02/12/2013

Informatica per la Finanza
Informatica per le Scienze Economiche
Informatica e Programmazione per il Turismo

Complementi sull'uso delle funzioni

Puntatore a funzione (*function handle*)

- Modo indiretto per richiamare una funzione:
  - Si fa precedere il nome della funzione dal carattere @, il valore restituito può essere assegnato ad una variabile che potrà essere usata come la funzione
  - Si possono creare vettori di celle (non vettori convenzionali) i cui argomenti sono dei puntatori a funzioni
  - La variabile puntatore a funzione richiede sempre le parentesi per eseguire la funzione cui è associata

Complementi sull'uso delle funzioni

```
 >> a=@sin
a =
    @sin

>> a(pi/2)
ans =
     1

>> C={@sin, @cos, @tan}
C =
    @sin    @cos    @tan

>> C{3}(0)
ans =
     0
```

Alle celle si accede elemento per elemento.

Esempio

- dato il vettore x applicare su esso la funzione sin, cos o tan a seconda se la variabile y vale 1, 2 o 3

```
vFunz={@sin, @cos, @tan}
vFunz{y}(x)
```

```
function opVarie(y, x)
   vfunz={@sin, @cos, @tan};
   vfunz{y}(x)
end
```

Complementi sull'uso delle funzioni

Funzioni **anonime**

– Modalità rapida per definire delle funzioni senza utilizzare M-file

```
nomeFunz = @(arg1, …,argn) expr
```

- **`nomeFunz`**: nome della funzione
- **`argX`**: argomento X-mo della funzione
  - se non ci sono argomenti le parentesi tonde vanno comunque indicate
- **`expr`**: espressione MathLab
  - va scritta su di una sola riga
  - se la funzione restituisce più valori va creato un vettore

Complementi uso funzioni

- Scrivere la funzione anonima che converte gradi (sessagesimali) in radianti

  ∏: 180 = rad : grad

  rad = ∏*grad/180

```
convDrgRad = @(grad) { pi*grad/180}
```

Complementi uso funzioni

- Scrivere la funzione anonima che restituisce
  - 1 se il valore in ingresso rappresenta un intero (non ha cifre dopo la virgola)
  - 0 altrimenti

```
intero=@(x){floor(x) == x}
```

Complementi uso funzioni

```
>> nuovaFz=@() (4*sin(4))
nuovaFz =
    @()(4*sin(4))

>> nuovaFz()
ans =
   -3.0272

>> anFz=@(a,b)[a*b,b-a; 4, a+b]
anFz =
    @(a,b)[a*b,b-a;4,a+b]

>> anFz(6,7)
ans =
    42     1
     4    13
```

Restituisce più valori

Importare/Esportare

- MatLab dispone di numerosi modi per esportare o importare dei dati.
- Si accennerà solo ai principali:
  - In formato binario da una sessione MatLab
  - In formato testo
    - Attraverso Import Wizard
  - In formato Excel

Formato MatLab: esportazione

- In questo modo si esportano in formato binario a doppia precisione

  le variabili contenute nell'ambiente di lavoro:
  - Per esportare tutte le variabili:

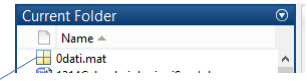
    `save` ***`nomefile`***
    - Viene creato un file denominato *`nomefile.mat`*
  - Per esportate solo alcune variabili

    `save` ***`nomefile var1 … varN`***
    - Viene creato il file *`nomefile.mat`* solo con le variabili indicate
  - Usando
    - opzione **`-append`** si aggiunge ad un file esistente
    - opzione **`-ascii`** si salva in formato ASCII
    - opzioni **`-ascii -tabs`** in formato ASCII delimitato da tabulazione
  - Con il comando **`whos -file`** ***`nomefile`*** si può vedere il contenuto del file creato

Formato MatLab: esportazione

```
>> clear
>> a=[5 6 7 8]
a =
     5     6     7     8

>> b=23.4
b =
   23.4000

>> c= a*b
c =
  117.0000  140.4000  163.8000  187.2000

>> save 0dati a b
```

Cancella tutte le variabili



Formato MatLab: importazione

- Per importare i dati da un file in formato MatLab si usa
  - **`load`** ***`nomefile`***
    - Importa tutte le variabili dal file sovrascrivendo le omonime nel Workspace
  - **`load`** ***`nomefile var1 … varN`***
    - Importa dal file solo le variabili indicate sovrascrivendo le omonime nel Workspace
      - In questo caso è opportuno usare prima il comando

        `whos -file` ***`nomefile`***
  - I dati salvati in formato ASCII possono essere importati con l'istruzione **`load`** ma
    - debbono essere
      - in formato numerico,
      - separati da spazio, virgola o tab,
      - contenere lo stesso numero di elementi per riga
    - sono assegnati ad una unica variabile matrice

Formato MatLab: importazione

```
>> a = 56
a =
    56

>> b= 67
b =
    67

>> load 0dati
>> a
a =
     5     6     7     8

>> b
b =
   23.4000
```

Valori iniziali di **a** e **b**

Nuovi valori di **a** e **b** dopo importazione

Import Wizard

- E' il modo più semplice per importare dati in MatLab:
  - Si sceglie la voce **`Import Data`** dal menù **`File`**
  - Si seguono le indicazioni fornite
  - E' possibile salvare il codice generato per l'importazione dei dati
    - Spuntare la casella ***`Generate M-File`***
      - Non sempre è possibile dipende dalla versione

Importare/Esportare da Excel

- E' possibile esaminare preliminarmente il contenuto di un file Excel con l'istruzione:

  `xlsfinfo(`*`nomefile`*`)`

  – Restituisce un vettore con due elementi
    - Tipo
    - Nome dei fogli di lavoro (Sheets)
- (ATTENZIONE FUNZIONA SOLO IN MATLAB)

Importare/Esportare da Excel

– Per esportare in un file Excel:
  - Si usa **`xlswrite`** con i seguenti parametri:
    – Nome file Excel in cui immettere i dati
    – Nome variabile da esportare
    – Nome del foglio di lavoro in cui inserire i dati
    – Cella da cui deve partire l'inserimento dei dati

– Per importare in MatLab:
  - Si usa **`xlsread`** con i seguenti parametri:
    – Nome file Excel da cui prendere i dati
    – *Nome del foglio di lavoro da cui prelevare i dati*
    – *Intervallo da cui prelevare i dati*
  - Restituisce
    – Matrice con i valori numerici
    – Matrice con le stringhe
    – Matrice con valori non tradotti

Esempio

- Prelevare dal file Book1 nel foglio Sheet2 i valori numerici compresi nelle celle da A1 a C3
  - Di questi dati calcolare la somma lungo le righe e le colonne
  - scrivere la matrice ottenuta dal primo punto con i valori delle somme dei suoi margini nel foglio sommeFinali

Esempio

```
A ris = xlsread('Book1.xlsx', 'Sheet1','A1:C3');

  %somma delle colonne
  sC = sum(ris);

  %somma delle righe
  sR = sum(ris, 2);

  %matrice finale
B usc=[ris sR; sC sum(sR)];

  %scrittura sul foglio Excel
C xlswrite('Book1.xlsx',usc,'sommaFinali');
```

L'estensione è obbligatoria quando si usano i nuovi formati (il default è **.xls**)

Se il foglio non esiste viene creato

Quando si scrive sul file, questo deve essere chiuso

Esempio

A

	A	B	C	D	E
1	45	23	89		
2	4	458,00	4,36		
3	12,25	2	45		
4					
5					
6					

Sheet1 sommaFinali

B

```
>>usc
usc =
   45.0000   23.0000   89.0000  157.0000
    4.0000  458.0000    4.3600  466.3600
   12.2500    2.0000   45.0000   59.2500
   61.2500  483.0000  138.3600  682.6100
```

C

	A	B	C	D	E
1	45	23	89	157	
2	4	458	4,36	466,36	
3	12,25	2	45	59,25	
4	61,25	483	138,36	682,61	
5					
6					

Sheet1 sommaFinali

Esempio

A

	A	B	C	D	E
1	45	23	89		
2	sfda	12/08/2103	4,36		
3	12,25	VERO	45		
4					
5					
6					

Sheet1 sommaFinali

B

```
>> usc
usc =
   45.0000   23.0000   89.0000  157.0000
       NaN       NaN    4.3600       NaN
   12.2500    1.0000   45.0000   58.2500
       NaN       NaN  138.3600       NaN
```

I valori **NaN** non sono riportati

C

	A	B	C	D	E
1	45	23	89	157	
2			4,36		
3	12,25	1	45	58,25	
4			138,36		
5					
6					

sommaFinali

Esempio

- Leggere il contenuto del foglio 2 (Sheet2) del file provaMatlab.xls dalla cella c3 alla d4 e scriverlo nella variabile `lettura`
  - Su ogni elemento di `lettura` verificare se
    - Contiene un numero Intero
      - Se è intero positivo se è primo
    - Scrivere nella matrice `risposta`
      - 1 se è primo
      - 0 se non è primo
      - -1 se non è intero positivo

Soluzione 1

```
function [pos0]=leggi(nomeFile, nomeFoglio, intervallo)
   H=xlsread(nomeFile,nomeFoglio,intervallo)
   pos0 = testa(H);
end

function pos=testa(H)
    pos=[];
    for i=1:size(H,1)
        for j=1:size(H,2)
            if (~isnan(H(i,j)))
                if (H(i,j)>0 && floor(H(i,j))==H(i,j))
                    pos(i,j) = isprime(H(i,j));
                else
                    pos(i,j) = -1;
                end
            else
                pos(i,j) = -1;
            end
        end
    end
end
```

Si richiama:
`>> leggi('provaMatlab.xls','Sheet2','C3:D4')`

Verifica se il numero è intero

Il codice della funzione **`testa()`** può essere inserito direttamente in **`leggi()`**

Soluzione 2

```
function [risposta]=leggi(nomeFile, nomeFoglio, …
      intervallo)
    lettura=xlsread(nomeFile,nomeFoglio,intervallo);
    risposta = testa1(lettura);
end

function pos=testa1(H)
  H(isnan(H))=0;
  H(H<0) = 0;
  H(floor(H) ~= H)=0;
  zeri = H == 0;
  pos=isprime(H) + zeri*-1;
end
```


Soluzione 2: analisi

```
H=[NaN 10 17 NaN;
   8.36 NaN -10 2;
   1 31 3.9 4.58]
```

isnan(H)

```
     1     0     0     1
     0     1     0     0
     0     0     0     0
```

H(isnan(H))=0

```
H([1 0 0 1; 0 1 0 0; 0 0 0 0])=0
```

Essendo una matrice di valori logici è inutile e più lento usare la forma **H(find(isnan(H)))=0;**

```
H=[       0   10.0000   17.0000        0
    8.3600         0  -10.0000   2.0000
    1.0000   31.0000    3.9000   4.5800 ]
```

Soluzione 2: analisi

```
H=[      0   10.0000   17.0000         0
    8.3600         0  -10.0000    2.0000
    1.0000   31.0000    3.9000    4.5800 ]
```

H<0

```
     0     0     0     0
     0     0     1     0
     0     0     0     0
```

H(H<0) = 0;

```
H=[      0   10.0000   17.0000         0
    8.3600         0         0    2.0000
    1.0000   31.0000    3.9000    4.5800 ]
```


Soluzione 2: analisi

```
H=[      0   10.0000   17.0000         0
    8.3600         0         0    2.0000
    1.0000   31.0000    3.9000    4.5800 ]
```

floor(H)

```
     0    10    17     0
     8     0     0     2
     1    31     3     4
```

floor(H) ~= H

```
     0     0     0     0
     1     0     0     0
     0     0     1     1
```

H(floor(H) ~= H)=0

```
H=[  0    10    17     0
     0     0     0     2
     1    31     0     0 ]
```

H == 0

```
     1     0     0     1
     1     1     1     0
     0     0     1     1
```

zeri = H == 0

```
zeri=[ 1     0     0     1
       1     1     1     0
       0     0     1     1 ]
```

Soluzione 2: analisi

```
H=[  0    10    17     0
     0     0     0     2
     1    31     0     0 ]
```

```
zeri=[ 1     0     0     1
       1     1     1     0
       0     0     1     1 ]
```

```
zeri*-1
```

```
   -1     0     0    -1
   -1    -1    -1     0
    0     0    -1    -1
```

```
isprime(H)
```

```
    0     0     1     0
    0     0     0     1
    0     1     0     0
```

```
pos=isprime(H)+zeri*-1;
```

```
pos = [-1    0    1   -1
       -1   -1   -1    1
        0    1   -1   -1]
```


Soluzione 3

```
function [risposta2]=leggi(nomeFile, nomeFoglio, intervallo)
    H=xlsread(nomeFile,nomeFoglio,intervallo);
    risposta2 = testa2(H);
end

function pos=testa2(H)
    H(isnan(H)) = 0;                        %0 nelle posizioni non  numeriche

    interi = (floor(H) == H) .* (H > 0);  %azzera anche valori negativi

    temp = H .* interi;                     %numeri non interi mascherati

    tp = isprime(temp);                     %verifica primalita' dei numeri interi

    pos = double(tp);                       %valori logici trasformati in double

    % nelle posizioni 0 di interi: -1
    %(nelle posizioni dei non interi -1)
    pos(interi == 0) = -1;
end
```

Confronto velocità

```
function [t0 t1 t2]=leggi(nomeFile, nomeFoglio, intervallo)
   H=xlsread(nomeFile,nomeFoglio,intervallo);
   t0=[];    t1=[];    t2=[];
   for i=1:1000
       tic
       testa(H);
       q=toc;
       t0=[t0 q];

       tic
       testa1(H);
       q=toc;
       t1=[t1 q];

       tic
       testa2(H);
       q=toc;
       t2=[t2 q];
   end
   t0=mean(t0);    t1=mean(t1);    t2=mean(t2);
end
```

```
t0 =   9.6335e-05

t1 =   7.5297e-05

t2 =   7.2685e-05
```


La successione di Collatz

- E' una successione discreta che si pensa converga sempre ad 1 per x(0)=y (y>0)
- I termini della successione sono definiti nel seguente modo:

$$x(n+1)=\begin{cases} x(n)/2 & x(n) \quad pari \\ 3\cdot x(n)+1 & x(n) \quad dispari \\ stop & x(n)=1 \end{cases}$$

```
x(0) =500

X(1) = X(0)/2=250
X(2) = X(1)/2=125
X(3)= 3*X(2) + 1 = 376
X(4) = X(3)/2=188
:
```

Successione di Collatz

– Scrivere la funzione Matlab **`Collatz`** che permette di calcolare in quante iterazioni la successione che parte dal valore x > 0, passato come argomento, converge al valore 1
  - La funzione restituisce:
    – Il numero di iterazioni necessarie
    – La successione generata

Successione di Collatz

```
function [it, succ]=Collatz(x)
    if (nargin ~= 1)
      error('numero argomenti errato')
   end

  if  (x <  0)
        error('argomento deve essere positivo')
   end

  if (floor(x) ~=x)
       error('numero deve essere intero')
  end

  succ=[x];
  while (x ~= 1)
      if (mod(x,2) == 0)
          x = x/2;
          succ = [succ x];
      else
          x = 3*x +1;
          succ = [succ x];
      end
  end
  it = length(succ);
end
```

Successione di Collatz

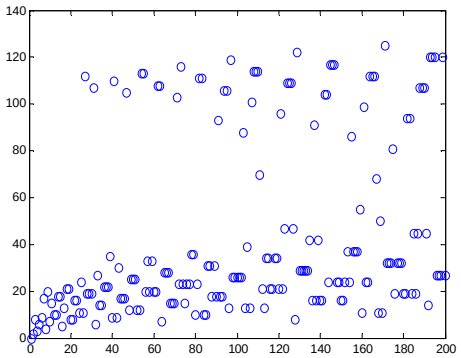
- Modifichiamo il codice precedente in modo che la funzione abbia un secondo argomento che indichi dopo quanti passi debba terminare
  - Se non specificato vale 10000

```
function [it, succ]=Collatz(x,passi)
   if (nargin ~= 1 && nargin ~= 2)
      error('numero argomenti errato')
   end

  if  (x <  0)
      error('argomento deve essere positivo')
   end

  if (floor(x) ~=x)
       error('numero deve essere intero')
  end

  if (nargin == 1)
      passi = 10000;
  end
  succ=[]; it=0;
  while (x ~= 1 &&  it <= passi)
      if (mod(x,2) == 0)
          x = x/2;
          succ = [succ x];
      else
          x = 3*x +1;
          succ = [succ x];
      end
      it=it+1;
  end
end
```


Successione di Collatz

- Il codice che segue permette di rappresentare graficamente il numero di iterazioni necessarie per raggiungere il valore 1 per una serie di valori da 2 a 200:

```
clear totIter;
for i=2:200
 totIter(i)=Collatz(i);
end
plot(totIter,'o');
```



Successione di Collatz

– Scrivere una funzione che generalizza questo comportamento e che:
  - Ha come parametro
    – Vettore con i valori iniziali del generatore della successione
    – il numero massimo di iterazioni per ogni successione
  - Restituisce
    – il numero massimo di iterazioni effettuate ed il valore per cui ciò è accaduto
    – il numero minimo di iterazioni effettuate ed il valore per cui ciò è accaduto
    – Il numero di iterazioni per le quali si è andato in overflow ed i valori per i quali ciò è accaduto

Successione di Collatz

```
function [maxIter, vtMax, ...
         minIter, vtMin, ...
         totOv vtOv] ...
      = collatzRipetuto (limite, ...
   limIter)

   iter = [];
   for i=2:limite
    iter=[iter, collatz(i,limIter)];
   end

   [vtMax, maxIter] = max(iter);
   [vtMin, minIter] = min(iter);
   ov = (iter ==  limIter);
%punti in cui le iterazioni sono uguali al limIte
   totOv = sum(ov);

 %parto da 2 e non da 1
   vtOv = find(ov) + 1;
end
```

```
function v=collatz (x, limite)
%visibile solo in questo file
%ed è la prima funzione trovata
%da collatzRipetuto

   s=[x];
   while s(length(s)) ~= 1
    if mod(s(length(s)),2) == 0
    %pari
       s =[s s(length(s))/2];
    else
       s = [s, (s(length(s))*3 + 1)];
    end
    if length(s) == limite
      break;
    end
   end
   v = length(s);
end
```

Si può richiamare così: `[a b c d e f]=collatzRipetuto(randi(1000,1,30),100)`