

Laboratorio 1 – Introduzione a Scilab

Scilab è un software scientifico per il calcolo numerico che fornisce un ambiente di calcolo per varie applicazioni ingegneristiche e scientifiche. Sviluppato inizialmente dai ricercatori dell'INRIA e dell'ENPC, è attualmente curato dal Consorzio Scilab e distribuito come software libero e open-source. Maggiori informazioni possono essere reperite all'indirizzo www.scilab.org Sullo stesso sito è inoltre possibile reperire l'ultima versione stabile (e alcune versioni precedenti) e manuali per l'utilizzo di tale software.

Prima di passare all'introduzione di alcuni comandi elementari che verranno utilizzati nei successivi laboratori, ricordiamo che per eventuali dubbi sull'uso di una funzione è possibile usare l'help, digitando il comando `help nome_comando`.

Assegnazione di scalari. Cominciamo con l'assegnare il valore 2.45 alla variabile a:

```
-->a = 2.45
a =

    2.45
```

Assegnamo ora il valore 3.1 alla variabile A. Osserviamo che Scilab fa distinzione tra le lettere maiuscole e le lettere minuscole.

```
-->A=3.1
A =

    3.1
```

Le variabili sono sovrascrivibili, cioè se ora assegnamo ad A un nuovo valore:

```
-->A=7.2
A =

    7.2
```

il precedente valore 3.1 viene definitivamente perso.

Osserviamo che possiamo far seguire un comando da una virgola, senza avere nessuna differenza. Tale virgola è però necessaria per separare più comandi scritti sulla stessa linea.

```
-->a=1.2,
a =

    1.2
```

```
-->a=1.7, a=2.45
a =

    1.7
a =

    2.45
```

Se invece si fa seguire il comando da un punto e virgola, Scilab non stamperà sulla command window il risultato dell'operazione. Il punto e virgola può essere usato per separare due comandi sulla stessa riga.

```
-->a=1.2;
```

```
-->a=1.7; a=2.45
```

```
a =
```

```
2.45
```

Le variabili possono essere cancellate utilizzando il comando **clear**. Possiamo ad esempio cancellare la variabile A digitando:

```
-->clear A
```

Digitando il comando:

```
-->clear
```

vengono cancellate tutte le variabili definite dall'utente. Scilab considera tra le variabili anche tutte le funzioni richiamate, direttamente o indirettamente (cioè richiamate da altre funzioni utilizzate), dall'utente, siano esse definite dall'utente o già presenti in Scilab (per sapere come definire e richiamare una funzione si veda nel seguito di quest'introduzione).

Per sapere quali sono le variabili (incluse le funzioni) dell'utente attualmente in memoria si utilizza il comando:

```
-->who_user
```

```
User variables are:
```

```
a          %scipad_fontsize  show_startupinfo  LCC
%toolboxes_dir  %toolboxes          modelica_libs      with_gtk
with_tk
```

```
using 1004 elements out of 984819
```

Per sapere quali sono le variabili (comprese le funzioni) in memoria definite dall'utente e/o da Scilab (come le variabili permanenti definite nel seguito) è possibile usare i comandi **who** o **whos**. Quest'ultimo, a differenza dei precedenti, mostra anche il tipo, la dimensione e l'occupazione di memoria in numero di bytes.

Variabili permanenti. Alcune variabili, proprie di Scilab, non necessitano di alcuna definizione e non possono essere modificate. Tra queste ricordiamo:

- %pi: il numero 3.1415927...;
- %i: l'unità immaginaria $\sqrt{-1}$.
- %eps: il minimo valore tale che, per il calcolatore, sia $(1 + \%eps) > 1$, pari a (sul calcolatore che stiamo usando):

```
-->%eps
%eps =
```

```
2.220D-16
```

Quindi due numeri che differiscono per meno di %eps per il calcolatore sono uguali. Infatti:

```
-->5+%eps/2-5
ans =
```

```
0.
```

- %e = 2.7182818... il numero di Nepero.

Istruzione format. Serve per modificare il formato di visualizzazione dei risultati ma NON la precisione con cui i calcoli vengono condotti. Il comando ha la seguente sintassi: `format(type,long)`, dove `type` è una stringa che può assumere i valori 'v', per visualizzare in formato floating point, o 'e', per la notazione esponenziale, e `long` è il numero *massimo* di cifre (10 per default). Si osservi che, nel caso 'v' almeno tre cifre vengono lasciate a sinistra della virgola. Se si omette uno dei due parametri in input, questo viene mantenuto nello "stato" in cui attualmente si trova. Otteniamo così ad esempio:

```
-->format(3)
```

```
-->x=0.1, y=1000
```

```
x =
```

```
0.
```

```
y =
```

```
***
```

```
-->format('e',3)
```

```
-->x,y
```

```
x =
```

```
1.0D-01
```

```
y =
```

```
1.0D+03
```

```
-->format
```

```
ans =
```

```
! 0.0D+00 8.0D+00 !
```

```
-->format('v',10)
```

```
-->x,y
```

```
x =
```

```
0.1
```

```
y =
```

```
1000.
```

```
-->format()
```

```
ans =
```

```
! 1. 10. !
```

Il comando `format` senza alcun parametro, rende un vettore di due componenti: il primo elemento è 0 per il formato 'e' e 1 per il formato 'v' mentre il secondo elemento indica il numero di cifre.

È possibile che i numeri visualizzati, nel formato esponenziale, abbiano E (se il programma è usato sotto Linux) al posto di D (sotto Microsoft Windows).

Assegnamento di vettori. Riportiamo alcuni modi per l'inserimento di vettori.

Vettori riga:

```
-->b=[1 2 3 4]
```

```
b =
```

```
! 1. 2. 3. 4. !
```

```
-->b1 = [1,2,3, 4]
```

```
b1 =
```

```
! 1. 2. 3. 4. !
```

```
-->b2 = [1:1:4]
```

```
b2 =
```

```
! 1. 2. 3. 4. !
```

```
-->b3 = [1:4]
```

```
b3 =
```

```
! 1. 2. 3. 4. !
```

```
-->d = [1:0.1:2]
```

```
d =
```

```
! 1. 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. !
```

```
-->c = [0:-0.3:-1]
```

```
c =
```

```
! 0. - 0.3 - 0.6 - 0.9 !
```

Vettori colonna:

```
-->e = [1; 2; 3; 4]
```

```
e =
```

```
! 1. !
```

```
! 2. !
```

```
! 3. !
```

```
! 4. !
```

Operazioni su vettori.

- Vettore trasposto:

```
-->a=[1:4]
```

```
a =
```

```
! 1. 2. 3. 4. !
```

```
-->b=a'
```

```
b =
```

```
! 1. !
```

```
! 2. !
```

```
! 3. !
```

```
! 4. !
```

- Somma e sottrazione (attenzione alla compatibilità delle dimensioni dei vettori!):

```
-->c=[2 5 4 1]
```

```
c =
```

```
! 2. 5. 4. 1. !
```

```
-->a+c
```

```
ans =
```

```
! 3. 7. 7. 5. !
```

```
-->a-c
```

```
ans =
```

```
! - 1. - 3. - 1. 3. !
```

```
-->d=[1 2 3]
```

```
d =
```

```
! 1. 2. 3. !
```

```
-->a+d
```

```
!--error 8  
inconsistent addition
```

```
-->c=c'
```

```
c =
```

```
! 2. !
```

```
! 5. !
```

```
! 4. !
```

```
! 1. !
```

```
-->a-c
```

```
!--error 9  
inconsistent subtraction
```

- Prodotto di vettori: prodotto (righe per colonne) vettore riga - vettore colonna (della stessa lunghezza)

```
-->a*b
```

```
ans =
```

```
30.
```

prodotto vettore colonna per vettore riga (della stessa lunghezza)

```
-->b*a
```

```
ans =
```

```
! 1. 2. 3. 4. !
```

```
! 2. 4. 6. 8. !
```

```
! 3. 6. 9. 12. !
```

```
! 4. 8. 12. 16. !
```

prodotto elemento per elemento (attenzione alla compatibilità delle dimensioni!):

```
-->b.*c
```

```
ans =
```

```
! 2. !  
! 10. !  
! 12. !  
! 4. !
```

```
-->a.*b  
!--error 9999  
inconsistent element-wise operation
```

- Elevamento a potenza di ogni componente:

```
-->a.^2  
ans =  
  
! 1. 4. 9. 16. !
```

Funzioni intrinseche per vettori.

```
-->v1=[200;300;100];
```

```
-->v2=[200,300,100];
```

- **size**: restituisce la dimensione del vettore passato in input, restituisce cioè un vettore contenente due interi, il primo indicante il numero di righe (1 se il vettore è riga) e il secondo indicante il numero di colonne (1 se il vettore è colonna) del vettore passato in input.

```
-->size(v1)  
ans =  
  
! 3. 1. !
```

```
-->size(v2)  
ans =  
  
! 1. 3. !
```

```
-->size(v1,1)  
ans =  
  
3.
```

```
-->size(v1,2)  
ans =  
  
1.
```

- `length`: restituisce la lunghezza del vettore passato in input.

```
-->length(v1)
ans =

    3.
```

```
-->length(v2)
ans =

    3.
```

`max`, `min`: calcolano il massimo e il minimo valore delle componenti del vettore passato in input.

```
-->max(v1)
ans =

    300.
```

```
-->min(v1)
ans =

    100.
```

- `sum`, `prod`: calcolano la somma e il prodotto degli elementi del vettore passato in input.

```
-->sum(v1)
ans =

    600.
```

```
-->prod(v1)
ans =

    6000000.
```

- `norm`: ha due parametri in input: un vettore `v` e un numero intero `n` o la stringa `"inf"`. Se viene passato solo il vettore, calcola la norma 2 (*norma euclidea*) di `v`, definita come $\|v\|_2 = \sqrt{\sum_{i=1}^{\text{length}(v)} v_i^2}$. Se viene passato un numero intero `n`, calcola la norma `n` di `v` definita come $\|v\|_n = \left(\sum_{i=1}^{\text{length}(v)} |v_i|^n\right)^{\frac{1}{n}}$. Infine se viene passata la stringa `"inf"`, calcola la norma infinito di `v` definita come $\|v\|_\infty = \max_{1 \leq i \leq \text{length}(v)} |v_i|$.

```
-->norm(v1)
ans =
```

```
374.16574
```

```
-->norm(v1,2)  
ans =
```

```
374.16574
```

```
-->norm(v1,1)  
ans =
```

```
600.
```

```
-->norm(v1,'inf')  
ans =
```

```
300.
```

- **abs**: restituisce un vettore avente come componenti il valore assoluto di ciascuna componente del vettore passato in input.

```
-->w=[1 -4 5 -9]  
w =
```

```
! 1. - 4. 5. - 9. !
```

```
-->abs(w)  
ans =
```

```
! 1. 4. 5. 9. !
```

Funzioni matematiche elementari.

- **sqrt**: rende la radice quadrata degli elementi dello scalare, vettore o matrice passato in input.

```
-->sqrt(4)  
ans =
```

```
2.
```

```
-->sqrt([4, -1])  
ans =
```

```
! 2. i !
```

```
-->sqrt([4 -1; 2 -2])  
ans =
```

```
! 2.          i          !
! 1.4142136   1.4142136i !
```

- **exp(x)**: rende e^x . Se x è un vettore o una matrice esegue l'operazione per ogni componente.

```
-->exp(1)
ans =
```

```
2.7182818
```

```
-->exp([1;0])
ans =
```

```
! 2.7182818 !
! 1.          !
```

```
-->exp([0,1; 2 -1])
ans =
```

```
! 1.          2.7182818 !
! 7.3890561   0.3678794 !
```

- **log(x)**: rende il logaritmo in base e (*logaritmo naturale*) di x , $\ln(x)$. Se x è una matrice o un vettore esegue l'operazione su ogni elemento.

```
-->log(%e)
ans =
```

```
1.
```

```
-->log([1 %e])
ans =
```

```
! 0.    1. !
```

```
-->log([1 %e; 2 1])
ans =
```

```
! 0.          1. !
! 0.6931472   0. !
```

- **log10(x)**: rende il logaritmo in base 10 di x , $\text{Log}(x)$. Se x è una matrice o un vettore esegue l'operazione su ogni elemento.

```
-->log10(10)
```

```

ans =

    1.

-->log10([1 10])
ans =

!   0.    1. !

-->log10([%e 1; 10 2])
ans =

!   0.4342945    0.    !
!   1.          0.30103 !

```

- **log₂(x)**: rende il logaritmo in base 2 di x, $\log_2(x)$. Se x è una matrice o un vettore esegue l'operazione su ogni elemento.

```

-->log2(2)
ans =

    1.

-->log2([1 2])
ans =

!   0.    1. !

-->log2([1 2; %e 10])
ans =

!   0.          1.          !
!   1.442695    3.3219281 !

```

- **sin(x)**: rende il seno di x, dove x è espresso in *radianti*. Se x è un vettore o una matrice esegue l'operazione elemento per elemento.

```

-->sin(%pi/2)
ans =

    1.

-->sin([%pi/2,0])
ans =

!   1.    0. !

```

```
-->sin([%pi/2,0; %pi, %pi/4])
ans =

! 1.          0.          !
! 1.225D-16   0.7071068 !
```

- $\cos(x)$: rende il coseno di x , dove x è espresso in *radianti*. Se x è un vettore o una matrice esegue l'operazione elemento per elemento.

```
-->cos(%pi/2)
ans =

6.123D-17
```

```
-->cos([%pi/2,0])
ans =
```

```
! 6.123D-17   1. !
```

```
-->cos([%pi/2,0; %pi, %pi/4])
ans =
```

```
! 6.123D-17   1.          !
! - 1.         0.7071068 !
```

- $\tan(x)$: rende la tangente di x , dove x è espresso in *radianti*. Se x è un vettore o una matrice esegue l'operazione elemento per elemento.

```
-->tan(%pi)
ans =

- 1.225D-16
```

```
-->tan([%pi;0])
ans =
```

```
1.0D-15 *

! - 0.1224606 !
! 0.          !
```

```
-->tan([%pi,0;%pi/4,%pi/2])
ans =
```

```
! - 1.225D-16   0.          !
! 1.            1.633D+16 !
```

- `asin(x)`: rende l'arcoseno di x , per $x \in [-1, 1]$. Se x è un vettore o una matrice esegue l'operazione elemento per elemento.

```
-->asin(1)
ans =

    1.5707963

-->asin([1,0.5])
ans =

!   1.5707963    0.5235988 !

-->asin([1,0.5;0,1])
ans =

!   1.5707963    0.5235988 !
!   0.          1.5707963 !
```

- `acos(x)`: rende l'arcocoseno di x , per $x \in [-1, 1]$. Se x è un vettore o una matrice esegue l'operazione elemento per elemento.

```
-->acos(0)
ans =

    1.5707963

-->acos([0, 1])
ans =

!   1.5707963    0. !

-->acos([0,1; 0.5,1])
ans =

!   1.5707963    0. !
!   1.0471976    0. !
```

Funzioni per definire vettori o matrici particolari.

- Vettore o matrice nulla: `zeros(n,m)`: rende una matrice $n \times m$ di elementi pari a zero.

```
-->zeros(1,3)
ans =

!   0.    0.    0. !

-->zeros(3,1)
```

```

ans =

!  0. !
!  0. !
!  0. !

-->zeros(3,3)
ans =

!  0.  0.  0. !
!  0.  0.  0. !
!  0.  0.  0. !

```

`zeros(A)`, dove `A` è una matrice, rende una matrice della stessa dimensione di `A` con elementi tutti nulli.

```

-->zeros(3)
ans =

0.

-->zeros([-1; 2; -3])
ans =

!  0. !
!  0. !
!  0. !

-->zeros([1 3])
ans =

!  0.  0. !

-->zeros([1,1;-1, 6])
ans =

!  0.  0. !
!  0.  0. !

```

- Matrice di 1: `ones(n,m)` : rende una matrice `n x m` di elementi pari a uno.

```

-->ones(1,3)
ans =

!  1.  1.  1. !

-->ones(3,1)

```

```
ans =  
  
! 1. !  
! 1. !  
! 1. !
```

```
-->ones(3,3)  
ans =
```

```
! 1. 1. 1. !  
! 1. 1. 1. !  
! 1. 1. 1. !
```

`ones(A)`, dove A è una matrice, rende una matrice della stessa dimensione di A con elementi tutti pari a uno.

```
-->ones(3)  
ans =
```

```
1.
```

```
-->ones([-1; 2; -3])  
ans =
```

```
! 1. !  
! 1. !  
! 1. !
```

```
-->ones([1 3])  
ans =
```

```
! 1. 1. !
```

```
-->ones([1,1;-1, 6])  
ans =
```

```
! 1. 1. !  
! 1. 1. !
```

- Matrice identità: `eye(n,n)` rende la matrice identità di dimensione n .

```
-->eye(3,3)  
ans =
```

```
! 1. 0. 0. !  
! 0. 1. 0. !  
! 0. 0. 1. !
```

`eye(n,m)` rende una matrice $n \times m$ con 1 sulla diagonale principale e 0 altrove.

```
-->eye(2,3)
```

```
ans =
```

```
! 1. 0. 0. !  
! 0. 1. 0. !
```

```
-->eye(3,2)
```

```
ans =
```

```
! 1. 0. !  
! 0. 1. !  
! 0. 0. !
```

`eye(A)`, dove A è una matrice, rende una matrice della stessa dimensione di A con elementi pari a uno sulla diagonale principale e zero altrove.

```
-->eye(3)
```

```
ans =
```

```
1.
```

```
-->eye([1 2 ; -1 2])
```

```
ans =
```

```
! 1. 0. !  
! 0. 1. !
```

```
-->eye([2 3])
```

```
ans =
```

```
! 1. 0. !
```

- Matrice di numeri casuali: `rand(n,m)` rende una matrice $n \times m$ di numeri casuali nell'intervallo $[0,1]$.

```
-->rand(1,3)
```

```
ans =
```

```
! 0.2113249 0.7560439 0.0002211 !
```

```
-->rand(3,1)
```

```
ans =
```

```
! 0.3303271 !  
! 0.6653811 !
```

```
! 0.6283918 !
```

```
-->rand(3,3)
```

```
ans =
```

```
! 0.8497452 0.0683740 0.7263507 !
```

```
! 0.6857310 0.5608486 0.1985144 !
```

```
! 0.8782165 0.6623569 0.5442573 !
```

`rand(A)`, dove A è una matrice, rende una matrice della stessa dimensione di A di numeri casuali nell'intervallo $[0,1]$.

```
-->rand(3)
```

```
ans =
```

```
0.2320748
```

```
-->rand(1,3)
```

```
ans =
```

```
! 0.2312237 0.2164633 0.8833888 !
```

```
-->rand(2,2)
```

```
ans =
```

```
! 0.6525135 0.9329616 !
```

```
! 0.3076091 0.2146008 !
```

`rand()` rende un numero casuale nell'intervallo $[0,1]$, diverso ogni volta che il comando viene invocato.

```
-->rand()
```

```
ans =
```

```
0.3616361
```

```
-->rand()
```

```
ans =
```

```
0.2922267
```

Introduciamo le matrici in Scilab risolvendo un semplice esercizio.

Esercizio 1: quadrato magico di ordine 4.

$$A = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$$

a) Inserire la matrice.

I Modo)

```
-->A = [16 2 3 13; 5 11 10 8; 9 7 6 12; 4 14 15 1]
```

```
A =
```

```
! 16.    2.    3.    13. !
!  5.    11.   10.    8.  !
!  9.     7.    6.   12. !
!  4.    14.   15.    1.  !
```

II Modo)

```
-->A=[16 2 3 13
```

```
-->5 11 10 8
```

```
-->9 7 6 12
```

```
-->4 14 15 1]
```

```
A =
```

```
! 16.    2.    3.    13. !
!  5.    11.   10.    8.  !
!  9.     7.    6.   12. !
!  4.    14.   15.    1.  !
```

III Modo) Usare il comando `testmatrix('magi',4)` che rende la matrice magica di ordine 4.

```
-->A=testmatrix('magi',4)
```

```
A =
```

```
! 16.    2.    3.    13. !
!  5.    11.   10.    8.  !
!  9.     7.    6.   12. !
!  4.    14.   15.    1.  !
```

b) Sommare per colonne.

I Modo)

```
-->sum(A,1)
ans =
```

```
! 34. 34. 34. 34. !
```

II Modo)

```
-->sum(A,'r')
ans =
```

```
! 34. 34. 34. 34. !
```

c) Sommare per righe.

I Modo)

```
-->sum(A,2)
ans =
```

```
! 34. !
! 34. !
! 34. !
! 34. !
```

II Modo)

```
-->sum(A,'c')
ans =
```

```
! 34. !
! 34. !
! 34. !
! 34. !
```

III Modo)

```
-->A=A'
A =
```

```
! 16. 5. 9. 4. !
! 2. 11. 7. 14. !
! 3. 10. 6. 15. !
! 13. 8. 12. 1. !
```

```
-->sum(A,1)
ans =
```

```
! 34. 34. 34. 34. !
```

Il comando `A'` rende la matrice trasposta della matrice `A`. Osserviamo che il trasposto di un vettore riga è un vettore colonna e viceversa.

Osserviamo che il comando `sum(A)` rende la somma di tutti gli elementi di `A`.

d) Sommare gli elementi della diagonale principale.

I Modo) Scriviamo la somma degli elementi richiesti. Con il comando `A(n,m)` otteniamo il valore dell'elemento di `A` posizionato sull'`n`-esima riga e sull'`m`-esima colonna.

```
-->A(1,1)+A(2,2)+A(3,3)+A(4,4)
ans =
```

34.

oppure, usando il comando `sum` visto per i vettori:

```
-->sum([A(1,1) A(2,2), A(3,3) A(4,4)])
ans =
```

34.

II Modo) Cominciamo con l'estrarre la diagonale principale:

```
-->a=diag(A)
a =
```

```
! 16. !
! 11. !
! 6.  !
! 1.  !
```

```
-->a=diag(A,0)
a =
```

```
! 16. !
! 11. !
! 6.  !
! 1.  !
```

Il comando `diag(M,k)`, dove `M` è una matrice, rende un vettore colonna avente per elementi gli elementi della diagonale `k`-esima di `M`, dove `k = 0` indica la diagonale principale, `k > 0` indica la `k`-esima sopradiagonale mentre `k < 0` indica la `k`-esima sottodiagonale. Se `k` viene omesso, si suppone `k=0`.

Osserviamo infine che il comando `diag(v,k)`, dove `v` è un vettore, rende una matrice di dimensione $(\text{length}(v)+k) \times (\text{length}(v)+k)$ con gli elementi di `v` sulla diagonale `k`-esima e gli altri elementi nulli.

Una volta estratta la sottodiagonale, possiamo usare il comando `sum` visto nel caso dei vettori.

```
-->sum(a)
ans =
```

34.

L'operazione poteva essere fatta in una sola riga scrivendo:

```
-->sum(diag(A))
ans =
```

34.

e) Sommare gli elementi dell'antidiagonale (elementi nei riquadri).

I Modo) Procedendo in maniera analoga al I Modo del punto precedente possiamo scrivere:

```
-->A(1,4)+A(2,3)+A(3,2)+A(4,1)
ans =
```

34.

oppure

```
-->sum([A(1,4),A(2,3),A(3,2),A(4,1)])
ans =
```

34.

II Modo) Non esiste un comando per estrarre l'antidiagonale. Possiamo, utilizzando un ciclo `for`, estrarre la sottodiagonale (difficilmente potremmo usare il primo modo ad esempio con una matrice $50 \times 50!$)

```
-->n=size(A,1)
n =
```

4.

```
-->for i=1:n
-->ad(i) = A(i,n-i+1);
-->end
```

```

-->ad
ad =

! 13. !
! 10. !
! 7. !
! 4. !

-->sum(ad)
ans =

34.

```

oppure, effettuando direttamente la somma:

```

-->s=0;

-->for i=1:n
-->s = s+ A(i,n-i+1);
-->end

-->s
s =

34.

```

Al posto di eseguire i comandi direttamente da linea di comando, possiamo scriverli su un file .sce, salvarli e successivamente eseguirli. A questo scopo apriamo scipad utilizzando il comando `scipad()`; o cliccando sulla seconda icona in alto a sinistra (l'icona contenente un foglio bianco) e riportiamo i comandi usati.

```

n=size(A,1)
for i=1:n
    ad(i) = A(i,n-i+1);
end
ad
sum(ad)

```

Salviamo il file modificato, cliccando sul menù file e successivamente su save o save as, come (ad esempio) `antidiag.sce` (l'estensione `sce` indica un file eseguibile in scilab). Eseguiamo il file (e quindi i comandi salvati) digitando: `exec antidiag.sce` se il file si trova nella cartella in cui stiamo lavorando oppure `exec ...\antidiag.sce`, dove `...` indicano il percorso dalla cartella dove stiamo lavorando alla cartella in cui il file è salvato. Possiamo cambiare la cartella di lavoro selezionando Change Directory dal menù File.

```

-->exec antidiag.sce

-->n=size(A,1)
n =

    4.

-->for i=1:n
-->    ad(i) = A(i,n-i+1);
-->end

-->ad
ad =

!  13. !
!  10. !
!   7. !
!   4. !

-->sum(ad)
ans =

    34.

```

III Modo) Scriviamo la funzione `adiag` che presa in input una matrice M , restituisce in output il vettore v degli elementi sull'antidiagonale di A . A questo scopo apriamo una nuova finestra nell'editor `scipad` selezionando `New` dal menù `File`. Scriviamo quindi la seguente funzione:

```

function ris=adiag(a)
// Funzione che estrae; l'antidiagonale della matrice a. Se a non e' quadrata
// mostra un messaggio di errore.

ris=[];
n=size(a)
if n(1)~=n(2)
    disp('Errore:');
    disp(' la matrice deve essere quadrata')
    return;
end
dim=n(1);
for i=1:dim
    ris(i)=a(i,dim-i+1);
end
endfunction

```

Le funzioni devono sempre cominciare con la parola **function** e terminare con la parola **endfunction**. Prima del nome della funzione, bisogna inserire, tra parentesi quadre, le variabili in output (le parentesi quadre possono essere omesse se deve essere restituita una sola variabile). Dopo il nome della funzione si scrivono, tra parentesi tonde, le variabili in input. Le doppie barre // indicano una riga di commento, una riga cioè che non sarà eseguita. E' buona norma commentare le funzioni indicando cosa fanno e quali sono i parametri in input ed in output ed inizializzare, all'inizio della funzione, le variabili in output. Il comando **disp** scrive a video, sulla command window, la stringa passata in input. Il comando **return** rende il controllo al chiamante, restituendo le variabili indicate come output. Le funzioni devono essere salvate con l'estensione .sci, che indica i file di Scilab che devono essere caricati prima di essere usati

Per poter usare la funzione, come abbiamo usato le funzioni già implementate da Scilab, dobbiamo caricarla, usando il comando **getf** seguito dal nome della funzione (se necessario il nome deve essere preceduto dal percorso per arrivare alla cartella in cui la funzione si trova).

```
-->getf addiag.sci
```

Se la funzione viene modificata, bisogna caricarla nuovamente, sempre con il comando **getf**, altrimenti in memoria rimane (e verrà quindi utilizzata) la versione "vecchia".

```
-->getf addiag.sci
```

```
Warning :redefining function: addiag
```

Scilab ci avverte che stiamo ridefinendo una funzione già presente. A questo punto possiamo usare **adiag** per estrarre la sottodiagonale e ottenerne la somma:

```
-->ad = addiag(A)
ad =
```

```
!  13. !
!  10. !
!   7. !
!   4. !
```

```
-->sum(ad)
ans =
```

```
34.
```

oppure

```
-->sum(adiag(A))
ans =
```

```
34.
```

f) Estrarre la prima riga di A.

Per estrarre la prima riga di A, dobbiamo prendere gli elementi di A che stanno sulla prima riga e su tutte le colonne di A:

```
-->a1 = A(1,:)
a1 =

! 16.    2.    3.    13. !
```

Il simbolo : significa “tutti”. Se si trova al posto del primo indice, indica a Scilab di prendere tutte le righe, mentre se si trova al posto del secondo indice indica di prendere tutte le colonne.

g) Estrarre i primi tre elementi della seconda colonna di A.

Per estrarre gli elementi richiesti, dobbiamo prendere gli elementi che stanno sulla seconda colonna e sulle righe dalla prima alla terza. Scriviamo quindi:

```
-->a2= A(1:3,2)
a2 =

! 2. !
! 11. !
! 7. !
```

h) Calcolare $A \cdot A'$, A^2 , $A \cdot A$ e $A.^2$.

```
-->A*A
ans =

! 345.    257.    281.    273. !
! 257.    313.    305.    281. !
! 281.    305.    313.    257. !
! 273.    281.    257.    345. !
```

```
-->A^2
ans =

! 345.    257.    281.    273. !
! 257.    313.    305.    281. !
! 281.    305.    313.    257. !
! 273.    281.    257.    345. !
```

```
-->A.*A
ans =

! 256.    4.    9.    169. !
! 25.    121.    100.    64. !
! 81.    49.    36.    144. !
! 16.    196.    225.    1. !
```

```
-->A.^2
ans =
```

!	256.	4.	9.	169.	!
!	25.	121.	100.	64.	!
!	81.	49.	36.	144.	!
!	16.	196.	225.	1.	!

Con l'operatore $*$ si intende il prodotto matriciale righe per colonne (attenzione alla compatibilità delle dimensioni). Con l'operatore \wedge si intende l'elevamento a potenza, rispetto al prodotto righe per colonne. Quindi $A^{\wedge n} = A * \dots * A$ n volte. Per avere compatibilità di dimensioni nel prodotto righe per colonne, tale operazione può essere eseguita solo su matrici quadrate. L'operatore $.$ indica che l'operazione che lo segue deve essere eseguita elemento per elemento (le due matrici coinvolte devono quindi avere le stesse dimensioni). Quindi $A.*A$ rende una matrice con la stessa dimensione di A e con elementi pari al prodotto degli elementi e $A.^2$ rende una matrice con la stessa dimensione di A e con elementi pari al quadrato degli elementi di A .

Esercizio 2: Cicli for.

1. Inserire la matrice di Hilbert di dimensione 5x5 definita da:

$$a_{ij} = \frac{1}{i+j-1}$$

(SUGGERIMENTO: utilizzare due cicli for annidati.)

2. Scrivere una function che fornisca la matrice di Hilbert di una generica dimensione `n` inserendo un commento.
3. Verificare il corretto funzionamento della function del punto precedente, confrontandone il risultato con la matrice restituita dal comando `inv(testmatrix('hilb',5))`.