Università degli Studi di Ferrara

Corso di Laurea in Matematica - A.A. 2018 - 2019

Programmazione
Lezione 17 – Codice e funzioni

Docente: Michele Ferrari - michele.ferrari@unife.it

Nelle lezioni precedenti

- Grafici Sovrapposti
- Commentare un grafico
- Sottografici
- Grafici di superfici

In questa lezione

- Script
- Funzioni
- Debug
- Esercizi

M-file

Le successioni di comandi viste nelle precedenti lezioni possono essere memorizzate direttamente in un file di testo.

In questo modo si crea uno script o m-file MATLAB (in quanto è obbligatorio assegnare al file l'estensione ".m").

Scrivendo il nome del file nella finestra principale di MATLAB si ottiene lo stesso risultato che si sarebbe ottenuto scrivendo uno ad uno i comandi elencati nello script.

- In sostanza uno script è un vero e proprio programma che ci consente di memorizzare e organizzare istruzioni MATLAB al fine di realizzare progetti di una certa complessità.

Script: pulizia

È consigliabile iniziare uno script “azzerando” la memoria e dichiarando il formato prescelto (se non specificato, di default si ha format short)

```
clear all;
clc;
clf;
```

- **clear all** pulisce la memoria di lavoro
- **clc** ripulisce il quadro comandi da tutto ciò che vi è stato scritto in precedenza.
- **clf** ripulisce lo schermo grafico

Esempio: Disegnare figure

Supponiamo di volere disegnare nel piano una poligonale chiusa.

La funzione **plot** si presta bene a questo scopo in quanto raccorda tramite segmenti di retta in modo ordinato i vertici dei vettori argomento.

Quindi per disegnare una figura basterà memorizzarne le coordinate dei vertici in due vettori x e y e utilizzare l'istruzione plot(x,y).

Esempio: Disegnare figure

Per disegnare una poligonale chiusa ad n vertici il primo e l'ultimo vertice memorizzati devono coincidere, quindi dovremo memorizzare n + 1 vertici.

Ad esempio nel caso di un quadrato di lato unitario avremo bisogno di memorizzare 4 + 1 vertici:

```
>> x=[0 1 1 0 0];
>> y=[0 0 1 1 0];
>> plot(x,y)
>> axis('square')
```

L'istruzione **axis('square')** fa sì che la figura venga visualizzata utilizzando la stessa scala per entrambi gli assi.

Esempio 2: script per disegnare figure

Se ora vogliamo realizzare il progetto più complesso di disegnare una stella con un numero prefissato di punte possiamo memorizzare la successione dei comandi nello script **stella.m**

Esempio 2: script stella.n

```
% stella.m
% Disegna una stella a n punte
%
n=5;a=0.5;
t=linspace(0,2*pi,n+1);
s=linspace(pi/n,2*pi-pi/n,n);
x(1:2:2*n+1)=cos(t);
x(2:2:2*n)=a*cos(s);
y(1:2:2*n+1)=sin(t);
y(2:2:2*n)=a*sin(s);
plot(x,y);
axis('square');
```

Esempio 2: script stella.m

In generale il carattere % serve per introdurre un commento all'interno dello script, MATLAB ignora il contenuto alla destra del carattere % fino alla linea successiva.

Il commento all'inizio dello script file è particolarmente importante in MATLAB, infatti richiamando il comando help seguito dal nome dello script otteniamo come risposta il commento inserito all'inizio dello script stesso

>> help stella

stella.m

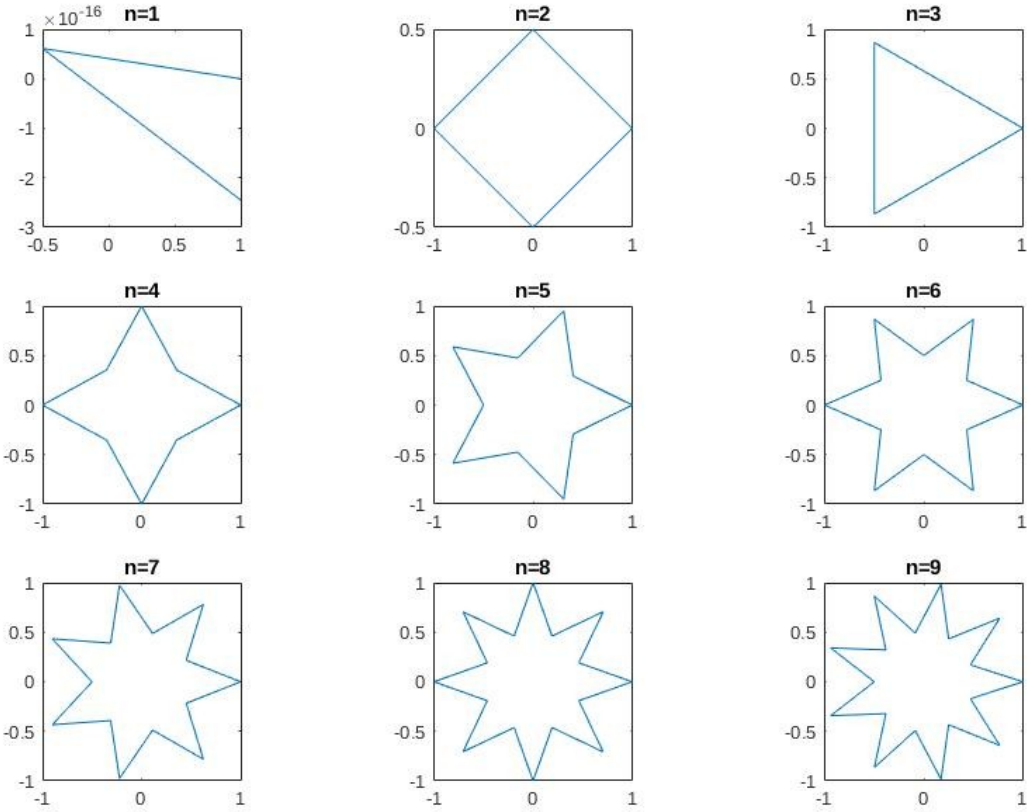
Disegna una stella a n punte

Possiamo ora scrivere nella finestra di comando di MATLAB

>> stella

per ottenere il risultato desiderato.

Esempio 2: script stella.m



Il comando input

Il comando input che consente all'utente di assegnare ad alcune variabili il valore che si desidera tramite tastiera

```
>> help input
 input  Prompt for user input.
    RESULT = input(PROMPT) displays the PROMPT string on the screen, waits
    for input from the keyboard, evaluates any expressions in the input,
    and returns the value in RESULT. To evaluate expressions, input accesses
    variables in the current workspace. If you press the return key without
    entering anything, input returns an empty matrix.
```

Il comando input: esempio

Supponiamo di realizzare uno script che calcoli l'area di un rettangolo:

```
% arearettangolo.m
% calcola l'area di un
% rettangolo di lati a e b
a = 2;
b = 5;
Area = a*b
```

Il comando input: esempio

In questo caso, se volessimo agire modificando i valori di base e altezza per il rettangolo, dovremmo ogni volta modificare lo script.

Per ovviare a questo potremmo quindi far inserire all'utente, da tastiera, i parametri dello script utilizzando il comando input.

Il comando input: esempio

Nell'esempio precedente ciò comporta la sostituzione dei comandi:

```
a=2;
b=5;
```

con

```
a=input('lato minore del rettangolo:');
b=input('lato maggiore del
rettangolo:');
```

Script vs Function

Una caratteristica degli script è quella di non avere parametri in ingresso modificabili.

Abbiamo infatti visto come, non utilizzando il comando input, se vogliamo modificare i valori di a e b dobbiamo modificare ogni volta lo script.

Un'altra particolarità consiste nel fatto che tutte le variabili usate nello script vengono automaticamente messe nella memoria di lavoro di MATLAB: si tratta quindi di variabili **globali** che contribuiscono ad aumentare l'occupazione di memoria.

E' necessario osservare come le variabili globali rimangano nella memoria del quadro comandi di MATLAB anche nel caso delle variabili "di servizio", di cui è superfluo conservare i valori.

Script vs Function

Una funzione MATLAB risponde esattamente alle precedenti due necessità, ossia è uno script al quale è possibile passare parametri in ingresso ed ottenerne in uscita, inoltre le variabili utilizzate durante l'esecuzione della funzione vengono automaticamente cancellate dalla memoria di MATLAB al termine della stessa.

Funzioni in MATLAB

A differenza di uno script standard la sintassi di una funzione richiede che la prima riga abbia la struttura:

```
function Parametri in uscita = Nomefunzione(Parametri in Ingresso)
```

generalmente seguita da alcune righe di commento, che possono ancora essere visualizzate tramite help Nomefunzione.

Altre regole importanti da ricordare nella costruzione di funzioni consistono nell'obbligo di salvare la funzione come script file Nomefunzione.m e che nel corpo della funzione devono essere assegnati dei valori ai Parametri in Uscita.

Ad esempio possiamo creare una funzione MATLAB che rende più versatile il precedente script stella.m

Funzione puntistella

```
function [xv,yv]=puntistella(ap,np)
% Costruisce due vettori xv e yv contenenti i vertici
% di una stella a np punte. Il parametro ap, 0 < ap < 1,
% consente di modificare la lunghezza delle punte
%
t=linspace(0,2*pi,np+1);
s=linspace(pi/np,2*pi-pi/np,np);
xv(1:2:2*np+1)=cos(t);
xv(2:2:2*np)=ap*cos(s);
yv(1:2:2*np+1)=sin(t);
yv(2:2:2*np)=ap*sin(s);
```

Lo script stella2.m

Chiaramente anche in matlab abbiamo necessità di sviluppare un “main” che chiami la nostra funzione, lo script stella2.m:

```
% stella2.m
% Disegna una stella a n punte
%
n=5;a=0.5;
[x,y]=puntistella(a,n);
plot(x,y);
axis('square');
```

Funzioni in MATLAB

Si noti come i parametri formali ap, np assumano i valori assegnati ad a e n al momento della chiamata della funzione, così come i parametri formali xv, yv forniscono i vettori risultato alle variabili x e y.

Notiamo che:

- Al fine di poter usare una funzione all'interno di uno script, la funzione dovrà trovarsi nella stessa directory dello script oppure in una directory tra quelle predefinite da MATLAB (si scriva help path per maggiori informazioni al riguardo)
- Simile meccanismo vale se si desidera eseguire uno script dalla finestra di comando di MATLAB
- Notiamo infine come sia possibile scrivere e memorizzare più funzioni direttamente all'interno dello stesso script file

Path

Per scrivere degli script possiamo digitare il comando edit ’nome file’: si aprirà la finestra che ci permette di scrivere e salvare i nostri lavori.

Quando scriviamo/salviamo/richiamiamo file è necessario ricordarsi che Matlab vede solamente i file che sono all’interno della cartella di lavoro o all’interno di cartelle predefinite.

E’ opportuno settare il percorso in maniera adeguata:

MATLAB supporta diversi comandi che sono proprio di un prompt o una shell testuale:

il comando per spostarsi attraverso le directory è cd.

Ad esempio:

>> cd .. → Torna indietro nell’albero delle directory

>> mkdir WORK → crea una nuova directory

>> cd WORK → si sposta all’interno della directory indicata

Debug

L’utilizzo di strategie di organizzazione del codice opportune, come l’utilizzo di function o una corretta e chiara indentazione, possono minimizzare l’insorgere di errori.

Nonostante questo inevitabilmente gli errori capitano.

Gli errori di programmazione sono chiamati bugs mentre il processo di ricerca ed eliminazione debugging.

Quando Matlab incontra un errore restituisce il nome della routine che l’ha causato insieme ad un messaggio.

Se la funzione che ha causato l’errore era chiamata da un altra funzione il nome della funzione chiamante e il numero della linea vengono inoltre forniti.

Viene quindi fornita tutta la gerarchia della chiamata finché non si raggiunge la linea di comando che ha generato l’errore.

Debug: i comandi type e dbtype

Il comando

>> type filename

ci restituisce l'intero file che vogliamo analizzare

Il comando

>> dbtype filename

ci restituisce solo alcune righe vicine alla riga incriminata

Debug: i comandi type e dbtype

Un altra forma del comando dbtype è la seguente:

>> dbtype filename start:stop

che restituisce le linee dal numero coincidente con start fino alla linea coincidente con stop.

Debug: il comando pause

Un altro comando che viene comunemente utilizzato nelle operazioni di debug è pause, che ci consente di sospendere temporaneamente l'esecuzione di un file .m

Questo comando può essere una buona strategia per visualizzare un messaggio senza interrompere la procedura.

Quando Matlab incontra il comando pause interrompe l'esecuzione e cambia la riga di comando con una P:

```
P>>
```

Debug: il comando pause

Esempio:

```
function y=nonnegativo(x)
% restituisce x se x>0 altrimenti visualizza un messaggio
% e si mette in pausa
if x<0
    disp('x \'e negativo')
    pause;
end
y=x;
```

Per continuare l'esecuzione è sufficiente premere invio, se si vuole terminare l'esecuzione si può utilizzare la combinazione di tasti ctrl-C: questa causa l'interruzione del programma in qualsiasi momento.

Debug: il comando keyboard

Il comando keyboard risulta molto più potente del comando pause: consente infatti di interrompere l'esecuzione di un programma e restituire il controllo all'utilizzatore.

L'aspetto più importante è che consente all'utente di accedere alle variabili interne di una function contenente tale comando.

Quando Matlab incontra il comando keyboard interrompe l'esecuzione e cambia la riga di comando con una K:

K>>

In questo modo possiamo visualizzare tutte le variabili e cercare di identificare l'errore.

Si osserva che possiamo ottenere lo stesso effetto attraverso il debugger interattivo senza introdurre il comando keyboard.

Esempio: comando keyboard

Esempio:

la funzione quadroot calcola le radici di un polinomio di secondo grado. Supponiamo che un bug da qualche parte causi un errore quando la funzione ci restituisce valori complessi.

```
function r=quadroot(a,b,c)
d=b^2-4*a*c;
if d<0
    fprintf('Warning in quadroot');
    keyboard;
end
q1=-0.5*(b+sign(b)*sqrt(b^2-4*a*c));
q2=-0.5*(b-sign(b)*sqrt(b^2-4*a*c));
r=[q1/a; q2/a];
```

Esercizio

Risolvere il problema dell'area del rettangolo implementando:

- La richiesta dei dati all'utente
- Una funzione che esegue il calcolo (rettangolo.m)
- Uno script che realizzi l'inserimento dei dati, chiami la funzione e visualizzi il risultato

Esercizio 2

Implementare un programma che sfruttando la funzione puntistella.m e subplot disegni nove stelle in una matrice di figure 3x3 per n=3, 4, 5, 6, 7, 8, 10, 20 , 40

Grazie per l’attenzione

Riferimenti

Il corso di programmazione per il primo anno della Laurea Triennale in Matematica nasce con l'intento di unire ai principi di programmazione una conoscenza basilare di uno degli strumenti software più diffusi nell'ambito matematico: Matlab.

Per la parte introduttiva di MATLAB:

L. Pareschi, G. Dimarco “Introduzione a MATLAB”, corso di Laboratorio di Calcolo Numerico 2006