appendice A

Esercitazioni con GNU Octave

GNU Octave può essere scaricato dalla seguente pagina:

`https://www.gnu.org/software/octave/download.html`

L'elenco dei concetti, delle funzioni e degli operatori di Octave possono essere trovati ai seguenti indirizzi:

`https://www.gnu.org/software/octave/doc/interpreter/Concept-Index.html`
`https://www.gnu.org/software/octave/doc/interpreter/Function-Index.html`
`https://www.gnu.org/software/octave/doc/interpreter/Operator-Index.html`

A.1 Introduzione a GNU Octave

A.1.1 Alcune funzioni

funzione	applicazione	significato
`dlmread`	input/output	input di un array da un file
`dlmwrite`	input/output	output di un array su un file
`disp`	input/output	scrive un testo o una variabile a video
`input`	input/output	scrive un messaggio ed attende un input
`printf`	input/output	formatta un array e produce l'output a video
`fprintf`	input/output	formatta un array e produce l'output su file
`sprintf`	input/output	formatta un array e restituisce una stringa a video
`str2num`	conversione	converte una stringa in numero
`num2str`	conversione	converte un numero in stringa
`zeros`	array	inizializza un array di elementi nulli

funzione	applicazione	significato
`ones`	array	inizializza un array di elementi unitari
`eye`	array	matrice identica
`repmat`	array	inizializza un array con termini uguali
`size`	array	numero di righe o di colonne di un array
`max`	array	massimo degli elementi di un array
`min`	array	minimo degli elementi di un array
`sum`	array	somma degli elementi di un array
`mean`	array	media degli elementi di un array
`abs`	math	valore assoluto di un numero/array
`log`	math	logaritmo naturale di un numero/array
`exp`	math	esponenziale di un numero/array
`sin`	math	seno di un numero/array
`cos`	math	coseno di un numero/array
`tan`	math	tangente di un numero/array
`cot`	math	cotangente di un numero/array
`asin`	math	arcoseno di un numero/array
`acos`	math	arcocoseno di un numero/array
`atan`	math	arcotangente di un numero/array
`acot`	math	arcocotangente di un numero/array
`fix`	math	parte intera di un numero
`round`	math	arrotondamento intero di un numero
`rem`	math	resto della divisione intera

A.1.2 Matrici

La creazione di una matrice si ottiene semplicemente scrivendone i termini separati da virgole o punti e virgole. Ad esempio, il comando

```
m=[1,2,3;4,5,6;7,8,9]
```

produrrà il seguente output

```
1 2 3
4 5 6
7 8 9
```

Se vogliamo aggiungere una riga costituita dagli elementi 1, 2, 3 alla matrice è sufficiente scrivere

```
m=[m;[1,2,3]]
```

Otterremo il seguente output

```
1 2 3
4 5 6
7 8 9
1 2 3
```

Se vogliamo aggiungere una colonna costituita dai termini 4, 5, 6, 7 alla nuova matrice è sufficiente scrivere

```
m=[m,[4;5;6;7]]
```

Otterremo il seguente output

```
1 2 3 4
4 5 6 5
7 8 9 6
1 2 3 7
```

Vediamo ora come inizializzare una matrice. L'inizializzazione a zero di una matrice 3x3 si ottiene con il seguente comando

```
m=zeros(3,3)
```

Otterremo il seguente output

```
0 0 0
0 0 0
0 0 0
```

Una matrice identica 3x3 si ottiene invece con il seguente comando

```
m=eye(3)
```

Otterremo il seguente output

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

Se, invece, vogliamo inizializzare una matrice 2x3 avente tutti gli elementi uguali ad un valore (ad esempio 7) abbiamo due possibilità. La prima è quella di eseguire il seguente comando

```
m=7*ones(2,3)
```

Ottenendo il seguente output

```
 7  7  7
 7  7  7
```

La seconda è quella di eseguire il comando

```
m=repmat(7,[2,3])
```

ottenendo lo stesso output.

A.2 Funzioni

Le funzioni sono programmi che data una serie di variabili in entrata restituiscono uno o più valori in uscita. La definizione di una funzione prevede il salvataggio di un file con estensione "**.m**" avente come nome lo stesso nome della funzione. Ad esempio, se vogliamo definire una funzione "funztest" che restituisca i risultati delle due operazioni $c = \sin(a - b)$ e $d = \exp(a + b)$ è necessario salvare un file chiamato "funztest.m" contenente il seguente testo:

```
function [c,d]=funztest(a,b)
    c=sin(a-b);
    d=exp(a+b);
end
```

A.3 Costrutti

A.3.1 Costrutto if

```
if (condizione 1)
    istruzioni se la condizione 1 è vera
elseif (condizione 2)
    istruzioni se la condizione 2 è vera
.
.
.
elseif (condizione n)
    istruzioni se la condizione n è vera
else
    istruzioni se tutte le condizioni precedenti sono false
endif
```

Esempio: test per stabilire che un numero intero sia pari o dispari

```
a=input("inserisci un numero intero ","s");
a=str2num(a);
if (a!=fix(a))
    disp "il numero inserito non è intero"
elseif (rem(a,2)==0)
    disp "il numero inserito è pari"
else
    disp "il numero inserito è dispari"
endif
```

A.3.2 Costrutto for

```
for cont=cont1:cont2
    istruzioni se cont1≤cont≤cont2
endfor
```

Esempio: costruzione della tavola pitagorica

```
a=input("inserisci un numero intero ","s");
a=str2num(a);
if (a!=fix(a))
    disp "il numero inserito non è intero"
else
    tab=[];
    for i=1:a
        for j=1:a
            tab(i,j)=i*j;
        endfor
    endfor
endif
disp (tab)
```

A.3.3 Costrutto while

```
while (condizione)
    istruzioni se condizione è vera
endwhile
```

Esempio: calcolo della radice numerica di una funzione. Costruiamo una funzione

"ff_bisection" salvando il file "ff_bisection.m". Il testo contenuto nel file è il seguente:

```
function [root,err,iter]=ff_bisection(ff,aa,bb,tol,itermax)
    iter=0;
    if (ff(aa)*ff(bb)>=0)
         disp "no solutions"
         return
    endif
    while (abs(bb-aa)>tol)
         root=0.5*(aa+bb);
         if (ff(root)*ff(aa)>0)
             aa=root;
         else
             bb=root;
         endif
         iter=iter+1;
         if (iter==itermax)
             disp (["max iterations (",num2str(itermax),")"])
             return
         endif
    endwhile
    err=abs(bb-aa);
end
```

Nella precedente function l'argomento "ff" rappresenta una funzione qualsiasi. Supponiamo, ad esempio, di volere calcolare la radice della funzione $\tan(x) - 1$. Definiamo innanzitutto una funzione salvando un file "tanxeq1.m" contenente il testo:

```
function [val]=tanxeq1(x)
    val=tan(x)-1;
end
```

In uno script dovremo poi chiamare la funzione di bisezione. La funzione della quale vogliamo calcolare la radice viene passata alla funzione di bisezione attraverso una "anonymous function". Quest'ultima viene definita come segue: "`ffh=@(x)tanxeq1(x)`".

```
aa=0;
bb=1;
tol=1.0e-12;
itermax=1000;
ffh=@(x)tanxeq1(x);
[root,err,iter]=ff_bisection(ffh,aa,bb,tol,itermax);
disp ""
disp (["numero di iterazioni  = ",sprintf("%i",iter)])
```

```
disp (["tolleranza            = ",sprintf("%.14e",tol)])
disp (["radice numerica       = ",sprintf("%.14e",root)])
disp (["arctan(1)             = ",sprintf("%.14e",atan(1))])
disp (["differenza            = ",sprintf("%.14e",root-atan(1))])
```