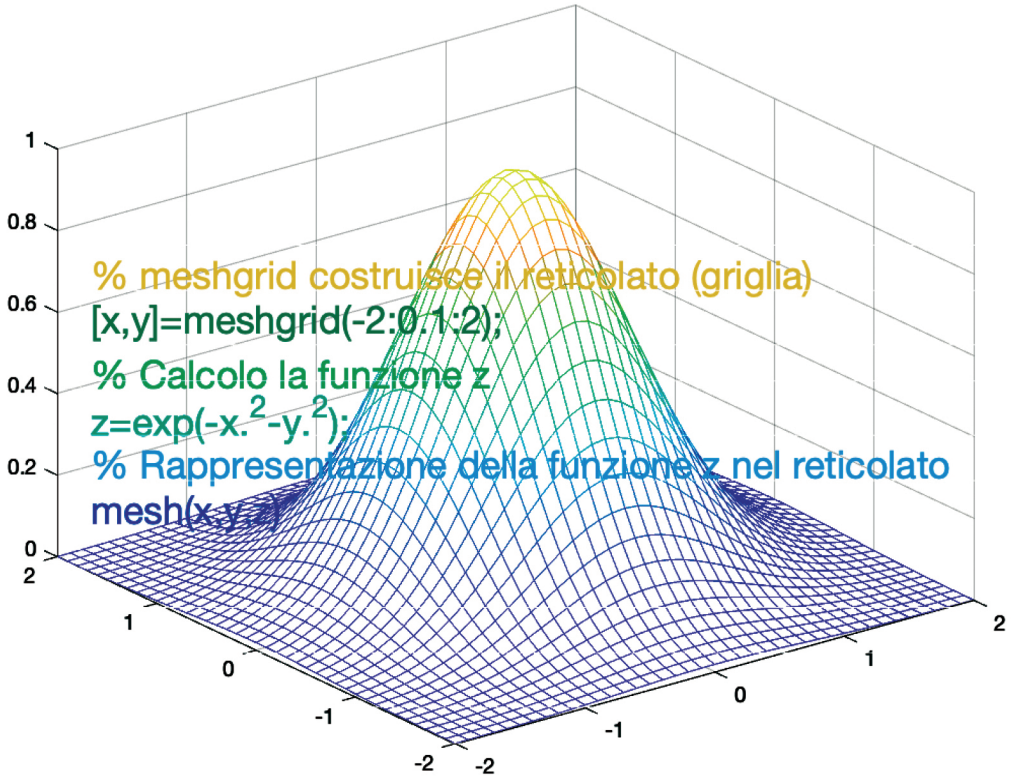
M. Riani, A. Corbellini, F. Laurini, G. Morelli
T. Proietti, E. Fibbi, D. Perrotta, F. Torti

Data Science con MATLAB



SECONDA EDIZIONE

Giappichelli

Software e materiale di corredo

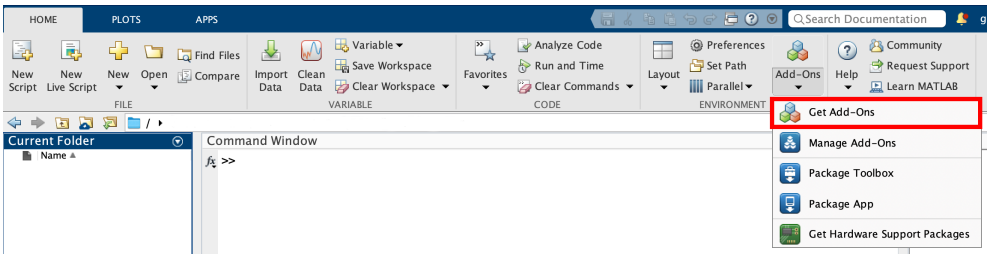
Il testo presuppone che il lettore abbia installato l'ultima versione di MATLAB (o almeno la 2021a) e lo Statistics and Machine Learning toolbox che consente di avere a disposizione una collezione di funzioni di data science. Nel capitolo relativo all'importazione dei dati dal mondo web (in tempo reale), si utilizza molto il DataFeed toolbox. Nel capitolo sulle serie storiche si utilizza in maniera estensiva l'Econometric, il Financial ed il Curve Fitting toolbox.

L'installazione di questi toolboxes aggiuntivi può avvenire al momento dell'installazione di MATLAB oppure successivamente. In ogni caso, se l'utente chiama una funzione che necessita di un determinato toolbox MATLAB, propone un collegamento ipertestuale per l'installazione del toolbox mancante.

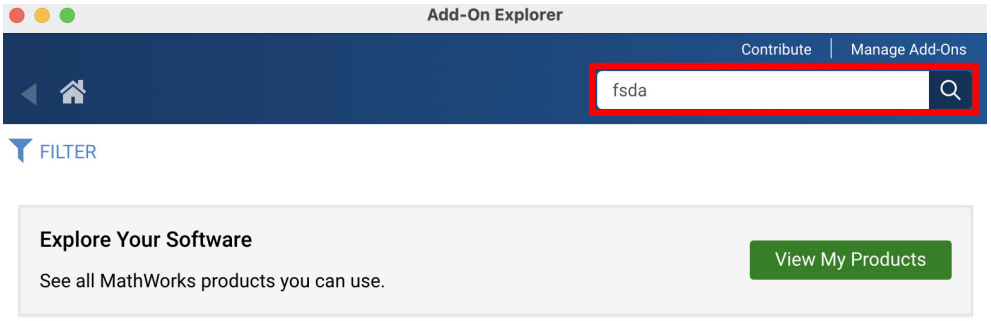
Nella *Command Window* di MATLAB digitando `ver` è possibile visualizzare sia la versione di MATLAB che si sta utilizzando sia i toolbox aggiuntivi che sono stati installati insieme alla loro versione.

Si suppone, inoltre che l'utente abbia installato un "Add-ons" aggiuntivo chiamato FSDA (Flexible Statistics and Data Analysis) Toolbox sviluppato dal Centro Interdipartimentale di Ricerca Ro.S.A. (Robust Statistics Academy) dell'Università di Parma congiuntamente con il J.R.C. (Joint Research Centre) della Commissione Europea. L'installazione di FSDA (essendo un Add-on) può essere eseguita tramite i seguenti passaggi:

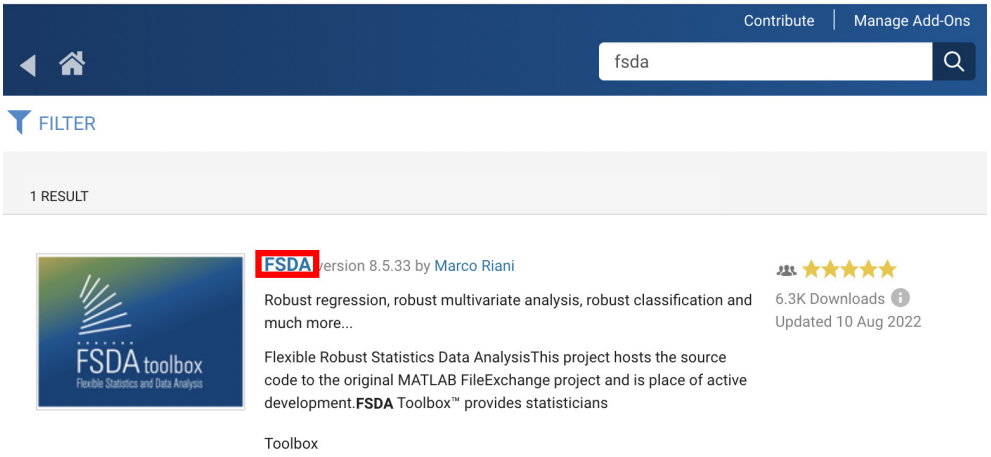
1. Nel tab "Home" selezionare "Add-ons" e successivamente "Get Add-ons"



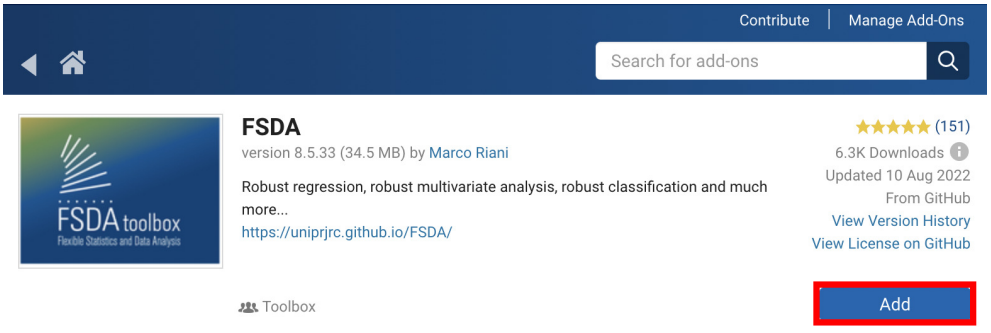
2. All'apertura della finestra "Add-ons Explorer" digitare "FSDA" nel riquadro di ricerca.



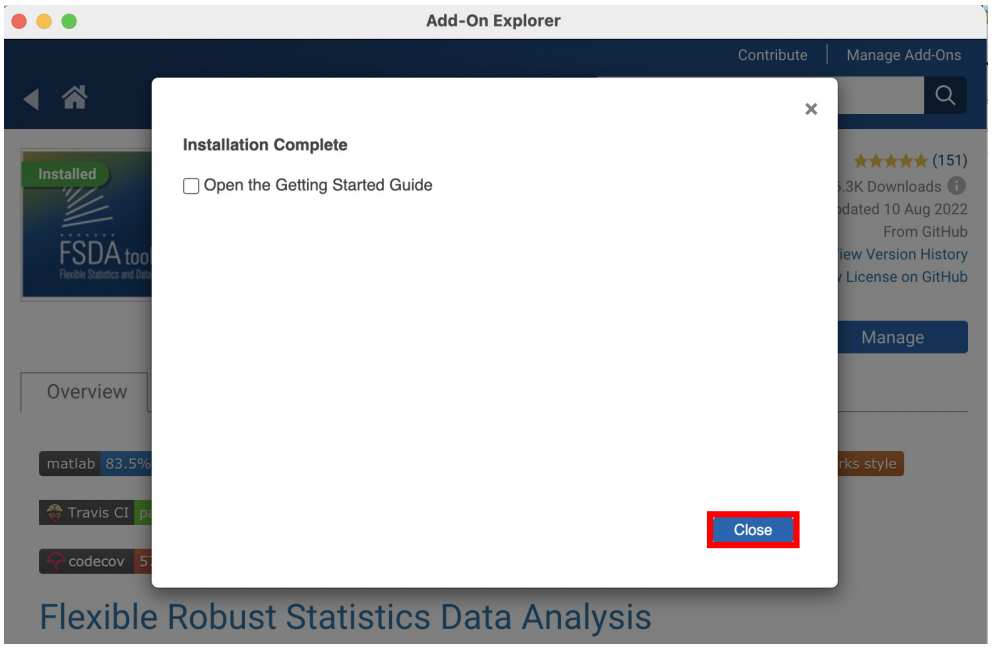
3. Selezionare “FSDA” nella pagina del Toolbox.



4. Selezionare “Add” e l’installazione si avvierà in automatico.



5. Alla fine dell'installazione, che richiederà qualche secondo, si aprirà la seguente finestra e dopo aver selezionato "Close" il Toolbox "FSDA" sarà pronto per essere utilizzato.



Tutti i file che sono menzionati in questo libro sono scaricabili da GitHub (la piattaforma di condivisione software leader nel mondo degli sviluppatori). Per scaricare questo materiale, una volta aperto MATLAB è sufficiente digitare il seguente comando dal prompt:

```
1  >> !git clone https://github.com/UniprJRC/DSconMATLAB
```

Se il comando precedente non dovesse funzionare, nel caso dei sistemi operativi Windows occorre scaricare il software open source `git` dal seguente indirizzo: https://git-scm.com. Per MAC OS è necessario installare `XCode` dall'App Store.

Come mantenere "vivo" il progetto

Il software ha un suo ciclo vitale e se non viene mantenuto ed utilizzato difficilmente mantiene interesse. Per questo motivo incoraggiamo il lettore a segnalare errori e proporre modifiche e miglioramenti accedendo alla pagina

GitHub del libro:

https://github.com/UniprJRC/DSconMATLAB.

È possibile inserire la segnalazione di inesattezza o la proposta dettagliata di modifica nella sezione Issues. Sarà nostra cura rispondere in tempi brevi ed eventualmente aprire un canale di comunicazione con l'autore della segnalazione.

1

Introduzione all'utilizzo di MATLAB e alla gestione dei dati

La prima parte del presente capitolo introduttivo ha lo scopo di mostrare i concetti di base per un utilizzo razionale dell'interfaccia grafica di MATLAB. Il capitolo nelle parti successive affronterà i temi legati alla formattazione dei diversi tipi di variabili, l'organizzare di tali variabili nelle strutture dati più comuni e la gestione dei dati all'interno delle suddette strutture.

1.1 L'interfaccia di MATLAB

MATLAB (abbreviazione di MATrix LABoratory) è un ambiente di calcolo numerico che consente di eseguire operazioni su matrici, gestire dati, implementare algoritmi, generare grafici e interfacciarsi con altri software. MATLAB è un software basato su un linguaggio di programmazione di alto livello caratterizzato dai costrutti tipici dei linguaggi di programmazione che viene utilizzato sia per la scrittura di piccoli programmi che per lo sviluppo di applicazioni particolarmente complesse.

MATLAB è un software a interfaccia grafica che può essere sintetizzata in una barra dei menu e degli strumenti e cinque finestre principali, come mostrato in Figura 1.1. Le finestre dell'interfaccia principale possono essere affiancate, spostate, ridotte a icona, ridimensionate ancorate ecc. . . , ciò consente di riorganizzare e personalizzare lo spazio di lavoro in base alle preferenze dell'utente.

I rettangoli colorati di Figura 1.1 mostrano rispettivamente:

- *barra dei menu e barra degli strumenti* (rettangolo giallo): sono caratterizzate da una serie di schede per navigare nei menu e da icone che permettono di accedere agli strumenti più comuni attraverso un click del mouse;

- *Command Window* (rettangolo verde): permette all'utente di digitare i comandi, le funzioni e le istruzioni che il software deve eseguire e di visualizzare in tempo reale i risultati;

- *Workspace* (rettangolo magenta): è lo spazio di lavoro o di memoria contenente le variabili dichiarate. Il valore di ciascuna variabile dichia-

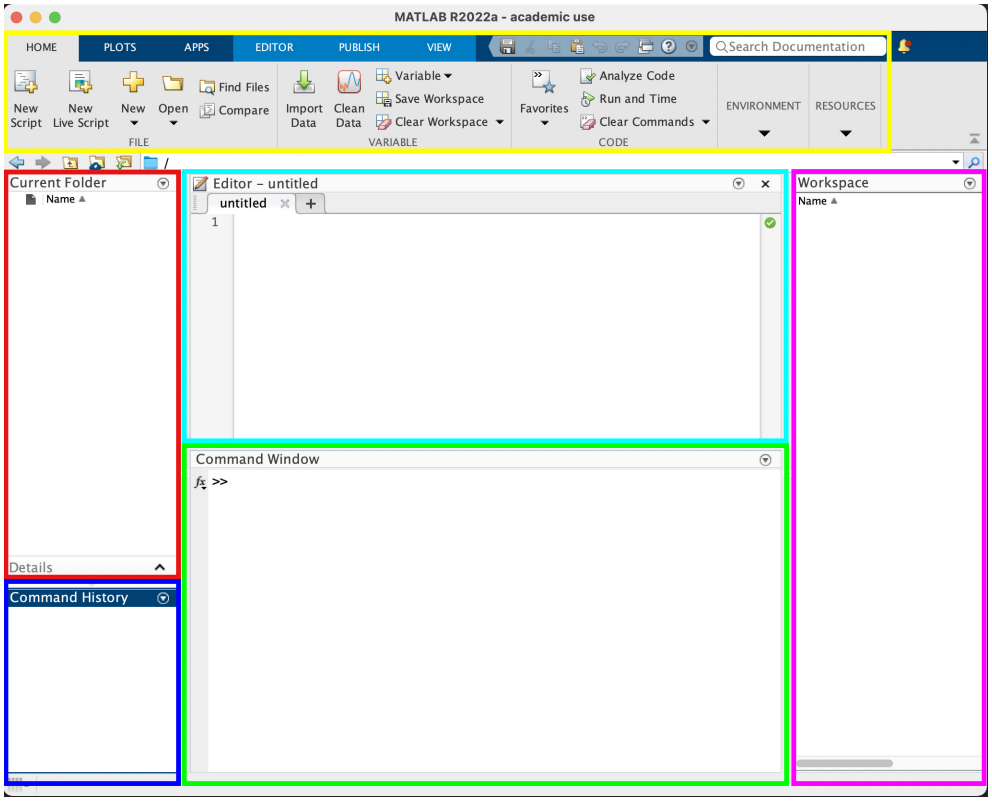


Figura 1.1: Struttura dell'interfaccia utente di MATLAB.

rata contenuta nel *Workspace* può essere visualizzato nella *Command Window* semplicemente scrivendo il nome della variabile stessa;

- *Current Folder* (rettangolo rosso): permette di esplorare il contenuto delle cartelle sul proprio disco o altro supporto di memoria e consente di aprire direttamente i file compatibili con un doppio click;

- *Command History* (rettangolo blu): in questa finestra sono elencati tutti i comandi digitati di recente, i comandi elencati possono essere rilanciati con un doppio click;

- *Editor* (rettangolo ciano): permette di scrivere codice che può essere salvato in un file con estensione nativa `.m` o `.mlx`. Si segnala fin da subito che all'apertura di MATLAB la finestra dell'*Editor* non compare in automatico ma deve essere aperta cliccando sulla barra dei menu o *New Script* o *New Live Script*. Come accennato, i comandi vengono eseguiti

nella *Command Window*, ma quando si deve far eseguire una lunga serie di comandi è preferibile creare uno script. Il file `.m` è un documento di testo che, come detto, può contenere una lunga serie di operazioni da eseguire e può essere creato e modificato attraverso l'*Editor*. Analogamente ai file `.m`, possono essere utilizzati i file `.mlx` i quali consentono non solo di scrivere comandi, ma anche di visualizzare l'output accanto al codice che li ha generati. Permettono di gestire il codice e i risultati con testo formattato, titoli, immagini e collegamenti ipertestuali e di salvare il codice, i risultati e il testo formattato in un singolo documento eseguibile. Si veda un esempio dei due tipi di file in Figura 1.2 e Figura 1.3.

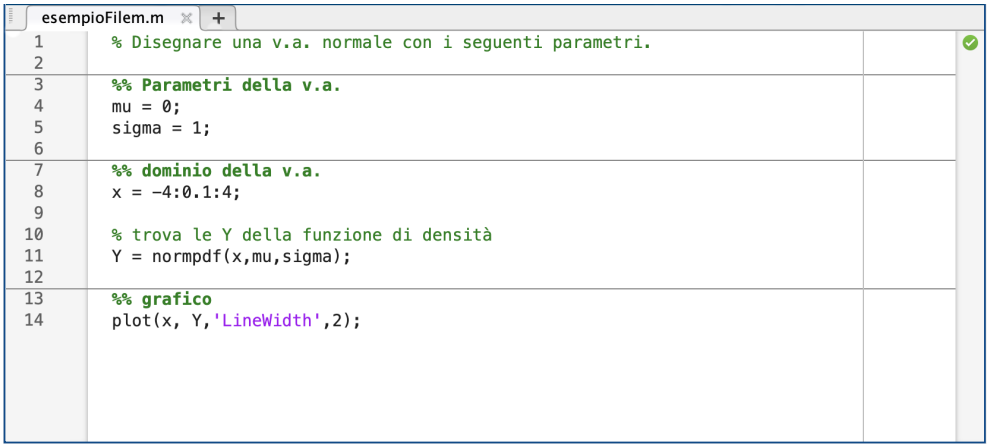


Figura 1.2: Esempio di codice contenuto nel formato file `.m`.

Sempre con riferimento alle Figure 1.2 e 1.3, dove a titolo di esempio viene mostrato uno script per la creazione della funzione di densità di una v.a. $X \sim N(0,1)$, si possono notare le differenze tra i due diversi formati. Il file `.m`, infatti, ha una funzione di semplice scrittura dello script da eseguire nella *Command Window*. Il testo preceduto dal simbolo % rappresenta un commento, ossia una parte di script che il software non esegue. L'output dello script in questo caso sarà dato dalla visualizzazione in una nuova finestra del grafico. Il file `.mlx`, invece, si presenta come un contenitore più versatile in cui si può scrivere uno script da eseguire (definito nel riquadro grigio), un testo al di fuori della finestra dedicata allo script (ad esempio la richiesta di disegnare un grafico) e infine il grafico ottenuto.

Figura 1.3: Esempio di codice contenuto nel formato file `.mlx`.

1.2 L'esecuzione del codice

Nei precedenti esempi è stato mostrato come creare gli script, vediamo adesso come eseguirli. Il modo più facile ma non particolarmente comodo è quello di copiare lo script dall'*Editor* alla *Command Window*, ma questo può avere senso pratico nel caso in cui si voglia eseguire una riga di codice o poco più. Riprendendo l'esempio di script di Figura 1.2, andiamo a mostrare i vari tipi di esecuzione dello script. I comandi mostrati in Figura 1.4 sono:

- *Run* (rettangolo rosso): esegue l'intero script (cliccando sulla freccia nera al di sotto del comando, inoltre, si apre una finestra con delle opzioni di esecuzione per il debugging del codice utili nella programmazione avanzata);
- *Step* (rettangolo nero): esegue lo script una riga per volta;
- *Run Section* (rettangolo blu): esegue una section. La section è una sezione di codice che può essere creata inserendo nello script il doppio simbolo di % quindi %%, come mostrato nel rettangolo magenta. MATLAB separa visivamente ciascuna sezione mediante delle linee orizzontali. La section può essere eseguita anche semplicemente cliccando sul segmento azzurro a sinistra dei numeri di riga dell'*Editor*;
- *Run and Advance* (rettangolo verde): esegue una section e passa alla successiva;
- *Run to End* (rettangolo ciano): partendo da una section qualsiasi esegue i comandi fino alla fine dello script;

- *Section Break* (rettangolo giallo): inserisce il doppio simbolo di % per creare una nuova section.

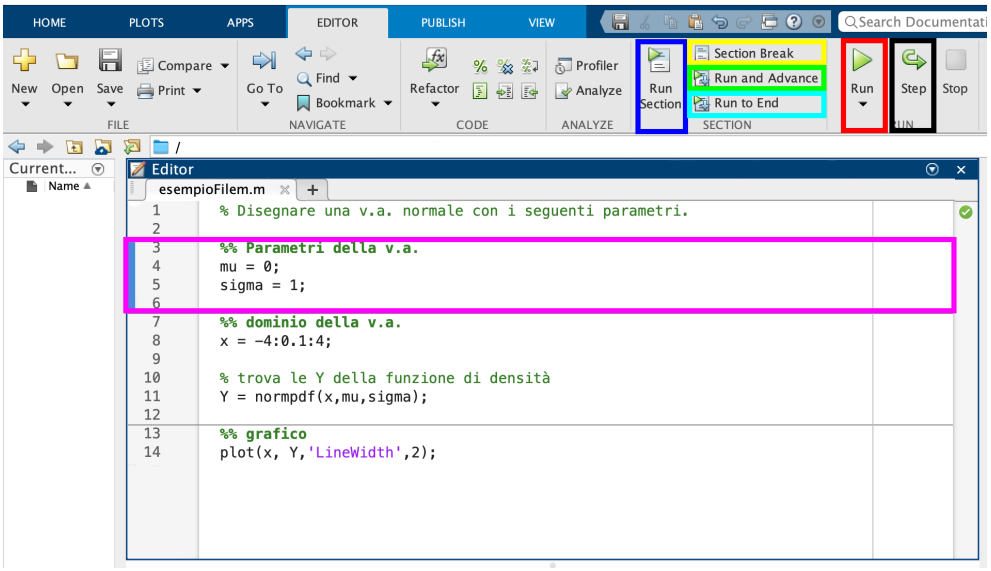


Figura 1.4: Presentazione dei vari comandi di esecuzione di uno script.

Seppur limitatamente ad un semplice accenno, è utile introdurre il concetto di debugging. Il debugging è una pratica essenziale nella programmazione poiché è lo strumento di diagnostica che consente di identificare e correggere eventuali errori commessi durante la programmazione.

Il debugger di MATLAB permette, cliccando sui numeri riga desiderati, di impostare dei Breakpoint che consentono di eseguire lo script fino alla riga prima del Breakpoint stesso. Come mostrato in Figura 1.5, per inserire un Breakpoint è sufficiente cliccare sul numero della riga nella quale si ha il sospetto della presenza di un errore.

Nelle operazioni di debugging l'utilizzo dei Breakpoint si rivela particolarmente utile quando lo script richiama funzioni esterne oppure quando si eseguono comandi all'interno di cicli. Da un punto di vista operativo, una volta posizionato il Breakpoint su riga 11 ed eseguito lo script, MATLAB eseguirà il codice fino alla riga 8 compresa dove si fermerà e mostrerà i nuovi comandi di debugging (Figura 1.6), i quali consentono di eseguire le seguenti operazioni:

- *Continue* (rettangolo rosso): esegue lo script dal Breakpoint al successivo se presente, altrimenti esegue lo script fino alla fine;

- *Step* (rettangolo verde): esegue lo script dopo il Breakpoint una riga per volta;

- *Step in* (rettangolo ciano): se la riga sotto indagine contiene una funzione esterna, questa viene aperta dal comando;

- *Step out* (rettangolo blu): la funzione aperta con Step in viene chiusa e viene riproposta la finestra dell'*Editor* che contiene lo script sotto verifica;

- *Stop* (rettangolo magenta): interrompe le operazioni di debugging.

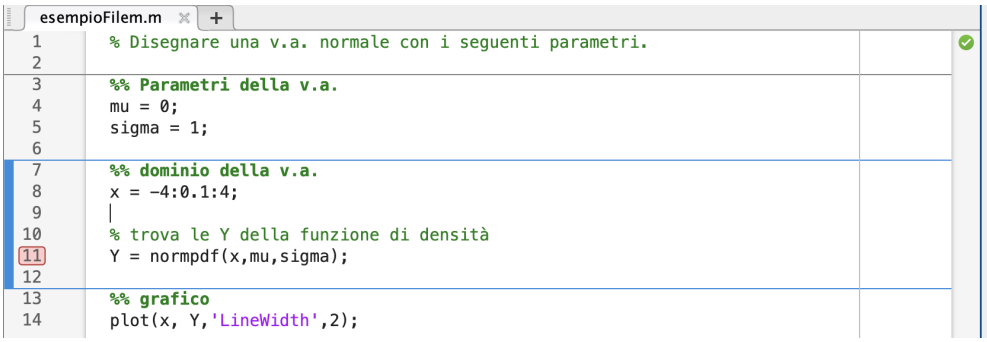


Figura 1.5: La riga 11 presenta un Breakpoint. Eseguendo lo script MATLAB eseguirà il codice fino alla riga 8 compresa.

Si rammenta che dopo aver eseguito le operazioni di debugging è necessario rimuovere i Breakpoint.

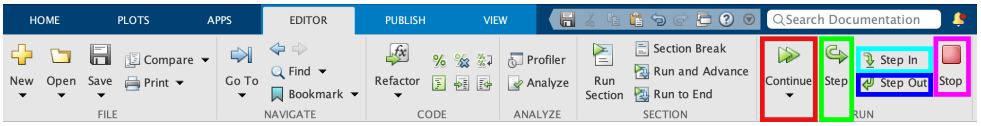


Figura 1.6: Barra dei menu della scheda EDITOR in modalità debugging.

1.3 Classi di dati

In MATLAB esistono molti tipi di classi di variabili, per brevità, in questo testo ci limiteremo ad introdurre solamente quelle funzionali alla gestione degli argomenti trattati nei capitoli successivi. Richiamando i principi di statistica

di base, distingueremo due macro-categorie di variabili, quelle quantitative (classe numerica) e quelle qualitative (classe di testo).

Le classi numeriche sono definite come `double` (impostazione di default per array numerici) e supportano tutte le operazioni di base sulle matrici, come indicizzazione, rimodellamento e operazioni matematiche.

Le classi di testo sono, invece, definite come "carattere" (`char`) o "stringa" (`string`) e sono in grado di archiviare variabili con modalità espresse in formato di testo.

Nel caso di costruzione di una variabile numerica, sarà quindi sufficiente assegnare un determinato valore numerico alla variabile: tale valore di default assumerà la classe `double`. Nel caso della creazione di una variabile destinata a contenere testo, MATLAB necessita di maggiori informazioni sulle proprietà da associare a tale oggetto, in quanto le classi di testo possono essere di tipo "carattere" (`char`) o "stringa" (`string`).

Nel primo caso, si definisce variabile `char` un vettore di caratteri e per creare tale variabile è necessario inserire il testo tra apici, ad esempio `c = ... 'Buongiorno a tutti'`. Nel secondo caso, una variabile `string` è un contenitore per un testo. È possibile creare una variabile `string` usando le doppie virgolette, ad esempio `s = "Saluti amico"`. I due metodi di creazione del testo differiscono in termini di dimensioni, infatti, la variabile `char` avrà come dimensione 1×18 (dove ogni carattere del testo occupa una colonna) mentre la variabile `string` è di dimensione 1×1 (in questo caso il testo viene interpretato come un blocco unico). Per le finalità delle prossime sezioni, verrà presa in considerazione solamente la classe `string` poiché è di più facile utilizzo e può essere gestita con le operazioni standard comuni agli array numerici.

Come mostrato nell'Esercizio 1.1, al fine di verificare la classe di una variabile esistente o creata è possibile utilizzare il comando `class()` dove all'interno delle parentesi tonde dovrà essere digitato il nome della variabile. Per verificare, invece, le dimensioni della variabile sarà necessario utilizzare il comando `size()`, sempre mettendo all'interno delle parentesi il nome della variabile stessa.
N.B.: alla fine di ogni riga di codice è consigliato utilizzare il punto e virgola in modo da eseguire la riga di codice senza che il risultato venga mostrato a monitor. Per richiamare il risultato ottenuto sarà sufficiente digitare il nome della variabile nella *Command Window* o fare doppio click sul nome della variabile contenuto nel *Workspace*.

Quando non si specifica una variabile di output, MATLAB usa la variabile `ans`, per immagazzinare i risultati di un calcolo.

Esercizio 1.1

Assegnare alla variabile `d` un numero, verificarne la classe e le dimensioni.

```
1 d = 3;
2 class(d)
3 size(d)
```

L'output di questo codice è:

```
1 ans =
2
3     'double'
4
5 ans =
6
7      1 1
```

Assegnare alla variabile `c` un carattere, verificarne la classe e le dimensioni.

```
1 c = 'Buongiorno a tutti';
2 class(c)
3 size(c)
```

L'output di questo codice è:

```
1 ans =
2
3     'char'
4
5 ans =
6
7      1 18
```

Assegnare alla variabile `s` una stringa, verificarne la classe e le dimensioni.

```
1 s = "Saluti amico";
2 class(s)
3 size(s)
```

L'output di questo codice è:

```
1 ans =
2
3     'string'
4
5 ans =
6
7      1 1
```

1.4 Gli array

Un *array* è definito come una struttura righe per colonne ($n \times p$) capace di contenere variabili appartenenti alle diverse classi descritte in precedenza. Dunque, possiamo dire che tutte le variabili di MATLAB sono array multidimensionali, indipendentemente dal tipo di dati.

1.4.1 Creazione dei diversi tipi di array

I diversi tipi di array possono essere creati includendo il contenuto tra operatori. Ad esempio, per creare un array numerico, ad esempio un vettore riga (orizzontale), è sufficiente inserire i numeri desiderati all'interno dell'operatore `[]` (parentesi quadre) separando gli elementi con una virgola o uno spazio. Se, invece, si necessita di un vettore colonna (verticale), gli stessi numeri devono essere separati dal simbolo del punto e virgola. Per creare una matrice 2×3 sarà sufficiente inserire all'interno dell'operatore `[]` i primi 3 numeri nella prima riga il punto e virgola e i successivi 3 numeri nella seconda riga. Un altro tipo di array numerico che è possibile creare è il cosiddetto vettore equispaziato, dove la distanza tra i numeri della successione in esso contenuto è costante.

Come precedentemente detto, agli oggetti contenuti nei due array dell'esempio 1.2 di default MATLAB associa la classe `double`.

Esercizio 1.2

Creare un vettore orizzontale chiamato `ono` composto dai primi tre numeri naturali.

```
1 ono = [0 1 2];
```

Creare un vettore verticale chiamato `vn` composto dai primi tre numeri naturali.

```
1 vn = [0; 1; 2];
```

Creare una matrice 2×3 chiamata `mn` composto dai primi sei numeri naturali.

```
1 mn = [0 1 2; 3 4 5];
```

Creare un vettore equispaziato chiamato `nd` composto dai numeri dispari nell'intervallo [1, 15].

```
1 nd = 1:2:15;
```

La creazione di array di testo utilizzando la classe `string` è basata sulla stessa logica vista negli array numerici, con la sola differenza di dover inserire il testo tra le doppie virgolette. Nell'Esercizio 1.3 seguente mostriamo come creare un array orizzontale o verticale.

Esercizio 1.3

Creare un vettore riga (orizzontale) di dimensione 1×2 chiamato `ot` con il nome "John" e il cognome "Doe".

```
1 ot = ["John" "Doe"];
```

Creare un vettore colonna (verticale) di dimensione 2×1 chiamato `vt` con il nome "John" e il cognome "Doe".

```
1 vt = ["John"; "Doe"];
```

Anche se per la creazione degli array sarà utilizzata la classe `string`, approfittiamo di questa sede per mostrare la differenza di comportamento delle classi di testo nella creazione degli array. Come appena visto nell'esercizio, la classe `string` consente di ottenere array di dimensione 1×2 o 2×1 poiché nome e cognome hanno lunghezza unitaria (in quanto interpretati come blocchi di testo). A parità di operazioni, se fosse stata utilizzata la classe `character` come risultato del vettore orizzontale avremmo ottenuto il concatenamento di nome e cognome quindi `JohnDoe`. Nel caso di creazione di un array verticale, invece, sarebbe stato impossibile ottenere il risultato poiché la lunghezza del nome e quella del cognome sono differenti e ciò rende impossibile l'operazione. In altri termini:

```
>> ['John' 'Doe']

ans =

    'JohnDoe'

>> ['John'; 'Doe']
Error using vertcat
Dimensions of arrays being concatenated are not consistent.
```

MATLAB mette a disposizione anche un altro tipo di array chiamato `cell`. Questo tipo di contenitore è caratterizzato da celle indicizzate, in cui ogni cella può contenere qualsiasi tipo di dato, si veda l'Esercizio 1.4. Gli array di celle contengono comunemente elenchi di testo, combinazioni di testo e numeri o array numerici di dimensioni diverse. Come nel caso degli array numerici o di testo anche la dimensione dei `cell` array deve essere consistente, in altre parole non è possibile creare un array che abbia un numero di elementi differenti tra una riga e la successiva. A differenza dei precedenti tipi di array le `cell` sono definite dall'operatore `{}`.

Esercizio 1.4

Creare un `cell` array chiamato `cm` di dimensione 2×3. La prima riga deve contenere i numeri 1, 2, 3. L'elemento 2, 1, deve contenere il testo `"Buongiorno a tutti"`. L'elemento 2, 2, deve contenere una matrice di dimensioni 3×2 con numeri a piacere nell'intervallo $[0, 1]$ tramite la funzione `rand`. L'elemento 2, 3 deve contenere un vettore colonna composto dai numeri 11, 22, 33.

```
cm= {1,2,3; "Buongiorno a tutti", rand(3,2),[11; 22; 33]};
```

L'output del precedente esercizio è il seguente:

```
m =

  2x3 cell array

    {[ 1]} {[ 2]} {[ 3]}
    {["Buongiorno a tutti"]} {3x2 double} {3x1 double}
```

Un oggetto, sempre appartenente alla famiglia degli array, abbastanza simile alle `cell`, è la `struct` o struttura. Una struttura è una tipologia di dati che raggruppa i dati correlati utilizzando contenitori chiamati campi. Ogni

campo può contenere qualsiasi tipo di dato. Per accedere ai dati in una struttura si utilizza la notazione `nomeStruttura.nomeCampo`. Nell'Esercizio 1.5 si propone la procedura per la creazione, attraverso una `struct`, di quanto è stato proposto nell'Esercizio 1.4.

Esercizio 1.5

Creare una `struct` chiamata `st`. il primo oggetto deve contenere il numero 1, il secondo il numero 2, il terzo il numero 3, il quarto il testo `"Buongiorno a tutti"`, il quinto una matrice di dimensione 3×2 con numeri a piacere nell'intervallo $[0, 1]$, il sesto un vettore colonna composto dai numeri 11, 22, 33.

```
st = struct;
st.a = 1;
st.b = 2;
st.c = 3;
st.d = "Buongiorno a tutti";
st.e = rand(3,2);
st.f = [11; 22; 33];
```

L'output del precedente Esercizio 1.5 è il seguente:

```
st =
 struct with fields:
   a: 1
   b: 2
   c: 3
   d: "Buongiorno a tutti"
   e: [3x2 double]
   f: [3x1 double]
```

1.4.2 Estrazione dei dati da un array

In questa sezione si mostrerà come accedere agli elementi dei vettori e/o matrici. Si prenda ad esempio la matrice di dimensioni 3×2 e denominata `A` riportata nella Tabella 1.1.

Nella Tabella 1.2 sono riportati alcuni comandi di esempio per estrarre dalla matrice A i suoi elementi.

L'estrazione di elementi dalle `cell`, essendo queste strutture più complesse, necessitano di un metodo a due livelli. Facendo riferimento all'esempio del

Operazione	Comando	Risultato
Creazione della matrice A di numeri causali	`A = rand(3, 2);`	$A = \begin{bmatrix} 0.2085 & 0.8592 \\ 0.4817 & 0.1712 \\ 0.4205 & 0.3389 \end{bmatrix}$

Tabella 1.1: Codice per la generazione di A dopo aver fissato un determinato seed di numeri casuali. Il comando `rng()` è necessario per fissare il cosiddetto seed il quale consente di controllare un generatore random di numeri. In altre parole, inserendo un numero all'interno delle parentesi tonde del comando `rng()` è possibile evitare che i numeri generati casualmente siano diversi ad ogni generazione, ciò consente di rendere ripetibili gli esperimenti. In questo esempio è stato utilizzato il comando `rng(22)`.

Operazione	Comando	Risultato
Estrazione dell'elemento in posizione 3, 2	`A(3,2)`	`0.3389`
Estrazione della riga 3	`A(3,:)`	`0.4205 0.3389`

Estrazione della colonna 2	`A(:,2)`	`0.8592`
		`0.1712`
		`0.3389`

Estrazione degli elementi all'incrocio delle righe 2 e 3 e colonne 1 e 2	`A(2:3,1:2)`	`0.4817 0.1712` `0.4205 0.3389`
Estrazione delle righe 1 e 3	`A([1 3],:)`	`0.2085 0.8592` `0.4205 0.3389`
Estrazione dell'elemento all'incrocio dell'ultima riga e della colonna 2	`A(end, 2)`	`0.3389`

Tabella 1.2: Comandi per l'estrazione degli elementi dell'array.

`cell` array `cm` creato precedentemente, se si vuole estrarre la cella è necessario racchiudere gli indici tra parentesi tonde, se invece si vuole accedere al contenuto indicizzato della cella bisogna inserire gli indici tra parentesi graffe. Riportiamo di seguito, in Tabella 1.3, alcuni comandi per estrarre gli elementi da una `cell`.

Come intuitivamente si può dedurre dalla descrizione delle differenze tra `cell` e `struct`, l'estrazione da una struttura ha similitudini con le estrazioni dagli altri array ma questa avviene utilizzando i nomi delle variabili contenute nella struttura. La Tabella 1.4 presenta alcuni esempi di estrazione relativi

Operazione	Comando	Risultato
Estrazione della cella in posizione 2, 3	`cm(2,3)`	`{3x1 double}`

Accesso al contenuto della cella in posizione 2, 2	`cm{2,2}`	`0.2085 0.8592`
		`0.4817 0.1712`
		`0.4205 0.3389`

Tabella 1.3: Comandi per l'estrazione degli elementi da un `cell` array. Il primo comando è utilizzato per l'estrazione di una cella, mentre il secondo per accedere al contenuto di una cella. Anche in questo caso la funzione `rand` ha come seed del comando `rng()` il numero 22.

alla `struct` creata nell'Esercizio 1.5.

Operazione	Comando	Risultato
Estrazione del terzo elemento	`st.c`	`3`

Estrazione della seconda colonna del quinto elemento	`st.e(:,2)`	`0.8592`
		`0.1712`
		`0.3389`

Tabella 1.4: Comandi per l'estrazione degli elementi da una `struct` array. Il primo comando è utilizzato per l'estrazione del contenuto della terza variabile, mentre il secondo comando estrae la seconda colonna della quinta variabile. La funzione `rand` della `struct` ha sempre come seed del comando `rng()` il numero 22.

1.5 Le tabelle

Le `table` possono essere definite come array caratterizzati da un nome per ciascuna colonna dove ogni variabile può contenere un diverso tipo di dati. La tabella è un tipo di struttura adatto per importare dati in colonna o in forma tabulare che vengono spesso archiviati in file di testo o in fogli di calcolo.
Una `table` ha la struttura mostrata in Figura 1.7, dove le righe sono numerate progressivamente e le colonne hanno un etichetta che indica il nome della variabile. Come è mostrato nell'esempio riportato in figura, la `table` può contenere le diverse classi di dati introdotte in precedenza.

Firm

107x10 table

	1 Code	2 Surname	3 Name	4 Gender	5 BirthDate	6 Education	7 Wage	8 CommutingTime	9 SmartWorkHours	10 Seniority
1	'P0211'	'BASILICO'	'TIZIANA'	'F'	16-Jun-1991	'A'	1877	27	10	11
2	'P0212'	'BONINO'	'PAOLO'	'M'	01-Jul-1993	'B'	2375	35	12	8
3	'P0213'	'GRI'	'ALBERTO'	'M'	19-Oct-1987	'C'	2450	39	16	11
4	'P0214'	'BARBIERI'	'ROSSANA'	'F'	28-Jan-1992	'B'	2037	16	6	7
5	'P0215'	'PRENNA'	'FEDERICA'	'F'	22-Jan-1966	'C'	3893	23	8	30
6	'P0216'	'MONTAN...	'GIACOMO'	'M'	24-Sep-1974	'A'	2314	29	10	23
7	'P0217'	'MAROVINO'	'ELISABETTA'	'F'	14-Oct-1966	'A'	1893	29	6	12

Figura 1.7: Esempio di visualizzazione della struttura di una table.

1.5.1 Creazione di una tabella

Vediamo adesso uno dei modi possibili con cui creare una tabella partendo da un array. Per parsimonia verrà mostrato il codice MATLAB necessario alla creazione delle prime due righe della table mostrata in figura. Come primo passo è necessario creare un array.

```
data = ["P0211", "BASILICO", "TIZIANA", "F", "16-Jun-1991", "A", 1877, 27, 10, 11;
    "P0212", "BONINO", "PAOLO", "M", "01-Jul-1993", "B", 2375, 35, 12, 8];
```

Una volta creato l'array che MATLAB interpreta come `string` array è necessario convertirlo in tabella con il comando `array2table` e definire le proprietà della tabella. Il comando `array2table` richiede come primo argomento l'array creato in precedenza e come secondo argomento `"VariableNames"`. A seguito del secondo argomento andremo ad inserire i nomi delle variabili che dovranno essere contenuti in uno string array.

```
Firm = array2table(data,"VariableNames",["Code" "Surname" "Name" "Gender" ...
    "BirthDate" "Education" "Wage" "CommutingTime" "SmartWorkHours" ...
    "Seniority"]);
```

La creazione delle tabelle attraverso l'utilizzo di uno script è una prassi molto frequente quando si creano tabelle contenenti delle statistiche descrittive. Mostriamo di seguito un esempio di creazione di una tabella contenente le statistiche descrittive di un dataset relativo ad alcune variabili rilevate in un certo punto vendita.

```
data = [110.63, 3.7; 736871, 12157];

Summary = array2table(data,"VariableNames",["Acquisti in euro", "Numero ...
    visite"], "RowNames", ["Media mensile", "Totale mensile"]);
```

Da notare che in questo caso è stata introdotta l'opzione `RowNames` poiché anche le righe, oltre alle colonne, devono riportare la descrizione dell'operazione eseguita sui dati (Media mensile e Totale degli acquisti e delle visite). Il risultato dello script è mostrato in Figura 1.8.

	Acquisti in euro	Numero visite
Media mensile	110.63	3.7
Totale mensile	736871	12157

Figura 1.8: Tabella delle statistiche descrittive.

1.5.2 Importazione di una tabella

Come già visto, la creazione di una tabella è una tipica operazione che si esegue per salvare i risultati ottenuti a seguito dell'applicazione di una elaborazione. Più frequentemente le tabelle sono strutture caratterizzate da dati grezzi da elaborare e già esistenti in altri formati. In generale, MATLAB supporta l'importazione di formati di testo, immagini, video e audio. Rimanendo sempre in linea con le finalità del testo, si analizza la procedura per importare, sotto forma di tabella, dati testuali da due sorgenti di esempio: i file di testo e i fogli di Excel. In questa sede mostreremo due modi per importare un file Excel in MATLAB, attraverso: menu o riga di comando.

Il processo di importazione di un file dati esterno da menu, avviene attraverso il comando "Import Data" (rettangolo rosso) presente nella scheda "HOME" della barra degli strumenti di MATLAB, mostrato in Figura 1.9

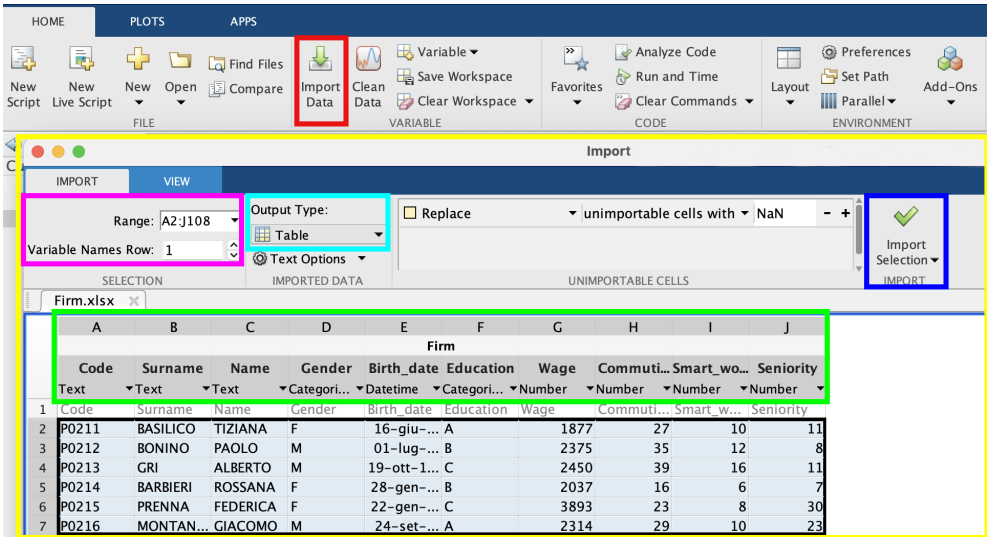


Figura 1.9: Schermata di importazione dei dati esterni.

Cliccando sul comando “Import Data”, MATLAB apre una finestra di navigazione che consente di andare a scegliere il file da importare, nell’esempio viene utilizzato il file `Firm.xlsx`. Una volta selezionato il file, si apre l’interfaccia di importazione (rettangolo giallo), all’interno di tale interfaccia il software propone (rettangolo magenta) il “Range” di righe e di colonne del file esterno contenenti dati e la riga in cui sono contenuti i nomi delle variabili. MATLAB mostra un’anteprima dei dati che si sta per importare (rettangolo nero) e propone la struttura dati più coerente, che in questo esempio corrisponde ad una tabella (rettangolo ciano). Sempre attraverso una lettura preliminare dei dati esterni vengono suggerite le classi delle variabili da importare (rettangolo verde). Da notare che tutto ciò che il software propone può essere modificato dall’utente nel caso in cui compaiano incongruenze (non del tutto impossibili perché MATLAB esegue la lettura preliminare dei dati in modo campionario). Una volta verificata la proposta di importazione, sarà sufficiente cliccare su “Import Selection” (rettangolo blu scuro) per caricare i dati nel *Workspace*.

Un metodo alternativo per importare un file Excel senza passare dagli strumenti del menu è il comando `readtable`. L’Esercizio 1.6 mostra la sintassi del comando per l’importazione del file `Firm.xlsx`.

Esercizio 1.6

Importare il file `Firm.xlsx` attraverso il comando `readtable`.

```
1 Firm = readtable('Firm.xlsx','Sheet','data','Range','A1:J108', ...
      'ReadRowNames', 0);
```

Il comando `readtable` come primo argomento, se il file da importare è presente nel *Current Folder*, necessita solo del nome del file stesso, altrimenti necessita dell'intero percorso che identifica la posizione del file nel disco. Nelle opzioni aggiuntive è possibile specificare il foglio (`'Sheet'`) del file di Excel da importare e la porzione del foglio contenente i dati `'Range'` e come gestire la prima colonna della tabella. Attraverso l'opzione `'ReadRowNames'` uguale a 0 (`false`) o a 1 (`true`) è possibile gestire il contenuto della colonna rispettivamente come modalità della variabile o nomi delle righe. A titolo di esempio la Figura 1.10 mostra gli output relativi all'importazione del file `Firm.xlsx`

107×10 table

Code	Surname	Name
{'P0211'}	{'BASILICO'}	{'TIZIANA'}
{'P0212'}	{'BONINO'}	{'PAOLO'}
{'P0213'}	{'GRI'}	{'ALBERTO'}
{'P0214'}	{'BARBIERI'}	{'ROSSANA'}
{'P0215'}	{'PRENNA'}	{'FEDERICA'}
{'P0216'}	{'MONTANARI'}	{'GIACOMO'}
{'P0217'}	{'MAROVINO'}	{'ELISABETTA'}

107×9 table

	Surname	Name
P0211	{'BASILICO'}	{'TIZIANA'}
P0212	{'BONINO'}	{'PAOLO'}
P0213	{'GRI'}	{'ALBERTO'}
P0214	{'BARBIERI'}	{'ROSSANA'}
P0215	{'PRENNA'}	{'FEDERICA'}
P0216	{'MONTANARI'}	{'GIACOMO'}
P0217	{'MAROVINO'}	{'ELISABETTA'}

Figura 1.10: Nel pannello di sinistra è mostrato un estratto dell'output della tabella importata con il comando `readtable` con l'opzione `ReadRowNames` = 0, mentre nel pannello di destra con l'opzione `ReadRowNames` = 1.

Osservazione: dalla versione di MATLAB 2021a è possibile utilizzare al posto della coppia (`'Name', Value`) la sintassi con l'uguale (`nome=valore`). Ad esempio, il codice riportato nell'Esercizio 1.6 poteva anche essere scritto come:

```
1 Firm = readtable('Firm.xlsx',Sheet='data',Range='A1:J108',ReadRowNames=0)
```

1.5.3 Introduzione ai task

Nel linguaggio di MATLAB i *task* sono delle finestre che appaiono dentro il file .mlx che consentono di generare in automatico le righe di codice necessarie per raggiungere un determinato obiettivo (da qui il nome *task*). Ad esempio, se si apre un nuovo file di tipo .mlx una volta digitato la parte iniziale della parola import, appare in automatico la finestra riportata nella Figura 1.11. Se si fa click sulla voce "Import Data", appare in automatico un menu con

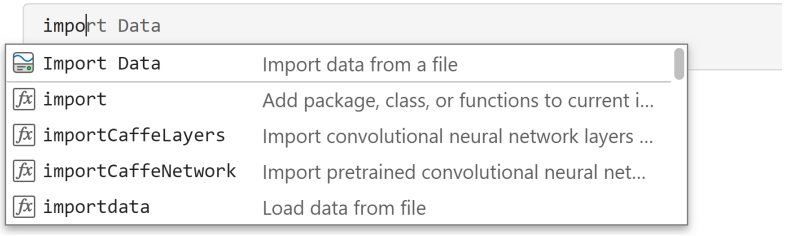


Figura 1.11: Schermata di autocompletamento che appare dopo aver digitato la parte iniziale della parola import.

il pulsante "Browse..." dove è possibile caricare manualmente il file che si desidera importare. Una volta selezionato il file, appare la schermata riportata nella Figura 1.12. La voce "Show Code" (in basso a sinistra) consente

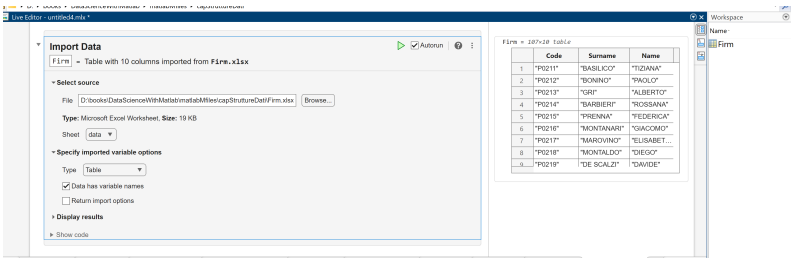


Figura 1.12: Schermata che appare dopo aver selezionato il file da importare (in questo caso `Firm.xlsx`). Nella finestra di output a destra appare in automatico l'anteprima dell'importazione dei dati nella table `Firm`. La variabile `Firm` viene aggiunta al *Workspace* (v. riquadro a destra).

di visualizzare in automatico il codice che viene generato da MATLAB per caricare i dati (in questo caso viene chiamata la funzione `readtable` e l'output viene inserito in una table denominata `Firm`). Il pulsante ⋮ dei 3 puntini verticali consente di alternare tra la Visualizzazione solo codice "Code Only", Controlli e Codice "Controls and Code" oppure solo controlli "Controls Only". La casella di controllo "Autorun" consente di specificare se il codice deve essere automaticamente eseguito. Lasciamo al lettore l'esplorazione delle altre opzioni.

In generale, se si fa click sulla freccia in basso del pulsante Task nella scheda *LIVE EDITOR*, è possibile selezionare l'obiettivo che interessa. Ad esempio, supponiamo che il nostro obiettivo sia quello di capire come il salario (variabile "Wage") dipende dagli anni di anzianità di servizio (variabile "Seniority"). Se

si vuole creare un diagramma di dispersione tra queste due variabili, dopo aver selezionato il task "Create Plot" (v. Figura 1.13), è possibile dalla casella a

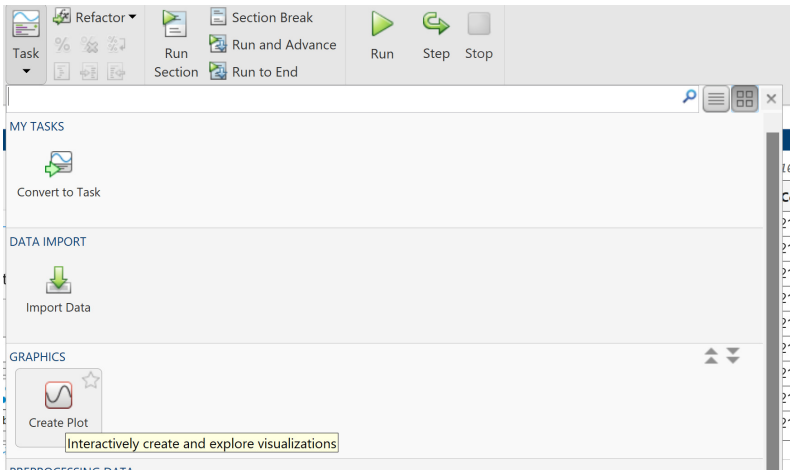


Figura 1.13: Schermata che mostra come selezionare il task Create Plot dal menu Task della scheda *LIVE EDITOR*.

discesa "Select Data" selezionare la table che interessa e dopo aver fatto click sul grafico "Scatter" selezionare le due variabili della table che interessano. L'anteprima output è riportata nella Figura 1.14.

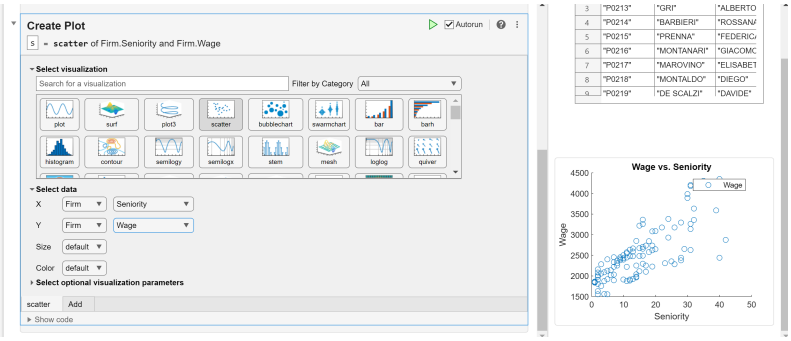


Figura 1.14: Costruzione del diagramma a dispersione tra le variabili "Seniority" e "Wage" della table `Firm` tramite il task "Create Plot".

1.5.4 Estrazione dei dati da una tabella

Nelle sezioni precedenti è stato mostrato come estrarre dati da array e da cell, adesso vediamo come estrarre dati da una tabella. Se si necessita di estrarre subset di dati da una tabella è possibile utilizzare lo stesso metodo visto con gli array oppure, come accennato nella sezione 1.5.1, è possibile sfruttare (per definire le variabili sulle quali vogliamo eseguire l'estrazione) il metodo `nomeTabella.nomeVariabile`. Nella Figura 1.14 in alto sotto la voce Create Plot MATLAB aveva scritto **scatter** of `Firm.Seniority` and `Firm.Wage` per indicare che erano state estratte le variabili denominate "Seniority" e "Age" dalla table `Firm`.

Esercizio 1.7

1. Estrarre dal dataset `Firm.xlsx` (importato con `RowNames` = 0 in una table denominata `Firm`) i primi 5 valori della variabile "Wage".

   Soluzione con metodo degli array.
   N.B.: il risultato sarà una `table`.

```
1 solarray1 = Firm(1:5,7);
```

   Soluzione con metodo delle tabelle.
   N.B.: il risultato sarà o uno `string` array o un `double` a seconda della classe delle variabili estratte.

```
1 soltable1 = Firm.Wage(1:5);
```

```
solarray1 =

  5×1 table

    Wage
    ____

    1877
    2375
    2450
    2037
    3893
```

Output metodo array.

```
soltable1 =

        1877
        2375
        2450
        2037
        3893
```

Output metodo table.

Figura 1.15: Output dell'estrazione dei primi 5 valori della variabile "Wage" ottenuto con i due metodi proposti.

2. Estrarre dal dataset Firm i primi 5 valori della variabile "Wage" e della variabile "Seniority".

   Soluzione con metodo degli array.
   N.B.: questo metodo consente di estrarre valori indipendentemente dalla loro classe e il risultato sarà una `table`.

```
1 solarray2 = Firm(1:5,[7, 10]);
```

   Soluzione con metodo delle tabelle.
   N.B.: questo metodo consente di estrarre valori appartenenti alla stessa classe o a limite da classi "compatibili" e il risultato avrà la classe delle variabili estratte o nel secondo caso il risultato sarà uno `string` array.

```
1 soltable2 = [Firm.Wage(1:5), Firm.Seniority(1:5)];
```

```
solarray2 =

  5×2 table

    Wage    Seniority
    ____    _________

    1877        11
    2375         8
    2450        11
    2037         7
    3893        30
```

Output metodo array.

```
soltable2 =

    1877        11
    2375         8
    2450        11
    2037         7
    3893        30
```

Output metodo table.

Figura 1.16: Output dell'estrazione dei primi 5 valori delle variabili "Wage" e "Seniority" ottenuto con i due metodi proposti.

3. Estrarre il soggetto P0219 dal dataset Firm importato con l'opzione `ReadRowNames=1`.

```
1 rowP0219 =Firm('P0219',:);
```

```
rowP0219 =

  1×9 table

          Surname          Name        .....    SmartWorkHours    Seniority
          _____________    _________   .....    ______________    _________

P0219     {'DE SCALZI'}    {'DAVIDE'}  .....          4               2
```

Figura 1.17: Output dell'estrazione di un soggetto del dataset utilizzando il nome della riga.

4. Estrarre i soggetti P0219 e P0476 dal dataset Firm importato con l'opzione `ReadRowNames=1`.

```
1 rowP0219P0476 =Firm({'P0219', 'P0476'},:);
```

```
rowP0219P0476 =

  2×9 table

            Surname          Name        .....    SmartWorkHours    Seniority
            ____________    _________             ______________    _________

    P0219   {'DE SCALZI'}    {'DAVIDE'}    .....          4              2
    P0476   {'FAVELLA'  }    {'FELICE'}    .....         12              7
```

Figura 1.18: Output dell'estrazione di più soggetti del dataset utilizzando i nomi delle righe.

1.6 Gestione dei dati

In questo capitolo verranno mostrati alcuni metodi di gestione dei dataset. Molto spesso, infatti, nel processo di analisi dati si è davanti a dataset che contengono più informazioni di quelle necessarie allo scopo della ricerca, si deve quindi ricorrere ad alcuni metodi di estrazione di segmenti contenenti solo le osservazioni che rispondono a precisi criteri. Esistono, altresì, casi in cui i dati da analizzare sono contenuti in tabelle diverse ed è dunque necessario, preliminarmente, unire le tabelle stesse. Prima di procedere è necessario, sia per concludere gli argomenti dei precedenti paragrafi, sia per avere nuovi strumenti funzionali alle sezioni successive, introdurre i metodi di salvataggio dei risultati in MATLAB.

1.6.1 Salvataggio dei risultati ottenuti

MATLAB mette a disposizione molti metodi di salvataggio dei dati ma per gli scopi di questo testo ne verranno presentati solo alcuni. Affronteremo due macro-categorie di metodi di salvataggio dei risultati legati al tipo di utilizzo che di questi verrà fatto. Prima di salvare, infatti, la domanda che l'utente deve porsi è se i dati saranno nuovamente utilizzati in MATLAB oppure se dovranno essere "digeribili" da altri software. Nel caso in cui si decida di riutilizzare ciò che è stato salvato esclusivamente su MATLAB il file nativo per il salvataggio ha estensione `.mat`. A titolo di esempio riprendiamo una `cell` vista nella sezione 1.4.1.

```
1 cm = {1,2,3; "Buongiorno a tutti", rand(3,2),[11; 22; 33]};
2 save('myCell.mat', 'cm');
```

L'estensione `.mat` è di particolare interesse, non solo perché permette di salvare in formato nativo i singoli risultati ma consente di salvare anche l'in-

tero contenuto del *Workspace*. Questo aspetto è di fondamentale importanza quando si eseguono script computazionalmente molto onerosi e si ha bisogno di salvare ogni passaggio intermedio dei calcoli eseguiti. Il salvataggio del *Workspace* ha una sintassi molto elementare che può essere generalizzata con `save('nomefile')`. Per utilizzare un file `.mat` salvato, indipendentemente dal fatto che esso contenga una sola variabile o un intero *Workspace*, è sufficiente caricarlo utilizzando il comando `load('nomefile')`.

Nei casi in cui i risultati ottenuti, ad esempio in formato `table`, devono poter essere utilizzati su altri software, MATLAB consente il salvataggio in una grande quantità di formati (ad esempio `.txt`, `.xlsx`, ecc...). Sempre a titolo di esempio, prendiamo nuovamente un oggetto già visto in precedenza (sezione 1.5.1)

```
1 data = [110.63, 3.7; 736871, 12157];
2
3 Summary = array2table(data,"VariableNames",["Acquisti in euro", "Numero ...
      visite"], "RowNames", ["Media mensile", "Totale mensile"]);
4
5 % Salvataggio della matrice "data" in ".txt" e in ".xlsx":
6 writematrix(data,'data.txt','Delimiter','tab');
7 writematrix(data,'data.xlsx');
8
9 % Salvataggio della tabella "Summary" in ".txt" e in ".xlsx":
10 writetable(Summary,'Summary.txt','Delimiter','tab','WriteRowNames',true);
11 writetable(Summary,'Summary.xlsx','WriteRowNames',true);
```

L'esempio appena mostrato comprende due modalità di salvataggio. L'oggetto `data` è una `matrix` e l'oggetto `Summary` è una `table`. Esiste un metodo di salvataggio dedicato per ciascuno dei due oggetti: `writematrix` e `writetable`. Entrambi i metodi di salvataggio possono generare file `.txt` o `.xlsx`. Nel caso del salvataggio in file di testo è consigliabile definire il delimitatore dei campi (di default MATLAB usa la virgola), nell'esempio proposto è stato utilizzato il tabulatore. Da notare che, per quanto riguarda il salvataggio della tabella, contenendo questa i `RowNames` è stata utilizzata l'opzione `'WriteRowNames', true`. Nel caso in cui il primo campo non fosse stato caratterizzato dai nomi di riga avremmo dovuto utilizzare l'opzione `'WriteRowNames',false` o più semplicemente ometterla. Come già detto nella sezione relativa all'importazione dei dati, anche nel contesto del salvataggio, MATLAB salva i file nel *Current Folder* se non specificato un percorso diverso all'interno del comando.

1.6.2 Estrazione dei dati in base a criteri

Un subset può essere estratto da un dataset utilizzando uno o più criteri e quando i criteri sono almeno due, è necessario stabilire quale relazione deve esistere tra essi. Si supponga di voler estrarre dal dataset Firm un subset composto da donne con un certo titolo di studio. Una volta individuati i criteri, è necessario stabilire come questi devano essere soddisfatti, in altre parole, è necessario definirne le relazioni. Le relazioni tra i criteri, in informatica, sono gestite dagli operatori logici "AND" o "OR". L'operatore logico "AND" prevede che tutti i criteri siano soddisfatti contemporaneamente, mentre l'operatore logico "OR", meno restrittivo, richiede che i criteri siano soddisfatti uno alla volta. In MATLAB, a livello di codice, gli operatori "AND" e "OR" sono sostituiti rispettivamente dai simboli `&` e |. Andando a definire i criteri e le relazioni tra gli operatori si ottiene la vera e propria estrazione del subset come mostrato nell'Esercizio 1.8.

Esercizio 1.8

Estrarre dalla table `Firm` tutte le donne.

```
1 subset1 = Firm(Firm.Gender == "F", :);
```

Estrarre dalla table `Firm` tutte le donne con titolo di studio "B".

```
1 subset2 = Firm(Firm.Gender == "F" & Firm.Education == "B", :);
```

Estrarre dalla table `Firm` tutte le donne con titolo di studio "B" oppure gli uomini con reddito maggiore di 4000 euro.

```
1 subset3 = Firm((Firm.Gender == "F" & Firm.Education == "B") | ...
      (Firm.Gender == "M" & Firm.Wage > 4000), :);
```

Estrarre dalla table `Firm` coloro che hanno un reddito maggiore uguale di 3000 euro e minore di 3500 euro.

```
1 subset4 = Firm(Firm.Wage >= 3000 & Firm.Wage < 3500, :);
```

Il `subset1` è caratterizzato da 46 osservazioni, quindi da tutte le donne presenti nella table di origine. Il `subset2`, al quale è stato applicato l'operatore logico "AND", mostra chiaramente una riduzione delle sue dimensioni (15 osservazioni) poiché le condizioni che devono essere soddisfatte contem-

document location specificare Installed Locally e/o fare click sul pulsante di installazione locale della documentazione. L'help delle funzioni di FSDA può essere consultato sia all'indirizzo web:

http://rosa.unipr.it/FSDA.

oppure installandolo dentro MATLAB (dopo aver impostato l'opzione Installed Locally vista poc'anzi) come segue. Nella schermata iniziale di help nella sezione Supplemental Software del riquadro denominato CONTENTS appare la voce FSDA toolbox. Una volta che si fa click su FSDA toolbox appare in automatico la finestra denominata "Copy FSDA HTML help files" che consente di copiare i file di help di FSDA nella cartella dove si trovano tutte le documentazioni di tutti gli altri toolboxes di MATLAB che sono stati scaricati.

1.8.5 Il comando `plot`

Per rappresentare graficamente una funzione si utilizza il comando `plot`; la sintassi del comando è: `plot(x, y, 'opzioni')` dove x e v sono i vettori di dati (rispettivamente ascisse e ordinate dei punti) e opzioni è un char di tre elementi che definisce il tipo di colore, di simbolo e di linea che si vogliono usare nel grafico.

All'indirizzo web https://it.mathworks.com/help/matlab/ref/plot.html è possibile visualizzare i diversi colori, gli stili di lineae ed i marcatori messi a disposizione da MATLAB. Ad esempio, se `x = 0:0.05:2*pi; y = 3*sin(y); plot(x, y, 'r:+')` traccia in rosso una linea a puntini e pone un marcatore + a ciascun dato. Se specifica un tipo di marcatore, ma non uno stile di linea, MATLAB disegna solamente il marcatore.

Esercizi di riepilogo

HW 1.1: Sequenza formata da stringhe

Creare un array di stringhe denominato `a` con i seguenti elementi "A6", "A9", "A12", ..., "A24".

HW 1.2: Creazione di un cell array

Creare un cell array denominato `CA`, di dimensione 2×4. L'elemento 1,1, deve contenere la sequenza 11, 12, 13, ..., 20. L'elemento 1,2 deve contenere il numero 5. L'elemento 1,3 il numero 10 e l'elemento 1,4 la sequenza 12,10,...,2. L'elemento 2,1 deve contenere il testo "Vado al

mare". L'elemento 2,2, deve contenere una matrice di dimensioni 5×6 con numeri estratti dalla distribuzione uniforme nell'intervallo [0, 1]. L'elemento 2,3 deve contenere un vettore colonna composto dai numeri 11, 14, 17, ..., 32. L'elemento 2,4 deve contenere una matrice identità di ordine 3.

HW 1.3: Creazione di una table

Creare la table riportata di seguito. Denominare questa table `Mt`.

```
1                          POP       SUP       DENS     ALT
2                          _____     ______    ____     ___
3       ANDRIA             99307     402.89     246     151
4       AREZZO             99258      384.7     258     296
5       UDINE              99051      57.17    1733     113
```

HW 1.4: Aggiungere i `RowNames` ad una table e filtrare i dati numerici

Caricare in memoria la table denominata `TableSenzaNomiRighe` tramite l'istruzione `load TableSenzaNomiRighe`. Questa table contiene 3 indicatori di criminalità dei 50 stati Americani (*Murder*, *Assault*, *Rape*), Murder = Assassini, Assault = Aggressioni, Rape = Stupri ed un indicatore legato alla densità della popolazione urbana (*UrbanPop*). I nomi dei 50 Stati Americani sono contenuti nel file `NomiStati.xlsx`.

1. Rinominare la table `TableSenzaNomiRighe` `USA`.

2. Assegnare alla table `USA` i nomi delle righe (`RowNames`) contenuti nel file `NomiStati.xlsx`.

3. Mostrare nella *Command Window* le righe relative al "Texas" e al "North Dakota". Commentare i due profili riga.

4. Mostrare nella *Command Window* tutti gli stati che hanno una popolazione urbana superiore a 70 e l'indicatore di assassini superiore a 12.

5. Mostrare nella *Command Window* tutti gli stati che hanno una popolazione urbana superiore a 70 oppure l'indicatore di assassini superiore a 12.

HW 1.5: Esercizi di base con le stringhe

Caricare in memoria la table che contiene una serie di indicatori riferiti alle città italiane, denominata citiesItaly tramite l'istruzione `load ... citiesItaly`. Estrarre i nomi delle province in un vettore denominato prov. Mostrare nella *Command Window* le province il cui nome:

1. inizia con "Pa";
2. finisce con "mo";
3. inizia con la "P" e termina con il carattere "o";
4. inizia con la "P" e termina o con il carattere "o", oppure con il carattere "a";
5. inizia per "P", termina per "o" oppure per "a" ed hanno al massimo un numero di caratteri pari a 5.

HW 1.6: Calcolo dei termini di una successione tramite ciclo for

Sia $\{a_n\}$ la successione definita da $a_1 = \log 10$ e $a_n = log(1 + a_{n-1})$ per $n \geq 2$. Calcolare e disegnare (tramite la funzione `plot`) i primi dieci termini della successione.

HW 1.7: Aggiunta di una variabile ad una table, salvataggio di un file .mat

Caricare in memoria il file `citiesItaly.mat`. Aggiungere alla table `citiesItaly` di dimensione 103×7, una nuova variabile denominata `nuovaVar` che contiene la successione 205, 203, ..., 1. Salvare la nuova table in un nuovo file .mat denominato `mioDataset.mat`. *Osservazione*: il file `mioDataset.mat` deve contenere solo la table `citiesItaly` di dimensione 103×8.

Data Science con MATLAB (seconda edizione)
Materiale aggiuntivo

Marco Riani[1], Aldo Corbellini[2], Fabrizio Laurini[3],
Gianluca Morelli[4],
Dipartimento di Scienze Economiche e Aziendale
and Interdepartmental Research Centre for Robust Statistics,
Università di Parma, 43123 Parma, Italy
Tommaso Proietti[5],
Università degli Studi di Roma "Tor Vergata"
Edoardo Fibbi[6], Domenico Perrotta[7], Francesca Torti[8],
Commissione Europea, Centro Comune di Ricerca
Ispra (VA), Italy

8 settembre 2023

[1]e-mail: marco.riani@unipr.it
[2]e-mail: aldo.corbellini@unipr.it
[3]e-mail: fabrizio.laurini@unipr.it
[4]e-mail: gianluca.morelli@unipr.it
[5]e-mail: tommaso.proietti@uniroma2.it
[6]e-mail: edoardo.fibbi@ec.europa.eu
[7]e-mail: domenico.perrotta@ec.europa.eu
[8]e-mail: francesca.torti@ec.europa.eu

Indice degli argomenti

1 Materiale extra: Introduzione all'utilizzo di MATLAB e alla gestione dei dati **9**

1.0.1 Intersezione di due table con le stesse variabili . . . 9

1.0.2 Unione e intersezione di due table 10

Bibliografia **15**

1

Materiale extra: Introduzione all'utilizzo di MATLAB e alla gestione dei dati

1.0.1 Intersezione di due table con le stesse variabili

La creazione di subset all'interno di un dataset, come visto, è spesso un passo fondamentale per l'analisi dei dati poiché consente di isolare solamente le unità statistiche di interesse. Non infrequente nella preparazione dei dati per l'analisi è il caso in cui si hanno le informazioni sparse su più dataset. Mostriamo, quindi, come unire o intersecare due dataset caratterizzati dalle stesse variabili. Per questi esempi utilizzeremo due tabelle contenute nella `struct` "Tinter.mat". Le due `table` che chiameremo `ds1` e `ds2` sono caratterizzate da alcune osservazioni comuni. Vediamo come unire le due `table` al fine di ottenerne una sola senza osservazioni ripetute. Innanzitutto il file "Tinter.mat" deve essere caricato nel *Workspace* con il comando `load` e poi devono essere definiti `ds1` e `ds2`.

```
1 load("Tinter");
2 ds1 = Tinter.ds1;
3 ds2 = Tinter.ds2;
```

L'operazione di unione in questo caso è un semplice accodamento delle informazioni delle due `table` dopo aver eliminato gli eventuali duplicati attraverso il comando `unique`.

```
1 dsu = [ds1; ds2];
2 dsu = unique(dsu);
```

L'operazione di intersezione, a differenza di quella di unione, mira a creare una nuova `table` con le sole osservazioni comuni ad entrambi i dataset di partenza. In questo caso per ottenere il risultato sarà utilizzato il comando `intersect`.

```
1 dsi = intersect(ds1, ds2);
```

Gli output delle operazioni appena eseguite sono mostrati in Figura 1.1.

dsu =

7×4 table

Code	Gender	Education	Wage
{'P0211'}	{'F'}	{'A'}	1877
{'P0212'}	{'M'}	{'B'}	2375
{'P0213'}	{'M'}	{'C'}	2450
{'P0214'}	{'F'}	{'B'}	2037
{'P0215'}	{'F'}	{'C'}	3893
{'P0217'}	{'F'}	{'A'}	1893
{'P0219'}	{'M'}	{'B'}	2016

dsi =

3×4 table

Code	Gender	Education	Wage
{'P0211'}	{'F'}	{'A'}	1877
{'P0213'}	{'M'}	{'C'}	2450
{'P0215'}	{'F'}	{'C'}	3893

Figura 1.1: Nel pannello di sinistra è mostrato il risultato dell'operazione di unione senza duplicati, mentre nel pannello di destra quello di intersezione.

1.0.2 Unione e intersezione di due table

Per completezza introduciamo un altro caso legato alla gestione dei dataset non raro nel contesto dell'analisi dati. Partiamo dal caso in cui i dati da analizzare sono archiviati su due tabelle e che le tabelle non contengano le medesime variabili. Premesso che le due tabelle devono avere una variabile in comune, ad esempio la variabile "Code" vista nella tabella `Firm`, tale variabile deve avere modalità distinte per osservazioni distinte, in sostanza stiamo parlando dei cosiddetti identificatori univoci. In altre parole la stessa osservazione presente nelle due tabelle deve essere associata allo stesso identificatore univoco. Alcuni esempi di identificatori univoci nella vita reale sono il codice fiscale, il numero di un documento di identità, la matricola universitaria, ecc.... Al fine di specificare meglio il problema si veda la Figura 1.2 la quale mostra due tabelle accomunate dalla variabile "Code" (identificatore univoco) e una serie di variabili da associare.

Tleft

9×4 table

	1 Code	2 Gender	3 Education	4 Wage
1	"P0211"	F	A	1877
2	"P0212"	M	B	2375
3	"P0213"	M	C	2450
4	"P0214"	F	B	2037
5	"P0215"	F	C	3893
6	"P0216"	M	A	2314
7	"P0217"	F	A	1893
8	"P0218"	M	A	2443
9	"P0219"	M	B	2016

Tright

9×4 table

	1 Code	2 Commuting_time	3 Smart_work_hours	4 Seniority
1	"P0211"	27	10	11
2	"P0212"	35	12	8
3	"P0213"	39	16	11
4	"P0214"	16	6	7
5	"P0215"	23	8	30
6	"P0216"	29	10	23
7	"P0217"	29	6	12
8	"P0218"	16	6	28
9	"P0219"	19	4	2

Figura 1.2: La tabella di sinistra mostra una parte dell'informazione associata ai vari soggetti e quella di destra le informazioni mancanti alla prima.

Le tabelle di Figura 1.2 sono contenute nella `struct` "Tjoin.mat". Per caricare la `struct` .mat e le tabelle in essa contenute è sufficiente scrivere i seguenti comandi.

```
load("Tjoin");
Tleft = Tjoin.Tleft;
Tright = Tjoin.Tright;
```

Assumendo che gli identificatori univoci siano i medesimi nelle due tabelle e ordinati allo stesso modo, il metodo più rapido per raggiungere lo scopo può essere mutuato da un metodo già mostrato in precedenza.

```
T = [Tleft, Tright(:,2:end)];
```

Il risultato ottenuto è mostrato in Figura 1.3

T

9x7 table

	1 Code	2 Gender	3 Education	4 Wage	5 Commuting_time	6 Smart_work_hours	7 Seniority
1	"P0211"	F	A	1877	27	10	11
2	"P0212"	M	B	2375	35	12	8
3	"P0213"	M	C	2450	39	16	11
4	"P0214"	F	B	2037	16	6	7
5	"P0215"	F	C	3893	23	8	30
6	"P0216"	M	A	2314	29	10	23
7	"P0217"	F	A	1893	29	6	12
8	"P0218"	M	A	2443	16	6	28
9	"P0219"	M	B	2016	19	4	2

Figura 1.3: Risultato del concatenamento di due tabelle in base alle assunzioni enunciate.

Il caso appena mostrato è piuttosto raro, per questo è necessario imparare a gestire concatenamenti di tabelle in cui il numero delle osservazioni non necessariamente è il medesimo in entrambe. Le tipologie di concatenamento tra tabelle sono molte e seguono la logica delle operazioni "Join" mutuate dai linguaggi di interrogazione delle basi di dati (ad esempio SQL). Al fine di non appesantire il lettore con un numero troppo elevato di file di corredo, procederemo alla modifica delle tabelle già esistenti per mostrare i modi con cui le tabelle possono essere unite. Le funzioni che saranno utilizzate per gestire le diverse fattispecie di unione o intersezione delle tabelle sono chiamate `innerjoin` e `outerjoin`.

Come prima cosa, attraverso il seguente codice andremo a rimuovere alcune righe dalla tabella `Tleft` trasformando le sue osservazioni in un di cui di quelle di `Tright`.

```
1 Tleft(3:7,:)=[];
```

La nuova dimensione di `Tleft` è 4×4 e per associare gli identificatori univoci rimasti alle righe di `Tright` coincidenti utilizzeremo i seguenti comandi ottenendo il risultato mostrato in Figura 1.4.

```
1 ij = innerjoin(Tleft, Tright,'LeftKeys',1,'RightKeys',1);
```

N.B.: LeftKeys e RightKeys posti entrambi uguali a uno indicano che sia nella `Tleft` che nella `Tright` l'identificatore univoco è la variabile contenuta nella prima colonna.

ij

4x7 table

	1 Code	2 Gender	3 Education	4 Wage	5 Commuting_time	6 Smart_work_hours	7 Seniority
1	"P0211"	F	A	1877	27	10	11
2	"P0212"	M	B	2375	35	12	8
3	"P0218"	M	A	2443	16	6	28
4	"P0219"	M	B	2016	19	4	2

Figura 1.4: Associazione del contenuto di `Tright` a `Tleft` con `innerjoin`.

Sempre facendo riferimento a `Tleft` di dimensione 4×4, mostriamo adesso un approccio più conservativo del dato. Come visto, il comando `innerjoin` associa alla prima tabella solo ciò che è comune nella seconda, ma esistono casi in cui può essere necessario che a `Tleft` venga associata tutta l'informazione contenuta in `Tright`. In queste occasioni il comando da utilizzare è `outerjoin`.

```
1 oj = outerjoin(Tleft, Tright,'LeftKeys',1,'RightKeys',1);
```

Come mostrato in Figura 1.5, in questo caso la tabella finale conserva il numero di righe della tabella con più osservazioni. Laddove l'identificativo univoco di `Tright` non trova corrispondenza in `Tleft` le righe relative alle variabili originariamente contenute in `Tleft` vengono riempite con dei valori mancanti.

oj

9x8 table

	1 Code_Tleft	2 Gender	3 Education	4 Wage	5 Code_Tright	6 Commuting_time	7 Smart_work_hours	8 Seniority
1	"P0211"	F	A	1877	"P0211"	27	10	11
2	"P0212"	M	B	2375	"P0212"	35	12	8
3	*<missing>*	<undefin...	<undefined>	NaN	"P0213"	39	16	11
4	*<missing>*	<undefin...	<undefined>	NaN	"P0214"	16	6	7
5	*<missing>*	<undefin...	<undefined>	NaN	"P0215"	23	8	30
6	*<missing>*	<undefin...	<undefined>	NaN	"P0216"	29	10	23
7	*<missing>*	<undefin...	<undefined>	NaN	"P0217"	29	6	12
8	"P0218"	M	A	2443	"P0218"	16	6	28
9	"P0219"	M	B	2016	"P0219"	19	4	2

Figura 1.5: Associazione del contenuto di Tright a Tleft con `outerjoin`.

Infine, deve essere affrontato l'ultimo caso rappresentato da una particolarità dei due precedenti e deducibile da essi. In questo caso gli identificatori univoci di una tabella non sono più un di cui di quelli dell'altra, ma le tabelle hanno solo alcuni identificatori comuni. Attraverso il seguente codice sarà possibile ricaricare le tabelle di partenza e cancellare alcune righe da entrambe.

```
load("Tjoin");
Tleft = Tjoin.Tleft;
Tright = Tjoin.Tright;
Tleft(3:7,:)=[];
Tright([1,8,],:)=[];
```

Un primo approccio utilizzabile per combinare le informazioni contenute nelle due tabelle è quello di ottenere come risultato una tabella contenente solamente gli identificatori univoci comuni ad entrambe le tabelle. Per fare ciò è sufficiente eseguire nuovamente il comando `innerjoin` esattamente come visto in precedenza ottenendo il risultato mostrato in Figura 1.6.

```
iij = innerjoin(Tleft, Tright,'LeftKeys',1,'RightKeys',1);
```

iij

2x7 table

	1 Code	2 Gender	3 Education	4 Wage	5 Commuting_time	6 Smart_work_hours	7 Seniority
1	"P0212"	M	B	2375	35	12	8
2	"P0219"	M	B	2016	19	4	2

Figura 1.6: Associazione del contenuto di `Tright` a `Tleft` con `innerjoin`.

Nel caso in cui si volesse, comunque, conservare tutta l'informazione delle due tabelle originarie, il comando da utilizzare sarebbe `outerjoin` ottenendo così il risultato mostrato in Figura 1.7.

```
1 uoj = outerjoin(Tleft, Tright,'LeftKeys',1,'RightKeys',1);
```

uoj

9x8 table

	1 Code_Tleft	2 Gender	3 Education	4 Wage	5 Code_Tright	6 Commuting_time	7 Smart_work_hours	8 Seniority
1	"P0211"	F	A	1877	*<missing>*	NaN	NaN	NaN
2	"P0212"	M	B	2375	"P0212"	35	12	8
3	*<missing>*	<undefin...	<undefined>	NaN	"P0213"	39	16	11
4	*<missing>*	<undefin...	<undefined>	NaN	"P0214"	16	6	7
5	*<missing>*	<undefin...	<undefined>	NaN	"P0215"	23	8	30
6	*<missing>*	<undefin...	<undefined>	NaN	"P0216"	29	10	23
7	*<missing>*	<undefin...	<undefined>	NaN	"P0217"	29	6	12
8	"P0218"	M	A	2443	*<missing>*	NaN	NaN	NaN
9	"P0219"	M	B	2016	"P0219"	19	4	2

Figura 1.7: Associazione del contenuto di `Tright` a `Tleft` con `outerjoin`.