Elaborazione delle Immagini

Sara Angeretti
@Sara1798

Fabio Ferrario
@fefabo

2023/2024

Indice

Capitolo 1

Introduzione a MatLab

1.1 Appunti video lezione 2020 da Moodle

1.1.1 Ambienti di lavoro

Command Window e Workspace

- Command Window: finestra di comando, dove andiamo effettivamente a scrivere i comandi.
- Workspace: variabili in memoria, vediamo le variabili che abbiamo istanziato, mi dà un feedback immediato di quello che ho fatto.

Variabili

```
>> pippo = 10
pippo =
    10
>>
```

Questo vuol dire che istanziare una variabile assegnandole anche un valore numerico, dopo che ho premuto invio mi viene creata in memoria la variabile che ho creato (lo vedo nel workspace che l'ho creata). Però scrivendo così mi viene anche stampato a schermo il comando che ha eseguito.
Nel workspace vedo che ho una variabile chiamata pippo che ha valore 10, size 1x1 e class double.
Questo vuol dire che ha dimensione 1x1, cioè che è una singola cella (MatLab lavora con dati che sono matrici). Inoltre, double è il valore base, di default, quasi tutto viene fatto in virgola mobile (ci possono essere forzature se necessario).

Ci sono operazioni più complesse però di semplici assegnamenti, per cui potrebbe tornare utile evitare di stampare a schermo, perché non è assolutamente necessario vedere live l'output del comando che viene eseguito. Perciò:

```
>> pluto = 10;
>>
```

1.1.2 Lista di comandi utili

¿¿ clear: pulisce il workspace, cancella tutte le variabili che ho istanziato.

¿¿ clc: pulisce il command window, cancella tutti i commenti che ho scritto.

freccia su tastiera: va a riprendere l'*history* dei miei comandi nella command window, che apre in una finestrella pop-up.

1.1.3 Operazioni matematiche

Addizione

```
>> a = 2;
>> b = 1;
>> c = a + b;
```

Sottrazione

```
>> a = 2;
>> b = 1;
>> c = a - b;

ans =

    1
```

Prodotto

```
>> a = 2;
>> b = 1;
>> c = a * b;

ans =

    2
```

Divisione

```
>> a = 2;
>> b = 1;
>> c = a / b;

ans =

    2
```

Elevamento a potenza

```
>> a = 2;
>> a^3;

ans =

    8
```

MatLab richiede che il valore di un'operazione venga associato ad una variabile. Se non la creo io, lo fa MatLab per me (qua la variabile "`ans`"). Visibile nel *Workspace*.
Questa variabile può essere riutilizzata.

```
>> a + ans;

ans =

    10
```

Però ocio che ad ogni operazione verrà sovrascritta.

Altre operazioni

Si può operare in tanti modi con gli scalari su MatLab; per vedere le operazioni possibili, si deve consultare la documentazione.

1.1.4 La documentazione

1. Tasto "*Help*" in "Resources"
2. In Command Window possiamo chiederlo direttamente a MatLab:

- ▷ comando "`help`", mi dà dei suggerimenti sull'ultima operazione eseguita.
- ▷ comando "`help max`", (dove "`max`" è un esempio di operazione che dà in output il massimo numero in un array di numeri dati in input) mi dice quale è la sintassi con tutte le sue alternative dell'operazione richiesta.
- ▷ comando "`lookfor max`", (dove "`max`" è un esempio di parte del nome di un'operazione che non ci ricordiamo nella sua interezza) mi dà una lista di possibili operazioni che potrebbero essere quella che stiamo cercando e che contengono il pezzo di nome che gli ho assegnato.

1.1.5 Gli array

Sintassi varie

I vettori sono rappresentati come concatenazione di valori, secondo la sintassi:

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

v =

    1   2     3     4     5     6

>>
```

Nel workspace vedremo:

Name	Value	Size	Class
`v`	`[1,2,3,4,5,6]`	`1x6`	`double`

In questo caso è un **vettore riga**.
Le *parentesi quadre* si usano per **concatenare** i valori, le *virgole* per separare i valori in un *vettore riga*, i *punti e virgola* per separare i valori in un *vettore colonna*.
Se invece volessimo un **vettore colonna**:

```
>> w = [10;20;30;40];
>> w

w =
```

```
    10
    20
    30
    40

>>
```

Nel workspace vedremo

Name	Value	Size	Class
`w`	`[10;20;30;40]`	`4x1`	`double`

N.B.: sono ***indicizzabili***. Per indicizzare si usano le *parentesi tonde* e, cosa importantissima, *in MatLab* ***non esiste l'indice 0****, perciò si parte da 1 e non da 0.* Se si scrive lo 0 come indice, dà errore.
Se volessimo ad esempio il terzo elemento del vettore `v`:

```
>> v(3)

ans =

    3

>>
```

Questa indicizzazione la possiamo usare anche in assegnazione:

```
>> v(3) = 100;
>> v

v =

    1   2     100     4     5     6

>>
```

Posso lavorare anche su *serie di cellette* invece di una celletta singola:

```
>> v = (1:3);

ans =
```

```
    1   2     100

>> v = (1:3);

ans =

    4     5     6

>>
```

Se volessi, senza sapere quanto è lungo un vettore (c'è un modo per saperlo, ma non lo vediamo ora), prendere tutti gli elementi da un certo indice fino alla fine, posso fare:

```
>> v(4:end)

ans =

    4     5     6

>>
```

Scoprire la lunghezza di un vettore:

```
>> size(v)

ans =

    1   6

>>
```

Ovvero una riga e 6 colonne. La convenzione (è sempre così) è prima righe e poi colonne, perciò se volessi conoscere solo le righe:

```
>> size(v,1)

ans =

    1

>>
```

se volessi conoscere solo le colonne:

```
>> size(v,2)

ans =

    6

>>
```

Operazioni sugli array

Somma: come sappiamo da GAL, se sommo due vettori devono avere la stessa dimensione, altrimenti non posso sommarli.

Prodotto: come sappiamo da GAL, se moltiplico due vettori devono avere gli stessi indici interni, altrimenti non posso moltiplicarli. Es.: noi abbiamo `v 1x6` e `w 4x1`, non posso moltiplicarli perché $6 \neq 4$.
Una cosa che potrei fare per ovviare il problema sarebbe selezionare un sotto vettore da `v` che abbia la stessa dimensione di `w`:

```
>> v(1:4)*w

ans =

    3210

>>
```

Per chi non ha ancora visto GAL, questo è un prodotto scalare, ma perché `ans` vale `3210`? Praticamente ho due matrici $m \times p$ e $p \times n$, la dimensione della matrice prodotto sarà $m \times n$. Quindi qua ho un vettore riga 1×4 e un vettore colonna 4×1, il prodotto sarà un vettore riga 1×1, che è quello che vediamo in `ans`. Inoltre il prodotto fra:

una matrice			e una matrice	
a_1	a_2	a_3	b_1	b_2
a_4	a_5	a_6	b_3	b_4
			b_5	b_6

mi dà una matrice 2×2 con i valori:

$a_1 * b_1 + a_2 * b_3 + a_3 * b_5$	$a_1 * b_2 + a_2 * b_4 + a_3 * b_6$
$a_4 * b_1 + a_5 * b_3 + a_6 * b_5$	$a_4 * b_2 + a_5 * b_4 + a_6 * b_6$

È quello che succede se:

```
>> z = v(1:4)

z =

    1   2     100     4

>> w*z

ans =

    10   20     1000     40
    20   40     2000     80
    30   60     3000     120
    40   80     4000     160

>>
```

Valore massimo: `max(v)` mi ritorna il valore massimo fra i numeri del vettore. Per visualizzare le varianti, basta scrivere `max` e aprire la parentesi tonda