12 Function Handles e Ricorsione in MATLAB

Un "function handle" è una variabile il cui valore è una funzione. Data una funzione esistente, il suo function handle è dichiarato come `handle = @nome` dove `nome` è il nome di tale funzione, a patto che essa sia già definita in `MATLAB` . Una volta dichiarato l'handle, è possibile invocarlo come fosse una funzione:

```
handle_sum = @sum;

handle_sum([3 2]);
```

Un function handle può essere dichiarato anche "al volo", ossia su una funzione non esistente. Ad esempio: `handle = @(x)(x*3)` è un handle di una funzione "anonima" che accetta in ingresso un parametro formale x e svolge su di esso una moltiplicazione. Si noti che $x * 3$ è a tutti gli effetti il corpo della funzione anonima.

```
handle = @(x)(x*3);

handle(10)
```

Essendo variabili, gli handle possono essere passati ad altre funzioni creando le cosiddette "funzioni di ordine superiore", in quanto ricevono in ingresso altre funzioni (come argomenti).

```
handle_es = @(vett, func)(func(vett));

handle_es([3 2], @sum)
handle_es([3 2], handle_sum)
```

12.1 Esercizi

Esercizio 12.1

Si definisca una funzione che riceve in ingresso un function handle e due variabili x e y contenenti numeri. Tale funzione, chiamata `controlla_funzione`, dovrà controllare se applicando ad x la funzione passata tramite function handle si ottiene y, e restituire 1 o 0 a seconda che tale controllo vada a buon fine o meno.

Estendere l'esempio ai vettori, ovvero costruire `controlla_funzione_vettori` affinchè riceva in ingresso un vettore X e un vettore Y, e controlli che la condizione di cui sopra valga per tutti gli elementi corrispondenti di X e Y.

Si scriva poi una funzione `operazione` che applichi un'operazione matematica a scelta (ad esempio, la moltiplicazione per 2) a un valore di input, e la si usi per testare il funzionamento di `controlla_funzione` e `controlla_funzione_vettori`.

Esercizio 12.2

Scrivere una funzione `filtra` che riceva come parametri una funzione `cond` e due vettori x e y delle stesse dimensioni.

La funzione `cond` riceve in ingresso due parametri, a e b, e restituisce 1 (true) se a è maggiore di b, altrimenti restituisce 0 (false). La funzione di ordine superiore `filtra` dovrà utilizzare la funzione `cond` al fine di filtrare il primo vettore ricevuto in ingresso, costruendo un nuovo vettore che contenga solo quegli elementi di x che sono maggiori dei corrispondenti elementi di y.

Sperimentare con altre funzioni al posto di `cond`. Ad esempio "minore di", oppure "uguale a", oppure "uno il doppio dell'altro", etc.

Esercizio 12.3

Implementare la funzione di ordine superiore `vrand` che riceve in ingresso un function handle e un numero reale tra 0 e 1. Non si sa nulla sul function handle, se non che va chiamato senza alcun argomento (ad esempio `handle()`). La funzione `vrand` dovrà chiamare la funzione passata come handle. Se il numero restituito dalla chiamata ad `handle()` è maggiore del numero reale ricevuto come parametro, questo viene restituito immediatamente, se invece è inferiore, si dovrà continuare a chiamare la funzione `handle()` finchè questa non restituirà un numero maggiore del reale passato.

Utilizzare `vrand` per realizzare un dado a sei facce truccato. Chiamare 100 volte la `vrand` e salvare i valori così ottenuti in un vettore `V`. Realizzare poi altre 100 estrazioni usando la `rand` al posto della `vrand` e salvare tali valori in un vettore `R`.

Infine, utilizzare `plot()` per visualizzare le estrazioni di `R` e `V`:

- su due grafici diversi;
- sullo stesso grafico con colori diversi.

Esercizio 12.4

Implementare una funzione iterativa (e poi una sua versione ricorsiva) per tradurre i caratteri di una stringa da minuscoli a maiuscoli. Assumere che la funzione riceva in ingresso una stringa di caratteri minuscoli.

Suggerimento: la traduzione viene effettuata semplicemente sottraendo 32 al carattere da tradurre, e applicando `char()`. Ad esempio:

```
1 trad = char('a' - 32)
```

stampa a video il carattere `A`.

Esercizio 12.5

Si implementi in MATLAB uno script che, preso un vettore v, lo ordini ricorsivamente utilizzando l'algoritmo **Mergesort**, definito come segue:

- Se la sequenza da ordinare ha lunghezza 1, è già ordinata;
- Se la sequenze è lunga 2 o più, la sequenza viene divisa in due metà;
- Ognuna di queste sotto-sequenze viene ordinata, applicando ricorsivamente l'algoritmo;
- Le due sotto-sequenze ordinate vengono fuse. Per fare questo, si estrae ripetutamente il minimo delle due sotto-sequenze e lo si pone nella sequenza in uscita, che risulterà ordinata.

Esercizio 12.6

(TdE 5 febbraio 2015) Data una matrice quadrata di dimensione N, definiamo *somma del quadrato concentrico di ordine* k la somma degli elementi che si trovano sul k-simo quadrato concentrico della matrice ($k \leq \text{ceil}(N/2)$). Per esempio nella matrice:

$$\begin{bmatrix} 1 & 2 & 3 & 2 & 5 \\ 0 & 2 & 3 & 7 & 3 \\ 3 & 2 & 2 & 5 & 4 \\ 1 & 8 & 4 & 6 & 6 \\ 5 & 4 & 9 & 1 & 2 \end{bmatrix}$$

il quadrato concentrico di ordine $k = 2$ è definito dagli elementi

$$\begin{bmatrix} 2 & 3 & 7 \\ 2 & \mathtt{x} & 5 \\ 8 & 4 & 6 \end{bmatrix}$$

(dove `x` sottolinea il fatto che l'elemento non è da considerare parte del quadrato concentrico) e la somma del quadrato concentrico è pari a 37, mentre il quadrato concentrico di ordine 3 è definito dall'elemento

$$\begin{bmatrix} 2 \end{bmatrix}$$

(e la somma del quadrato concentrico è pari a 2).

Si sviluppi in MATLAB una funzione *ricorsiva* `quadratiConcentrici` che, data una matrice quadrata, restituisca il vettore contenente le somma dei quadrati concentrici che la compongono. Per esempio, per la matrice di esempio avremo il vettore deve contenere: `[51 37 2]`.

Per sviluppare questa funzione si scriva prima la funzione `sommaDiCorniceEsterna` che, data una matrice quadrata di ordine N, restituisce la somma del quadrato concentrico di ordine 1, cioè la somma degli elementi sulla riga 1, riga N, colonna 1, colonna N (senza contare due volte gli elementi ai vertici della matrice). Nel caso in cui la matrice non sia quadrata, la funzione restituisce 0. La funzione applicata all'esempio restituirebbe 51.

Infine, si mostri in uno script un esempio di chiamata della funzione `quadratiConcentrici` sulla matrice di esempio.

Esercizio 12.7

(TdE 28 gennaio 2013)

Scrivere la funzione **ricorsiva** `cifra` che riceve come parametri due numeri interi `num` e `k` (si supponga che entrambi i numeri siano sempre strettamente positivi). La funzione cifra restituisce la `k`-esima cifra del numero `num` a partire da destra. Per esempio:

- `cifra(1456, 1)` deve restituire 6;
- `cifra(5136, 4)` deve restituire 5;
- `cifra(512, 2)` deve restituire 1.

Suggerimento: un numero intero positivo di n cifre, $c_n, \ldots, c_1$ può essere rappresentato in termini di potenze di 10 come segue: $c_n 10^{n-1} + c_{n-1} 10^{n-2} + \ldots + c_2 10 + c_1$. Per esempio $1456 = 1 \cdot 10^3 + 4 \cdot 10^2 + 5 \cdot 10 + 6$.

Scrivere la funzione ricorsiva `cifraConControllo` che, in modo analogo alla funzione precedente, riceve come parametri due numeri interi `num` e `k` (si supponga che entrambi i numeri siano sempre strettamente positivi). La funzione `cifraConControllo` restituisce la `k`-esima cifra del numero `num` a partire da destra. Nel caso in cui `k` sia maggiore del numero effettivo di cifre che compongono `num`, la funzione restituisce -1. Per esempio:

- `cifraConControllo(512, 2)` deve restituire 1;
- `cifraConControllo (123, 5)` deve restituire -1 (perché in questo caso `num` è composto da sole 3 cifre).

Nota: Non è consentito l'uso della funzione `num2str`.

Esercizio 12.8

(TdE 20 febbraio 2012) Si assuma di avere una matrice di 2 righe ed un numero arbitrario di colonne, ad esempio:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \tag{12.1}$$

Ogni colonna della matrice rappresenta una frazione, ae esempio la matrice precedente rappresenta:

$$\begin{bmatrix} \frac{1}{5} & \frac{2}{6} & \frac{3}{7} & \frac{4}{8} \end{bmatrix}$$

Si scriva una funzione ricorsiva `frac` che, ricevuta una qualsiasi matrice in ingresso, restituisca i parametri `n` e `d` che rappresentano rispettivamente il numeratore ed il denominatore della frazione risultante dalla somma delle frazioni contenute nella matrice. Ad esempio:

```
1 [n d] = frac([1 3 1; 2 4 3])
```

restituisce `n = 19` e `d = 12`. La frazione deve essere minimizzata, ovvero non vi devono essere divisori comuni tra `n` e `d`. Nel progettare frac, si può utilizzare la funzione `gcd(a,b)` che restituisce il massimo comun divisore tra `a` e `b` per minimizzare una frazione in quanto:

$$\frac{a}{b} = \frac{\frac{a}{gcd(a,b)}}{\frac{b}{gcd(a,b)}}$$

Soluzioni

Soluzione dell'esercizio 12.1

```m
clear;
clc;
close all;

func = @operazione;

x = rand();

%Controllo singolo numero
controlla_funzione(func, x, 2*x) % vero
controlla_funzione(func, x, x+3) % falso

X = rand([10 1]);
Y = 2 * X;

%Controllo su vettore
controlla_funzione_vettori(func, X, Y) % vero
Y(3) = Y(3) - 1;
controlla_funzione_vettori(func, X, Y) % falso
```

```m
function ok = controlla_funzione(handle, x, y)

ok = (y == handle(x));
```

```m
function ok = controlla_funzione_vettori(func, X, Y)

ok = true;
for ii = 1:length(X)
    if ~controlla_funzione(func, X(ii), Y(ii))
        ok = false;
        return;
    end
end
```

```m
function risultato = operazione(input)

risultato = 2 * input;
```

Soluzione dell'esercizio 12.2

```
clc
clear
close all

%Riempimento del vettore in modo casuale
x = randi(10, 1, 10)
y = randi(10, 1, 10)

% Filtraggio del vettore
x_mod = filtra(@cond, x, y)
x_mod = filtra(@(a,b)(a > b), x, y) % equivalente
```

```
function x_new = filtra(condizione, x, y)

idx = condizione(x,y);
x_new = x(idx);
```

```
function ris = cond(a, b)

ris = (a > b);
% ris = (a < b);
% ris = (a == b);
% ris = (a == 2 * b);
```

Soluzione dell'esercizio 12.3

```
clear
clc
close all;

%Estrazioni da dado truccato e regolare
for ii = 1:100
    V(ii) = ceil(6 * vrand(@rand, 0.5));
    R(ii) = ceil(6 * rand());
end

%Plot su due grafici diversi
figure(1);
plot(V, 'o');
title('Dado truccato');
ylim([1 6])
```

```
yticks(1:6)
grid on

figure(2);
plot(R, 'o');
ylim([1 6])
yticks(1:6)
grid on
title('Dado regolare');

%Plot sullo stesso grafico con colori diversi
figure(3);
plot(V, 'ro');
hold on;
plot(R, 'o');
ylim([1 6])
yticks(1:6)
grid on
legend({'Dado truccato' 'Dado regolare'});
xlabel('Numero estrazione');
ylabel('Valore estrazione');
```

```
function ris = vrand(r, m)

ris = r();
while ris < m
    ris = r();
end
```

Soluzione dell'esercizio 12.4

```
clc
clear
close all

s = 'ciaocomestai';

fprintf('
   ------------------------------------------------------\n');
tic
S = maiuscola_iterativa(s);
disp(S)
toc
```

```
12
13 fprintf('
      ------------------------------------------------------------\n');
14 tic
15 S = maiuscola_ricorsiva1(s);
16 disp(S);
17 toc
18
19 fprintf('
      ------------------------------------------------------------\n');
20 tic
21 S = maiuscola_ricorsiva2(s);
22 disp(S);
23 toc
```

```
1 function S = maiuscola_iterativa(s)
2
3 S = s;
4 for ii = 1:length(s)
5     S(ii) = char(s(ii) - 32);
6 end
```

```
1 function S = maiuscola_ricorsiva1(s)
2
3 % Caso base: stringa di un carattere
4 if length(s) == 1
5     S = [char(s(1) - 32)];
6 else
7     S = [char(s(1) - 32) maiuscola_ricorsiva1(s(2:end))];
8 end
```

```
1 function S = maiuscola_ricorsiva2(s)
2
3 % Caso base: stringa di un carattere
4 if length(s) == 1
5     S = char(s(1) - 32);
6 else
7     m = round(length(s) / 2);
8     S = [maiuscola_ricorsiva2(s(1:m)) maiuscola_ricorsiva2(s((m
          +1):end))];
9 end
```

Soluzione dell'esercizio 12.5

```
clear
clc
close all

v = rand(100, 1);
plot(v, 'o');
hold on;
sorted_v = merge_sort(v);
plot(sorted_v,'ro');
```

```
function sorted_v = merge_sort(v)

len_v = length(v);
if  len_v > 1
    %Divide
    middle_point = round(len_v / 2);
    v1 = merge_sort(v(1:middle_point));
    v2 = merge_sort(v(middle_point+1:end));

    %Impera
    len_v1 = length(v1);
    len_v2 = length(v2);
    ii = 1;
    jj = 1;
    sorted_v = [];
    while ii <= len_v1 && jj <= len_v2
        if v1(ii) < v2(jj)
            sorted_v = [sorted_v v1(ii)];
            ii = ii + 1;
        else
            sorted_v = [sorted_v v2(jj)];
            jj = jj + 1;
        end
    end
    if ii <= len_v1
        sorted_v = [sorted_v v1(ii:end)];
    else
        sorted_v = [sorted_v v2(jj:end)];
    end
    assert(length(sorted_v) == len_v1 + len_v2)
else
    sorted_v = v;
```

```
33 end
```

Soluzione dell'esercizio 12.6

```
1  clear;
2  clc;
3  close all;
4
5  M = [1 2 3 2 5; ...
6       0 2 3 7 3; ...
7           3 2 2 5 4; ...
8           1 8 4 6 6; ...
9           5 4 9 1 2 ];
10
11  quadratiConcentrici(M)
```

```
1  function ris = sommaDiCorniceEsterna(m)
2
3  [r, s] = size(m);
4  if (r == 0 || r ~= s)
5      ris = 0;
6  else
7      v = [m(1, :) m(2:end, 1)' m(end, 2:end) m(2:end-1, end)'];
8      ris = sum(v);
9  end
```

```
1  function vetSomme = quadratiConcentrici(m)
2
3  [r, c] = size(m);
4  assert(r == c);
5
6  if (r == 0)
7      vetSomme = [];
8  else
9      s = sommaDiCorniceEsterna(m);
10     vetSomme = [s quadratiConcentrici(m(2:end-1, 2:end-1))];
11 end
```

Soluzione dell'esercizio 12.7

```
1  clear
2  clc
```

```
close all

cifra(645, 2)

cifraConControllo(112, 5)
```

```
function ris = cifra(num, k)

if k == 1
    ris = mod(num, 10);
else
    ris = cifra(floor(num/10), k-1);
end
```

```
function ris = cifraConControllo(num, k)

if (k > 1 && num < 10)
    ris = -1;
else
    if k == 1
        ris = mod(num, 10);
    else
        ris = cifraConControllo(floor(num/10), k-1);
    end
end
```

Soluzione dell'esercizio 12.8

```
clear
clc
close all

M = [1 2 3 4; 5 6 7 8];

[n, d] = frac(M)
```

```
function [n, d] = frac(M)

if size(M,2) == 1
    n = M(1,1);
    d = M(2,1);
else
    [n_tot, d_tot] = frac(M(:,2:end));
```

```
8
9       n1 = M(1,1);
10      d1 = M(2,1);
11
12      n = n1 * d_tot + n_tot * d1;
13      d = d1 * d_tot;
14
15      mcm = gcd(n,d);
16      n = n / mcm;
17      d = d / mcm;
18  end
```