldots, bn)$

$\gg [c1, c2, \ldots, cm] = NomeFile(d1, d2, \ldots, dn)$

Esempio 230 *Supponendo di aver caricato in memoria le variabili* $m = 4$ *e* $n = 2$*, mandiamo in esecuzione la function dell'esempio precedente lanciando*
$\gg [r1, r2, r3, r4] = operazioni(m, n)$
Otteniamo:
$r1 =$

6

$r2 =$

2

$r3 =$

8

$r4 =$

2

Cosa è accaduto? I valori delle variabili del workspace m e n sono stati scaricati come dati di input della function operazioni, cioè sono stati riversati nelle variabili locali a e b della function. Mediante questi valori sono stati calcolati i dati di output della function: s, d, p e q. Questi risultati sono stati infine riversati nelle variabili $r1$, $r2$, $r3$ e $r4$.

Abbiamo visto come vengono gestiti i dati quando la linea di intestazione della function è

$function\ [a1, a2, \ldots, am] = NomeFunction(b1, b2, \ldots, bn)$

e dal prompt di MATLAB viene lanciato

$\gg [c1, c2, \ldots, cm] = NomeFile(d1, d2, \ldots, dn)$

Osserviamo che le variabili d_i e b_i e le variabili c_i e a_i rispettivamente sono diverse, cioè indirizzano a locazioni di memoria differenti; ne segue che esse possono avere lo stesso nome.

Simbolicamente:

$$function\ [a1, a2, \ldots, am] = NomeFunction(b1, b2, \ldots, bn)$$

$$\gg [a1, a2, \ldots, am] = NomeFile(b1, b2, \ldots, bn)$$

Esempio 231 *Supponendo di aver caricato in memoria le variabili $a = 4$ e $b = 2$, mandiamo in esecuzione la function precedente lanciando*

$\gg [s, d, p, q] = operazioni(a, b)$

Otteniamo:

$s =$

6

$d =$

2

$p =$

8

$q =$

2

Tranne nel caso in cui nel programma non sia prevista la possibilità di un numero variabile di dati di input (ad esempio mediante un controllo della variabile *nargin*), il numero di variabili in input nell'istruzione lanciata dal prompt di MATLAB per eseguire la function deve coincidere con il numero di variabili in input presenti nella linea di intestazione della function, altrimenti MATLAB segnala l'errore e non esegue il programma.
Il numero di variabili in output nell'istruzione lanciata dal prompt di MATLAB per eseguire la function può essere minore o uguale al numero di variabili in output presenti nella riga di intestazione della function: nel caso in cui il numero coincide MATLAB visualizza tutte le variabili di output, altrimenti vengono visualizzate solo le prime variabili (tante quante ne vengono immesse nell'istruzione lanciata dal prompt).
MATLAB lancia un messaggio d'errore quando il numero delle variabili in input o in output nell'istruzione che serve per eseguire la function supera il numero corrispondente nella function.

Esempio 232 *Abbiamo analizzato cosa accade lanciando dal prompt di MATLAB* $\gg [s, d, p, q] = operazioni(a, b)$ *cioè un'istruzione che contiene* ***tanti dati di input e di output quanti ne contiene la function****. Analizziamo tutti gli altri possibili casi:*

- ***numero di dati in input minore rispetto a quello previsto dalla function****, cioè ad esempio* $\gg [s, d, p, q] = operazioni(20)$*. Si ottiene:*
  ??? Input argument 'b' is undefined. [5]

  Error in $==> \; C : \backslash MATLABR11\backslash work\backslash operazioni.m$
  On line 17 $==> \;$ *s = a + b;* [6]

- ***numero di dati in input maggiore rispetto a quello previsto dalla function****, cioè ad esempio* $\gg [s, d, p, q] = operazioni(20, 5, 2)$*. Si ottiene:*
  ??? Error using ==> operazioni
  Too many input arguments. [7]

[5] Traduzione:
??? Il dato di input '*b*' non è stato definito

[6] Path del file (nell'ipotesi che il file operazioni.m si trovi nella cartella work di MATLAB), numero della riga e riga di istruzioni in cui è stato commesso l'errore.

[7] Traduzione:
??? Errore nell'uso di ==> operazioni
Ci sono troppi dati di input.

- ***numero di dati in output minore rispetto a quello previsto dalla function**, cioè ad esempio* $\gg [s, d] = operazioni(20, 5)$. *Si ottiene:*
  $s =$

  25

  $d =$

  15

- ***numero di dati in output maggiore rispetto a quello previsto dalla function**, cioè ad esempio* $\gg [s, d, p, q, e] = operazioni(20, 5)$. *Si ottiene:*
  ??? Error using ==> operazioni
  Too many output arguments. [8]

Per concludere, è possibile schematizzare analogie e differenze tra uno script file e un function file in una tabella:

SCRIPT	**FUNCTION**
Non ha una linea di intestazione.	Ha una linea di intestazione nella quale si definiscono i dati di input e di output.
Contiene una sequenza di istruzioni MATLAB che vengono eseguite una dopo l'altra quando il file viene lanciato.	Contiene una sequenza di istruzioni MATLAB che vengono eseguite una dopo l'altra quando il file viene lanciato.
Modifica il workspace poichè l'esecuzione di uno script file equivale all'esecuzione di ogni sua singola istruzione direttamente dal prompt di MATLAB. Uno script file non consente l'uso di variabili locali.	Tutte le variabili gestite da un function file sono locali, ovvero esistono solo durante l'esecuzione del file. Ne segue che una function non modifica il workspace.
Non accetta dati di input, ma lavora sui dati presenti nel workspace.	Ammette variabili in input e restituisce variabili di output.

[8]Traduzione:
??? Errore nell'uso di ==> operazioni
Ci sono troppi dati di output.

5.2 Programmazione strutturata

Qualsiasi programma può essere sviluppato in un linguaggio di programmazione che disponga delle tre strutture fondamentali: sequenza, selezione, iterazione.

5.2.1 Sequenza

La sequenza in MATLAB è ovviamente ottenuta dalla sequenza lessicografica delle istruzioni.

5.2.2 Selezione

La selezione si può ottenere nei seguenti modi:

- mediante il costrutto

  if prima condizione

  istruzioni

  elseif seconda condizione

  istruzioni

  ⋮

  else

  istruzioni

  end

  che può essere utilizzato anche solo come if ... *end*, if ... *else* ... *end* oppure if ... $elseif$ ... *end*.
  Se l'espressione logica che rappresenta la condizione dell'if è logicamente vera, MATLAB esegue le istruzioni comprese tra la linea dell'if e la linea dell'*end* oppure dell'$elseif$ (o *else*) successivo. L'esecuzione del programma viene ripresa dalla linea successiva all'*end* del ciclo.
  Se l'espressione logica che rappresenta la condizione dell'if è logicamente falsa, MATLAB salta tutte le istruzioni comprese tra l'if e l'$elseif$ (o *else*) successivo o la linea dell'*end*, valutando la veridicità della condizione successiva (se ce n'è una) oppure riprendendo l'esecuzione del programma dalla linea successiva all'*end*.
  La differenza tra l'uso dell'*else* e dell'$elseif$ sta nel fatto che il primo non è seguito da nessuna ulteriore condizione logica e il blocco delle istruzioni

che lo seguono viene eseguito nell'eventualità in cui nessuna delle condizioni precedenti sia verificata, invece il secondo prevede una ulteriore condizione logica da verificare.

Osservazione 60 L'istruzione if può essere fatta seguire anche soltanto da una variabile che contiene una matrice o uno scalare: la condizione logica viene considerata vera se la matrice ha tutti gli elementi diversi da 0, falsa altrimenti.

- mediante il costrutto

  switch variabile (scalare oppure stringa)

  case primo valore

  istruzioni

  case secondo valore

  istruzioni

  $\vdots$

  otherwise [9]

  istruzioni

  end

  che esegue gruppi di istruzioni dipendenti da una variabile da valutare. Nel caso in cui la variabile assuma il valore presente in uno dei possibili *case*, MATLAB esegue le istruzioni successive al *case*, arrestandosi alla linea del *case* successivo (o dell'*otherwise*) e riprendendo l'esecuzione del programma dalla prima linea successiva all'*end*.
  Nel caso in cui il valore della variabile non sia compreso in nessuno dei possibili *case*, MATLAB esegue le istruzioni successive all'*otherwise*, se questa alternativa è possibile, in caso contrario riprende l'esecuzione del programma dalla linea successiva all'*end* del ciclo.
  L'uso di questo costrutto si rivela molto utile quando la variabile da confrontare è una stringa: in tal caso si evita l'uso di *strcmp*, indispensabile nel caso in cui si vuole lavorare con if ... *end*.

Esempio 233 *Costruiamo una function che utilizza il costrutto* if ... *else ... end:*

[9] Traduzione: altrimenti.

```
function multiplo(m, n)

% Permette di stabilire se un numero è multiplo di un altro
% numero oppure no
%
% Sintassi:          multiplo(m, n)
%
% Dati di input
%            m e n:  numeri dei quali si vuole verificare
%                    se il primo è multiplo del secondo
% Dati di output
%                    non si hanno valori numerici in output,
%                    ma una stringa che fornisce la risposta
%                    del quesito

% calcoliamo il quoziente tra m ed n e valutiamone la parte
% intera:
quoz = floor(m/n);

% valutiamo il resto della divisione tra m ed n:
resto = m - (quoz * n);

% se il resto è nullo il numero m è multiplo del numero n:
if resto == 0
    str = sprintf(' %g e'' multiplo di %g ', m, n);
    disp(str)

% in caso contrario il numero m non è multiplo di n:
else
    str = sprintf(' %g non e'' multiplo di %g ', m, n);
    disp(str)
end
```

Dopo aver salvato la function con il nome multiplo.m, lanciando, ad esempio, le seguenti istruzioni, si ottiene:

$\gg multiplo(9, 3)$

9 *è multiplo di* 3

$\gg multiplo(19, 3)$
19 *non è multiplo di* 3

Esempio 234 *Costruiamo una function che utilizza il costrutto* if ... $elseif$ ... *else* ... *end:*

```
function confronto(m, n)

% Permette di stabilire se un numero è più grande, più
% piccolo oppure uguale ad un altro
%
% Sintassi:        confronto(m, n)
%
% Dati di input
%          m e n:  numeri da confrontare
% Dati di output
%                  non si hanno valori numerici in output,
%                  ma una stringa che fornisce la risposta
%                  del quesito

if m > n
    str = sprintf(' %g  e'' piu'' grande di %g ',m,n);
    disp(str)
elseif m == n
    disp('i due numeri sono uguali ')
else
    str = sprintf(' %g  e'' piu'' piccolo di %g ',m,n);
    disp(str)
end
```

Dopo aver salvato la function con il nome confronto.m, si ottiene ad esempio:
$\gg confronto(5, 4)$
5 *è più grande di* 4
$\gg confronto(-5, 4)$
-5 *è più piccolo di* 4
$\gg confronto(-5, -5)$
i due numeri sono uguali

Esempio 235 *Costruiamo una function che utilizza il costrutto switch ... case ... case ... otherwise ... end:*

```
function ris=operazione(m,n,str)

% Consente di effettuare una determinata operazione (a scelta tra
% addizione, sottrazione, moltiplicazione e divisione) tra due nu-
% meri
%
% Sintassi:          ris = operazione(m, n, str)
%
% Dati di input
%                  m e n: numeri tra i quali eseguire l'operazione
%                    str: stringa contenente l'operazione da eseguire
% Dati di output
%                    ris: risultato dell'operazione tra m ed n

switch str
case '+'
        ris = m + n;
case '-'
        ris = m - n;
case '*'
        ris = m * n;
case ':'
        ris = m/n;
otherwise
        ris = 'indeterminato';
end
```

Dopo aver salvato la function con il nome operazione.m, si ottiene ad esempio:

```
>> operazione(4, 2, '+')
ans =
      6
>> operazione(4, 2, '-')
ans =
      2
```

$\gg operazione(4, 2, \ '*' \ )$
$ans =$
        8
$\gg operazione(4, 2, \ ':' \ )$
$ans =$
        2
$\gg operazione(4, 2, \ '.' \ )$
$ans =$
$indeterminato$

5.2.3 Iterazione

L'iterazione si può ottenere nei seguenti modi:

- mediante il costrutto

  for ind = val1 : step : val2

  istruzioni

  end

  che esegue in maniera ripetuta il gruppo di istruzioni interne al ciclo per un numero di volte prestabilito, uguale al numero di volte in cui varia l'indice *ind* (detto contatore) fra il valore $val1$ e il valore $val2$ con un incremento pari a *step*. Sono consentiti incrementi positivi e negativi: per incrementi positivi nel caso in cui $val1 < val2$ il ciclo si arresta quando l'indice *ind* diventa più grande del valore di $val2$, nel caso in cui $val1 > val2$ le istruzioni interne al ciclo non vengono mai eseguite; per incrementi negativi nel caso in cui $val1 > val2$ il ciclo si arresta quando l'indice *ind* diventa più piccolo di $val2$, nel caso in cui $val1 < val2$ le istruzioni interne al ciclo non vengono mai eseguite.

  Osservazione 61 Se l'incremento *step* non è specificato esplicitamente (cioè se il ciclo è del tipo $for\ ind = val1 : val2$) esso viene scelto per default uguale a +1.

  Osservazione 62 L'istruzione for può ammettere come indice i anche soltanto il nome di una matrice A di dimensioni $m \times n$: questo equivale a considerare i uguale ai vettori $A(:, k)$, al variare di k fra 1 e n.

- mediante il costrutto

  while condizione

  istruzioni

  end

  che esegue in maniera ripetuta il gruppo di istruzioni interne al ciclo fino a quando la condizione resta verificata. Ovviamente, qualora la condizione risulti non verificata in partenza, MATLAB salta tutto il blocco delle istruzioni, passando alla linea successiva all'*end*.

Osservazione 63 L'istruzione *while* può essere fatta seguire anche soltanto dal nome di una matrice: la condizione viene considerata vera se tutti gli elementi della matrice A sono diversi da 0, falsa altrimenti. La condizione è logicamente falsa anche se A è la matrice vuota.

Esempio 236 *Costruiamo una function che utilizza il costrutto for* $ind = val1 : val2$*:*

```
function n1 = norma1(x)

% Calcola la norma 1 di un vettore x
%
% Sintassi:          n1 = norma1(x)
%
% Dati di input
%                      x: vettore di cui calcolare la norma 1
% Dati di output
%                      n1: norma 1 di x

% calcoliamo la somma dei valori assoluti degli elementi
% del vettore x, cioè la sua norma 1:
n1 = 0;
for i = 1 : length(x)
    n1 = n1 + abs(x(i));
end
```

Salvata la function con il nome norma1.m, per testarla consideriamo, ad esempio, il vettore x *e lanciamo*

$\gg n = norma1(x)$

Si ottiene:

$n =$

15

che è lo stesso risultato che si ottiene lanciando l'apposito comando MATLAB per il calcolo della norma 1 di un vettore, cioè lanciando

$\gg n = norm(x, 1)$

Esempio 237 *Costruiamo una function che utilizza il costrutto* for $ind = val1 : step : val2$ *con step negativo:*

```
function fatt=fattoriale(n)

% Calcola il fattoriale di un numero intero positivo
%
% Sintassi:         fatt = fattoriale(n)
%
% Dati di input
%                   n: numero intero del quale si intende calcolare
%                      il fattoriale
% Dati di output
%                fatt: fattoriale di n

% se n è negativo oppure non è un numero intero il fattoriale
% non è definito:
if n < 0 | fix(n) ~= n
    disp('il dato di input deve essere positivo')

% se n è maggiore di 0 il fattoriale di n è il prodotto
% dei numeri n,(n-1),(n-2), ...,2,1:
else
    fatt = 1;
    for i = n : -1 : 2
```

```
        fatt = fatt * i;
    end
end
```

Salvata la function con il nome fattoriale.m, lanciando, ad esempio, $\gg m = fattoriale(6)$ *si ottiene:*

$m =$

   720

che è lo stesso risultato che si ottiene lanciando l'apposito comando MATLAB per il calcolo del fattoriale di un numero, cioè lanciando $\gg m = factorial(6)$.

Esempio 238 *Costruiamo una function che utilizza il costrutto for ind = NomeMatr ... end*

```
function C = formatr(A, B)

% Esegue il prodotto fra due matrici sfruttando il co-
% strutto for i = NomeMatr
%
% Sintassi:        C = formatr(A, B)
%
% Dati di input
%               A e B: matrici da moltiplicare
% Dati di output
%                   C: matrice risultato del prodotto delle
%                      matrici A e B

% escludiamo il caso in cui non è possibile effettuare il
% prodotto:
[m, n] = size(A);
[o, p] = size(B);
if n ~= o
    error('dimensioni non compatibili');
end
```

```
% preallochiamo la matrice risultato, ponendo tutti i suoi
% elementi uguali a zero:
C = zeros(m,p);

% consideriamo ad una ad una tutte le colonne della matrice
% trasposta di A, che corrispondono alle righe della matrice
% A di partenza:
ind_riga = 0;
for i = A'
    % per ogni riga considerata aumentiamo di un'unità il con-
    % tatore dell'indice di riga:
    ind_riga = ind_riga + 1;

    % consideriamo ad una ad una tutte le colonne della matri-
    % ce B:
    ind_colonna = 0;
    for j = B
        % per ogni colonna considerata aumentiamo di un'unità
        % il contatore dell'indice di colonna:
        ind_colonna = ind_colonna + 1;

        % nell'elemento corrispondente all'indice di riga e
        % all'indice di colonna che si stanno considerando del-
        % la matrice risultato memorizziamo il prodotto della
        % colonna della matrice A' per la colonna della matrice
        % B in esame (ovviamente è necessaria una trasposizione
        % del primo vettore per poter effettuare questo prodotto
        % scalare):
        C(ind_riga,ind_colonna) = i' * j;
    end
end
```

Salviamo il file con il nome formatr.m e costruiamo, ad esempio, le matrici $A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$ *e* $B = \begin{bmatrix} 3 & 6 & 9 \\ 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix}$.

Lanciando $\gg formatr(A, B)$ *si ottiene:*

$ans =$

$$\begin{array}{rrr} 18 & 43 & 68 \\ 24 & 56 & 88 \\ 30 & 69 & 108 \end{array}$$

che è lo stesso risultato che si otterrebbe lanciando $\gg A * B$.

Esempio 239 *Costruiamo una function che utilizza il costrutto while ... end:*

```
function [an, n] = successione(tol, nmax)


% Calcola il valore verso cui converge la successione
%        a1 = 1;
%                          an^2 + 6
%        a(n + 1)    =    − − − − − − − −
%                              5
% nei margini della precisione richiesta.
%
% Sintassi:        [an, n] = successione(tol, nmax)
%
% Dati di input
%                       tol: precisione richiesta
%                      nmax: numero massimo di iterate consentite
% Dati di output
%                        an: valore dell'ultimo termine della successione
%                            considerato
%                         n: numero di iterate effettuate dall'algoritmo
%                            per approssimare il valore della successio-
%                            ne entro la precisione richiesta


% assegnamo il primo e il secondo elemento della successione,
% che si ottengono per n = 1 e per n = 2:
```

```
n = 1;
a1 = 1;
n = 2;
a2 = 7/5;

% inizializziamo il residuo:
residuo = abs(a2 - a1)/(1 + a2);

% finché il residuo non è minore della precisione richiesta
% effettuiamo le seguenti operazioni:
while residuo > tol & n <= nmax

    % aumentiamo di un'unità l'indice della successione:
    n = n + 1;

    % valutiamo il nuovo termine della successione:
    an = (a2^2 + 6)/5;

    % aggiorniamo il residuo:
    residuo = abs(an - a2)/(1 + an);

    % fissiamo come punto di partenza a2 per il calcolo della
    % somma successiva il valore di an calcolato:
    a2 = an;
end
```

Salviamo il file con il nome successione1.m e lanciamo:

```
>> [val, it] = successione1(1e-2, 100)
```

Si ottiene:

```
val =

    1.9023

it =

     8
```

Questo significa che l'algoritmo ha approssimato, entro una precisione

dell'ordine di 10^{-2}*, il valore verso cui converge la successione con il valore* 1.9023*, effettuando* 8 *iterate.*
Considerando una precisione maggiore, si ottiene una migliore approssimazione del valore verso cui converge la successione, ma un numero maggiore di iterate necessarie per raggiungere tale valore. Il lettore può verificarlo lanciando, ad esempio:
$\gg [val, it] = successione1(1e-3, 100)$
$\gg [val, it] = successione1(1e-4, 100)$
$\gg [val, it] = successione1(1e-5, 100)$
$\gg [val, it] = successione1(1e-6, 100)$

Osservazione 64 Si può arrestare l'esecuzione di un ciclo for oppure $while$ mediante l'istruzione **break**: quando questa istruzione è inserita all'interno di un ciclo, l'esecuzione riprende dalla riga successiva all'*end* del ciclo.

Osservazione 65 L'esecuzione di un M-file può essere interrotta in qualsiasi punto inserendo nel file l'istruzione **return**: quando MATLAB, durante l'esecuzione delle istruzioni del file, incontra questa istruzione, arresta l'esecuzione del file a quel punto.

Osservazione 66 Per ottenere una velocità maggiore di esecuzione di un algoritmo è preferibile, quando possibile, vettorizzare le variabili che intervengono negli M-file. Dove gli altri programmi procedono unicamente attraverso cicli for e *do* [10], MATLAB consente l'utilizzo di matrici e vettori.

Esercizio 33 *Si verifichi che* $\sum_{x=1}^{n} x = \frac{n*(n+1)}{2}$ *, scrivendo un'opportuna function.*
Soluzione.

Esercizio 34 *Si scriva una function per il calcolo della norma* 2 *di un assegnato vettore.*
Soluzione.

Esercizio 35 *Si scriva una function per il calcolo della norma infinito di un assegnato vettore.*
Soluzione.

[10] In alcuni linguaggi di programmazione, come per esempio Fortran, un ciclo *do* consente di effettuare iterativamente una serie di istruzioni, come in MATLAB il ciclo for.

Esercizio 36 *Si scriva una function per il calcolo della norma di Frobenius di un'assegnata matrice.*
Soluzione.

Esercizio 37 *Si scriva una function per il calcolo della norma* 1 *di un'assegnata matrice.*
Soluzione.

Esercizio 38 *Si verifichi, attraverso esempi numerici, che la successione definita per ricorrenza:*
$$\begin{cases} a_1 = 1 \\ a_{n+1} = \frac{a_n^2+6}{5} \end{cases}$$
converge al valore 2.
Soluzione.

Esercizio 39 *Si verifichi che la somma della serie* $\displaystyle\sum_{n=1}^{\infty} \frac{1}{n*(n+2)}$ *è uguale a* $\dfrac{3}{4}$.
Soluzione.

5.3 Il comando *diary*

Il comando **diary** viene utilizzato per memorizzare in un file una sequenza di operazioni che si sono eseguite dal prompt di MATLAB allo scopo di poterle rivedere in sessioni o tempi successivi.
Per utilizzare questa istruzione bisogna digitare il file direttamente dal prompt di MATLAB, anziché dall'editor. L'istruzione *diary NomeFile* permette di memorizzare nel file NomeFile tutte le istruzioni scritte dall'utente fino a quella che precede il *diary* successivo e tutti i messaggi d'errore lanciati da MATLAB in fase di esecuzione.

Esempio 240 *Supponiamo di lanciare, dal prompt di MATLAB, le seguenti istruzioni:*
$\gg$ *diary scrivi.m*
$\gg v = 0$
$v =$
$\quad 0$
$\gg for\ i = 1 : 11$
$x(i) = v;$

y(i) = sen(v);
v = v + 6.28;
end
??? Undefined function or variable 'sen'. [11]

≫ diary

In questo modo viene creato il file scrivi.m all'interno del quale si può leggere (mediante ≫ edit scrivi.m) quanto segue:
v = 0
v =
0
for i = 1 : 11
x(i) = v;
y(i) = sen(v);
v = v + 6.28;
end
??? Undefined function or variable 'sen'.

diary

Come si può osservare, tutte le istruzioni inserite dal prompt sono state riportate in un file, compresi i messaggi d'errore.
Si può notare che, se alla fine di una linea di istruzioni viene omesso il punto e virgola, la variabile calcolata in quella riga di istruzioni viene visualizzata anche nel file ottenuto con il comando diary.

5.4 Variabili globali

Abbiamo visto che nelle function le variabili interne sono tutte locali, quindi per utilizzare i valori delle variabili che compaiono in una function all'esterno della function stessa bisogna passarle come dati di output e, analogamente, per utilizzare variabili esterne in una function bisogna inserire i valori delle variabili come dati di input. In alternativa MATLAB dispone dell'istruzione **global**: mediante *global NomeVar* la variabile *NomeVar* viene resa global-

[11] Traduzione:
??? La funzione o variabile 'sen' non è definita.

mente visibile. Essa sarà condivisa da tutte le function (o dal workspace di MATLAB) in cui la variabile viene dichiarata globale.

Osservazione 67 Ovviamente l'istruzione *global* $NomeVar$ deve essere inserita in una function prima che la variabile stessa venga utilizzata.

Esempio 241 *Supponiamo di voler risolvere il seguente problema di cinematica mediante due function: un corpo, partito dal punto* $x0 = 0$ *all'istante* $t0 = 0$ *con velocità iniziale* $v0 = 0$ *e soggetto ad un'accelerazione uniforme* $a = 5\ m/s^2$*, ha raggiunto la velocità* $v = 30\ m/s$*. Quanto spazio ha percorso?*

Supponiamo di aver scritto le due seguenti function tempo.m e spazio.m:

- $function\ t = tempo(v0, v, t0, a)$

  % La seguente function consente di ricavare il tempo trascorso
  % dall'istante iniziale $t0$ *all'istante in cui il corpo, partito*
  % con velocità $v0$ *e soggetto all'accelerazione uniforme* a*, ha*
  % raggiunto la velocità v*, sfruttando la relazione:*
  % $v = v0 + a * (t - t0)$

  $t = (v - v0)/a + t0$;

- $function\ s = spazio(x0, v0, t0, t, a)$

  % La seguente function consente di ricavare lo spazio percorso
  % all'istante t *da un corpo partito dal punto* $x0$ *all'istante* $t0$
  % con velocità $v0$ *e soggetto all'accelerazione uniforme* a*, sfrut-*
  % tando la relazione:
  % $x = x0 + v0 * (t - t0) + 1/2 * a * (t - t0)$^2

  $s = x0 + v0 * (t - t0) + 1/2 * a * (t - t0)$^2;

Come è evidente, i dati $v0$*,* $t0$ *ed* a *sono comuni alle due function. Inoltre la variabile* t *che si ricava in output dalla prima function è necessaria in ingresso nella seconda function.*
Per utilizzare gli stessi dati si può procedere nei due seguenti modi:

1. *assegnare i valori comuni nel workspace e far si che le due function li leggano come dati di input:*
   $\gg v0 = 0$; $t0 = 0$; $a = 5$;
   $\gg t = tempo(v0, 30, t0, a)$
   $t =$

   6

   $\gg s = spazio(0, v0, t0, t, a)$
   $s =$

   90

2. *sfruttare l'istruzione global, modificando le due function tempo.m e spazio.m nelle nuove function tempo1.m e spazio1.m:*

   - $function\ t = tempo1(v)$

     global v0 t0 a t
     $t = (v - v0)/a + t0$;

   - $function\ s = spazio1(x0)$

     global v0 t0 a t
     $s = x0 + v0 * (t - t0) + 1/2 * a * (t - t0)$^2;

   Le variabili v0, t0 e a verranno assegnate dall'esterno, quindi devono essere condivise con il workspace. Questo significa che l'istruzione global deve essere lanciata anche dal prompt di MATLAB per queste tre variabili. Invece la variabile t calcolata nella prima function viene utilizzata anche nella seconda. Per questo la variabile t deve essere condivisa dalle due function.
   Per ottenere gli stessi risultati precedenti bisogna lanciare:
   $\gg$ *global v0 t0 a*
   $\gg v0 = 0$; $t0 = 0$; $a = 5$;
   $\gg t = tempo1(30)$
   $t =$

   6

   $\gg s = spazio1(0)$
   $s =$

   90

5.5 Le variabili *nargin* e *nargout*

Abbiamo visto che le function in MATLAB permettono di avere più parametri in ingresso ed in uscita. MATLAB memorizza il numero di variabili in ingresso nella variabile **nargin**, quelle in uscita nella variabile **nargout**.
Utilizzate all'interno di una function, le variabili *nargin* e *nargout* indicano rispettivamente quante variabili in ingresso e in uscita un utente fornisce alla function.
Utilizzate all'esterno di qualsiasi function *NomeFunction*, cioè dal prompt mediante la sintassi *nargin('NomeFunction')* e *nargout('NomeFunction')* forniscono rispettivamente il numero di parametri in input e in output per una data funzione.

Esempio 242 *Consideriamo il seguente function file:*

```
function [r, it] = radquad(a, nmax, tol)

% Calcola la radice quadrata aritmetica del numero (positi-
% vo) a mediante la successione:
% x(n + 1) = 1/2 * (x(n) + a/x(n))
%
% Sintassi:        [r, it] = radquad(a, nmax, tol)
%
% Dati di input
%                a: numero del quale si intende calcolare la
%                   radice quadrata
%             nmax: numero massimo di iterate consentite
%              tol: precisione richiesta
% Dati di output
%                r: radice quadrata di a
%               it: numero di iterate impiegate dall'algoritmo
%                   per ottenere il risultato

% escludiamo le radici immaginarie:
if a < 0
    error('radice quadrata immaginaria')
end
```

```
% inizializziamo la variabile logica che contiene la condizione
% d'arresto, il punto della successione e il numero di iterate:
arresto = 0;
x0 = a;
n = 0;

% valutiamo il numero di dati in input: se questo non è uguale a tre,
% vuol dire che non sono state assegnate tutte le variabili in input;
% in questo caso assegnamo quelle che mancano per default:
switch nargin

case 1
    % nel caso in cui in ingresso venga fornito soltanto il valore del
    % numero del quale calcolare la radice, assegnamo per default i va-
    % lori di nmax e tol:
    nmax = 100;
    tol = 1e-3;

case 2
    % nel caso in cui in ingresso venga fornito soltanto il valore del
    % numero del quale calcolare la radice e il numero massimo di itera-
    % te consentite, assegnamo per default il valore di tol:
    tol = 1e-3;
end

% effettuiamo ciò che segue finché la condizione d'arresto
% non è verificata e finché il numero di iterate effettua-
% te non supera il numero massimo di iterate consentite:
while ~arresto & n <= nmax

    % aumentiamo di un'unità il numero di iterate effettuate:
    n = n + 1;

    % valutiamo un altro punto della successione:
    x1 = 1/2 * (x0 + a/x0);

    % verifichiamo se si è raggiunta o meno la precisione ri-
```

```
    % chiesta:
    arresto = abs(x1 - x0)/(1 + abs(x1)) < tol;

    % consideriamo il punto x1 come punto iniziale per
    % il calcolo del nuovo punto della successione:
    x0 = x1;
end

% assegnamo alla radice quadrata il valore dell'ultimo punto
% calcolato e al numero di iterate eseguito il valore di n:
r = x0;
it = n;
```

Salvata la function con il nome radquad.m, per testarla lanciamo:

$\gg [r, it] = radquad(8, 100, 1e-3)$

Si ottiene:

```
r =
      2.8284
it =
        5
```

Questo significa che l'algoritmo ha approssimato, entro una precisione dell'ordine di 10^{-3}*, la radice quadrata del numero* 8 *(che si può calcolare lanciando* $\gg sqrt(8)$*) con il valore* 2.8284*, effettuando* 5 *iterate.*

Il lettore, al fine di verificare come opera questa function nel caso in cui si lanci la riga di intestazione con numeri di variabili in input e in output differenti, può lanciare, ad esempio, le seguenti istruzioni, che sono rappresentative di tutti i casi possibili:

$\gg [r, it] = radquad(2, 100, 1e-9)$

$\gg [r, it] = radquad(2, 100)$

$\gg [r, it] = radquad(2)$

$\gg r = radquad(2, 100, 1e-9)$

$\gg radquad(2, 100, 1e-9)$

5.6 Correzione dei programmi

È molto probabile che nella digitazione di un programma si commettano alcuni errori, dei quali ci si può accorgere soltanto in fase di esecuzione. Il processo di *debugging* consiste nell'isolare un problema all'interno di un M-file e nel risolverlo.

5.6.1 Tipi di errori e modalità di correzione

Gli errori che si possono commettere all'interno di un file sono di due tipi:

- **errori di sintassi**: consistono nello scrivere in maniera scorretta il nome di una variabile o di una funzione, nel dimenticare di chiudere una parentesi ... Questo tipo di errori può essere facilmente individuato e corretto servendosi dei messaggi d'errore lanciati da MATLAB in fase di esecuzione del file.

  Esempio 243 $\gg sen(2 * pi)$
  ??? Undefined function or variable 'sen'. [12]

  Come si può osservare, si è scritta la funzione seno (sin) in maniera sbagliata e MATLAB lo ha segnalato.

- **errori logici** (o **run-time errors**): sono quelli che si commettono a livello concettuale all'interno dell'algoritmo (per esempio impostando operazioni non corrette) e che sono evidenti quando MATLAB restituisce risultati diversi da quelli attesi in teoria.

  Esempio 244 *Consideriamo il seguente script file (che salviamo con il nome script_err.m) nel quale vogliamo memorizzare nella matrice C il risultato del prodotto di una matrice 3×3 per un vettore 1×3, teoricamente sbagliato:*

  $A = eye(3, 3)$;
  $B = ones(1, 3)$;
  $C = A * B$;

  Eseguendo il file (mediante $\gg$ script_err) MATLAB lancia il seguente messaggio di errore:

[12] Traduzione:
??? La funzione o variabile 'sen' non è definita.

$??? \; Error \; using == > \; *$
$Inner \; matrix \; dimensions \; must \; agree.$ [13]

$Error \; in ==> C : \backslash MATLABR11 \backslash work \backslash script\_err.m$
$On \; line \; 3 ==>$

Osservazione 68 Come è evidente dall'esempio, MATLAB segnala l'errore commesso, il path del file in cui è stato individuato l'errore e la linea del file alla quale si trova l'errore.

Vediamo come si possono individuare e correggere gli errori negli M-file in MATLAB, analizzando diversi casi a seconda della modalità in cui si sta lavorando:

- quando si sta lavorando in **modalità interattiva** (cioè direttamente dal prompt di MATLAB), si ha una visualizzazione immediata degli eventuali errori sintattici commessi: se una linea di istruzioni contiene un errore, MATLAB lo segnala, indicando anche il comando per il quale è stato commesso l'errore.
  Può risultare utile la visualizzazione di una o più variabili che intervengono nell'area di lavoro, al fine di correggere eventuali errori logici: per fare questo è sufficiente non digitare il punto e virgola alla fine della riga di istruzioni in cui si assegna o si calcola la variabile cui si è interessati.

  Osservazione 69 Nel caso in cui si voglia interrompere la visualizzazione di una variabile (per esempio perché questa operazione richiede troppo tempo) è sufficiente premere contemporaneamente i tasti **Ctrl** e **c**.

- abbiamo visto che utilizzare uno **script file** in MATLAB equivale a far eseguire un blocco di istruzioni, come se si stesse lavorando in modalità interattiva. Questo vuol dire che la correzione di uno script file è analoga al tipo di correzione analizzata nel caso precedente: quando viene eseguito il file, MATLAB visualizza il messaggio relativo al primo degli eventuali errori commessi e la riga del file alla quale l'errore si trova, bloccando l'esecuzione.

[13] Traduzione:
??? Errore nell'uso di $== > \; *$
Le dimensioni interne delle matrici devono coincidere.

Esempio 245 *Consideriamo il seguente script file som_primi_pari.m:*

```
% Effettua la somma dei primi 100 numeri pari
%
% Sintassi:        som_primi_pari

somma = 0;
num = 0;
for i = 1 : 100
    num = num + 2;
    somma = somma + nu;
end
str = sprintf('La somma dei primi 100 numeri pari e'' %g', somma);
disp(str)
```

Come si può osservare, nella seconda riga del ciclo for è stato commesso un errore: il nome della variabile num è stato digitato in maniera scorretta. Eseguendo il file, MATLAB visualizza l'errore:

```
>> som_primi_pari:
??? Undefined function or variable 'nu'. [14]

Error in ==> C:\MATLABR11\work\som_primi_pari.m
On line 9 ==> somma = somma + nu;
```

Apportando la correzione e salvando le modifiche, il file verrà eseguito in maniera corretta (in quanto non ci sono altri errori):

```
>> som_primi_pari:
La somma dei primi 100 numeri pari è 10100
```

- nella maggior parte dei casi in MATLAB vengono usati function file. Per correggere questo tipo di file si può procedere come già visto per gli script file (cioè eseguendo la function e aspettando che MATLAB segnali gli eventuali errori), ma questa è la strada più scomoda. Ci sono altri metodi per correggere una function:

[14] Traduzione:
??? La funzione o variabile 'nu' non è definita.

– si può interrompere temporaneamente l'esecuzione del file utilizzando il comando *keyboard*;

– si può utilizzare il Debugger di MATLAB.

Osservazione 70 Ovviamente questi ultimi metodi per correggere gli errori possono essere utilizzati anche nel caso in cui si stia lavorando in modalità interattiva oppure con uno script file, anche se non sono necessari: in questi due casi non c'è il problema che le variabili sono soltanto locali, come nel caso in cui si stia utilizzando una function.

5.6.2 Il comando *keyboard*

Il comando **keyboard** è un efficace strumento per la correzione di un M-file: il suo posizionamento all'interno di un M-file provoca l'interruzione momentanea dell'esecuzione del file a quel punto. Da quel momento in poi il comando viene dato alla tastiera (in inglese keyboard), attraverso la quale si può prendere visione delle variabili utilizzate fino a quel momento. Questo risulta particolarmente utile quando si sta lavorando con una function: diversamente non si potrebbe avere accesso alle variabili utilizzate nel file, essendo queste locali e quindi non disponibili nel workspace dopo l'esecuzione.
Inserendo il comando *keyboard* all'interno di un file, dopo che l'utente lancia dal prompt l'istruzione che serve per eseguire il file, MATLAB esegue il file fino al punto in cui esso è inserito e trasforma il prompt da
$\gg$
in
$K \gg$
per segnalare la sospensione dell'esecuzione.
Per riprendere l'esecuzione è sufficiente scrivere, accanto al nuovo prompt, il comando **return** e premere il tasto INVIO. L'esecuzione riprende dal punto in cui è stata sospesa fino al *keyboard* successivo o fino alla fine del file (se non ci sono altri *keyboard*).
Quando viene raggiunta la fine del file il prompt ritorna ad essere quello classico (cioè $\gg$) e le variabili locali tornano ad essere non visibili nell'area di lavoro.

Esempio 246 *Consideriamo il seguente file somma_primi_pari.m:*

```
function somma_primi_pari(n)

% Effettua la somma dei primi n numeri pari
%
% Sintassi:          somma_primi_pari(n)
%
% Dati di input
%                 n: numero di interi pari da sommare
% Dati di output
%                    in output si ottiene una stringa che forni-
%                    sce il risultato della somma richiesta

somma = 0;
if n ~= 0
    num = 0;
    for i = 1 : n
        num = num + 2;
        somma = somma + num;
    end
end
str = sprintf('La somma dei primi %g numeri pari e'' %g',n,somma);
disp(str)
```

Aggiungiamo alla fine del ciclo for il comando keyboard (dopo l'end) e salviamo il file con le modifiche apportate.
Supponiamo di non avere variabili nel workspace (se ce ne sono è sufficiente lanciare ≫ clear).
Lanciando dal prompt
≫ *somma_primi_pari*(3)
si ottiene:
K ≫
Questo significa che l'esecuzione è stata interrotta e che il comando è stato dato alla tastiera. Lanciando a questo punto il comando
K ≫ *whos*
si può prendere visione di tutte le variabili intervenute nel file fino a quel momento, cioè i, n, num e somma.
Per far riprendere l'esecuzione è sufficiente digitare

$K \gg$ *return*
da cui si ottiene:
La somma dei primi 3 *numeri pari è* 12
$\gg$
Si può osservare che lanciando a questo punto il comando
$\gg$ *whos*
non viene visualizzata nessuna variabile: questo perché è stata raggiunta la fine del file quindi le variabili della function (che sono locali) non possono più essere visualizzate.

Osservazione 71 Se prima di lanciare la function fossero state presenti delle variabili nel workspace, queste non sarebbero state elencate in seguito al comando $K \gg$ *whos*: sarebbero state visualizzate anche in questo caso soltanto le variabili intervenute nella function fino a quel momento, cioè i, n, *num* e *somma*.

Si possono inserire anche più *keyboard* nello stesso file: in tal modo è possibile isolare e quindi correggere eventuali errori di programmazione.

5.6.3 Il Debugger

Il Debugger di MATLAB aiuta l'utente ad individuare gli eventuali errori commessi all'interno degli M-file: se è molto semplice correggere gli errori sintattici mediante i messaggi di errore lanciati da MATLAB, non è altrettanto semplice individuare gli errori concettuali, soprattutto se si stanno utilizzando function file, per i quali le variabili utilizzate sono locali e quindi non gestibili dal workspace dopo che si è eseguito il file. Il Debugger viene quindi utilizzato soprattutto per individuare errori concettuali non segnalati esplicitamente da MATLAB.
Usando il Debugger è possibile sospendere l'esecuzione del file, controllare il contenuto del workspace raggiunto fino a quel punto dell'esecuzione, apportare al file le modifiche desiderate e poi continuare l'esecuzione interrotta.
Analizziamo alcuni fra i più importanti comandi del Debugger di MATLAB:

- innanzitutto occorre sapere come iniziare una sessione di Debugger:
  - se il file da esaminare è stato scritto mediante l'Editor/Debugger di MATLAB si può procedere;
  - se il file è stato scritto con un editor esterno, bisogna aprirlo con l'editor di MATLAB prima di procedere;

Esempio 247 *Apriamo, con l'editor di MATLAB il file som_primi_pari (lanciando ≫ edit som_primi_pari, nell'ipotesi che il file sia stato salvato nella cartella work di MATLAB, altrimenti bisogna prima raggiungere la cartella in cui è stato salvato il file e poi lanciare questa istruzione) per iniziare una sezione di Debugger.*

Osservazione 72 Ricordiamo che nel file è stato commesso un errore nella scrittura del nome della variabile *num* all'interno del ciclo for, quindi un errore sintattico ma, vista la semplicità del file in questione, lo utilizziamo negli esempi che seguono per comprendere il funzionamento del Debugger.

- il Debugger consente di porre all'interno di un file dei punti di arresto (breakpoint) alla linea specificata, alla quale l'esecuzione del file verrà bloccata: questo consente di esaminare le variabili presenti nell'area di lavoro fino a quel momento e, eventualmente, cambiare parte della function prima che l'esecuzione riprenda. Tutto questo si può realizzare in diversi modi:
  - lanciando dal prompt l'istruzione **dbstop in** $NomeFile$ che arresta l'esecuzione del file NomeFile alla prima istruzione eseguibile, saltando le linee di commento.
    Dopo aver lanciato l'istruzione, sullo schermo riappare il prompt di MATLAB, mentre nel file, accanto alla prima istruzione eseguibile, appare un pallino rosso, che indica che l'esecuzione verrà arrestata in quel punto.
    Eseguendo il file, sullo schermo appare il prompt $K \gg$, mentre nel file, accanto al pallino rosso, appare anche una freccetta gialla rivolta verso destra, che denota che l'esecuzione è in corso ed è stata bloccata prima dell'istruzione segnalata dalla freccia.

    Esempio 248 *Supponiamo di voler arrestare alla prima linea eseguibile l'esecuzione del file som_primi_pari: è sufficiente lanciare*
    ≫ dbstop in som_primi_pari
    Se si guarda l'editor, si vede che affianco all'istruzione somma $= 0$
    (che è la prima eseguibile) è apparso un pallino rosso.
    Nel workspace, invece, si ottiene:
    ≫

da cui è possibile lanciare il comando per eseguire il file, cioè:
$\gg$ *som_primi_pari*
Nel workspace si ottiene:
$K \gg$
e nel file è apparsa la freccetta gialla accanto al pallino rosso.
È possibile vedere quali variabili sono presenti nel workspace fino a quel punto dell'esecuzione lanciando:
$K \gg$ *whos*
Supponendo che in memoria non fossero precedentemente presenti altre variabili, si ottiene:
$K \gg$
cioè non sono presenti variabili fino a quel punto dell'esecuzione.

– se si vuole interrompere l'esecuzione del file NomeFile ad una linea precisa (non necessariamente coincidente con la prima), si può lanciare dal prompt **dbstop in** $NomeFile$ **at** $NumLinea$, che arresta l'esecuzione del file NomeFile alla riga numero $NumLinea$. Si osservi che le righe vanno conteggiate dalla prima riga di intestazione o di commento, includendo nel conto anche le eventuali righe bianche.

Osservazione 73 Un'alternativa al conteggio delle linee è quella di posizionare nell'editor il cursore del mouse sulla linea cui si è interessati: il numero corrispondente viene indicato nell'angolo in basso a destra della finestra dell'editor.

Osservazione 74 L'esempio che segue non potrà essere eseguito mentre si stanno provando gli esempi precedenti: deve prima essere conclusa la sessione di Debugger cominciata, lanciando $\gg$ *dbquit* (per arrestare il Debugger) e $\gg$ *dbclear all* (per eliminare i breakpoint inseriti). L'esempio può essere eseguito al posto del precedente, per poi continuare con la sessione di Debugger (cioè con gli esempi successivi).

Esempio 249 *Supponiamo di voler arrestare l'esecuzione del file som_primi_pari all'ottava riga: lanciamo*
$\gg$ *dbstop in som_primi_pari at* 8
Si ottiene una situazione analoga a quella dell'esempio precedente, solo che il pallino rosso compare accanto all'ottava riga (num =

$num + 2$;).
Eseguendo il file (mediante l'istruzione $\gg$ som_primi_pari), nel workspace si ottiene il prompt K $\gg$, che segnala che l'esecuzione è stata interrotta; nell'editor, accanto al pallino rosso, appare la freccetta gialla. Lanciando
K $\gg$ whos
si ottiene, in questo caso, che le nuove variabili presenti in memoria sono i, num e somma (cioè le variabili intervenute nel file fino alla ottava riga).

- si può posizionare il cursore del mouse accanto alla linea alla quale si intende arrestare l'esecuzione e

Osservazione 75 Se il breakpoint viene inserito ad una linea non valida (cioè una linea che non è eseguibile) l'esecuzione viene arrestata alla prima linea eseguibile successiva al breakpoint.

- nel caso in cui si voglia prendere visione del numero corrispondente alla linea alla quale si intende arrestare l'esecuzione del file NomeFile in una sessione di Debugger già avviata è sufficiente lanciare **dbtype** *NomeFile*, che visualizza tutte le linee di programma del file con i numeri corrispondenti.

Esempio 250 *Supponiamo di trovarci nella situazione finale dell'esempio precedente e lanciamo:*
K $\gg$ dbtype som_primi_pari
Si ottiene:

```
1       % Effettua la somma dei primi 100 numeri pari
2       %
3       % Sintassi:        som_primi_pari
4
5       somma = 0;
6       num = 0;
7       for i = 1 : 100
8           num = num + 2;
9           somma = somma + nu;
10      end
```

```
11    str = sprintf('La somma dei primi 100 numeri pari e'' %g',somma);
12    disp(str)
```

$K \gg$

- per riprendere l'esecuzione dopo aver inserito un breakpoint è sufficiente lanciare, accanto al cursore $K \gg$, l'istruzione **dbcont**, che riprende l'esecuzione dal punto in cui è stata interrotta e si arresta al breakpoint successivo (se ce n'è uno) oppure alla fine del file.

  Esempio 251 *Nella situazione dell'esempio precedente, lanciamo*
  $K \gg$ dbcont
  Si ottiene il messaggio d'errore visto in precedenza. Questo significa che l'errore commesso si trova tra l'ottava riga e la riga finale (e infatti MATLAB segnala, in questo caso molto semplice, che l'errore è stato commesso alla nona linea).
  Poichè nel workspace riappare il prompt tradizionale ($\gg$) è evidente che l'esecuzione del file è terminata in quanto non ci sono altri breakpoint. Per riprendere a lavorare sullo stesso file è necessario cancellare i breakpoint inseriti (infatti MATLAB li memorizza e li conserva finché non vengono esplicitamente eliminati), mediante l'istruzione dbclear all.

- se si vuole avanzare di una riga per volta nel file non conviene utilizzare *dbstop* specificando ogni volta il numero di linea successivo, ma è sufficiente lanciare **dbstep**. Questa istruzione deve essere lanciata in una sessione di Debugger.

  Esempio 252 *Supponiamo di aver lanciato l'istruzione:*
  $\gg$ dbstop in som_primi_pari
  con la quale, come abbiamo visto, nel workspace compare il prompt tradizionale ($\gg$) e nel file appare il pallino rosso accanto alla prima riga eseguibile.
  Mandiamo in esecuzione il file con $\gg$ som_primi_pari. Sappiamo che questo fa apparire nel workspace il prompt $K \gg$ e nel file la freccetta gialla accanto al pallino rosso.
  Lanciando $K \gg$ whos possiamo notare che non ci sono nel workspace variabili nuove rispetto a quelle preesistenti.
  Se vogliamo far proseguire l'esecuzione, bloccandola al rigo successivo, lanciamo:

$K \gg$ dbstep
Nel file la freccetta gialla viene spostata alla riga successiva, per denotare che l'esecuzione è stata sospesa in quel punto.
Nel workspace appare ancora $K \gg$, ma questa volta lanciando
$K \gg$ whos
si trova una nuova variabile, somma, che è stata ottenuta dall'esecuzione della prima riga eseguibile.

- se si vuole avanzare di n righe per volta nel file è possibile lanciare **dbstep n**.

Esempio 253 *Supponiamo di essere nella situazione finale dell'esempio precedente e di voler avanzare di altre 2 righe: è sufficiente lanciare*
$K \gg$ dbstep 2
da cui si ottiene ancora il prompt $K \gg$ nel workspace, mentre nel file la freccetta avanza di due righe, segnalando il nuovo punto di arresto dell'esecuzione.

Osservazione 76 Quando l'avanzamento mediante il comando *dbstep* oppure *dbstep n* dovrebbe concludersi all'ultima riga eseguibile del file, nell'editor, accanto all'ultima riga, appare una freccetta gialla rivolta verso il basso: questo segnala che l'esecuzione si sarebbe dovuta concludere perché è stata raggiunta la fine del file, ma è stata arrestata da MATLAB per consentire all'utente la visualizzazione delle variabili presenti nella function, prima che esse tornino ad essere non più visualizzabili.

Esempio 254 *Sospendiamo la sessione di Debugger dell'esempio precedente, lanciando $\gg$ dbquit e $\gg$ dbclear all.*
Consideriamo il file somma_primi_pari e fissiamo questo breakpoint:
$\gg$ dbstop in somma_primi_pari at 22
Eseguiamo la function per n uguale a 10:
$\gg$ somma_primi_pari(10)
Il pallino rosso e la freccetta gialla sono apparsi nell'editor accanto all'ultima riga di istruzioni, a significare che l'esecuzione è stata sospesa prima dell'ultima riga. Supponiamo di voler far avanzare l'esecuzione del file di una riga: lanciamo
$K \gg$ dbstep
in questo modo viene raggiunta la fine del file, quindi nel workspace si

ottiene il risultato della function, cioè:
La somma dei primi 10 *numeri pari è* 110
Ma non è finita la sessione di Debugger, infatti nel workspace c'è ancora il prompt $K \gg$ *e, andando a visualizzare l'editor sul monitor, osserviamo che la freccetta gialla alla fine del file è rivolta verso il basso. Questo consente di visualizzare le variabili intervenute nel programma prima che non sia più possibile farlo, lanciando* $K \gg$ *whos.*
Per terminare l'esecuzione bisogna lanciare $K \gg$ *dbcont, che fa finire la sessione di Debugger e fa ricomparire il prompt* $\gg$*.*

- per programmi più complessi (che chiamano dall'interno di una function altre function) è facile perdersi in una serie di diramazioni, quindi può risultare utile il comando **dbstack**, che consente di visualizzare in quale punto ci si trova.

  Esempio 255 *Supponiamo di essere nella situazione finale dell'esempio che precede l'ultima osservazione e di voler vedere dove ci troviamo. Lanciando*
  $K \gg$ *dbstack*
  si ottiene:
  $>$ *In* $C:\backslash MATLABR11\backslash work\backslash som\_primi\_pari.m$ *at line* 8
  $K \gg$
  (nell'ipotesi che il file fosse stato salvato nella cartella work di MATLAB).

- il comando **dbstatus** $NomeFunction$ elenca i punti di arresto definiti nella function $NomeFunction$.
  Il comando *dbstatus* (senza il nome di alcuna function) fornisce invece tutti i breakpoint della sessione di Debugger in corso (la differenza con il caso precedente è evidente soltanto nel caso in cui si stia utilizzando una function che richiama altre function, altrimenti le due sintassi del comando sono equivalenti).

  Esempio 256 *Supponiamo di essere nella situazione finale dell'esempio precedente e di lanciare*
  $K \gg$ *dbstatus som_primi_pari (che equivale, in questo caso, a lanciare* $K \gg$ *dbstatus).*
  Si ottiene:

Breakpoint for $C:\backslash MATLABR11\backslash work\backslash som\_primi\_pari$ *is on line* 5 *(nell'ipotesi che il file fosse stato salvato nella cartella work di MATLAB), che indica che l'unico breakpoint inserito si trova alla quinta riga del file som_primi_pari (del quale viene fornito anche il path).*

Osservazione 77 Se nella sessione di Debugger in corso non sono stati definiti breakpoint, MATLAB lo segnala con un messaggio.

Esempio 257 *Supponiamo di non aver ancora lanciato la sessione di Debugger (e quindi di non aver fissato alcun breakpoint) e di lanciare:*
$\gg$ *dbstatus som_primi_pari*
si ottiene
No breakpoints are set for som_primi_pari. [15]

- per cancellare un breakpoint alla linea n della function $NomeFunction$ si può utilizzare il comando **dbclear in** $NomeFunction$ **at** n oppure si può, dopo essersi posizionati con il mouse alla linea
- per cancellare tutti i breakpoint all'interno della function $NomeFunction$ si può utilizzare il comando **dbclear all in** $NomeFunction$.
- per cancellare i breakpoint da tutte le function della sessione di Debugger in corso si utilizza **dbclear all**.

Esempio 258 *Riprendiamo l'esempio precedente l'osservazione e supponiamo di voler cancellare il breakpoint inserito. Si può realizzare questo utilizzando una delle seguenti possibilità:*
$K \gg$ *dbclear in som_primi_pari at* 5
$K \gg$ *dbclear all in som_primi_pari*
$K \gg$ *dbclear all*
In ogni caso il breakpoint viene eliminato.
L'esecuzione resta ferma dove era stata interrotta (infatti la freccetta gialla nell'editor è sulla ottava riga come in precedenza e il prompt è ancora $K \gg$*). Lanciando* $K \gg$ *dbcont essa verrà ripresa e terminata.*

[15] Traduzione:
Non è stato fissato alcun breakpoint in *som_primi_pari*.

Osservazione 78 Il comando *dbclear* (con una delle varie opzioni possibili) può essere utilizzato anche dopo essere usciti dalla sessione di Debugger, per eliminare i breakpoint inseriti, che restano memorizzati nel file, finché non vengono esplicitamente eliminati.

Esempio 259 *Supponiamo di far partire una sessione di Debugger con l'istruzione*
$\gg$ *dbstop in som_primi_pari at* 6
Questo inserisce un breakpoint alla sesta riga del file. Eseguiamo il file mediante
$\gg$ *som_primi_pari*
L'esecuzione si ferma al breakpoint. Per riprenderla lanciamo
K $\gg$ *dbcont*
Si ottiene, alla fine, il prompt $\gg$ *nel workspace. Esaminando il file, però, notiamo che il pallino rosso è ancora presente, cioè il breakpoint non è stato eliminato alla fine dell'esecuzione. Per eliminarlo è sufficiente lanciare:*
$\gg$ *dbclear in som_primi_pari at* 6
oppure
$\gg$ *dbclear all in som_primi_pari*
oppure, più semplicemente,
$\gg$ *dbclear all*

- per uscire da una sezione di Debugger è sufficiente digitare **dbquit**.

  Esempio 260 *Supponiamo di far partire una sessione di Debugger con l'istruzione*
  $\gg$ *dbstop in som_primi_pari at* 6
  Questo inserisce un breakpoint alla sesta riga del file. Eseguiamo il file mediante
  $\gg$ *som_primi_pari*
  L'esecuzione si ferma al breakpoint.
  Supponiamo di voler interrompere la sessione di Debugger prima della fine dell'esecuzione del file: è sufficiente lanciare
  K $\gg$ *dbquit.*

Osservazione 79 Se durante la sessione di Debugger si decide di voler cambiare una parte della function, MATLAB lancia un avviso. Proseguendo con la modifica, viene sospesa la sessione di Debugger.

Esempio 261 *Supponiamo di far partire una sessione di Debugger con l'istruzione*
$\gg$ *dbstop in som_primi_pari at* 6
e eseguiamo il file mediante
$\gg$ *som_primi_pari*
Supponiamo poi di accorgerci di aver scritto in maniera sbagliata la variabile num nel ciclo for *e di andarla a correggere nell'editor. Prima di eseguire la modifica, MATLAB fa apparire una finestra nella quale scrive Editing this file will end your debugging session. Would you like to continue editing?* [16]*, nella quale è necessario scegliere tra OK oppure Annulla prima di poter proseguire. Apportando la modifica (dopo aver selezionato OK) nel workspace appare il prompt* $\gg$*, mentre il breakpoint non viene modificato. Salvando il file, il breakpoint scompare.*

Osservazione 80 Tutto ciò che abbiamo visto finora per un unico breakpoint all'interno di un file vale anche quando all'interno dello stesso file si inseriscono più breakpoint: questo consente di correggere un file suddividendone l'analisi in più parti.

Esercizio 40 *Si scriva un programma per il calcolo del coefficiente binomiale*
$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{k!(n-k)!} = \frac{n*(n-1)*\ldots*(n-k+1)}{k!}$$
scrivendo una function (binomiale.m) per calcolare il coefficiente binomiale, all'interno della quale venga chiamata la function per il calcolo del fattoriale di un numero, fattoriale.m, analizzata in precedenza.
Si faccia partire una sessione di Debugger, fissando un breakpoint alla ventunesima linea di binomiale.m e uno alla ventiduesima linea di fattoriale.m (si faccia riferimento ai file riportati nella soluzione dell'esercizio).
Quindi si calcoli il coefficiente binomiale $\begin{pmatrix} 15 \\ 4 \end{pmatrix}$*, osservando a quali punti si arresta l'esecuzione dei file nell'editor.*
Si elimini il breakpoint in fattoriale.m e si esegua nuovamente la function per il calcolo del coefficiente binomiale $\begin{pmatrix} 15 \\ 4 \end{pmatrix}$*. Quando l'esecuzione si arresta si avanzi nel file di* 2 *linee eseguibili, si verifichi ciò che accade nell'editor e si analizzi quali sono le variabili presenti nel workspace. Si controlli in quale posizione ci si trova e si sospenda la sessione di Debugger. Si eliminino tutti*

[16]Traduzione:
Apportando modifiche a questo file la tua sessione di debugging verrà sospesa. Vuoi continuare a editare?

i breakpoint.
Soluzione.

Esercizio 41 *Sfruttando le function fattoriale.m e binomiale.m scritte in precedenza, si costruisca una function che valuti, in un generico punto t, il polinomio di Bernstein così definito:*

$$B_{n,k}(t) = \left(\begin{array}{c} n \\ k \end{array}\right) * t^k * (1-t)^{n-k} \qquad \forall \begin{array}{c} n \\ k \end{array} \in \mathbb{N}, \;\; n \geq k, \;\; \forall t \in [0,1]$$

adottando le seguenti convenzioni: $\begin{cases} B_{n,k}(0) = 1 & se\ k = 0 \\ B_{n,k}(0) = 0 & se\ k > 0 \\ B_{n,k}(1) = 1 & se\ k = n \\ B_{n,k}(1) = 0 & se\ k < n \\ B_{n,k}(t) = 0 & se\ k > n \end{cases}$

Si valuti quindi il polinomio di Bernstein $B_{5,3}$ *nel punto* $t = 0.5$.
Soluzione.

Appendice A

Applicazioni all'analisi numerica

A.1 Programmazione

In questo paragrafo vengono mostrati alcuni algoritmi più complessi, relativi al capitolo Programmazione.

A.1.1 Algoritmo per il calcolo del prodotto scalare tra due vettori

Analizziamo un programma che utilizza un ciclo if ... end e un ciclo for $ind = val1 : val2$ ... end per calcolare il prodotto scalare tra due vettori:

$function\ p = vettvett(x, y)$

% Calcola il prodotto scalare di un vettore x per un vettore y
%
% Sintassi: $p = vettvett(x, y)$
%
% Dati di input
% x e y: vettori da moltiplicare scalarmente
% Dati di output
% p: risultato del prodotto scalare del vettore x
% per il vettore y

% escludiamo il caso in cui non è possibile effettuare
% il prodotto:

$nx = length(x);$
$ny = length(y);$
$if\ nx \sim= ny$
    $error('i\ \ due\ \ vettori\ \ devono\ \ avere\ \ la\ \ stessa\ \ lunghezza')$
end

% inizializziamo il risultato ponendolo uguale a zero:
$p = 0;$

% sommiamo tutti i prodotti fra gli elementi omologhi
% dei due vettori:
$for\ i = 1 : nx$
    $p = p + x(i) * y(i);$
end

Salvata la function con il nome vettvett.m, per testarla si può lanciare, ad esempio:
$\gg vettvett(x, y)$
Si ottiene:
$ans =$
    65
che è lo stesso risultato che si otterrebbe effettuando il prodotto scalare tra i vettori x e y, cioè lanciando $\gg x * y'$.

A.1.2 Algoritmo per il calcolo del prodotto tra una matrice ed un vettore

Analizziamo un programma che utilizza il costrutto
$for\ ind = val1 : val2$
    $for\ ind = val3 : val4$
        istruzioni
    end
end
per calcolare il prodotto tra una matrice ed un vettore:

$function\ y = matrvett(A, x)$

```
% Calcola il prodotto di una matrice A per un vettore x
%
% Sintassi:        y = matrvett(A,x)
%
% Dati di input
%                  A: matrice della quale si intende effettuare il
%                     prodotto per un vettore
%                  x: vettore da moltiplicare per la matrice
% Dati di output
%                  y: vettore prodotto della matrice A per il
%                     vettore x

% escludiamo il caso in cui non è possibile effettuare
% il prodotto:
[m,n] = size(A);
l = length(x);
if n ~= l
    error('dimensioni incompatibili')
end

% preallochiamo la memoria per il vettore risultato come vettore colon-
% na di lunghezza m, inizialmente nullo:
y = zeros(m,1);

% per ogni elemento del vettore risultato effettuiamo la se-
% guente costruzione:
for i = 1:m

    % inizializziamo a zero la somma che intendiamo effettuare:
    somma = 0;

    % sommiamo tutti i prodotti fra gli elementi della i-esima
    % riga della matrice A per i corrispondenti elementi del vet-
    % tore x:
    for j = 1:n
        somma = somma + A(i,j)*x(j);
```

end

% assegnamo all'elemento i-esimo del vettore risultato il va-
% lore della somma calcolata:
$y(i) = somma;$
end

Salvata la function con il nome matrvett.m, per testarla si può lanciare, ad esempio:
$\gg matrvett(A, c)$
Si ottiene:
$ans =$
80
90
90
80
che è lo stesso risultato che si otterrebbe lanciando $\gg A * c$.

A.1.3 Algoritmo per il calcolo del prodotto tra due matrici

Analizziamo un programma che utilizza il costrutto
$for\ ind = val1 : val2$
$for\ ind = val3 : val4$
$for\ ind = val5 : val6$
istruzioni
end
end
end
per calcolare il prodotto tra due matrici:

$function\ C = matrmatr(A, B)$

% Calcola il prodotto di una matrice A per una matrice B
%
% Sintassi: $y = matrmatr(A, B)$
%

```
% Dati di input
%                A e B: matrici da moltiplicare
% Dati di output
%                    C: matrice prodotto della matrice A per la
%                       matrice B

% escludiamo il caso in cui non è possibile effettuare
% il prodotto:
[m1, n1] = size(A);
[m2, n2] = size(B);
if n1 ~= m2
    error('dimensioni incompatibili')
end

% preallochiamo la matrice risultato:
C = zeros(m1, n2);

% per ogni riga e per ogni colonna della matrice risultato
% effettuiamo la seguente costruzione:
for i = 1 : m1
    for j = 1 : n2

        % inizializziamo a zero la somma che intendiamo effettuare:
        somma = 0;

        % sommiamo tutti i prodotti fra gli elementi della i-esima
        % riga della matrice A per gli elementi della j-esima colon-
        % na della matrice B:
        for k = 1 : n1
            somma = somma + A(i, k) * B(k, j);
        end

        % assegnamo all'elemento sulla i-esima riga e sulla j-esima
        % colonna della matrice risultato il valore della somma cal-
        % colata:
        C(i, j) = somma;
    end
```

end

Salvata la function con il nome matrmatr.m, per testarla si può lanciare, ad esempio:

$\gg matrmatr(C, D)$

Si ottiene:

$ans =$

$$\begin{matrix} -2 & 5 & 1 \\ -4 & 11 & 3 \end{matrix}$$

che è lo stesso risultato che si otterrebbe lanciando $\gg C * D$.

Analogamente, lanciando

$\gg matrmatr(G, H)$

si ottiene:

$ans =$

$$\begin{matrix} -2 & -6 \\ 41 & 46 \end{matrix}$$

che è lo stesso risultato che si otterrebbe lanciando $\gg G * H$.

A.1.4 Algoritmo di sostituzione in avanti

Analizziamo un programma che utilizza il costrutto

```
for ind = val1 : val2
        for ind = val3 : val4
                istruzioni
        end
end
```

per risolvere un sistema triangolare inferiore:

```
function x = trianginf(A, b)

% Consente di risolvere il sistema Ax = b con A matrice triango-
% lare inferiore mediante le relazioni:
%
%                b(1)
% x(1)   =   - - -- ;                                          (1)
%              A(1,1)
%
%              b(2) - A(2,1) * x(1)
```

```
% x(2)  =  − − − − − − − − − − − ;                              (2)
%                    A(2,2)
%
%                      ⋮
%                      ⋮
%
%             b(m) − (A(m,1) ∗ x(1) + ... + A(m,m − 1) ∗ x(m − 1))
% x(m)  =  − − − − − − − − − − − − − − − − − − − − − − − − − − −;      (m)
%                              A(m,m)
%
% Sintassi:          x = trianginf(A,b)
%
% Dati di input
%                    A: matrice del sistema
%                    b: vettore dei termini noti
% Dati di output
%                    x: vettore risultato

% escludiamo il caso in cui non è possibile effettuare
% il prodotto:
[m,n] = size(A);
l = length(b);
if m ~= l | n ~= m
    error( 'dimensioni  non  compatibili ' )
end

% preallochiamo il vettore risultato:
x = zeros(m,1);

% valutiamo gli elementi del vettore risultato mediante le relazioni
% (1), (2), ... (m):
for i = 1 : m

    % calcoliamo la somma dei prodotti degli elementi sulla riga
    % i-esima e sulla colonna j-esima della matrice A per i corri-
    % spondenti elementi x(j) del vettore, per j = 1,...,i − 1:
    somma = 0;
```

$for\ j = 1 : i - 1$

$somma = somma + A(i, j) * x(j);$

end

% calcoliamo l'incognita $x(i)$:

$x(i) = (b(i) - somma)/A(i, i);$

end

Salviamo la function con il nome trianginf.m e consideriamo la matrice P e il vettore z. La soluzione x del sistema $P * x = z$ si può calcolare lanciando
$\gg x = trianginf(P, z)$
da cui si ottiene:
$x =$

0.0833
0.9583
6.5833
8.2569
−5.3458

A.1.5 Algoritmo di sostituzione all'indietro

Analizziamo un programma che utilizza il costrutto
$for\ ind = val1 : val2$

$for\ ind = val3 : val4$

istruzioni

end

end

per risolvere un sistema triangolare superiore:

$function\ x = triangsup(A, b)$

% Consente di risolvere il sistema $Ax = b$ con A matrice triango-
% lare superiore mediante le relazioni:
%
% $\qquad\qquad b(m)$
% $x(m) \ = \ - - -- \ ;$ (1)
% $\qquad\qquad A(m, m)$

```
%
%                        b(m-1) - A(m-1,m) * x(m)
% x(m-1)  =  - - - - - - - - - - - - - - - - - - - ;                    (2)
%                           A(m-1,m-1)
%
%                                  ⋮
%                                  ⋮
%
%            b(1) - (A(1,2) * x(2) + A(1,3) * x(3) + ... + A(1,m) * x(m))
% x(1) = - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -;   (m)
%                                  A(1,1)
%
% Sintassi:          x = triangsup(A,b)
%
% Dati di input
%                    A: matrice del sistema
%                    b: vettore dei termini noti
% Dati di output
%                    x: vettore risultato

% escludiamo il caso in cui non è possibile effettuare
% il prodotto:
[m,n] = size(A);
l = length(b);
if m ~= l | n ~= m
    error( 'dimensioni  non  compatibili' )
end

% preallochiamo il vettore risultato:
x = zeros(m,1);

% valutiamo gli elementi del vettore risultato mediante le relazioni
% (1), (2), ... (m):
for i = m : -1 : 1

    % calcoliamo la somma dei prodotti degli elementi sulla riga
    % i-esima e sulla colonna j-esima della matrice A per i corri-
```

$\%$ spondenti elementi $x(j)$ del vettore, per $j = i+1, \ldots, m$:
$somma = 0;$
$for\ j = i+1 : m$
    $somma = somma + A(i,j) * x(j);$
end

$\%$ calcoliamo l'incognita $x(i)$:
$x(i) = (b(i) - somma)/A(i,i);$
end

Salviamo la function con il nome triangsup.m e consideriamo la matrice Q e il vettore z. La soluzione x del sistema $Q * x = z$ si può calcolare lanciando
$\gg x = triangsup(Q, z)$
da cui si ottiene:
$x =$
$$\begin{array}{r} -0.7847 \\ 6.5833 \\ 3.3333 \\ 0.1667 \\ 1.0000 \end{array}$$

Appendice B

Soluzioni degli esercizi proposti

B.1 Capitolo 1

Soluzione esercizio 3 Aperto MATLAB con la procedura descritta nel paragrafo Come avviare MATLAB, si può leggere la directory in cui ci si trova mediante l'istruzione $\gg$ *cd*. Viene visualizzato:
$C : \backslash MATLABR11 \backslash work$
Ciò significa che ci si trova nell'unità di memoria C, nella sottodirectory work della directory $MATLABR11$.
Per passare alla directory $C : \backslash MATLABR11 \backslash extern$ bisogna prima uscire dalla directory work mediante l'istruzione $\gg$ *cd*␣.. e poi lanciare l'istruzione $\gg$ *cd extern*.
Con l'istruzione $\gg$ *dir* viene visualizzato l'elenco delle sottodirectory presenti in extern, che sono:

.	..	examples	include	lib	src

Per passare alla directory examples si lancia l'istruzione $\gg$ *cd examples*.
Lanciando $\gg$ *dir mex* si ottiene la lista dei file contenuti nella directory mex:

.	mexcpp.cpp	mexget.c	yprime.m
..	mexeval.c	mexgetarray.c	yprimef.f
explore.c	mexeval.m	mexload.c	yprimefg.f
explore.dll	mexevalstring.c	mexload.m	
mexatexit.c	mexfeval.c	mexlock.c	
mexatexit.cpp	mexfeval.m	mexsettrapflag.c	
mexcallmatlab.c	mexfunction.c	yprime.c	

Si passa nella directory mex con il comando $\gg$ *cd mex* e si può vedere quali sono i file con estensione .m, .mat e i file MEX di questa directory con l'istruzione $\gg$ *what* che restituisce:

M-files in the current directory $C:\backslash MATLABR11\backslash extern\backslash examples\backslash mex$

mexeval mexfeval mexload yprime

MEX-files in the current directory $C:\backslash MATLABR11\backslash extern\backslash examples\backslash mex$

explore

Per visualizzare l'output sullo schermo leggendo 15 righe per volta si attivi l'opzione *more* mediante l'istruzione $\gg$ *more*(15). Si può leggere il file yprime.m mediante $\gg$ *type yprime.m* che visualizza la prima videata:

function yp = yprime(t,y)
% Differential equation system for restricted three body problem.
% Think of a small third body in orbit about the earth and moon.
% The coordinate system moves with the earth-moon system.
% The 1-axis goes through the earth and the moon.
% The 2-axis is perpendicular, in the plane of motion of the third body.
% The origin is at the center of gravity of the two heavy bodies.
% Let mu = the ratio of the mass of the moon to the mass of the earth.
% The earth is located at $(-mu,0)$ and the moon at $(1-mu,0)$.
% $y(1)$ and $y(3)$ = coordinates of the third body.
% $y(2)$ and $y(4)$ = velocity of the third body
%.
% Copyright (c) $1984-98$ by The MathWorks, Inc.
% All Rights Reserved.
$--more--$

Premendo una volta la barra della tastiera vengono visualizzate anche le 15 righe successive, cioè:

function yp = yprime(t,y)
% Differential equation system for restricted three body problem.
% Think of a small third body in orbit about the earth and moon.
% The coordinate system moves with the earth-moon system.

% The 1-axis goes through the earth and the moon.
% The 2-axis is perpendicular, in the plane of motion of the third body.
% The origin is at the center of gravity of the two heavy bodies.
% Let mu = the ratio of the mass of the moon to the mass of the earth.
% The earth is located at ($-mu$,0) and the moon at ($1-mu$,0).
% $y(1)$ and $y(3)$ = coordinates of the third body.
% $y(2)$ and $y(4)$ = velocity of the third body
%.

$mu = 1/82.45;$
$mus = 1 - mu;$
$r1 = norm([y(1) + mu, y(3)]);$ % Distance to the earth
$r2 = norm([y(1) - mus, y(3)]);$ % Distance to the moon
$yp(1) = y(2);$
$yp(2) = 2 * y(4) + y(1) - mus * (y(1) + mu)/r1\hat{}3 - mu * (y(1) - mus)/r2\hat{}3;$
$yp(3) = y(4);$
$yp(4) = -2 * y(2) + y(3) - mus * y(3)/r1\hat{}3 - mu * y(3)/r2\hat{}3;$
$yp = yp';$

$\gg$

Per disattivare l'opzione *more* si lancia $\gg$ $more\ \ off$.
Lanciando $\gg$ $which\ \ yprimef.f$ viene visualizzato:
$C : \backslash MATLABR11\backslash extern\backslash examples\backslash mex\backslash yprimef.f$
che ci dice dove si trova il file richiesto.

B.2 Capitolo 2

Soluzione esercizio 4 Scrivendo $\gg +3$ (oppure $\gg 3$) MATLAB visualizza:
$ans =$
    3
Scrivendo $\gg +3.52$ (oppure $\gg 3.52$) MATLAB visualizza:
$ans =$
    3.5200
Scrivendo $\gg -4.02$ MATLAB visualizza:

$ans =$

-4.0200

Soluzione esercizio 5 Scrivendo $\gg 2e6$ MATLAB visualizza:

$ans =$

2000000

Scrivendo $\gg 3e-2$ MATLAB visualizza:

$ans =$

0.0300

Soluzione esercizio 6 Uno dei possibili modi per soddisfare la richiesta è quello di scrivere $\gg 4-6i$, che restituisce:

$ans =$

$4.0000 - \ 6.0000i$

Nel paragrafo sono illustrate tutte le altre possibilità.

Soluzione esercizio 7 Si possono considerare numeri a piacere: a titolo di esempio vengono considerati, oltre a $\pm Inf$ e NaN, i numeri $+5$, 0 e -5. Addizione e sottrazione:

$\gg +Inf+5$
$ans =$
Inf

$\gg -Inf+5$
$ans =$
$-Inf$

$\gg +Inf-5$
$ans =$
Inf

$\gg -Inf-5$
$ans =$
$-Inf$

$\gg +Inf+0$
$ans =$
Inf

$\gg -Inf+0$
$ans =$
$-Inf$

$\gg +Inf+Inf$
$ans =$
Inf

$\gg -Inf+Inf$
$ans =$
NaN

$\gg +Inf-Inf$
$ans =$
NaN

$\gg -Inf-Inf$
$ans =$
$-Inf$

$\gg +Inf + NaN$
$ans =$
NaN

$\gg -Inf + NaN$
$ans =$
NaN

$\gg +Inf - NaN$
$ans =$
NaN

$\gg -Inf - NaN$
$ans =$
NaN

$\gg NaN + 5$
$ans =$
NaN

$\gg NaN + Inf$
$ans =$
NaN

$\gg NaN - 5$
$ans =$
NaN

$\gg NaN - Inf$
$ans =$
NaN

$\gg NaN + 0$
$ans =$
NaN

$\gg NaN + NaN$
$ans =$
NaN

<u>Moltiplicazione</u>:

$\gg +Inf * 5$
$ans =$
Inf

$\gg -Inf * 5$
$ans =$
$-Inf$

$\gg +Inf * (-5)$
$ans =$
$-Inf$

$\gg -Inf * (-5)$
$ans =$
Inf

$\gg +Inf * 0$
$ans =$
NaN

$\gg -Inf * 0$
$ans =$
NaN

$\gg +Inf * (+Inf)$
$ans =$
Inf

$\gg -Inf * (+Inf)$
$ans =$
$-Inf$

$\gg +Inf * (-Inf)$
$ans =$
$-Inf$

$\gg -Inf * (-Inf)$
$ans =$
Inf

$\gg +Inf * NaN$
$ans =$
NaN

$\gg -Inf * NaN$
$ans =$
NaN

$\gg NaN * 5$
$ans =$
NaN

$\gg NaN * (+Inf)$
$ans =$
NaN

$\gg NaN * (-5)$
$ans =$
NaN

$\gg NaN * (-Inf)$
$ans =$
NaN

$\gg NaN * 0$
$ans =$
NaN

$\gg NaN * NaN$
$ans =$
NaN

<u>Divisione a destra</u>:

$\gg +Inf/5$
$ans =$
Inf

$\gg -Inf/5$
$ans =$
$-Inf$

$\gg +Inf/(-5)$
$ans =$
$-Inf$

$\gg -Inf/(-5)$
$ans =$
Inf

$\gg +Inf/0$
Warning: Divide by zero.
$ans =$
Inf

$\gg -Inf/0$
Warning: Divide by zero.
$ans =$
$-Inf$

$\gg +Inf/(+Inf)$
$ans =$

$\gg -Inf/(+Inf)$
$ans =$

NaN

NaN

$\gg +Inf/(-Inf)$
$ans =$
NaN

$\gg -Inf/(-Inf)$
$ans =$
NaN

$\gg +Inf/NaN$
$ans =$
NaN

$\gg -Inf/NaN$
$ans =$
NaN

$\gg NaN/5$
$ans =$
NaN

$\gg NaN/(+Inf)$
$ans =$
NaN

$\gg NaN/(-5)$
$ans =$
NaN

$\gg NaN/(-Inf)$
$ans =$
NaN

$\gg NaN/0$
Warning: Divide by zero.
$ans =$
NaN

$\gg NaN/NaN$
$ans =$
NaN

<u>Divisione a sinistra</u>:

$\gg +Inf\backslash 5$
$ans =$
0

$\gg -Inf\backslash 5$
$ans =$
0

$\gg +Inf\backslash(-5)$
$ans =$
0

$\gg -Inf\backslash(-5)$
$ans =$
0

$\gg +Inf\backslash 0$
$ans =$
0

$\gg -Inf\backslash 0$
$ans =$
0

$\gg +Inf\backslash(+Inf)$

$\gg -Inf\backslash(+Inf)$

$ans =$
NaN

$ans =$
NaN

$\gg +Inf\backslash(-Inf)$
$ans =$
NaN

$\gg -Inf\backslash(-Inf)$
$ans =$
NaN

$\gg +Inf\backslash NaN$
$ans =$
NaN

$\gg -Inf\backslash NaN$
$ans =$
NaN

$\gg NaN\backslash 5$
$ans =$
NaN

$\gg NaN\backslash(+Inf)$
$ans =$
NaN

$\gg NaN\backslash(-5)$
$ans =$
NaN

$\gg NaN\backslash(-Inf)$
$ans =$
NaN

$\gg NaN\backslash 0$
$ans =$
NaN

$\gg NaN\backslash NaN$
$ans =$
NaN

Elevamento a potenza:

$\gg +Inf\hat{\ }5$
$ans =$
Inf

$\gg -Inf\hat{\ }5$
$ans =$
$-Inf$

$\gg +Inf\hat{\ }(-5)$
$ans =$
0

$\gg -Inf\hat{\ }(-5)$
$ans =$
0

$\gg +Inf\hat{\ }0$
$ans =$
1

$\gg -Inf\hat{\ }0$
$ans =$
-1

$\gg +Inf\hat{\ }(+Inf)$

$\gg -Inf\hat{\ }(+Inf)$

$ans =$
$\quad Inf$

$ans =$
$\quad -Inf$

$\gg +Inf$^$(-Inf)$
$ans =$
$\quad 0$

$\gg -Inf$^$(-Inf)$
$ans =$
$\quad 0$

$\gg +Inf$^NaN
$ans =$
$\quad NaN$

$\gg -Inf$^NaN
$ans =$
$\quad NaN$

$\gg NaN$^5
$ans =$
$\quad NaN$

$\gg NaN$^$(+Inf)$
$ans =$
$\quad NaN$

$\gg NaN$^(-5)
$ans =$
$\quad NaN$

$\gg NaN$^$(-Inf)$
$ans =$
$\quad NaN$

$\gg NaN$^0
$ans =$
$\quad 1$

$\gg NaN$^NaN
$ans =$
$\quad NaN$

Soluzione esercizio 8 Il m.c.m. si ottiene mediante $\gg lcm(15, 20)$ che restituisce:
$ans =$
60
mentre il M.C.D. si ottiene da $\gg gcd(15, 20)$ che restituisce:
$ans =$
5

Soluzione esercizio 9 Il numero può essere inserito mediante l'istruzione $\gg n = 2 + 3i$, che restituisce:
$n =$
2.0000 + 3.0000i
Il suo modulo si calcola mediante $\gg abs(n)$ che restituisce:
$ans =$
3.6056

mentre il suo angolo di fase è dato da $\gg angle(n)$ da cui si ottiene:

$ans =$

0.9828

La richiesta successiva può essere soddisfatta lanciando $\gg eps = n * pi$, da cui si ricava:

$eps =$

$6.2832 + \ 9.4248i$

Lanciando $\gg conj(eps)$ si ottiene il complesso coniugato di eps, cioè:

$ans =$

$6.2832 - \ 9.4248i$

L'arrotondamento richiesto si ottiene mediante $\gg ceil(ans)$ che restituisce:

$ans =$

$7.0000 - \ 9.0000i$

Per ripristinare il valore predefinito di eps basta lanciare $\gg clear\ \ eps$. Lanciando $\gg eps$ si ottiene:

$ans =$

$2.2204e - 016$

che è il valore predefinito di MATLAB.

Soluzione esercizio 10 Si consideri un numero complesso a piacere $z = x + iy$, ad esempio $z = 2 + 3i$.

- La prima uguaglianza può essere verificata nel seguente modo:
  $\gg sin(2) * cosh(3) + i * cos(2) * sinh(3)$
  $ans =$

  $9.1545 - \ 4.1689i$

  $\gg sin(2 + 3 * i)$
  $ans =$

  $9.1545 - \ 4.1689i$

  Come si vede l'identità è verificata;

- la seconda uguaglianza può essere verificata nel seguente modo:
  $\gg (exp(1)\hat{}(i * z) - exp(1)\hat{}(-i * z))/(2 * i)$
  $ans =$

  $9.1545 - \ 4.1689i$

  $\gg sin(z)$
  $ans =$

$9.1545 - \ 4.1689i$

Come si vede l'identità è verificata.

Lo stesso procedimento può essere utilizzato per verificare tutte le altre uguaglianze.

Soluzione esercizio 11 Si consideri un numero complesso a piacere $z = x + iy$, ad esempio $z = 2 + 3i$.

- La prima uguaglianza può essere verificata nel seguente modo:
  $\gg -i * log(i * z + (1 - z\hat{}2)\hat{}(1/2))$
  $ans =$

  $0.5707 + \ 1.9834i$
  $\gg asin(z)$
  $ans =$

  $0.5707 + \ 1.9834i$
  Come si vede l'identità è verificata.

Lo stesso procedimento può essere utilizzato per verificare tutte le altre uguaglianze.

Soluzione esercizio 12 Il valore di e si ottiene mediante $\gg e = exp(1)$, che restituisce:
$e =$
2.7183
La variabile a si ottiene mediante l'istruzione $\gg a = log2(e)$, che restituisce:
$a =$
1.4427
La variabile b si ottiene mediante l'istruzione $\gg b = log10(e)$, che restituisce:
$b =$
0.4343
Il resto della divisione intera tra a e b (che memorizziamo nella variabile $resto$) si ottiene lanciando $\gg resto = rem(a, b)$
$resto =$
0.1398
L'approssimazione razionale di questo risultato è data da:
$\gg rats(resto)$
$ans =$
282/2017
mentre la sua espansione razionale è data da:
$\gg rat(resto)$

$ans =$

$0 + 1/(7 + 1/(7 + 1/(-2 + 1/(-4 + 1/(5)))))$

Soluzione esercizio 13 Le variabili a e b si costruiscono con le istruzioni seguenti:

$\gg a = realmin * 1e308$

$a =$

2.2251

$\gg b = realmax\hat{}(1/1000)$

$b =$

2.0335

La somma delle due variabili è data da

$\gg c = a + b$

$c =$

4.2586

quindi la variabile d si ottiene mediante l'istruzione:

$\gg d = pow2(c)$

$d =$

19.1414

Per arrotondare d all'intero più vicino si lancia:

$\gg l = round(d)$

$l =$

19

La radice cubica di l è data da:

$\gg l\hat{}(1/3)$

$ans =$

2.6684

Per arrotondare questo valore verso lo 0 si usa:

$\gg m = fix(ans)$

che restituisce:

$m =$

2

La parte reale e la parte immaginaria di m sono rispettivamente:

$\gg real(m)$

$ans =$

2

$\gg imag(m)$

$ans =$

$\quad 0$

B.3 Capitolo 3

Soluzione esercizio 14 Per memorizzare $vett1$ si può usare l'istruzione
$\gg vett1 = [\ -2 \quad 12 \quad 0 \quad -14\ ]$.
Per memorizzare $vett2$ si possono usare le istruzioni
$\gg vett2 = [\ -2 \quad ; \quad 12 \quad ; \quad 0 \quad ; \quad -14\ ]$ oppure
$\gg vett2 = [\ -2 \quad 12 \quad 0 \quad -14\ ]\,'$ oppure
$\gg vett2 = [\ -2$
12
0
$-14\ ]$.

Soluzione esercizio 15 Si può costruire la matrice elemento per elemento con l'istruzione
$\gg I = [\ 1 \quad 0 \quad 0 \quad 0 \quad 0$
$0 \quad 1 \quad 0 \quad 0 \quad 0$
$0 \quad 0 \quad 1 \quad 0 \quad 0$
$0 \quad 0 \quad 0 \quad 1 \quad 0$
$0 \quad 0 \quad 0 \quad 0 \quad 1\ ]$
oppure mediante l'istruzione $\gg I = eye(5)$.

Soluzione esercizio 16 Lanciando l'istruzione:
$\gg v = [\ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1\ ]\,'$
si ottiene il vettore
$v =$

$$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{array}$$

Osservazione 81 Poteva essere utilizzata equivalentemente l'istruzione
$\gg v = [\ zeros(5, 1)$
$1\ ]$

La variabile v può essere salvata nel file in formato ASCII V.DAT mediante l'istruzione
$\gg save\ \ V.DAT\ \ v\ \ -ascii.$

Osservazione 82 L'ultima istruzione utilizzata salva il vettore v nel file ASCII che abbiamo chiamato V.DAT; questo significa che il vettore viene salvato con il nome V.

Per effettuare la sostituzione richiesta si può procedere in questo modo: aperto il blocco note di Windows e raggiunto il file V.DAT seguendo il suo path (che è $C:\backslash MATLABR11\backslash work$), si cambia l'ultimo elemento del vettore in 2, come richiesto e si salva il file. Si esce dal blocco note e, da MATLAB, si lancia $\gg load\ \ V.DAT$, che carica in memoria il nuovo vettore.
Per visualizzare gli elementi del vettore modificato si può lanciare $\gg V$, che restituisce:
$V =$

$$\begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \end{matrix}$$

Soluzione esercizio 17 Si consideri, ad esempio, la matrice A: $\gg A + A'$ restituisce:
$ans =$

$$\begin{matrix} 32 & 8 & 11 & 17 \\ 8 & 20 & 17 & 23 \\ 11 & 17 & 14 & 26 \\ 17 & 23 & 26 & 2 \end{matrix}$$

che, come si vede, è una matrice simmetrica.

Osservazione 83 Ovviamente, la matrice che si considera in questo esercizio deve essere quadrata, in modo tale che la trasposta sia delle stesse dimensioni della matrice di partenza e quindi sia possibile effettuare la somma delle due matrici.

Soluzione esercizio 18 Si consideri, ad esempio, la matrice A: $\gg A * A'$ restituisce:
$ans =$

438	236	332	150
236	310	278	332
332	278	310	236
150	332	236	438

che, come si vede, è una matrice simmetrica. Analogamente $\gg A' * A$ restituisce una matrice simmetrica:

$ans =$

378	212	206	360
212	370	368	206
206	368	370	212
360	206	212	378

La proprietà è valida anche per matrici complesse, infatti, considerando la matrice E e lanciando l'istruzione $\gg E * E.'$, in output si ottiene una matrice simmetrica:

$ans =$

$1.0e + 002*$

$-0.0300 + 0.1000i$	$0.6700 - 0.1900i$
$0.6700 - 0.1900i$	$1.3100 + 0.1200i$

Analogamente se si lancia l'istruzione $\gg E.' * E$.

Soluzione esercizio 19 Considerata la matrice A, lanciando l'istruzione

$\gg Y = inv(A)$

si ottiene:

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. $RCOND = 1.175530e - 017$.

$Y =$

$1.0e + 015 *$

0.1251	0.3753	−0.3753	−0.1251
−0.3753	−1.1259	1.1259	0.3753
0.3753	1.1259	−1.1259	−0.3753
−0.1251	−0.3753	0.3753	0.1251

Effettuando i due prodotti possibili, si ottengono le matrici:

$\gg Y * A$

$ans =$

$$\begin{array}{rrrr} 1.0000 & 0 & -0.5000 & 0 \\ 0 & 0 & 2.0000 & 2.0000 \\ 0 & 0 & 0 & -2.0000 \\ 0 & 0 & -0.5000 & 1.0000 \end{array}$$

$\gg A * Y$
$ans =$

$$\begin{array}{rrrr} 0.6250 & -1.5000 & 0.5000 & 0.2500 \\ 0 & 2.0000 & 0 & 0.5000 \\ 0 & -1.0000 & 2.0000 & 0 \\ 1.0000 & 0 & 0 & 1.0000 \end{array}$$

che, evidentemente, non rappresentano la matrice identica.

Soluzione esercizio 20 Considerata la matrice $A = \begin{pmatrix} 2 & 5 & -4 & 5 \\ 0 & 5 & -1 & 4 \\ 7 & 3 & 5 & -2 \\ -1 & 0 & 3 & 6 \end{pmatrix}$ e il vettore $b = \begin{pmatrix} 3 \\ 2 \\ -7 \\ 4 \end{pmatrix}$, il sistema si può scrivere in forma matriciale come $A * X = b$, dove X è il vettore delle incognite, con $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$.

Dopo aver inserito da tastiera la matrice A e il vettore b, si può calcolare la soluzione del sistema mediante l'istruzione $\gg X = inv(A) * b$, che restituisce:
$X =$

$$\begin{array}{r} -0.4077 \\ -0.2876 \\ -0.3476 \\ 0.7725 \end{array}$$

quindi la soluzione del sistema è data da:
$x_1 = -0.4077$; $\quad x_2 = -0.2876$; $\quad x_3 = -0.3476$; $\quad x_4 = 0.7725$.

Osservazione 84 Anziché mediante l'uso dell'inversa, la soluzione del sistema poteva essere calcolata mediante la divisione a sinistra, cioè con l'operazione $X = A\backslash b$.

Soluzione esercizio 21 I coefficienti di $p(\lambda)$ si ottengono mediante l'istruzione $\gg poly(G)$, che restituisce:

$ans =$

$$1.0000 \quad -10.0000 \quad 22.0000$$

quindi $p(\lambda) = \lambda^2 - 10\lambda + 22$.

Soluzione esercizio 22 Si consideri, ad esempio, la matrice A per la quale risulta:

$\gg size(A)$

$ans =$

$$4 \quad 4$$

Si costruisca la matrice di permutazione P di dimensioni 4×4

$$P = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

inserendola da tastiera.

Risulta:

1. $\gg A * P$

   $ans =$

   $$\begin{matrix} 13 & 2 & 3 & 16 \\ 8 & 11 & 10 & 5 \\ 12 & 7 & 6 & 9 \\ 1 & 14 & 15 & 4 \end{matrix}$$

   che è lo stesso risultato che si ottiene con l'istruzione $\gg fliplr(A)$;

2. $\gg P * A$

   $ans =$

   $$\begin{matrix} 4 & 15 & 14 & 1 \\ 9 & 6 & 7 & 12 \\ 5 & 10 & 11 & 8 \\ 16 & 3 & 2 & 13 \end{matrix}$$

   che è lo stesso risultato che si ottiene con l'istruzione $\gg flipud(A)$.

Soluzione esercizio 23 Lanciando $\gg M = hilb(3)$ e $\gg N = pascal(3)$ si ottengono le matrici richieste, cioè:

$M =$

1.0000	0.5000	0.3333
0.5000	0.3333	0.2500
0.3333	0.2500	0.2000

$N =$

1	1	1
1	2	3
1	3	6

Lanciando $\gg P = M. * N$ si memorizza il prodotto elemento per elemento delle due matrici nella nuova matrice P:

$P =$

1.0000	0.5000	0.3333
0.5000	0.6667	0.7500
0.3333	0.7500	1.2000

Per ruotare di 270° questa matrice si lanci l'istruzione $\gg K = rot90(P, 3)$ che restituisce:

$K =$

0.3333	0.5000	1.0000
0.7500	0.6667	0.5000
1.2000	0.7500	0.3333

Con l'istruzione $\gg trace(K)$ si ottiene la traccia della matrice K:

$ans =$

1.3333

Le norme richieste si calcolano con le istruzioni:

$\gg norm(K, 1)$

$ans =$

2.2833

$\gg norm(K, 2)$

$ans =$

2.0391

$\gg norm(K, inf)$

$ans =$

2.2833

$\gg norm(K, 'fro')$

$ans =$

2.1752

Soluzione esercizio 24 Lanciando $\gg H = invhilb(3)$ si ottiene la matrice richiesta, che è:
$H =$

$$\begin{array}{rrr} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{array}$$

Mediante l'istruzione $\gg flops(0)$ si azzera il contatore delle operazioni eseguite nella sessione di lavoro. Dopo aver lanciato questa istruzione, viene visualizzato nuovamente il prompt di MATLAB. A questo punto, lanciando
$\gg expm(H);$
e quindi
$\gg flops$
viene visualizzato:
$ans =$
1166
che è il numero di operazioni eseguite per calcolare $expm(H)$.
Azzerando nuovamente il contatore con $\gg flops(0)$ e lanciando
$\gg funm(H, 'exp');$
e quindi
$\gg flops$
si ottiene:
$ans =$
406
che è il numero di operazioni eseguite per calcolare $funm(H, 'exp')$.

Soluzione esercizio 25 Dopo aver inserito da tastiera il vettore riga con l'istruzione $\gg v = [\ 5, -1, 3, 2, 1, 4, -7\ ]$, per risolvere l'esercizio basta lanciare le seguenti istruzioni:

1. $\gg length(v)$
   $ans =$

   7

2. $\gg size(v)$
   $ans =$

   1 7

3. $\gg v(10) = -v(5)$

$v =$

$$5 \quad -1 \quad 3 \quad 2 \quad 1 \quad 4 \quad -7 \quad 0 \quad 0 \quad -1$$

4. bisogna a questo punto ricaricare il vettore v di partenza. Per farlo basta digitare $\gg v$ e premere $\uparrow$ finché non ricompare l'istruzione $\gg v = [\ 5, -1, 3, 2, 1, 4, -7\ ]$ con la quale si è costruito il vettore e premere INVIO.
   A questo punto si può lanciare $\gg [ymax, imax] = max(v)$, che restituisce:
   $ymax =$
   5
   $imax =$
   1
   il che significa che l'elemento massimo del vettore v è il primo, cioè 5;

5. analogamente al punto precedente, basta lanciare $\gg [ymin, imin] = min(v)$, che restituisce:
   $ymin =$
   -7
   $imin =$
   7
   il che significa che l'elemento minimo del vettore v è il settimo, cioè -7;

6. lanciando $\gg [vord, iord] = sort(v)$ si ottiene l'ordinamento in maniera decrescente degli elementi del vettore:
   $vord =$
   $$-7 \quad -1 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5$$
   $iord =$
   $$7 \quad 2 \quad 5 \quad 4 \quad 3 \quad 6 \quad 1$$

7. la somma degli elementi del vettore si ottiene con $\gg sum(v)$ che restituisce:
   $ans =$
   7

8. il valor medio degli elementi del vettore si ottiene con $\gg mean(v)$ che restituisce:
   $ans =$
   1

Soluzione esercizio 26 Mediante l'istruzione $\gg v = [\ 3 \quad 6 \quad 9 \quad 12 \quad 15\ ]$ si inserisce il vettore riga v:

$v =$

$$\begin{matrix} 3 & 6 & 9 & 12 & 15 \end{matrix}$$

Con l'istruzione $\gg v(2) = [\ ]$ si ottiene:

$v =$

$$\begin{matrix} 3 & 9 & 12 & 15 \end{matrix}$$

e con l'istruzione $\gg v(5) = 8$:

$v =$

$$\begin{matrix} 3 & 9 & 12 & 15 & 8 \end{matrix}$$

che è il vettore richiesto. Imponendo $\gg z = v$ si ottiene:

$z =$

$$\begin{matrix} 3 & 9 & 12 & 15 & 8 \end{matrix}$$

Per costruire la matrice richiesta si lanci $\gg M = diag(z, 1)$. Si ottiene:

$M =$

$$\begin{matrix} 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 15 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Lanciando $\gg det(M)$ si ottiene il determinante della matrice:

$ans =$

$$0$$

mentre con $\gg rank(M)$ si ottiene il rango:

$ans =$

$$5$$

Soluzione esercizio 27 Le seguenti istruzioni restituiscono, equivalentemente, il vettore richiesto:

$$\gg v = [\ -1 \quad -0.75 \quad -0.5 \quad -0.25 \quad 0 \quad 0.25 \quad 0.5 \quad 0.75 \quad 1\ ]\ ;$$

$$\gg v = linspace(-1, 1, 9);$$

$$\gg v = [\ -1 : 0.25 : 1\ ];$$

L'uso del ; dopo le istruzioni serve per non visualizzare sullo schermo gli elementi di v.

Per costruire il vettore z richiesto si lancia l'istruzione $\gg z = v(1:3)$:

$z =$

$$-1.0000 \quad -0.7500 \quad -0.5000$$

La matrice di Frobenius associata a questo vettore si ottiene mediante $\gg C = compan(z)$

$C =$

$$\begin{matrix} -0.7500 & -0.5000 \\ 1.0000 & 0 \end{matrix}$$

e l'approssimazione per difetto di questa matrice è data da $\gg D = floor(C)$:

$D =$

$$\begin{matrix} -1 & -1 \\ 1 & 0 \end{matrix}$$

Per calcolare la radice quadrata della matrice D si usa l'istruzione $\gg sqrtm(D)$, che restituisce:

$ans =$

$$\begin{matrix} 0.0000 & -\ 0.0000i & -1.0000 & -\ 0.0000i \\ 1.0000 & +\ 0.0000i & 1.0000 & \end{matrix}$$

mentre la radice quadrata di tutti gli elementi della matrice si ottiene mediante $\gg sqrt(D)$, che restituisce:

$ans =$

$$\begin{matrix} 0 & +\ 1.0000i & 0 & +\ 1.0000i \\ 1.0000 & & 0 & \end{matrix}$$

Soluzione esercizio 28 Per costruire il vettore richiesto si utilizza l'istruzione *logspace*, dopo aver osservato che $\log_b a = \dfrac{\log_c a}{\log_c b}$; per non visualizzare gli elementi di x sullo schermo usiamo un ; dopo le istruzioni:

$\gg x = logspace(-1, -5, 31);$

$\gg x = x./log(2);$

Memorizziamo nel vettore z i primi 5 elementi di x:

$z = x(1:5)$

$z =$

$$0.1443 \quad 0.1061 \quad 0.0781 \quad 0.0574 \quad 0.0423$$

Lanciando $\gg Z = vander(z)$ si ottiene la matrice richiesta:

$Z =$

0.0004	0.0030	0.0208	0.1443	1.0000
0.0001	0.0012	0.0113	0.1061	1.0000
0.0000	0.0005	0.0061	0.0781	1.0000
0.0000	0.0002	0.0033	0.0574	1.0000
0.0000	0.0001	0.0018	0.0423	1.0000

Memorizziamo la terza e la quarta colonna di Z nella matrice U mediante l'istruzione $\gg U = Z(:, 3:4)$, che restituisce:

$U =$

0.0208	0.1443
0.0113	0.1061
0.0061	0.0781
0.0033	0.0574
0.0018	0.0423

Lanciamo $\gg format\ long$ ed eleviamo al quadrato tutti gli elementi di U:

$\gg U = U.\hat{}2$

$U =$

0.00043320968351	0.02081368981006
0.00012687171850	0.01126373466019
0.00003715621688	0.00609558995320
0.00001088173526	0.00329874752900
0.00000318687348	0.00178518163846

La sottomatrice u si ottiene mediante l'istruzione $\gg u = U(2:3, :)$, che restituisce:

$u =$

0.00012687171850	0.01126373466019
0.00003715621688	0.00609558995320

Per calcolare e^u basta lanciare $\gg expm(u)$, che restituisce:

$ans =$

1.00012708946950	0.01129885087061
0.00003727205639	1.00611441598750

Per tornare al formato predefinito di MATLAB si lancia l'istruzione
$\gg format\ short$.

La potenza richiesta è data da:

$\gg u.\wedge - 0.1$

$ans =$

2.4528 1.5661

2.7733 1.6653

Il vettore v costituito dagli elementi della prima colonna di questa matrice si ottiene mediante:

$\gg v = ans(:, 1)$

$v =$

2.4528

2.7733

La norma richiesta è quindi:

$\gg norm(v, 5.4)$

$ans =$

2.9952

Soluzione esercizio 29 Definito il vettore $\gg x = linspace(1, 6, 6)$, si può usare, equivalentemente, una delle due seguenti procedure:

- $\gg x = [\ x \quad 20\ ]$
  $\gg y = [\ x \quad 20 \quad 6 : -1 : 1\ ]$
- $\gg x1 = [\ x \quad 20\ ]$
  $\gg x2 = fliplr(x1)$
  $\gg y = [\ x1 \quad x2\ ]$

Qualsiasi sia la scelta fatta, MATLAB visualizza:

$y =$

Columns 1 through 12 [1]

1	2	3	4	5	6	20	20	6	5	4	3

Columns 13 through 14 [2]

2 1

Lanciando $\gg y(13 : 14) = [\ ]$ si ottiene:

$y =$

1	2	3	4	5	6	20	20	6	5	4	3

Lanciando $\gg Y = reshape(y, 3, 4)$ si ottiene la matrice richiesta:

$Y =$

[1] Traduzione:
Colonne dalla prima alla dodicesima

[2] Traduzione:
Colonne dalla tredicesima alla quattordicesima

$$\begin{matrix} 1 & 4 & 20 & 5 \\ 2 & 5 & 20 & 4 \\ 3 & 6 & 6 & 3 \end{matrix}$$

Lanciando $Y(:,4) = [\ ]$ si ottiene la nuova matrice

$Y =$

$$\begin{matrix} 1 & 4 & 20 \\ 2 & 5 & 20 \\ 3 & 6 & 6 \end{matrix}$$

che deve essere moltiplicata per il quadrato magico di dimensione 3, che si ottiene lanciando $\gg magic(3)$

$ans =$

$$\begin{matrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{matrix}$$

L'operazione $\gg Z = Y * ans$ restituisce:

$Z =$

$$\begin{matrix} 100 & 201 & 74 \\ 111 & 207 & 87 \\ 66 & 87 & 72 \end{matrix}$$

Lanciando $\gg S = triu(Z,1)$ si ottiene la matrice richiesta:

$S =$

$$\begin{matrix} 0 & 201 & 74 \\ 0 & 0 & 87 \\ 0 & 0 & 0 \end{matrix}$$

La sottomatrice R si estrae mediante l'istruzione $\gg R = S(1:2, 2:3)$, che restituisce:

$R =$

$$\begin{matrix} 201 & 74 \\ 0 & 87 \end{matrix}$$

Modifichiamo l'elemento richiesto:

$\gg R(2,1) = 1$

$R =$

$$\begin{matrix} 201 & 74 \\ 1 & 87 \end{matrix}$$

La matrice logaritmo della matrice R si ottiene mediante $\gg L = logm(R)$, che restituisce:

$L =$

5.3018	0.5443
0.0074	4.4632

Il prodotto richiesto è dato da $\gg kron(R, L)$:

$ans =$

$1.0e + 003 *$

1.0657	0.1094	0.3923	0.0403
0.0015	0.8971	0.0005	0.3303
0.0053	0.0005	0.4613	0.0474
0.0000	0.0045	0.0006	0.3883

Soluzione esercizio 30 Per risolvere l'esercizio basta lanciare le istruzioni:

$\gg A1 = A(2:3, 2:4)$

$A1 =$

10	11	8
6	7	12

$\gg A2 = A(\ [\ 1\ \ 2\ ]\ , 1:3)$

$A2 =$

16	3	2
5	10	11

Costruiamo le matrici che hanno 2 righe e 3 colonne ed elementi tutti uguali rispettivamente a 0 e a 1:

$\gg Z = zeros(2, 3)$

$Z =$

0	0	0
0	0	0

$\gg U = ones(2, 3)$

$U =$

1	1	1
1	1	1

La matrice G richiesta si ottiene concatenando le quattro matrici precedentemente costruite in questo modo:

$\gg G = [\ A1\ \ Z$

$U\ \ A2\ ]$

$G =$

$$\begin{matrix} 10 & 11 & 8 & 0 & 0 & 0 \\ 6 & 7 & 12 & 0 & 0 & 0 \\ 1 & 1 & 1 & 16 & 3 & 2 \\ 1 & 1 & 1 & 5 & 10 & 11 \end{matrix}$$

Soluzione esercizio 31 Dopo aver inserito i due vettori mediante le istruzioni:
$\gg a = [\ 1\ \ 3\ \ 5\ \ 7\ ]$
$\gg b = [\ 2\ \ 4\ \ 6\ \ 8\ ]$
si può costruire il vettore c mediante il seguente procedimento: concateniamo i vettori a e b in questo modo:
$\gg c = cat(1, a, b)$
$c =$

$$\begin{matrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{matrix}$$

con l'istruzione $\gg c(:)$ si possono leggere per colonne tutti gli elementi di questa matrice:
$ans =$

$$\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

Questo vettore è il trasposto del vettore richiesto, quindi per risolvere l'esercizio basta lanciare $\gg c = ans\ '$, che restituisce:
$c =$

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix}$$

Per aggiungere l'elemento richiesto basta lanciare $\gg c = [\ c(1:3)\ \ 7\ \ c(4:8)\ ]$, che restituisce:

$$\begin{matrix} 1 & 2 & 3 & 7 & 4 & 5 & 6 & 7 & 8 \end{matrix}$$

La matrice S richiesta si ottiene in questo modo:
$\gg S = reshape(c, 3, 3)$
$S =$

$$\begin{matrix} 1 & 7 & 6 \\ 2 & 4 & 7 \\ 3 & 5 & 8 \end{matrix}$$

e la sottomatrice richiesta è:

$\gg S(1:2, 2:3)$

$ans =$

7	6
4	7

Soluzione esercizio 32 Effettuiamo la discretizzazione di $[\,0, 1\,]$ come richiesto:

$\gg x = linspace(0, 1, 6)\,'$

$x =$

0
0.2000
0.4000
0.6000
0.8000
1.0000

Il valore che la funzione coseno assume in questi nodi è dato da:

$\gg y = cos(x)$

$y =$

1.0000
0.9801
0.9211
0.8253
0.6967
0.5403

quindi la matrice richiesta si può ottenere mediante l'istruzione:

$\gg T = [\ x\ \ y\ ]$ che restituisce:

$T =$

0	1.0000
0.2000	0.9801
0.4000	0.9211
0.6000	0.8253
0.8000	0.6967
1.0000	0.5403

B.4 Capitolo 7

Soluzione esercizio 33 Si può scrivere la seguente function primiint.m:

$function\ [val\_calc, val\_teor] = primiint(n)$

% Calcola la somma dei primi n numeri interi e il valore
% teorico atteso per questa somma.
%
% Sintassi: $[val\_calc, val\_teor] = primiint(n)$
%
% Dati di input
% n: numero di interi da sommare
% Dati di output
% val_calc: valore calcolato dall'algoritmo
% val_teor: valore teorico atteso per la somma

% sommiamo tutti i primi n numeri interi:
$val\_calc = 0;$
$for\ i = 1 : n$
$\quad val\_calc = val\_calc + i;$
end

% calcoliamo il valore atteso:
$val\_teor = n * (n + 1)/2;$

Il lettore può testare la function lanciando, ad esempio:
$\gg [vc, vt] = primiint(3)$
$\gg [vc, vt] = primiint(10)$
$\gg [vc, vt] = primiint(100)$
$\gg [vc, vt] = primiint(1000)$

Soluzione esercizio 34 Si può scrivere il seguente file norma2.m:

$function\ n2 = norma2(x)$

% Calcola la norma 2 di un vettore x
%
% Sintassi: $n2 = norma2(x)$
%
% Dati di input
% x: vettore di cui calcolare la norma 2

% Dati di output

% $\qquad\qquad n2$: norma 2 di x

% calcoliamo la somma dei quadrati degli elementi

% del vettore x:

$n2 = 0;$

$for\ i = 1 : length(x)$

$\quad n2 = n2 + x(i)\hat{}2;$

end

% calcoliamo la radice quadrata della somma precedente,

% cioè la norma 2 del vettore x:

$n2 = sqrt(n2);$

Per testare la function, consideriamo il vettore x e lanciamo

$\gg n = norma2(x)$

Si ottiene:

$n =$

7.4162

che è lo stesso risultato che si ottiene lanciando l'apposito comando MATLAB per il calcolo della norma 2 di un vettore, cioè lanciando

$\gg n = norm(x, 2)$

Soluzione esercizio 35 Si può scrivere il seguente file normainf.m:

$function\ ninf = normainf(x)$

% Calcola la norma infinito di un vettore x

%

% Sintassi: $\qquad ninf = normainf(x)$

%

% Dati di input

% $\qquad\qquad x$: vettore di cui calcolare la norma infinito

% Dati di output

% $\qquad\qquad ninf$: norma infinito di x

% calcoliamo il massimo elemento in modulo del vettore x, cioè

% la sua norma infinito:

$ninf = abs(x(1));$
$for\ i = 2 : length(x)$
    $if\ abs(x(i)) > ninf$
        $ninf = abs(x(i));$
    end
end

Per testare la function, consideriamo il vettore x e lanciamo
$\gg n = normainf(x)$
Si ottiene:
$n =$
    5
che è lo stesso risultato che si ottiene lanciando l'apposito comando MATLAB per il calcolo della norma infinito di un vettore, cioè lanciando
$\gg n = norm(x, inf)$

Soluzione esercizio 36 Assegnata la matrice $A = \{a_{ij}\}_{\substack{i=1,\dots,n\\ j=1,\dots,m}}$, la sua norma di Frobenius si ottiene mediante la relazione $||A||_F = \sqrt{\sum_{j=1}^{m}\sum_{i=1}^{n}|a_{ij}|^2}$, quindi può essere scritta la seguente function normafro.m:

$function\ n = normafro(A)$

% Calcola la norma di Frobenius di un matrice A
%
% Sintassi: $n = normafro(A)$
%
% Dati di input
% A: matrice di cui calcolare la norma
% di Frobenius
% Dati di output
% n: norma di Frobenius di A

% valutiamo le dimensioni della matrice:
$[n, m] = size(A);$

% effettuiamo la somma dei quadrati dei valori assoluti

% di tutti gli elementi della matrice:
$sum = 0;$
$for\ j = 1 : m$
$\quad for\ i = 1 : n$
$\quad\quad sum = sum + (A(i, j))\hat{}2;$
$\quad end$
end

% valutiamo la radice quadrata della somma precedente, cioè
% la norma di Frobenius della matrice:
$n = sqrt(sum);$

La function può essere testata considerando, ad esempio, la matrice A e lanciando $\gg normafro(A)$ e osservando che il risultato coincide con quello che si ottiene lanciando $\gg norm(A,' fro')$.
Si può osservare che la function calcola anche la norma di vettori oltre che di matrici, lanciando ad esempio $\gg normafro(x)$ e verificando che l'output coincide con quello che si ottiene lanciando $\gg norm(x,' fro')$.

Soluzione esercizio 37 Assegnata la matrice $A = \{a_{ij}\}_{\substack{i=1,\ldots,n\\ j=1,\ldots,m}}$ la sua norma 1 si ottiene mediante la relazione $||A||_1 = \max_{j=1,\ldots,m} \sum_{i=1}^{n} |a_{ij}|$, quindi può essere scritta la seguente function norma1matr.m:

$function\ sum = norma1matr(A)$

% Calcola la norma 1 di un matrice A
%
% Sintassi: $sum = norma1matr(A)$
%
% Dati di input
% A: matrice di cui calcolare la norma 1
% Dati di output
% sum: norma 1 di A

% valutiamo le dimensioni della matrice:
$[n, m] = size(A);$

$\%$ calcoliamo la somma dei valori assoluti degli elementi
$\%$ della prima colonna della matrice:
$j = 1;$
$sum = 0;$
$for\ i = 1 : n$
    $sum = sum + abs(A(i, j));$
end

$\%$ calcoliamo la somma dei valori assoluti degli elementi
$\%$ di tutte le altre colonne della matrice:
$for\ j = 2 : m$
    $newsum = 0;$
    $for\ i = 1 : n$
        $newsum = newsum + abs(A(i, j));$
    end

    $\%$ se il valore della somma degli elementi della colonna
    $\%$ j-esima è più grande del valore della somma degli ele-
    $\%$ menti della colonna precedente, assumiamo questo nuo-
    $\%$ vo valore come norma 1 della matrice:
    $if\ newsum > sum$
        $sum = newsum;$
    end
end

La function può essere testata considerando, ad esempio, la matrice A e lanciando $\gg norma1matr(A)$ e osservando che il risultato coincide con quello che si ottiene lanciando $\gg norm(A, 1)$.

Soluzione esercizio 38 Oltre al file già considerato, si può scrivere il seguente file successione2.m:

$function\ an = successione(n)$

$\%$ Calcola il valore verso cui converge la successione
$\%$ $\qquad a1 = 1;$
$\%$ $\qquad\qquad\qquad an\hat{}2 + 6$

% $a(n+1) \quad = \quad - - - - - - - -$
% $\qquad\qquad\qquad 5$
%
% Sintassi: $an = successione(n)$
%
% Dati di input
% n: ultimo termine della successione che si
% intende considerare
% Dati di output
% an: valore del termine n-esimo della successione

% assegnamo il primo elemento della successione:
$an = 1;$

valutiamo tutti i termini della successione fino all'n-esimo:
$for\ i = 2 : n$
$\quad an = (an\hat{}2 + 6)/5;$
end

Scegliendo valori sempre più grandi per n, si ottiene, ad esempio:
$\gg a = successione2(10)$
$a =$
1.9402
$\gg a = successione2(30)$
$a =$
1.9994
$\gg a = successione2(50)$
$a =$
2.0000

Soluzione esercizio 39 Non è possibile in MATLAB far variare un indice fino al valore infinito, ma prendendo una precisione sempre maggiore, possiamo far vedere che la serie converge proprio al valore $\frac{3}{4} = 0.75$.
Per calcolare il valore della somma entro una fissata precisione *tol* e con un numero massimo di iterate *nmax* è sufficiente scrivere il seguente file serie.m:

$function\ [s, n] = serie(tol, nmax)$

```
% Calcola la somma della serie
%                        1
%        somma(---------)
%                   n*(n+2)
%
% per n = 1,2,3... nei margini della precisione richiesta.
%
% Sintassi:        [s,n] = serie(tol,nmax)
%
% Dati di input
%                    tol: precisione richiesta
%                   nmax: numero massimo di iterate consentite
% Dati di output
%                      s: somma della serie
%                      n: numero di iterate effettuate dall'algoritmo
%                         per approssimare la somma della serie entro
%                         la precisione richiesta

% assegnamo il primo e il secondo elemento della serie, che si
% ottengono per n = 1 e per n = 2:
n = 1;
s1 = 1/3;
n = 2;
s2 = 11/24;

% inizializziamo il residuo:
residuo = abs(s1 - s2)/(1 + s2);

% finché il residuo non è minore della precisione richiesta
% effettuiamo le seguenti operazioni:
while residuo > tol & n <= nmax

    % aumentiamo di un'unità l'indice della serie:
    n = n + 1;

    % aggiungiamo alla serie un altro elemento:
    s = s2 + 1/(n*(n + 2));
```

% aggiorniamo il residuo:
$residuo = abs(s2 - s)/(1 + s);$

% fissiamo come punto di partenza $s2$ per il calcolo della
% somma successiva il valore di s calcolato:
$s2 = s;$
end

Lanciando $\gg [somma, it] = serie(1e - 3, 100)$ si ottiene:
$somma =$
0.7108
$it =$
24
Questo significa che l'algoritmo ha approssimato, entro una precisione dell'ordine di 10^{-3}, la somma della serie con il valore 0.7108, effettuando 24 iterate. Considerando una precisione maggiore, si ottiene una migliore approssimazione del valore della somma, ma un numero maggiore di iterate necessarie per raggiungere tale valore. Ad esempio, lanciando:
$\gg [somma, it] = serie(1e - 5, 100)$
si ottiene:
$somma =$
0.7402
$it =$
101
Il fatto che il valore di it, cioè del numero di iterate effettuate dall'algoritmo, sia uguale a 101 ci dice che è stato superato il limite massimo di iterate consentite. Per ottenere effettivamente una precisione dell'ordine di 10^{-5} bisogna aumentare il valore massimo di iterate consentite $nmax$; ad esempio, lanciando:
$\gg [somma, it] = serie(1e - 5, 300)$
si ottiene:
$somma =$
0.7458
$it =$
239
Il lettore può verificare che il valore teorico della somma della serie viene rag-

giunto aumentando la precisione richiesta e, eventualmente, il numero massimo di iterate consentite. Ad esempio si può verificare cosa accade lanciando:
$\gg [somma, it] = serie(1e-7, 1e3)$
$\gg [somma, it] = serie(1e-7, 1e4)$
$\gg [somma, it] = serie(1e-8, 1e5)$
$\gg [somma, it] = serie(1e-9, 1e5)$

Soluzione esercizio 40 Oltre alla function fattoriale.m già analizzata, occorre scrivere una function per il calcolo del rapporto $\frac{n*(n-1)*...*(n-k+1)}{k!}$: binomiale.m. Si può procedere in questo modo:

$function\ b = binomiale(n, k)$

% Calcola il coefficiente binomiale dei due numeri n e k
%
% Sintassi: $b = binomiale(n, k)$
%
% Dati di input
% n: primo numero nel coefficiente binomiale
% k: secondo numero nel coefficiente binomiale
% Dati di output
% b: coefficiente binomiale di n e k

% calcoliamo il numeratore $n * (n-1) * \ldots * (n-k+1)$:
$num = n;$
$for\ i = n-1 : -1 : (n-k+1)$
  $num = num * i;$
end

% calcoliamo il denominatore, cioè il fattoriale del numero k:
$den = fattoriale(k);$

% calcoliamo il coefficiente binomiale:
$b = num/den;$

Per fissare i breakpoint richiesti è sufficiente lanciare:
$\gg$ *dbstop in binomiale at* 21

$\gg dbstop\ in\ fattoriale\ at\ 22$

Per calcolare il coefficiente binomiale si lanci $\gg binomiale(15, 4)$.

L'esecuzione procede in questo modo: il file binomiale.m viene eseguito fino al primo breakpoint, cioè fino alla ventunesima riga, dove viene chiamata la function fattoriale.m. Lanciando dal workspace l'istruzione $K \gg dbcont$, l'esecuzione riprende: viene eseguito il file fattoriale.m fino alla ventiduesima riga, dove l'esecuzione si arresta nuovamente. Lanciando ancora $K \gg dbcont$ viene terminata la function fattoriale.m e si torna in binomiale.m, dove l'esecuzione viene ripresa dal punto in cui era stata interrotta e viene terminata. Si ottiene:

$ans =$

1365

Per cancellare il breakpoint nel file fattoriale.m si può lanciare:

$\gg dbclear\ in\ fattoriale\ at\ 22$

Lanciando $\gg binomiale(15, 4)$ viene avviata l'esecuzione, che si arresta al breakpoint di binomiale.m.

Per avanzare di 2 linee eseguibili si lanci $K \gg dbstep\ 2$. Visualizzando il file nell'editor si può osservare che accanto all'ultima istruzione è apparsa una freccetta gialla rivolta verso il basso. Lanciando $K \gg whos$ vengono visualizzate le variabili presenti nel workspace (b, den, i, k, n, num).

Per visualizzare la posizione in cui ci si trova si lanci $K \gg dbstack$, da cui si ottiene:

$>$ In $C : \backslash MATLABR11\backslash work\backslash binomiale.m$ after line 24

(supponendo di aver salvato il file nella cartella work di MATLAB). Per sospendere la sessione di Debugger prima che l'esecuzione sia conclusa si lancia $K \gg dbquit$, si ottiene il risultato finale della function, cioè:

$ans =$

1365

Per cancellare tutti i breakpoint è sufficiente lanciare $\gg dbclear\ all$.

Osservazione 85 Ovviamente il coefficiente binomiale $\begin{pmatrix} n \\ k \end{pmatrix}$ può essere calcolato più rapidamente come $factorial(n)/(factorial(k) * factorial(n-k))$ oppure mediante $prod((n-k+1) : n)/factorial(k)$.

Soluzione esercizio 41 La function richiesta può essere scritta in questa maniera:

```
function b = bernstein(t, n, k)

% Valuta il polinomio di Bernstein di indici n e k nel
% punto t.
%
% Sintassi:          b = bernstein(t, n, k)
%
% Dati di input
%                         t: punto nel quale valutare il polinomio
%                     n e k: indici del polinomio
% Dati di output
%                         b: valore del polinomio di Bernstein di
%                            indici n e k nel punto t

if k > n
    b = 0;
else
    switch t
    case 0
        switch k
        case 0
            b = 1;
        otherwise
            b = 0;
        end
    case 1
        switch k
        case n
            b = 1;
        otherwise
            b = 0;
        end
    otherwise
        num = binomiale(n, k);
        b = num * (t^k) * ((1 - t)^(n - k));
    end
```

end

Salvata la function con il nome bernstein.m, per valutare il polinomio $B_{5,3}$ in $t = 0.5$ è sufficiente lanciare:
$\gg b = bernstein(0.5, 5, 3)$
da cui si ottiene:
$b =$
    0.3125

B.5 Capitolo 8

Soluzione esercizio 42 L'esercizio può essere risolto lanciando l'istruzione
$\gg \{\ 'oggi\,'\ blanks(10)\ 'fa\,'\ \ 'molto\,'\ \ 'caldo\,'\ \}$
dalla quale si ottiene il cell array:
$ans =$
    $'oggi\,' \quad ' \qquad\qquad ' \quad 'fa\,' \quad 'molto\,' \quad 'caldo\,'$

Soluzione esercizio 43 La stringa si costruisce mediante l'istruzione:
$\gg a =\ 'casa\ \sqcup\ \sqcup\ \sqcup\,'$
dalla quale si ottiene:
$a =$
casa
Per eliminare gli spazi bianchi è sufficiente lanciare:
$\gg deblank(a)$
che restituisce:
$ans =$
casa
Per verificare che gli spazi bianchi sono stati effettivamente eliminati basta lanciare $\gg whos$, che mostra come la lunghezza della nuova stringa *ans* è minore di quella della stringa *a*.

Soluzione esercizio 44 Le stringhe richieste si costruiscono mediante le istruzioni:
$\gg stri1 =\ 'salva\,';$
$\gg stri2 =\ 'guardare\,';$
$\gg stri3 =\ 'cassa\,';$
$\gg stri4 =\ 'forte\,';$
Per costruire le matrici di stringhe $m1$ ed $m2$ si utilizza il comando *strvcat*: lanciando $\gg m1 = strvcat(stri1, stri3)$ e $\gg strvcat(stri2, stri4)$ si ottengono

le matrici di stringhe richieste, cioè:

$m1 =$

$salva$

$cassa$

$m2 =$

$guardare$

$forte$

La matrice di stringhe $m3$ può essere costruita in vari modi: volendola introdurre da tastiera, come richiesto dall'esercizio, bisognerà lanciare:

$\gg m3 = [\ 'salvaguardare\ \ '$

$'cassaforte\ \sqcup\sqcup\sqcup\ \ '\ ]$

da cui si ottiene:

$m3 =$

$salvaguardare$

$cassaforte$

Lo stesso output si ottiene utilizzando le matrici di stringhe $m1$ ed $m2$, come richiesto dall'esercizio: è sufficiente concatenare gli elementi corrispondenti dei due array mediante il comando $strcat$, cioè lanciare l'istruzione $\gg m3 = strcat(m1, m2)$ da cui si ottiene proprio la matrice di stringhe $m3$.

La variabile $m4$ si ottiene lanciando:

$\gg m4 = m3(1, :)$

da cui si ottiene:

$m4 =$

$salvaguardare$

Lanciando $\gg ischar(m4)$ si ottiene:

$ans =$

1

e ciò significa che $m4$ è una matrice di caratteri.

Memorizzando in $m5$ la stringa gente, mediante l'istruzione:

$\gg m5 =\ 'gente\,';$

si ottiene la stringa salvagente tramite:

$\gg strrep(m4,\ 'guardare\,', m5)$

$ans =$

$salvagente$

La parte di questa stringa che precede la lettera g si può individuare mediante l'istruzione $\gg strtok(ans,\ 'g\,'\ )$ da cui si ottiene:

$ans =$

$salva$

Il corrispondente valore numerico di questa stringa si ottiene lanciando:

$\gg double(ans)$ (oppure, equivalentemente, $\gg abs(ans)$)

che restituisce:

$ans =$

115 97 108 118 97

Soluzione esercizio 45 L'esercizio può essere risolto in questo modo:

$\gg a = \ 'libero\,'$

$a =$

$libero$

$\gg a(4) = [\ ]$

$a =$

$libro$

$\gg b = strcat(\ 'segna\,', a)$

$b =$

$segnalibro$

$\gg length(b)$

$ans =$

10

$\gg b(1:5) = [\ ]$

$b =$

$libro$

$\gg b = [\ b(1:4)\ \ 'ai\,'\ \ b(5)\ ]$

$b =$

$libraio$

$\gg c = strvcat(\ 'Il\,',\ b,\ 'e\,'\,'\ \ ',\ 'simpatico\,'\ )$

$c =$

Il

$libraio$

$\grave{e}$

$simpatico$

$\gg d = upper(c)$

$d =$

IL

$LIBRAIO$

$\grave{E}$

$SIMPATICO$

$\gg strjust(d)$

$ans =$

IL

$LIBRAIO$

$\grave{E}$

$SIMPATICO$

$\gg isstr(d)$

$ans =$

1

Soluzione esercizio 46 La matrice S può essere introdotta da tastiera mediante $\gg S = [\ 67\ \ 79\ \ 50\ ];$

Lanciando $\gg stri5 = char(S)$ si ottiene la stringa richiesta, cioè:

$stri5 =$

$CO2$

Lanciando $\gg isletter(stri5)$ si ottiene:

$ans =$

1 1 0

Ciò significa che i primi due elementi di $stri5$ sono lettere, il terzo elemento non lo è.

Il cell array $cell1$ si ottiene mediante:

$\gg cell1 = \{\ stri5\ \ \ 'anidride\ \ carbonica\,'\ \}$

da cui si ricava:

$cell1 =$

$'CO2\,'\quad 'anidride\ \ carbonica\,'$

Il secondo elemento di questo cell array si ottiene mediante:

$\gg elem = cell1\,\{2\}$

$elem =$

$anidride\ \ carbonica$

Lanciando $\gg iscell(elem)$ si ottiene:

$ans =$

0

che vuol dire che il secondo elemento di $cell1$ non è un cell array.

Gli indici ai quali si trovano le lettere i si ottengono mediante:

$\gg findstr(elem,\ 'i\,'\ )$

$ans =$

3 6 16

mentre l'indice al quale si trova la sillaba dri si trova lanciando:

$\gg findstr(elem, \ 'dri\,' \ )$
$ans =$
        4
La stringa $stri6$ richiesta si costruisce mediante l'istruzione:
$\gg stri6 = \ 'anidride\,'$
$stri6 =$
$anidride$
Lanciando:
$\gg strcmp(elem, stri6)$
si ottiene
$ans =$
        0
che vuol dire che le variabili $elem$ e $stri6$ non sono coincidenti.
Lanciando:
$\gg strncmp(elem, stri6, 8)$
si ottiene:
$ans =$
        1
che vuol dire che i primi 8 elementi delle due stringhe coincidono.

Soluzione esercizio 47 Lanciando le seguenti istruzioni:
$\gg temperature.luogo = \ 'VE\,';$
$\gg temperature.min = 11;$
$\gg temperature.max = 21;$
si ottiene la struttura richiesta. Per visualizzarne gli elementi è sufficiente lanciare:
$\gg temperature$
da cui si ottiene:
$temperature =$

luogo:	$'VE\,'$
min:	11
max:	21

Le altre colonne della struttura si possono costruire mediante le seguenti istruzioni:
$\gg temperature(2).luogo = \ 'FI\,';$
$\gg temperature(2).min = 13;$
$\gg temperature(2).max = 23;$

$\gg temperature(3).luogo = \ 'RO\ ';$
$\gg temperature(3).min = 10;$
$\gg temperature(3).max = 23;$
$\gg temperature(4).luogo = \ 'BA\ ';$
$\gg temperature(4).min = 13;$
$\gg temperature(4).max = 19;$
Lanciando $\gg$ *temperature* si ottiene:
$temperature =$
1×4 struct array with fields:
    luogo
    min
    max

Osservazione 86 In alternativa le ulteriori 3 colonne della struttura potevano essere costruite mediante il comando *struct*.

B.6 Capitolo 9

Soluzione esercizio 48 La matrice sparsa richiesta può essere costruita mediante il seguente procedimento:

- sfruttando il comando *sparse* con 5 dati di input, costruiamo la matrice sparsa di dimensione 5×5 che abbia gli elementi della diagonale principale tutti uguali a 2:
  $\gg i1 = [1\ 2\ 3\ 4\ 5];$
  $\gg nn1 = [2\ 2\ 2\ 2\ 2];$
  $\gg dp = sparse(i1, i1, nn1, 5, 5)$
  da cui si ottiene:
  $dp =$

$(1,1)$	2
$(2,2)$	2
$(3,3)$	2
$(4,4)$	2
$(5,5)$	2

- sfruttando il comando *sparse* con 5 dati di input, costruiamo la matrice sparsa di dimensione 5×5 che abbia gli elementi della diagonale superiore

a quella principale tutti uguali a 1:
$\gg i2 = [1\ 2\ 3\ 4];$
$\gg j2 = [2\ 3\ 4\ 5];$
$\gg nn2 = [1\ 1\ 1\ 1];$
$\gg ds = sparse(i2, j2, nn2, 5, 5)$
da cui si ottiene:
$ds =$

$(1, 2)$	1
$(2, 3)$	1
$(3, 4)$	1
$(4, 5)$	1

- sfruttando il comando *sparse* con 5 dati di input, costruiamo la matrice sparsa di dimensione 5×5 che abbia gli elementi della diagonale inferiore a quella principale tutti uguali a 1:
$\gg di = sparse(j2, i2, nn2, 5, 5)$
da cui si ottiene:
$di =$

$(2, 1)$	1
$(3, 2)$	1
$(4, 3)$	1
$(5, 4)$	1

- la matrice richiesta è la somma delle tre matrici costruite, cioè:
$\gg O = dp + ds + di$
$O =$

$(1, 1)$	2
$(2, 1)$	1
$(1, 2)$	1
$(2, 2)$	2
$(3, 2)$	1
$(2, 3)$	1
$(3, 3)$	2
$(4, 3)$	1
$(3, 4)$	1
$(4, 4)$	2

$(5,4)$	1
$(4,5)$	1
$(5,5)$	2

La matrice piena corrispondente si ottiene lanciando $\gg\ F = full(O)$, che restituisce:

$F =$

2	1	0	0	0
1	2	1	0	0
0	1	2	1	0
0	0	1	2	1
0	0	0	1	2

Per mostrare che la matrice occupa una quantità di memoria differente a seconda di come viene memorizzata è sufficiente lanciare

$\gg whos\ O\ F$

ed analizzare l'output come si è fatto negli esempi precedenti.

Per valutare il diverso costo computazionale si può procedere come segue: mediante l'istruzione $\gg flops(0)$ si azzera il contatore delle operazioni effettuate fino a quel momento nella sessione di lavoro.

Lanciando: $\gg z = rand(5,1);$ si costruisce un vettore colonna con 5 elementi casuali scelti tra 0 e 1 con distribuzione uniforme. Per effettuare il prodotto della matrice con questo vettore possiamo effettuare due differenti moltiplicazioni: quella tra la matrice memorizzata in forma piena e il vettore e quella tra la matrice memorizzata in forma sparsa e il vettore.

Valutiamo, mediante il comando $flops$, il numero di operazioni richieste per effettuare il primo tipo di prodotto:

$\gg F * z;$

$\gg flops$

$ans =$

50

Azzeriamo nuovamente il contatore e valutiamo le operazioni richieste per effettuare il secondo tipo di prodotto:

$\gg flops(0)$

$\gg O * z;$

$\gg flops$

$ans =$

26

È evidente che non conviene memorizzare le matrici sparse come matrici piene.

La differenza fra il numero di operazioni richieste per gestire matrici memorizzate in forma sparsa e matrici memorizzate in forma piena cresce al crescere delle dimensioni delle matrici.

Appendice C

Variabili utilizzate

$$a = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix} \quad ; \quad A = \begin{pmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \end{pmatrix} \quad ; \quad b = 2 \quad ;$$

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{pmatrix} \quad ; \quad c = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \quad ; \quad C = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad ;$$

$$D = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 2 & 0 \end{pmatrix} \quad ;$$

$$E = \begin{pmatrix} 10 & 3+4i & 12i & -7+i \\ 2i & 8 & -6 & -6-i \end{pmatrix} \quad ;$$

$$f = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & -2 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 3 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & -1 & 0 & 0 \end{pmatrix} \quad ; \quad F = \begin{pmatrix} -2 & i \\ +3-i & 0 \end{pmatrix} \quad ;$$

$$g = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 6 & 3 & 1 & -2 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ -7 & 0 & 0 & 3 \end{pmatrix} \quad ; \quad G = \begin{pmatrix} 2 & -2 \\ 3 & 8 \end{pmatrix} \quad ;$$

$$H = \begin{pmatrix} 3 & 2 \\ 4 & 5 \end{pmatrix} \quad ;$$

$$I = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 1 & -2 & 3 \end{pmatrix} \quad ; \quad l = \begin{pmatrix} 0.73 & 1.2 \\ 3.4 & 0.8 \end{pmatrix} \quad ; \quad L = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & -1 \end{pmatrix} \quad ;$$

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 3 & 0 & 0 & 0 \end{pmatrix} \quad ; \quad \mathrm{N} = \begin{pmatrix} (1,1) & 1 \\ (3,1) & 5 \\ (5,2) & 3 \\ (2,3) & 3 \\ (4,4) & 2 \\ (3,5) & 1 \end{pmatrix} \quad ;$$

$$P = \begin{pmatrix} 12 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 3 & -4 & 1 & 0 & 0 \\ 5 & 7 & -8 & 6 & 0 \\ -7 & 1 & 1 & 3 & 5 \end{pmatrix} \quad ; \quad Q = \begin{pmatrix} 12 & 1 & 3 & 5 & -7 \\ 0 & 2 & -4 & 7 & 1 \\ 0 & 0 & 1 & -8 & 1 \\ 0 & 0 & 0 & 6 & 3 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix} \quad ;$$

$$S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad ;$$

$$x = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix} \quad ; \quad y = \begin{pmatrix} 1 & 10 & -5 & 6 & 7 \end{pmatrix} \quad ; \quad z = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} .$$