Messaggi d'errore in Matlab®

Quando si scrive del codice sorgente in un qualunque linguaggio di programmazione, gli errori sono sempre dietro l'angolo. Questo vale anche in Matlab®, che però cerca di aiutare il debugging fornendo dei messaggi di errore che aiutino il più possibile a trovare il bug all'interno del proprio codice. È quindi estremamente utile imparare a leggere questi messaggi, in modo da capire su quale parte del codice concentrarsi nella ricerca dell'errore. In questo documento elencheremo i messaggi d'errore più comuni, mostrando le probabili cause che possono generare tali messaggi.

1 Errori di sintassi

Gli errori di sintassi sono gli errori che Matlab® trova nel codice ancor prima di eseguirlo. Questi errori infatti non consentono l'esecuzione del sorgente in quanto Matlab® non riesce a capire come interpretare quello che è stato scritto. Gli esempi più comuni sono:

- `Error: Expression or statement is incorrect-possibly unbalanced (, {, or [.`
  Questo errore significa che non tutte le parentesi che sono state aperte hanno la corrispondente parentesi chiusa. La causa può essere sia una parentesi chiusa mancante che una parentesi aperta di troppo. Ad esempio:

  ```
  >> v (1
   v (1
       |
  Error: Expression or statement is incorrect--possibly unbalanced (, {, or [.
  ```

  Si noti che Matlab® tende a suggerire la posizione in cui ritiene che la parentesi vada inserita (utilizzando il simbolo `|` nelle riga sopra al messaggio di errore).

- `Error: Unbalanced or misused parentheses or brackets.`
  Simile al caso precedente, questo errore di solito significa che c'è una parentesi chiusa in più:

  ```
  >> v (1))
   v (1))
        |
  Error: Unbalanced or misused parentheses or brackets.
  ```

  Anche in questo caso, Matlab® prova a dare un suggerimento su quale possa essere la parentesi incriminata.

- `Error: Expression or statement is incomplete or incorrect.`
  Questo errore signfica che l'istruzione data è in qualche modo incompleta. Ad esempio:

  ```
  >> a = 1 + 3 +
   a = 1 + 3 +
               |
  Error: Expression or statement is incomplete or incorrect.
  ```

- `Error: Unexpected MATLAB operator.`
  In questo caso invece l'istruzione data contiene un operatore che Matlab® non riconosce. Per operatore si intende un qualunque simbolo che agisce su uno o più argomenti per generare un risultato. Gli esempi più semplici di operatori sono gli operatori aritmetici (`+`, `-`, `*` e `/`), ma anche il simbolo `:` nell'istruzione `x = 1:10` è un operatore. Se viene usato un operatore che Matlab® non conosce, viene stampato a video l'errore sopra, poiché Matlab® non sa come generare il risultato a partire dagli operandi. Ad esempio:

  ```
  >> x = 1 +/ 2
   x = 1 +/ 2
          |
  Error: Unexpected MATLAB operator.
  ```

  Matlab® non conosce l'operatore `+/` e non sa quindi come generare il valore da salvare in `x` a partire dai valori `1` e `2`.

È importante notare che se tali comandi sono inseriti in uno script Matlab®, il messaggio d'errore conterrà anche **la riga e la colonna** che indicano dove all'interno del file .m Matlab® ha trovato l'errore riportato. Se ad esempio salviamo i comandi:

```
1 a = 1;
2 b = 2;
3
4 c = a +/ b
```

nel file `prova.m` e cerchiamo di eseguirlo, otteniamo:

```
>> prova
Error: File: prova.m Line: 4 Column: 8
Unexpected MATLAB operator.
```

L'errore è effettivamente alla colonna 8 della quarta riga. Si noti inoltre che lo script **non viene eseguito**. Dopo la chiamata allo script, infatti, le variabili `a` e `b` non sono state definite: si possono ad esempio a dare i comandi

```
clear all
prova
whos
```

per vedere immediatamente che non c'è traccia di tali variabili.

2 Errori di esecuzione

A differenza degli errori di sintassi, gli errori di esecuzione vengono segnalati solo quando i comandi vengono effettivamente eseguiti. Si tratta infatti di errori generati da comandi formalmente corretti che contengono però errori logici che non consentono la corretta esecuzione del comando. Vediamone alcuni:

- `Error using ***`
  `Matrix dimensions must agree.`

  dove `***` indica una qualunque operazione fra `+`, `-`, `\`, `.*` e `./`. Questo errore indica che si è provato a fare un'operazione su operandi con dimensioni non compatibili rispetto a quell'operazione. Ad esempio in:

```
>> A = rand (3, 4);
>> B = rand (4, 3);
>> A + B
Error using  +
Matrix dimensions must agree.
```

la somma tra `A` e `B` non può essere calcolata, poiché la prima ha dimensione `3x4`, mentre la seconda ha dimensione `4x3`.
Nel caso della moltiplicazione, l'errore è leggermente diverso:

```
>> C = rand (3, 3);
>> D = rand (4, 3);
>> C * D
Error using  *
Inner matrix dimensions must agree.
```

L'errore in questo caso parla di `Inner matrix dimensions`, in quanto per la moltiplicazione matriciale sono solo il numero di colonne della prima matrice e il numero di righe della seconda a definire se l'operazione è effettuabile o meno.

- `Index exceeds matrix dimensions.`
  Questo errore indica che si è provato ad accedere a un elemento in una posizione non presente nella matrice (o vettore), in quanto l'indice utilizzato è maggiore delle dimensioni della matrice stessa. Ad esempio:

  ```
  >> v = [1 2 3];
  >> v (4)
  Index exceeds matrix dimensions.
  ```

  Un errore uguale negli effetti ma diverso nelle cause è:
  `Subscript indices must either be real positive integers or logicals.`
  In questo caso, l'indice utilizzato per accedere alla matrice non è un numero intero positivo:

  ```
  >> v (0)
  Subscript indices must either be real positive integers or logicals.

  >> v (1.2)
  Subscript indices must either be real positive integers or logicals.
  ```

- `Undefined function or variable '***'.`
  La variabile o la funzione usata non è mai stata definita. Nel caso delle variabili, controllate il vostro Workspace! Potreste aver cancellato per sbaglio la vostra variabile con i comandi `clear` o `clear all`.
  Se non avete definito, ad esempio, una variabile `c` all'interno di una function `esempio.m`:

  ```
  function [ b ] = esempio( a )
  b=a+c;
  end
  ```

  chiamandola dalla Command Window, otteniamo:

```
  >> esempio(4)
Undefined function or variable 'c'.
Error in esempio (line 3)
b=a+c;
```

Si noti che questo errore compare anche quando nella chiamata a funzione c'è un **errore di battitura** nel nome della funzione! Si ricordi anche Matlab distingue le lettere maiuscole dalle minuscole. Ad esempio:

```
>> A = rand (4, 4);
>> sums (A)
Undefined function or variable 'sums'.
>> Sum (A)
Undefined function or variable 'Sum'.
```

Il nome corretto della funzione è `sum`.

- `Not enough input arguments/Too many input arguments/Too many output arguments`
  Si è provato a chiamare la funzione con un numero di argomenti in ingresso o in uscita non appropriato. Si noti che questo stesso errore viene stampato quando si prova a usare una function come uno script:

  ```
  >> [V, D] = eig
  Error using eig
  Not enough input arguments.
  ```

  L'errore inverso (provare a eseguire uno script come una function, ossia chiamandolo dalla Command Window passando argomenti in ingresso) viene invece segnalato dal seguente errore:

  ```
  >> myscript (A)
  Attempt to execute SCRIPT myscript as a function
  ```

Come nel caso degli errori di sintassi, anche per gli errori di esecuzione viene segnalata la riga a cui l'errore è avvenuto nel caso in cui i comandi siano inseriti in uno script. Per esempio, salvando i comandi:

```
1 clc
2 clear
3
4 A = rand (3, 4);
5 B = rand (4, 3);
6
7 A - B
```

nel file `esempio_errori.m` e chiamandolo dalla Command Window, otteniamo:

```
>> esempio_errori.m
Error using  -
Matrix dimensions must agree.
```

```
Error in esempio_errori (line 7)
A - B
```

Si notino però due differenze fondamentali rispetto al caso trattato in precedenza:

1. il messaggio di errore riporta la riga a cui tale errore è avvenuto ma non la colonna. Questo è ovvio, se pensiamo al tipo di errore: Matlab® sa solo che quell'istruzione ha dato un errore, ma non può dire quale parte dell'istruzione stessa ha generato tale errore, in quanto la scrittura è formalmente corretta

2. lo script **viene eseguito fino alla riga precedente a quella che dà errore!** Il comando `whos` dato dopo la chiamata allo script infatti stampa:

   ```
   >> whos
     Name      Size            Bytes  Class     Attributes

     A         3x4                96  double
     B         4x3                96  double
   ```

   Le due variabili sono dunque state create e sono rimaste in memoria.