

Lab. 2

Laboratorio di Calcolo Numerico

Annalisa Buffa Fausto Cavalli

Corso di Laurea in Chimica

A.A. 2009-2010

Script

Matlab offre la possibilità di raggruppare più di comandi, che vengono usati spesso, in un unico file, detto **script** file.

Alcune caratteristiche:

→ Gli script file condividono l'area di memoria con la command window, cioè tutte le variabili presenti nello workspace possono essere lette, utilizzate e modificate da un file script e tutte le variabili create in un file script sono disponibili nella command window alla fine dell'esecuzione dello script.

→ Per eseguire uno script file è sufficiente digitare nella command window il nome con cui è stato salvato (prestando attenzione che si trovi nella cartella in cui si sta lavorando). Molti dei comandi predefiniti in matlab sono anch'essi degli script file.

→ Gli script file possono essere salvati con qualunque nome accettato dal sistema operativo ma devono terminare con l'estensione **.m** per poter essere letti da matlab (per questo vengono anche chiamati **M-file**).

Script

Per creare uno script è sufficiente un qualunque editor di testo: matlab ne offre uno molto versatile già predisposto per la creazione di file di matlab. Si può aprire l'editor attraverso in vari modi: attraverso il menu **file-new-M-file** oppure digitando il comando **edit [nomefile.m]**.

Esempio

Proviamo a creare uno script che raccolga i comandi visti per disegnare una funzione. Creiamo un nuovo file e apriamo l'editor con il comando

```
>> edit disegna.m
```

ed inseriamo le righe

```
x=linspace(-1,1);  
y=x.^2.*sin(x);  
plot(x,y)
```

Ora salviamo il file usando il menu, poi torniamo nella command window e digitiamo

```
>> disegna % senza l'estensione
```

Script

Proviamo a modificare lo script precedente per disegnare due grafici su due finestre diverse. Questo è possibile con il comando **figure(n)**, che apre e rende attiva la finestra n. I comandi di plot che vengono dati successivamente agiscono sulla finestra n. Apriamo lo script disegna.m appena creato e salviamolo con un altro nome, per esempio con duedisegni.m. Supponiamo di voler disegnare in $[-1, 1]$ le due funzioni

$$y = x^2 \sin(x) \quad y = x^2 \cos(x)$$

Una soluzione può essere

```
clear all
close all % chiude tutte le finestre aperte
x=linspace(-1,1);
y1=x.^2.*sin(x);
plot(x,y1)
y2=x.^2.*cos(x);
figure(2)
plot(x,y2)
```

Ora salviamo il file ed eseguiamolo nella command window

Script

Nella notazione scientifica di matlab i numeri come $3 \cdot 10^4$ o $4 \cdot 10^{-5}$ si scrivono

```
>> 3e4
```

```
>> 4e-5
```

Esercizio

Si crei uno script dal nome `confronto.m` in cui si disegnino su due finestre diverse le funzioni

$$y = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$$

$$y = (x - 1)^7$$

nell'intervallo $[1 - 2 \cdot 10^{-8}, 1 + 2 \cdot 10^{-8}]$.

Cosa si può osservare?

Attenzione!

Ricordarsi sempre di salvare il file prima di eseguirlo!

Operatori relazionali

Per confrontare dei valori, in matlab si usano gli **operatori relazionali**, che sono

Relazioni

Operatore	Significato
==	uguale a
~=	diverso da
>	maggiore di
>=	maggiore o uguale a
<	minore di
<=	minore o uguale a

Se una espressione è vera, ad essa viene assegnato il valore di 1, mentre se è falsa vale 0. Questi operatori possono lavorare anche tra array della stessa dimensione, operando elemento per elemento. Per esempio

```
>> v=[1 -1 0];
```

```
>> w=[0 -1 2];
```

```
>> v<=w
```

```
ans =
```

```
0     1     1
```

Operatori logici

Più espressioni condizionali possono essere legate da operatori logici: i due più importanti sono `&&` che rappresenta un **AND** logico e `||` che rappresenta un **OR** logico. Un'espressione del tipo `a && b` è vera se e solo se entrambe le proposizioni sono vere, mentre `a || b` è vera se almeno una delle due proposizioni è vera.

Nel caso in cui più proposizioni siano legati da operatori logici è necessario usare delle parentesi per racchiudere le proposizioni stesse. Per esempio

```
(a==b) && (c>d)
```

Cicli

Quando alcune istruzioni devono essere eseguite un numero prefissato di volte o ripetute finchè una condizione non viene soddisfatta si possono usare i **cicli**. Matlab offre due tipi di cicli: il **for** e lo **while**. Ora verrà preso in esame questo secondo. Il ciclo `while` ripete un insieme di istruzioni finchè una condizione è verificata: la sua sintassi è

```
while condizione per ripetere  
    comandi  
end
```

Il ciclo `while` è detto per questo ciclo condizionato e la condizione è costruita usando gli operatori relazionali e quelli logici. I *comandi* vengono eseguiti finchè la *condizione* rimane vera, poi si prosegue.

Cicli

Vediamo un esempio: calcolare la più piccola potenza di 2 maggiore del valore assoluto di un numero assegnato

```
x=input('Numero da superare ');  
n=0;  
while 2^n<abs(x)  
    n=n+1;  
end  
y=2^n
```

Attenzione

Se le condizioni non sono date attentamente, se il valore di un indice viene modificato in maniera sprovvista è possibile che non si verifichi mai la condizione di uscita da un ciclo. In tal caso si parla di loop infinito, cosa ovviamente da evitare. Se l'esecuzione di un codice ristagna all'interno di un ciclo per più del previsto si può interrompere l'esecuzione premendo CTRL-C.

Precisione di macchina

Precisione macchina

La precisione di macchina ϵ_M è il più piccolo numero binario (quindi potenza di 2) che sommato a 1 da' un risultato diverso da 1

Vediamo ora di calcolare il valore di ϵ_M

Esercizio

Si crei uno script di nome `precmacc.m` che effettui il calcolo della precisione di macchina.

Suggerimenti

- Inizializzare `em` col valore 1
- Iniziare un ciclo: finchè $1+em > 1$
 - Porre `em` pari a $em/2$
- Fine del ciclo
- Porre `em=2*em`

Precisione di macchina

Osservazione

Nel codice precedente è stato necessario moltiplicare per 2 il risultato finale. Il fatto è che si **esce** da un ciclo `while` quando la condizione **risulta falsa**. Se come, in questo caso, si vuole conoscere lo stato delle variabili all'ultimo verificarsi delle condizioni, bisogna ricalcolare all'indietro i loro valori. Nell'ultimo esempio si voleva il più piccolo valore di `em` per cui la condizione risultasse vera, ma il ciclo si interrompe la prima volta che la condizione è stata trovata falsa. Si è in sostanza diviso per 2 una volta di troppo, e quindi alla fine bisogna moltiplicare per due

Si può confrontare il risultato `em` con la variabile predefinita di matlab `eps`.

Osservazione

Se si prova a fare `1+em`, matlab dà come risultato `1.0000`: in realtà per matlab `1+em` è comunque diverso da 1. Infatti si confronti il risultato dell'espressione `1+em>1` con quello di `1+em/2>1`

Istruzioni condizionali

In un programma è possibile avere la necessità di dover agire in modo diverso a seconda del verificarsi o meno di certe condizioni. Matlab offre il costrutto **if...elseif...else...end** per poter strutturare il proprio codice. Ci sono varie possibilità, la più semplice è

```
if condizione
    comandi
end
```

che esegue i comandi se la condizione è verificata.

Si possono poi porre più alternative usando la sintassi seguente

```
if condizione 1
    comandi 1
elseif condizione 2
    comandi 2
    ...
elseif condizione n
    comandi n
end
```

Istruzioni condizionali

che permette di eseguire diversi comandi a seconda della condizione trovata vera.

Si può poi aggiungere sia nella prima sintassi sia nella seconda l'opzione `else`, che viene eseguita se tutte le condizioni precedenti risultano false. La sintassi che comprende tutti i costrutti è la seguente

```
if condizione 1
    comandi 1
elseif condizione 2
    comandi 2
    ...
elseif condizione n
    comandi n
else    % else non vuole nessuna condizione
    comandi alternativi
end
```

Definizione di nuove funzioni

In Matlab è possibile definire nuove funzioni, da utilizzare come quelle predefinite. Questo si può fare attraverso il comando **inline**, la cui sintassi è la seguente

$$\text{nomefunzione} = \text{inline}(\text{'definizione'}, [\text{'elenco variabili'}])$$

Per definire la funzione $f(x) = xe^{-x^2}$ si usa per esempio

```
>> f=inline('x.*exp(-x.^ 2'),'x')
f =
      % con ' ' in matlab si delimitano le
      Inline function:      % stringhe di testo
      f(x) = x.*exp(x.^2)
```

Nella finestra di workspace, si può notare come `f` sia un oggetto di tipo `inline`, diverso da quelli presi in esame sinora. Se la funzione è di una sola variabile, non è necessario specificare l'elenco variabili, come

```
>> f=inline('x.*exp(-x.^ 2')
f =
      Inline function:
      f(x) = x.*exp(x.^2)
```

Definizione di nuove funzioni

Attenzione!

Negli esempi precedenti le operazioni di prodotto e di elevamento a potenza sono state definite usando il punto. Questo, si ricorda è il modo con cui matlab permette di valutare le funzioni su dei vettori. Se si definisce

```
>> h=inline('x*exp(-x^2)')
```

e si prova a calcolare

```
>> x=0:0.1:1;
```

```
>> f(x)
```

si ottiene un messaggio di errore. In generale nelle funzioni si vorrà sempre che siano calcolabili su vettori e quindi i simboli matematici $*$, $/$, $^$ saranno usati nella modalità $.*$, $./$, $.^$. Solo nel caso in cui si vogliono riprodurre le operazioni dell'algebra lineare (ma in genere nelle funzioni non succede) si dovranno usare $*$, $/$, $^$ senza il punto.

Metodo di bisezione

Esercizio

Si vuol risolvere l'equazione $e^x + x = 0$ con il metodo di bisezione. Si effettui uno studio grafico dell'equazione per individuare un intervallo in cui sia applicabile il metodo e si costruisca uno script che lo implementi.

Suggerimenti → Si disegni la funzione con i comandi `linspace` e `plot` in modo da trovare l'intervallo adatto.

→ Si inizializzi lo script, le variabili che servono sono

- `f` per la funzione
- `a, b` per gli estremi dell'intervallo di partenza
- `nmax, toll` per il numero massimo di iterazioni possibili e per la tolleranza del metodo (per esempio `nmax=100, toll=1e-6`)
- `it` un contatore per le iterazioni fatte (all'inizio `nit=0`)

Metodo di bisezione

→ Prima di iniziare il ciclo principale bisogna controllare che il metodo sia applicabile, cioè che $f(a) \cdot f(b) < 0$. Se questo non si verifica si può interrompere l'esecuzione del programma e dare un messaggio di errore con il comando `error('messaggio di errore')`

→ La parte principale sarà costituita da un ciclo del tipo:

- ripetere finchè $(b-a)/2 > \text{toll}$ e $\text{nit} < \text{nmax}$

- calcolare c , punto medio di $[a, b]$

- Se

• $f(c) == 0$ uscire (comando `return`)

• altrimenti se $f(a) \cdot f(c) > 0$ porre $a=c$

• altrimenti porre $b=c$

- aumentare it di 1

- fine del ciclo

Alla fine del programma la soluzione sarà nella variabile c .

Metodo di bisezione

Alcune osservazioni: per l'equazione $x^2 - 2 = 0$, è possibile ricavare entrambe le soluzioni?

Facoltativo - Si provi a stimare quante iterazioni in più sono necessarie per ottenere nella soluzione una cifra corretta in più (in sostanza si riduce di un fattore 10 la tolleranza). Che considerazioni si possono fare a tal riguardo?

Facoltativo - Si provi a considerare una tolleranza di $1e - 15$ e poi si provi con una tolleranza di $1e - 16$. Cosa succede? Perché?