UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
INDUSTRIALE

DIPARTIMENTO
MATEMATICA
DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

MATLAB STEP BY STEP

Materiale realizzato da Michela Redivo Zaglia

con il contributo di E. Bachini, L. Bruni, W. Erb, A. Larese, F. Piazzon

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

DIPARTIMENTO MATEMATICA

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Laboratorio di Calcolo Numerico LAB 4

Approfondimenti su vettori e funzioni in Matlab

Docenti: E. Bachini, L. Bruni

Email: elena.bachini@unipd.it Email: bruni@math.unipd.it

27 marzo 2024

Outline

Vettori riga equispaziati

Si è visto che vi sono due modi equivalenti per creare vettori che contengano valori equispaziati:

1. definito il passo `h` si può usare

   `x = a : h : b;`

   Se `h=1` posso scrivere semplicemente `x = a:b`.
2. definito il numero di componenti `n` si può usare il comando `Matlab`

   `x = linspace(a,b,n);`

Entrambe le modalità creano un vettore riga!

Esempio (viene creato lo stesso vettore riga)

```
>> n = 5;
>> x = 1:n
x =
     1     2     3     4     5
>> x = linspace(1,n,n)
x =
     1     2     3     4     5
```

L'operatore colon :

La modalità 1, ovvero la scrittura di un vettore utilizzando l'operatore Matlab colon : risulta molto utile per definire

- gli indici delle componenti di un vettore che si vogliono considerare;

```
>> n = 3; x = rand(1,5)
x =
    0.6555    0.1712    0.7060    0.0318    0.2769
>> x1 = x(2:n)
x1 =
    0.1712    0.7060
```

- i valori che deve assumere la variabile indice (riservata che non deve essere modificata ma solo utilizzata) in un ciclo `for`. Esempio (script):

```
n = 5; x = rand(n,1);
for i = 1:n
    x1(i) = 3*x(i);
end
```

ATTENZIONE: Si noti che `x` era vettore colonna, mentre `x1` è vettore riga! Un vettore creato componente per componente all'interno di un ciclo `for` è sempre riga.

L'operatore colon :

L'ultimo esempio può essere semplificato in Matlab:

```
n = 5; x = rand(n,1);
x1 = 3*x;
```

ATTENZIONE: Le strutture `for`, grazie alla vettorizzazione, possono essere spesso evitate nel linguaggio Matlab, e si deve sempre farlo, quando è possibile.
Altro esempio di ciclo `for`:

```
n = 1:2:7;
for i = n
    x(i) = i
end
```

Provate ad eseguirlo e cercate di capire perché crea il vettore

```
x =
     1     0     3     0     5     0     7
```

L'operatore colon : e la keyword end

Operatore : e keyword end sono utilissime per indicare quali componenti del vettore si desidera considerare. Vanno utilizzate nella descrizione degli indici di un vettore. Ovvero quando, dato un vettore x si usa la sintassi Matlab

x(indici)

- L'operatore : ha il significato di tutte le componenti del vettore;
- La keyword end ha il significato di l'ultima componente del vettore.

Esempio: voglio assegnare ad a l'ultima componente del vettore x:

```
>> x = [-2, 3, 50, -10, 44];
>> a = x(end)
a =
    44
```

E' equivalente (ma migliore) del comando seguente

```
>> a = x(length(x))
a =
    44
```

L'operatore colon : e la keyword end

Esempio: voglio creare un vettore `x1` che contenga le componenti dalla terza all'ultima del vettore `x`:

```
>> x = [-2, 3, 50, -10, 44];
>> x1 = x(3:end)
x1 =
    50   -10    44
```

Esempio: voglio creare un vettore `x1` che contenga TUTTE le componenti del vettore `x`:

```
>> x = [-2, 3, 50, -10, 44]; % x vettore riga
>> x1 = x(:)                 % x1 vettore colonna
x1 =
    -2
     3
    50
   -10
    44
```

Avrei potuto scrivere anche il comando seguente, ma con un risultato che sarebbe stato diverso!

```
>> x1 = x                    % x1 vettore riga
x1 =
    -2     3    50   -10    44
```

L'operatore colon : per forzare un vettore ad essere colonna

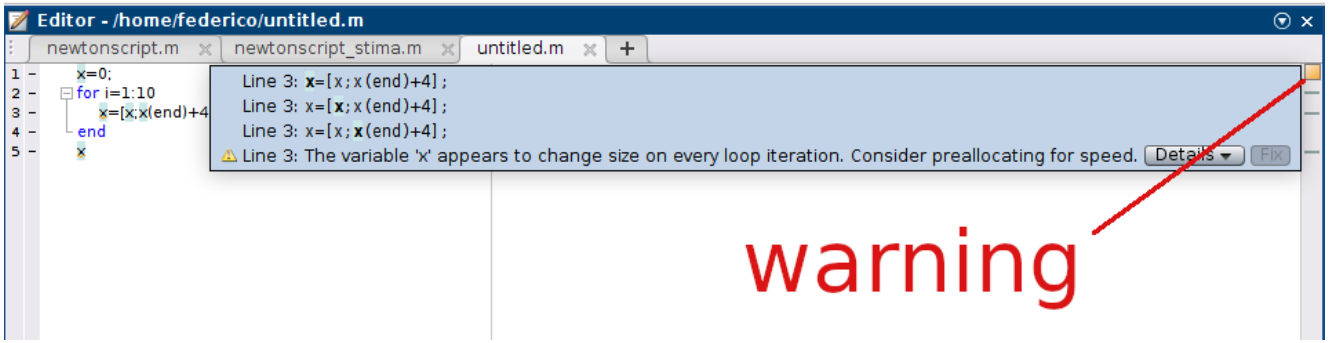
Abbiamo visto che in molti casi i vettori che vengono costruiti, sono vettori riga. Possiamo 'forzare' un vettore ad essere colonna proprio con l'operatore :
In Algebra lineare i vettori matematici sono sempre vettori colonna.

Non dobbiamo confondere tale operatore con l'operatore di trasposizione '

```
>> x = [-2, 3, 44]; % vettore riga
>> x = x'           % vettore colonna
x =
    -2
     3
    44
```

```
>> x = [-2, 3, 44]; % vettore riga
>> x = x(:)         % vettore colonna
x =
    -2
     3
    44
```

Preallocazione ed inizializzazione



Molti algoritmi prevedono l'utilizzo di cicli (`for` o `while`) nei quali vengono creati dei vettori, aggiungendo le componenti una alla volta ed aumentando quindi via via la dimensione del vettore stesso. Tale metodo impone al Matlab memory manager di riallocare la memoria per il vettore stesso (inefficiente!). Il Matlab lo segnala con un Warning.
Quando è possibile conoscere in anticipo le dimensioni massime che il vettore potrà o dovrà avere, è bene inizializzarlo (ad esempio, con il comando `zeros`), e poi assegnare uno ad uno i suoi elementi.

Preallocazione ed inizializzazione

Esempio: utilizziamo un ciclo `for` (in Matlab potrebbe essere scritto in un modo vettorizzato) solo per far capire il concetto. Creiamo un vettore `x` che contenga i valori `1,2,...,n`, con `n=100`

```
n = 100;
for i = 1:n
   x(i) = i;
end
```

Noi sappiamo esattamente quante componenti verranno create: esattamente `n`. Quindi prima del ciclo possiamo preallocare lo spazio per il vettore `x`:

```
n = 100;
x = zeros(1,n); % preallochiamo per il vettore riga spazio
                % per n componenti
for i = 1:n
   x(i) = i;
end
                % x sara' un vettore riga!
```

```
n = 100;
x = zeros(n,1); % preallochiamo per il vettore colonna spazio
                % per n componenti
for i = 1:n
   x(i) = i;
end
                % x sara' un vettore colonna!
```

Preallocazione ed inizializzazione

Possiamo preallocare lo spazio anche nei cicli `while` qualora si sia definita una variabile (ad esempio `nmax` che indichi in numero massimo di iterazione che possono essere effettuate. Ad esempio (script)

```
nmax = 10;
n = 1;              % Inizializza il contatore n
xv(1) = 10;         % Inizializza la prima componente
while (xv(n)>=0) &&  (n < nmax)
    n = n+1;        % incrementa il contatore delle iterate
    xv(n) = xv(n-1) - n^2;
end
```

Otteniamo il vettore di **3 componenti**

```
xv =
    10     6    -3
```

Si noti che nell'editor all'interno del ciclo il vettore `xv` ha una sottolineatura rossa e se mettiamo puntatore del mouse ci fornisce il warning precedentemente visto.

Preallocazione ed inizializzazione

Preallochiamo:

```
nmax = 10;
n = 1;                  % Inizializza il contatore n
xv = zeros(1,nmax); % preallochiamo
xv(1) = 10;             % Inizializza la prima componente
while (xv(n)>=0) &&  (n < nmax)
    n = n+1;            % incrementa il contatore delle iterate
    xv(n) = xv(n-1) - n^2;
end
```

Otteniamo un vettore riga di **10 componenti** in quanto oltre le 3 definite, restano le restanti componenti create e poste uguali a zero.

```
xv =

    10     6    -3     0     0     0     0     0     0     0
```

ATTENZIONE: Se preallochiamo ed il vettore viene creato in una function m-file, bisogna tenere conto del fatto che verrà restituito sempre un vettore di `nmax` componenti e non di `n` componenti che corrispende al numero delle iterate calcolate. Quindi, ad es.

```
xv(end) % sara' uguale a -3 (SENZA preallocazione)
        % sara' uguale a  0 (CON preallocazione)
```

Creare vettori nei cicli per concatenazione

In un ciclo (`for, while`), anziché assegnare le componenti di un vettore utilizzando gli indici, è possibile (consigliato quando si ha difficoltà ad usare gli indici) utilizzare la concatenazione. Cerchiamo di capire come fare con un esempio

```
n = 100;
x = [];         % inizializziamo x come vettore vuoto
for i = 1:n
   x = [x;i]; % viene "aggiunta" una componente al precedente
              % vettore x;
end           % essa contiene il valore della variabile "i"
              % x sara' un vettore colonna!
```

Stessa modalità in un ciclo while (Es. function `bisezfun`)

```
% Inizializza i vettori xv e fxv
xv  = [];
fxv = [];
......
% Ciclo iterativo del metodo
while (amp >= toll) && (n < nmax) % TEST DI ARRESTO
    .....
    x = a + amp*0.5; % calcola il punto medio dell'intervallo
    fx = f(x);       % Calcola il valore del residuo f(x)
    xv = [xv; x];    % aggiunge al vettore la nuova iterata
    fxv = [fxv; fx]; % aggiunge al vettore il nuovo residuo
    .....
end
```

Creare vettori nei cicli per concatenazione (preallocazione)

Anche con la modalità di concatenazione il Matlab sottolinea in rosso i vettori le cui dimensioni aumentano ad ogni ciclo, fornisce il warning precedentemente visto, e suggerisce di preallocare.
Ma in questo caso, se preallochiamo, non otteniamo un risultato corretto.
Esempio:

```
n = 5;
x = [];          % inizializziamo x come vettore vuoto
for i = 1:n
   x = [x, i]; % viene "aggiunta" una componente al precedente
               % vettore x;
end            % essa contiene il valore della variabile "i"
               % x sara' un vettore riga!
```

```
x =
     1     2     3     4     5
```

Preallochiamo

```
n = 5;
x = zeros(1,n); % Preallochiamo x
for i = 1:n
   x = [x, i];  % viene "aggiunta" una componente al "precedente"
                % vettore x;
end             % x sara' un vettore riga!
```

```
x =
     0     0     0     0     0     1     2     3     4     5
```

Consigli per scrivere bene le functions m-file

- All'interno delle function m-file NON devono essere inseriti (di norma) comandi di lettura da tastiera (`input(...)`) o da file e comandi di visualizzazione (`disp, fprintf`) o scrittura su file di risultati.
- Rispettare pertanto il concetto di "black box" dove i valori di ingresso (dati su cui lavorare) vengono passati alla function SOLO tramite i parametri di ingresso ed i risultati vengono restituiti al programma chiamate SOLO attraverso i parametri di uscita.
- In una function non ha senso settare il formato (NO istruzioni `format`).
- In una function si deve scrivere SEMPRE la parte di commenti iniziale che descrive cosa fa la funzione, come si usa e le caratteristiche dei parametri (viene visualizzata tramite il comando Matlab `help nomefunction`).

Consigli per scrivere bene le functions m-file

- Il nome interno dato alla function DEVE coincidere con il nome esterno dato al file.
- Cercare di inserire (senza esagerare) dei commenti anche all'interno della function (e degli script!) per descrivere brevemente il significato di quanto si sta facendo.