

Calcolo Numerico

A.A. 2013-2014

Esercitazione n. 4

24-03-2014

Script, elementi di programmazione

Files .m

- Al posto di eseguire i comandi direttamente da linea di comando, possiamo memorizzare la successione dei comandi in un file di testo, salvarli e successivamente eseguirli; file di questo tipo sono detti **m-file** (files **.m**)
- Un file **.m** può essere generato con qualsiasi editor di testo ASCII (ad es. notepad di windows o l'editor di Matlab)
- Il **path** di Matlab è un insieme di directory sul computer locale in cui vengono cercate le funzioni o gli script che vengono chiamati dalla linea di comando
- Per visualizzare l'attuale path
 - dalla GUI di Matlab
 - dal prompt digitare il comando **path**

Path

- Se creiamo un nuovo file `.m` possiamo
 - salvare il file in una directory contenuta nel path
 - aggiungere la directory, in cui è salvato il file, al path
 - cambiare la directory di lavoro (Current Directory)
- **Per aggiungere una directory al path**
 - Menù **File** -> **Set Path** apre il **Path Browser**
 - da Path Browser: selezionare la directory con il pulsante **Add Folder** (o **Add with subfolders**)
 - questa operazione è consigliata solo per inserire nuovi pacchetti o quando si vuole aggiungere una nuova funzione ai toolbox esistenti. Negli altri casi è consigliato lavorare nella directory di lavoro o sotto-directory
 - con il comando `addpath('nome_dir')`

Script e funzioni

■ Script files: `nome_file.m`

- eseguono una lista di istruzioni
- non prevedono parametri di ingresso
- utilizzano il workspace di MATLAB, **le variabili usate sono messe nella memoria di lavoro di MATLAB**

■ Funzioni: `nome_funzione.m`

- si possono passare parametri in ingresso ed ottenerne in uscita
- sintassi `function [y1,...,yn] = nome_funzione(x1,...,xn)`
 - `y1,...,yn` -> parametri in uscita
 - `x1,...,xn` -> parametri in entrata
- **le variabili usate all'interno sono locali**

Creazione di m-files

- **Per creare un nuovo m-file**
 - Menù **File** -> **New** -> **M-File**
- Per lanciare uno **script** (e quindi eseguire i comandi in esso contenuti)
 - **se siamo nella stessa directory** dove è salvato il file digitare il nome dello script nella linea di comando
 - **se siamo in una directory** diversa rispetto a quella in cui è salvato lo script
 - digitare dal prompt **run ../file.m** ... indica il percorso dalla cartella dove stiamo lavorando alla cartella in cui il file è salvato
 - è possibile cambiare la directory di lavoro utilizzando il comando **cd** (si digiti **help cd** per maggiori chiarimenti) oppure utilizzando le apposite icone nella barra dei comandi.

Script

- Tutte le variabili utilizzate nello script durante l'esecuzione dell' M-file vengono automaticamente messe nella memoria di lavoro di MATLAB
 - vedremo come questo non valga nel caso in cui si crei una funzione
- Per una minima manipolazione dei file su disco, MATLAB mette a disposizione alcuni comandi

`dir, delete, cd, pwd, mkdir, copyfile`

- **Esempio:** disegniamo una retta e una parabola

Esempio

- Creiamo il file `disegna.m` e scriviamo la lista di comandi

```
% Disegna una retta e una parabola
%
n = 5;
x = linspace(-n,n);
y1 = x;
y2 = x.^2+5*x+1;

figure, hold on
plot(x,y1)
plot(x,y2,'r')
```

- Per eseguire il file, dal prompt

```
>> disegna
```

N.B. Il file deve essere salvato nella directory di lavoro che è quella in cui ci troviamo!!!

Commenti

- Il carattere `%` serve per introdurre un commento all'interno dello script, MATLAB ignora il contenuto alla destra del carattere `%` fino alla linea successiva
 - **CTRL R**(**CTRL T**) per commentare (eliminare il commento da) una riga
- 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 disegna
```

```
disegna.m
```

```
Disegna una retta e una parabola
```

- Una caratteristica degli script è quella di non avere parametri in ingresso modificabili. Ad esempio se vogliamo modificare i valori di `n` dobbiamo modificare ogni volta lo script.

Programmare in MATLAB

- Esistono molti comandi per creare m-file complessi e versatili e strutture logiche simili a quelle usate nei linguaggi di programmazione
 - alcune strutture di programmazione elementari
- **Operatori relazionali:** `<`, `<=`, `>`, `>=`, `==`, `=`
- **Operatori logici:** `&` (and), `|` (or), `~` (not)
- **Cicli controllati da un contatore:** `for...end`
- **Cicli condizionati:** `while...end`
- **Strutture condizionali:** `if - elseif - else - end`
- **Uscita incondizionata:** `break`

Nota: tutte le strutture possono essere scritte su più righe oppure su un'unica riga separate da virgole

Programmare in MATLAB

- Anche gli algoritmi più semplici richiedono l'**esecuzione ripetuta di istruzioni** e l'esecuzione condizionata di alcune parti
- La **costruzione ripetuta di blocchi di codice** in MATLAB viene eseguita tramite cicli. Esistono due diversi modi per realizzare cicli
 - il **ciclo incondizionato `for...end`**
 - il **ciclo condizionato `while...end`**
- Importante è l'uso degli **operatori relazionali**

Operatori relazionali

Operatori relazionali

< , <= , > , >= , == , ~=

si usano per **confrontare** tra di loro gli elementi di 2 vettori (matrici); il risultato dell'operazione sarà una variabile di tipo **logical**

- ❑ 0 se la relazione è falsa
- ❑ 1 se la relazione è vera

Esempio:

```
>> 2==3
```

```
ans =
```

```
0
```

```
>> 2 ~ =3
```

```
ans =
```

```
1
```

Operatori logici

Operatori logici

& , | , ~

si usano per **combinare** tra loro gli operatori relazionali

Operatore	Significato	a	b	a & b	a b	~a	xor(a,b)
&	and	0	0	0	0	1	0
	or	1	0	0	1	0	1
~	not	0	1	0	1	1	1
xor	or esclusivo	1	1	1	1	0	0

Esempio:

```
>> a = [0 0 1 1]; b = [0 1 0 1]
```

```
>> c = xor(a,b)
```

```
c =
```

```
0 1 1 0
```

Osservazioni

- Sono **operazioni binarie** come la somma e il prodotto e danno luogo ad un risultato
 - la differenza è che il risultato può assumere solo 2 valori: 0 (se la relazione è falsa) e 1 (se la relazione è vera)
 - il risultato è una variabile di tipo **logical**
- Gli operatori logici e relazionali possono essere applicati a scalari, vettori e matrici
 - l'operatore relazionale è applicato ad ogni elemento dei vettori (matrici) che si confrontano per cui i vettori (matrici) devono avere le stesse dimensioni, oppure uno dei due deve essere uno scalare
- Gli operatori logici e relazionali sono usati nei **test condizionali** (**if...end, while...end**)

Osservazioni

- I vettori (matrici) di tipo logical possono essere usati per selezionare elementi di un vettore (matrice)

Esempio:

```
>> x = [30 12 19 7 5]
```

```
>> x > 15
```

```
ans =
```

```
1 0 1 0 0
```

```
>> x(x>15)
```

```
ans =
```

```
30 19
```

- In alcune circostanze questa notazione compatta risulta particolarmente utile, ad esempio per individuare gli elementi non nulli di una matrice o di un vettore

Ciclo for...end

Ciclo incondizionato

```
for indice = m:p:n
    blocco di istruzioni
end
```

Ripete il blocco di istruzioni un numero fissato di volte

- **indice** contatore
- **m** valore iniziale del contatore
- **p** incremento del contatore (positivo o negativo)
- **n** valore finale del contatore

m, **p**, **n** possono essere variabili intere o reali

Nota: `indice = m:n` equivale a `indice = m:1:n`

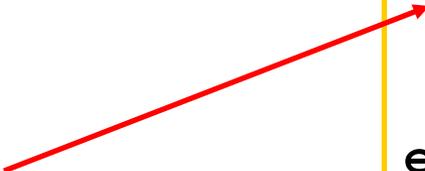
Ciclo for...end - esempio

Consideriamo il semplice problema del calcolo del **valore medio** di un vettore o di una matrice. Dato un vettore x di n componenti il valore medio è definito come

$$m = \frac{1}{n} \sum_{i=1}^n x_i$$

```
In uno script
n=length(x);
somma=0;
for i=1:n
    somma=somma+x(i);
end
m=somma/n;
```

```
for i = n1:passo:n2
    blocco di istruzioni
end
```



Ad esempio provare per $\mathbf{x}=[1 \ 2 \ 3 \ 4 \ 5 \ 6]$

Ciclo for...end

- Nell'esempio di prima l'uso del ciclo for può essere evitato utilizzando la funzione MATLAB **sum** nella forma

```
somma=sum(x)
```

che assegna alla variabile somma la somma degli elementi del vettore x

- In MATLAB esiste la funzione predefinita per il calcolo della media di un vettore

```
help mean
```

Osservazioni

- Non è obbligatorio, ma è fortemente consigliato, scrivere il blocco di istruzioni all'interno del ciclo allineandolo con l'espressione che governa il ciclo. Questo modo di procedere consente di evidenziare bene le parti di codice che vengono eseguite nei cicli o sotto opportune condizioni
- È possibile **annidare** più cicli **for**

```
for i = n1:passo1:n2
    for j = m1:passo2:m2
        blocco di istruzioni
    end
end
end
```

Ciclo while...end

Ciclo condizionato

```
while condizione
    blocco di istruzioni
end
```

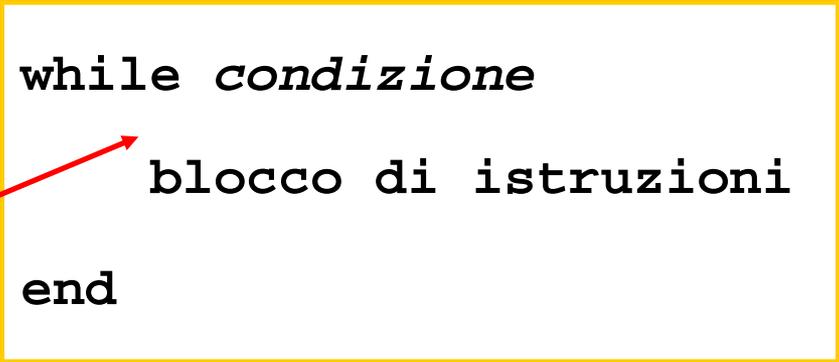
Esegue un blocco di istruzioni un numero indefinito di volte fino al persistere di una certa condizione: **condizione** è un'espressione che MATLAB valuta numericamente e che viene interpretata come vera se diversa da zero

- ❑ in un contesto logico una variabile numerica restituisce 1 (vero) se la variabile è diversa da zero, altrimenti 0 (falso). Non è quindi noto a priori il numero di ripetizioni del blocco di istruzioni
- ❑ nel ciclo for si ripete il blocco d'istruzioni tot volte. In molte circostanze si ha la necessità di ripetere un certo numero di operazioni diverse volte a seconda che una certa condizione sia verificata oppure no. In questo caso si utilizza il costrutto **while**

Ciclo while...end - esempio

In uno script

```
n=length(x);  
somma=0;  
i=1;  
while i<=n  
    somma=somma+x(i);  
end  
m=somma/n;
```



```
while condizione  
    blocco di istruzioni  
end
```

Esegue il ciclo fino a quando i è minore uguale a n

Esempio: visualizzare i primi 3 numeri naturali

```
>>x=1;  
>>while x~=4, disp(x), x=x+1; end
```

1

2

3

Osservazioni

- Nel ciclo **while** si fa uso degli operatori relazionali
- L'operatore relazionale **==** che confronta il valore della variabile a sinistra dell'operatore con quello della variabile a destra, non va confuso con l'operatore di assegnazione **=** che invece assegna il valore della quantità alla destra dell'operatore alla variabile a sinistra dello stesso
- Qualora, a causa di un errore di programmazione, il programma dovesse ripetere un numero indefinito di volte il blocco di istruzioni perché la condizione di persistenza è sempre verificata, è possibile interrompere l'esecuzione del programma premendo contemporaneamente i tasti **Ctrl+C**

Osservazioni

Privilegiare operazioni vettoriali ai cicli **for...end** e **while...end**

- ❑ la caratteristica principale di MATLAB consiste nella possibilità di eseguire operazioni vettoriali. L'uso efficiente di MATLAB è strettamente legato alla capacità dell'utente di sfruttare tale caratteristica
- ❑ un ciclo di istruzioni di tipo **for ...end** o **while ...end** comporta l'esecuzione ripetuta in forma sequenziale di un blocco di istruzioni
- ❑ è buona regola prima di scrivere un ciclo cercare di vedere se è possibile evitarlo tramite un uso opportuno di istruzioni vettoriali
- ❑ MATLAB è un linguaggio interpretato e solo recentemente sono stati introdotti opportuni compilatori. L'efficienza e la velocità saranno quindi ridotte rispetto agli usuali programmi scritti in linguaggi ad alto livello, come il C o il Fortran, e compilati.

Comandi “utili”

- **break**

- per uscire in maniera forzata da un ciclo
- MATLAB salta automaticamente all’istruzione **end** che termina il ciclo

- **return**

- interrompe l’esecuzione della funzione
- svolge una funzione analoga a break, la differenza è che return interrompe l’esecuzione della funzione e ritorna al programma da cui tale funzione era stata chiamata

Funzioni di input\output

- Per introdurre il valore di una variabile dal prompt di MATLAB

```
nome_variabile=input('stringa di caratteri tra apici')
```

- il valore assegnato potrà essere di tipo scalare, vettoriale oppure matriciale
- si utilizzan la sintassi standard di MATLAB

Esempio:

```
>>n=input('inserisci il numero di righe: ')
```

```
inserisci il numero di righe: 3
```

```
n =
```

```
3
```

```
>>v=zeros(n,1); %crea un vettore di zeri di dimensione n
```

Funzioni di input\output

- Per visualizzare solo **stringhe** sullo schermo si può usare **disp**

```
disp('stringa di caratteri')
```

Esempio:

```
>> disp('Oggi e'' una bella giornata')
```

- Per visualizzare sullo schermo **stringhe e variabili**

```
fprintf('stringa val = %d', variabile)
```

o

```
str=sprintf('stringa val=%d',variabile),disp(str)
```

- vedremo in seguito la **formattazione** da usare per i vari tipi di variabili, per interi si usa **%d**

Esempio:

```
>> n = input('inserisci un intero');
```

```
>> s = sprintf('n = %d',n);
```

```
>> disp(s)
```

If...else...end

- Test condizionale

```
if condizione1  
    blocco di istruzioni  
elseif condizione2  
    blocco di istruzioni  
else  
    blocco di istruzioni  
end
```

- Il primo blocco di istruzioni sarà eseguito se e solo se la Condizione1 è verificata, il secondo se e solo se la Condizione1 risulta essere falsa e la Condizione 2 vera e così via
 - **condizione1, condizione2** coinvolgono operatore relazionali
- Il blocco di istruzioni che segue else sarà eseguito soltanto se nessuna delle precedenti condizioni risulti essere vera.

If...else...end - esempio

- Data la parabola $y = x^2 + 3x + 2$, dire se il punto P di coordinate (-3,4) vi appartiene

```
>> y = @(x)[x.^2+3*x + 2]
>> P = [-3,4];
>> if y(P(1))==P(2)
>>     fprintf('P(%d, %d) appartiene alla parabola',...
>>         P(1),P(2))
>> else
>>     fprintf('P(%d, %d) non appartiene alla parabola',...
>>         P(1),P(2))
>> end
```

Osservazioni

- Il test condizionale è usato quando si vuole che un comando venga eseguito se e solo se è verificata una certa condizione
- Le condizioni sono espresse mediante l'uso di operatori logici e relazionali
- **else** si può omettere

```
if condizione1  
    blocco di istruzioni  
end
```

Esercizio 1

- Scrivere uno script **fattoriale.m** che calcola il fattoriale di un numero intero positivo **n** passato come input
 - effettuare il controllo su **n** che deve essere un numero intero positivo
 - usare un ciclo **for...end**
 - aggiungere opportuni commenti
 - visualizzare il risultato
 - eseguire lo script per diversi valori di **n**
 - qual è la funzione di MATLAB che calcola il fattoriale?

```
% calcola il fattoriale di un numero positivo n
n=input('inserisci un numero ')
while n <=0 || round(n)~=n
    n=input('inserisci un numero ')
end

fatt = 1;
for i=1:n
    fatt = fatt*i;
end

fprintf('il fattoriale di %d e'' %d', n, fatt)
```

Esercizio 2

- Scrivere uno script (`script1.m`) che richieda il valore della variabile \mathbf{x} come input e calcoli il valore di $e^{\mathbf{x}}$ usando lo sviluppo in serie di Mac Laurin arrestato all'ordine \mathbf{n} tale che il termine n -simo sia trascurabile nella precisione di macchina
 - si confronti il risultato con il valore vero della funzione in \mathbf{x} scegliendo $\mathbf{x} = 2$ e $\mathbf{x} = -20$

```
x =input('introduci il valore in cui calcolare la funzione,x = ')
oldsum = 0;, newsum = 1;
term = 1; % primo termine della serie
n = 0;    % indica il termine della serie

while newsum ~= oldsum, %Itera finchè il termine successivo della
                        %serie diventa trascurabile (nella
                        %precisione di macchina)

    oldsum = newsum;
    n = n + 1;
    newsum = newsum + x^n/factorial(n);

end

approx_value = newsum;
true_value = exp(x);
disp('valore approssimato'),
disp(approx_value)
disp('valore vero')
disp(true_value)
```

Esercizio 3

Consideriamo per ogni $k \in \mathbb{R}$ la funzione seguente

$$f(x) = x e^{\frac{1+x^2}{kx^2}}$$

Costruire un m-file funzione di k t.c.

- ❑ esegua un controllo su k che deve essere non nullo
- ❑ disegni il grafico di $f(x)$ nell'intervallo $[1,2]$
- ❑ calcoli il punto di minimo e il valore di minimo della funzione nell'intervallo
- ❑ dica se il punto di minimo è interno all'intervallo o in uno degli estremi

```

k = input('inserire k ');
if k==0
    disp('introdurre un valore non nullo per k')
end
intervallo = [1 2];
x = linspace(intervallo(1),intervallo(2));
y = x.*exp((1+x.^2)./(k*x.^2));
figure, hold on
plot(x,y)
[y_min i_min] = min(y);

x_min = x(i_min);

% dire se x_min è interno a [1,2] o in uno degli estremi
if x_min ==intervallo(1) || x_min ==intervallo(2)
    fprintf('%2.4f coincide con uno degli estremi
dell''intervallo [%d %d]\n',x_min,intervallo(1),intervallo(2))
else
    fprintf('%2.4f e'' interno all''intervallo [%d
%d]\n',x_min,intervallo(1),intervallo(2))
end

```