

Introduzione ad Octave

versione 1

Roberto Boggiani

9 febbraio 2003

1 Introduzione

1.1 Presentazione

Octave è un programma matematico di calcolo numerico. Compito del presente manuale è quello di fornire una guida ai principali comandi di tale programma.

1.2 Operazioni preliminari

Modificando il file

```
/usr/share/octave/site/m/startup/octaverc
```

possiamo inserire alcune modificazioni alla configurazione iniziale che saranno conservate ogni volta che avviamo il programma. Alcune modifiche preliminari da effettuare potrebbero essere le seguenti:

- `LOADPATH=[DEFAULT_LOADPATH,“:/mydirectory/octave“]` in questo memorizzando le funzioni e gli script che scriveremo nella directory `/mydirectory/octave` tali funzioni e script saranno direttamente accessibili dal programma
- `EDITOR=“kwrite“` per rendere l’editor `kwrite` editor predefinito di Octave
- `default_eval_print_flag=0` per visualizzare in forma migliore alcuni risultati
- `gnuplot_has_multiplot=1` per poter utilizzare `gnuplot` per comporre grafici multipli

1.3 Come ottenere un aiuto su un comando

Il programma permette, conoscendo il nome di un comando, di ottenere un aiuto sul comando stesso relativo sia al suo utilizzo sia al risultato che da esso si ottiene. Per richiamare un aiuto su un qualunque comando si usa la seguente sintassi:

```
help comando
```

in cui `comando` è il nome del comando di cui vogliamo un aiuto.

1.4 Come ottenere il manuale in linea

Il programma dispone di un ottimo manuale che può essere richiamato digitando:

```
help -i
```

Tale manuale è ordinato per argomenti e molto ricco di esempi.

1.5 Formato di visualizzazione dei numeri

Il programma permette di visualizzare in parecchi formati i risultati ottenuti. Per stabilire un certo formato visualizzazione si usa il comando:

```
format opzione
```

in cui `opzione` può assumere uno dei seguenti valori:

- `short`
- `long`
- `long e`
- `short e`
- `long E`
- `short E`
- `free`
- `none`
- `bank`
- `+`
- `hex`
- `bit`

Per vedere come operano tali opzioni si veda l'help di `format`.

1.6 Memorizzare un valore in una variabile

Per memorizzare un certo valore in una variabile si usa la sintassi:

```
variabile=valore
```

in cui `variabile` rappresenta il nome della variabile in cui il valore deve essere memorizzato.

2 I tipi di dati

2.1 Introduzione

Il programma utilizza diverse tipologie di dati. Vedremo in questo paragrafo quali sono e il loro utilizzo.

2.2 Diverse tipologie di dati

Le tipologie di dati che possono essere utilizzate in Octave sono le seguenti:

- dati numerici
- stringhe
- strutture

2.3 Funzioni utilizzabili su tutti i tipi di dati

Mentre vi sono funzioni diverse per ogni tipologia di dati da utilizzare, vi sono anche delle funzioni che sono applicabili ad ogni tipo di dato. Esse sono le seguenti:

- `columns(a)` restituisce il numero di colonne di `a`
- `rows(a)` restituisce il numero delle righe di `a`
- `length(a)` restituisce il numero di righe di `a` o il numero di colonne sempre di `a` se più grande
- `size(a)` restituisce il numero delle righe e delle colonne di `a`
- `size(a,1)` restituisce il numero delle righe di `a`
- `size(a,2)` restituisce il numero delle colonne di `a`
- `isempty(a)` restituisce 1 se `a` è una matrice in cui sia il numero di riga che quello di colonna sono zero

Si noti che uno scalare è considerato una matrice di dimensioni 1 x 1.

3 I Dati numerici

3.1 Introduzione

I dati numerici possono essere

- scalari
- matrici
- vettori

e possono contenere sia numeri reali che numeri complessi

3.2 Gli scalari

Gli scalari rappresentano numeri singoli e possono essere sia reali che complessi.

I numeri reali si ottengono semplicemente digitando una sequenza di numeri ad esempio con:
`a=123.78`

si assegna alla variabile `a` il numero reale 123.78

I numeri complessi si ottengono invece nel seguente modo: `a+bi`
in cui `a` e `b` sono scalari reali mentre `i` rappresenta l'unità immaginaria $\sqrt{-1}$. Si noti che `i` può anche essere sostituito da uno dei seguenti simboli `j`, `J`, `I`.

3.3 Le matrici

Le matrici sono tabelle di numeri che possono essere sia reali che complessi. La dimensione della matrice è determinata automaticamente, ad esempio l'espressione:

`A=[1,2;3,4]`

oppure con l'espressione

`A=[1 2;3 4]`

viene generata la matrice A :

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Per inserire una matrice bisogna ricordare di:

- racchiudere i numeri tra parentesi quadrate
- separare i numeri della riga con `,` o con uno spazio
- creare una nuova riga con `;`

Supponendo che A sia una matrice di dimensioni $m \times n$ il programma permette di estrarre righe, colonne e submatrici da tale matrice. Vi sono varie possibilità di richiamo date da:

- `A(a,b)` richiama l'elemento di posizione (a,b) della matrice A
- `A(:,b)` richiama la colonna di posizione b della matrice A
- `A(a,:)` richiama la riga di posizione a della matrice A
- `A(:,a:b:c)` richiama le colonne di posizione da a a c con passo b della matrice A
- `A(a:b:c,:)` richiama le righe di posizione da a a c con passo b della matrice A
- `A(a:b:c,a1:b1:c1)` richiama righe e colonne della matrice A secondo quanto visto in precedenza

in cui $a, b, c, a1, b1, c1$ sono numeri naturali.

In modo del tutto naturale possiamo anche assegnare uno o più elementi nuovi alla matrice A . Nell'ipotesi di aver creato la matrice A con:

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

abbiamo i seguenti casi:

- `A(1,2)=100` viene sostituito nella matrice A l'elemento di posto $(1,2)$ con il numero 100
- `A(1,:)=2:2:6` viene sostituita la prima riga di A con la successione 2,4,6

altre combinazioni di simboli sono facilmente interpretabili.

Una operazione utile è anche quella di cancellare intere righe o colonne di una matrice ciò viene effettuato nel seguente modo:

- `A(1,:)=[]` viene cancellata da prima riga della matrice A
- `A(:,1)=[]` viene cancellata da prima colonna della matrice A

altre combinazioni di simboli sono facilmente interpretabili.

3.4 I vettori

I vettori sono matrici aventi una sola riga o una sola colonna. Essi possono essere composti come le matrici sia da numeri reali che da numeri complessi. Per creare un vettore si usa la seguente espressione:

```
a=[1,2,3,4,5,6]
```

che genera il vettore:

```
1 2 3 4 5 6
```

Per inserire un vettore bisogna ricordare di:

- racchiudere i numeri tra parentesi quadrate
- separare i numeri che lo compongono con `,` o con uno spazio

I vettori possono anche essere creati nel seguente modo:

- `a:b` definisce un vettore da `a` a `b` con incremento 1
- `a:b:c` definisce un vettore da `a` a `c` con incremento `b` si noti che `b` può essere negativo
- `linspace(a,b,c)` definisce un vettore da `a` a `b` composto da `c` elementi
- `logspace(a,b,c)` come sopra ma crea un vettore logaritmico

con `a,b,c` numeri reali.

Supponendo che `x` sia un vettore di dimensioni `n` il programma permette di estrarre singoli elementi da tale vettore. Vi sono varie possibilità di richiamo date da:

- `x(a)` richiama l'elemento di posizione `a` del vettore `x`
- `x(a:b:c)` richiama gli elementi del vettore `x` da `a` a `c` con passo `b`

in cui `a,b,c` sono numeri naturali.

In modo del tutto naturale possiamo anche assegnare uno o più elementi nuovi alla vettore `x`. Nell'ipotesi di aver creato un vettore `x` con:

```
x=[1 2 3 4 5 6 7 8 9]
```

abbiamo i seguenti casi:

- `A(1)=100` viene sostituito nel vettore `x` l'elemento di posto 1 con il numero 100
- `x(1:2:9)=[1 4 6 8 9]` vengono sostituiti nel vettore `x` gli elementi richiamati con quelli nuovi

altre combinazioni di simboli sono facilmente interpretabili.

Una operazione utile è anche quella di cancellare elementi dal vettore `x` nel seguente modo:

- `x(a)=[]` viene cancellato l'elemento di posto `a` del vettore `x`
- `A(a:b:c)=[]` vengono cancellati gli elementi richiamati del vettore `x`

altre combinazioni di simboli sono facilmente interpretabili.

3.5 La funzione rand

La funzione rand permette di ottenere una matrice o un vettore composto da numeri casuali uniformemente distribuiti nell'intervallo]0; 1[. Il suo uso è il seguente:

`rand(n)` viene generata una matrice di numeri casuali $n \times n$

`rand(n,m)` viene generata una matrice di numeri casuali di dimensione $n \times m$

`rand("seed",n)` viene inizializzato il seme al valore `n`

`rand("seed")` otteniamo il valore del seme

3.6 Predicati per dati numerici

Vi sono alcune funzioni molto utili nella programmazione che fanno riferimento ai dati di tipo numerico. Esse sono:

- `is.matrix(x)` : ritorna 1 se `a` è una matrice 0 altrimenti
- `is.vector(x)` : ritorna 1 se `a` è un vettore 0 altrimenti
- `is.scalar(x)` : ritorna 1 se `a` è uno scalare 0 altrimenti
- `is.square(x)` : ritorna 1 se `a` è una matrice quadrata le dimensioni di `x` altrimenti
- `is.symmetric(x,tol)` : ritorna 1 se `a` è una matrice simmetrica con la tolleranza `tol` 0 altrimenti, se `tol` è omissso si usa la tolleranza della macchina

4 Le stringhe

4.1 Introduzione

Le stringhe consistono in una sequenza di caratteri inclusi in virgolette semplici o in doppie virgolette. Esempi di stringhe sono le seguenti:

```
"parrot"  
'parrot'
```

4.2 Concatenazione di stringhe

Le stringhe possono essere concatenate usando la notazione vista per le matrici. Si noti che l'espressione:

```
["foo","bar","baz"]
```

da origine alla stringa:

```
"foobarbaz".
```

4.3 Funzioni applicabili alle stringhe

Vi sono una serie di funzioni che possono essere applicate alle stringhe. Esse sono:

- `blank(n)` ritorna una stringa di `n` caratteri bianchi
- `int2str(n)` converte un numero in una stringa
- `num2str(n)` converte un numero in una stringa

- `setstr(x)` converte una matrice in una stringa, ogni elemento della stringa è convertito nel corrispondente carattere ascii.
- `strcat(s1,s2,...)` ritorna una stringa contenente tutti gli argomenti concatenati
- `str2mat(s1,s2,...,sn)` ritorna una matrice contenente le stringhe `s1,s2,...,sn` come righe
- `isstr(a)` ritorna 1 se `a` è una stringa 0 altrimenti
- `findstr(s,t,overlap)` ritorna un vettore con gli indici relative alla posizione che il carattere `t` occupa nella stringa `s`. Se `overlap` è uguale a 0 restituisce un vettore solamente con la prima e l'ultima occorrenza.
- `index(s,t)` ritorna la prima posizione in cui il carattere `t` si trova nella stringa `s`
- `rindex(s,t)` ritorna l'ultima posizione in cui il carattere `t` si trova nella stringa `s`
- `split(s,t)` divide la stringa `s` in parti separati dal carattere `t`
- `strcmp(s1,s2)` ritorna 1 se le stringhe `s1` ed `s2` sono uguali 0 altrimenti
- `strrep(s,x,y)` rimpiazza tutte le occorrenze della sottostringa `x` presenti in `s` con la stringa `y`
- `substr(s,ben,len)` ritorna una sottostringa di `s` che parte dal posto `ben` e arriva al posto `len`
- `bin2dec(s)` ritorna il numero decimale corrispondente al numero binario rappresentato come stringa di 0 ed 1
- `dec2bin(n)` ritorna il numero binario corrispondente a un numero decimale non negativo `n` sotto forma di stringa contenente 0 ed 1
- `dec2hex(n)` ritorna il numero esadecimale corrispondente a un numero decimale non negativo `n` sotto forma di stringa
- `hex2dec(s)` ritorna il numero decimale corrispondente al numero esadecimale rappresentato come stringa
- `str2num(s)` converte la stringa `s` in numero
- `toascii(s)` ritorna la rappresentazione ascii di `s` come matrice
- `tolower(s)` ritorna una stringa composta da caratteri tutti minuscoli
- `toupper(s)` ritorna una stringa composta da caratteri tutti maiuscoli

4.4 Altre funzioni operanti con stringhe

Vi sono altre funzioni che possono essere applicate alle stringhe o che danno come risultato una stringa. Esse sono le seguenti:

- `isalnum(s)` ritorna 1 per caratteri che sono lettere o cifre
- `isalpha(s)` ritorna 1 per caratteri che sono lettere

- `isascii(s)` ritorna 1 per caratteri che sono ascii
- `iscntrl(s)` ritorna 1 per caratteri di controllo
- `isdigit(s)` ritorna 1 per caratteri che sono cifre decimali
- `isgraph(s)` ritorna 1 per caratteri stampabili
- `islower(s)` ritorna 1 per caratteri minuscoli
- `isprint(s)` ritorna 1 per caratteri stampabili
- `ispunct(s)` ritorna 1 per caratteri di punteggiatura
- `isspace(s)` ritorna 1 per caratteri bianchi
- `isupper(s)` ritorna 1 per caratteri maiuscoli
- `isxdigit(s)` ritorna 1 per caratteri che sono cifre esadecimali

5 Le strutture

5.1 Introduzione

Octave include un supporto per organizzare qualsiasi tipo di dati in strutture. Gli elementi di una struttura possono essere valori di ogni tipo. Per esempio se digitiamo:

```
x.a=1
x.b=[1,2;3,4]
x.c="string"
```

abbiamo creato una struttura con tre elementi. Per visualizzare tale struttura dobbiamo semplicemente digitare `x` ottenendo com risultato:

```
x=
      {
      a=1
      b=
         1  2
         3  4
      c= string
      }
```

5.2 Esempio di utilizzo di una struttura

Le strutture possono essere utilizzate in modo efficace per mezzo delle funzioni. Un esempio può essere il seguente:

```
octave:1>function [y]=f(x)
> y.re = real(x);
> y.im = imag(x);
> endfunction
```

in questo modo il risultato della funzione sarà una struttura con primo elemento la parte reale di x e come secondo elemento quella immaginaria.

6 Le variabili

6.1 Introduzione

Le variabili sono gli elementi fondamentali del programma Octave. I nomi che possiamo assegnare alle variabili non possono essere superiori ai 30 caratteri. Si noti che i simboli i ed A sono distinte variabili.

6.2 Assegnazione di valori alle variabili

Possiamo assegnare un qualsiasi valore ad una variabile attraverso la seguente sintassi:

```
nomevariabile=valore
```

in cui nome variabile è il nome della variabile e valore è il valore che assegniamo alla variabile numerico o stringa.

Possiamo anche utilizzare in Octave delle variabili che prendono il nome di variabili globali. Le variabili globali sono variabili a cui possiamo fare riferimento da ogni funzione senza che sia necessario passare ad esse alcun parametro. Una variabile globale può essere dichiarata in uno dei seguenti modi:

```
global a  
global b=2  
global c=3,d,e=5
```

6.3 Funzioni che riguardano le variabili

Le principali funzioni che riguardano le variabili sono le seguenti:

- `clear nome` elimina la variabile `nome`
- `who` fornisce la lista delle variabili memorizzate
- `whos` fornisce la lista delle variabili memorizzate
- `exist(x)` ritorna 1 se il nome esiste come variabile, 2 se il nome è un file di funzione 3 se il nome è un file con estensione `.oct`, 5 se è una funzione predefinita 0 altrimenti.

7 Le espressioni

7.1 Introduzione

Le espressioni sono i blocchi basilari per costruire funzioni o effettuare calcoli con Octave. Le espressioni si basano su:

- operatori aritmetici
- operatori di comparazione
- operatori logici

- operatori di assegnazione
- operatori di incremento

7.2 Operatori aritmetici

I seguenti operatori aritmetici sono applicabili ove compatibili sia su scalari che su matrici. Essi sono:

- addizione: $x+y$
- addizione elemento per elemento: $x.+y$
- sottrazione: $x-y$
- sottrazione elemento per elemento: $x.-y$
- moltiplicazione: $x*y$
- moltiplicazione elemento per elemento: $x.*y$
- divisione destra: x/y è concettualmente equivalente ad $\text{inverse}(y')*x'$
- divisione destra elemento per elemento: $x./y$
- divisione sinistra: $x\backslash y$ è concettualmente equivalente ad $\text{inverse}(x)*y$
- divisione sinistra elemento per elemento: $x.\backslash y$
- operatore esponenziale: x^y
- operatore esponenziale: $x**y$
- operatore esponenziale elemento per elemento: $x.^y$
- operatore esponenziale elemento per elemento: $x.**y$
- negazione: $-x$
- trasposta: $x.'$
- operatore complesso trasposta: x' concettualmente e equivalente ad $\text{conj}(x.')$

Si noti che se dopo aver inserito l'espressione contenente l'operatore aritmetico inseriamo il `;`, il risultato non viene visualizzato da Octave. Per visualizzare il risultato sarà necessario non inserire `;`.

7.3 Operatori di comparazione

I seguenti operatori sono applicabili ove compatibili sia su scalari che su matrici. Se sono applicati su matrici sono applicati su ogni elemento delle matrici che li utilizzano. Essi sono:

- minore: $x < y$
- minore uguale: $x \leq y$
- maggiore: $x > y$
- maggiore uguale: $x \geq y$
- uguaglianza: $x == y$
- disuguaglianza: $x \neq y$
- disuguaglianza: $x \neq y$
- disuguaglianza: $x <> y$

7.4 Operatori logici

Gli operatori logici sono operatori che permettono di eseguire espressioni booleane ossia espressioni che restituiscono solamente il valore vero o falso. Tali operatori possono essere applicati sia a scalari che a matrici. Essi sono:

- and elemento per elemento: $\&$
- or elemento per elemento: $|$
- negazione di una espressione booleana: $!$
- negazione di una espressione booleana: $'$
- and complessivo: $\&\&$
- or complessivo: $||$

7.5 Operatori di assegnazione

Tramite gli operatori di assegnazione, noi assegnamo un determinato valore ad una variabile. L'operatore unico di assegnazione che è utilizzato in Octave è:

=

Per assegnare un valore ad una variabile possiamo quindi utilizzare la seguente sintassi:

`nomevariabile=valore da assegnare`

Così per assegnare un valore numerico ad una variabile possiamo utilizzare la sintassi:

`a=5`

mentre per assegnare un valore numerico possiamo utilizzare la sintassi:

`a="good"`

Per assegnare una matrice utilizziamo la sintassi:

`a=[1,2;3,4]`

Si noti quindi che assegnare un valore significa anche inizializzare una variabile a quel valore.

7.6 Operatori di incremento

Gli operatori di incremento sono quegli operatori che permettono di incrementare o decrementare il valore numerico memorizzato in variabili. Supponendo che x sia una variabile contenente un valore numerico gli operatori di incremento che possiamo applicare sono i seguenti:

- pre incremento: $++x$ incrementa la variabile x di uno e restituisce il nuovo valore
- post incremento: $x++$ incrementa la variabile x di uno e restituisce il valore prima dell'incremento
- pre decremento: $--x$ decrementa la variabile x di uno e restituisce il nuovo valore
- post decremento: $x--$ decrementa la variabile x di uno e restituisce il valore prima dell'incremento

8 La valutazione

8.1 Introduzione

Normalmente per valutare una espressione basterà semplicemente digitare l'espressione e dare l'invio. In certi casi sarà però necessario procedere, soprattutto in caso di programmazione, ad effettuare valutazioni di tipo diverso.

8.2 La funzione eval

La funzione `eval` provvede ad applicare ad un vettore di valori assunti dalla variabile x una determinata funzione inserita come stringa. Il suo uso è il seguente:

```
x=[1:10]
eval("x.^2")
```

in questo caso si ottiene la sequenza di tutti i quadrati dei primi dieci numeri naturali

8.3 La funzione feval

La funzione `feval` permette di valutare una funzione definita come funzione di Octave ossia con una estensione `.m` in un valore precisato o in un vettore di valori. Il suo uso è il seguente:

```
definire una funzione e memorizzata in un file ad esempio funzione.m
feval("funzione",a)
```

in questo caso se a è uno scalare si ottiene la valutazione della funzione in questo punto, se a è un vettore si ottiene la valutazione di tale funzione in ogni elemento del vettore.

9 Le strutture di controllo

9.1 Introduzione

Per poter definire funzioni, anche di particolare complessità, sarà necessario una conoscenza preliminare di come operano le strutture di controllo in Octave. Ciò sarà effettuato nei paragrafi successivi.

9.2 Le strutture di controllo

Qualunque linguaggio di programmazione prevede l'utilizzo delle strutture di controllo. Tali strutture sono le seguenti:

- if
- switch
- while
- for
- break
- continue

Vedremo nei paragrafi successivi il loro uso.

9.3 La struttura di controllo if

Il suo uso è il seguente:

```
if(condizione)
```

```
endif
```

oppure:

```
if(condizione)
```

```
else
```

```
endif
```

oppure

```
if(condizione)
```

```
elseif(condizione)
```

```
else
```

```
endif
```

9.4 La struttura di controllo switch

Il suo uso è il seguente:

```
switch espressione
```

```
case label
```

```
case label
```

```
otherwise
```

```
endswitch
```

9.5 La struttura di controllo while

Il suo uso è il seguente:

```
while(condizione)

endwhile
```

9.6 La struttura di controllo for

Il suo uso è il seguente:

```
for(var= espressione)

endfor
```

9.7 La struttura di controllo break

La struttura di controllo break quando viene eseguita fa uscire il programma dal ciclo for o while che la contiene. Essa può essere inserita esclusivamente in cicli for o while.

9.8 La struttura di controllo continue

La struttura di controllo continue quando viene eseguita fa uscire il avanzare il ciclo for o while fino alla fine permettendo la ripetizione dello stesso. Essa può essere inserita esclusivamente in cicli for o while.

10 Funzioni e script

10.1 Introduzione

Una delle potenzialità più grandi che possiede Octave è quella relativa alla possibilità di creare funzioni e di creare script che saranno eseguibili con una semplice chiamata da tastiera. Le funzioni restituiscono sempre un valore mentre gli script non sono altro che una successione di comandi che generalmente non produce alcun risultato.

10.2 Definire una funzione

Il procedimento che deve essere seguito per definire una funzione in Octave è il seguente. Prima di tutto di deve aprire un file con un qualunque editor di testo. La prima riga di tale file dovrà necessariamente contenere

```
function [y1,y2,...,yn]=nomefunzione(x1,x2,...,xn)
%Funzione per
%da usare con la seguente sintassi
%      [y1,y2,...,yn]=nomefunzione(x1,x2,...,xn)
%in cui si avrà che
%      x1=
%      x2=
%      .....
```

```
%      xn=  
%la funzione restituisce  
%      y1=  
%      y2=  
%      .....  
%      yn=  
corpo della funzione
```

Si noti quanto segue:

- le variabili denotate con x sono le varibili di input e possono essere in numero da 1 a n
- le variabili denotate con y sono le varibili di output e possono essere in numero da 1 ad n
- quanto racchiuso dentro il simbolo di % può essere omesso ma è utile sia per informare su come opera la funzione sia perchè quanto inserito viene richiamato come jhelp della funzione stessa
- il corpo della funzione rappresenta le istruzioni che devono essere eseguite e si noti che esso dovrà sempre contenere le istruzioni `y1=` , `y2=` , `yn=` che rappresentano l'output della funzione stessa.

Il file così completato dovrà essere salvato in una directory accessibile da Octave con il nome `nomefunzione.m`

10.3 Definire uno script

Il procedimento che deve essere seguito per definire uno script in Octave è il seguente. Prima di tutto di deve aprire un file con un qualunque editor di testo. La prima riga di tale file dovrà necessariamente contenere

```
function nomefunzione  
%Script per  
%da usare con la seguente sintassi  
%      nomefunzione  
%lo script restituisce  
%  
corpo dello script
```

Il file così completato dovrà essere salvato in una directory accessibile da Octave con il nome `nomescript.m`

10.4 Richiamare una funzione

Se abbiamo memorizzato la funzione nel modo visto sopra, sarà sempre possibile ottenere un help con il comando:

```
help nomefunzione
```

In generale una funzione in Octave verrà richiamata con il comando:

```
[y1,y2,...,yn]=nomefunzione(x1,x2,...,xn)
```

in cui:

- i valori delle variabili denotate con x devono essere valori numerici
- i valori restituiti dalla funzione saranno automaticamente memorizzati nelle variabili denotate con y

Si faccia attenzione a conoscere prima della chiamata della funzione il numero delle variabili di input e quelle di output.

10.5 Richiamare uno script

Se abbiamo memorizzato uno script nel modo visto sopra, sarà sempre possibile ottenere un help con il comando:

```
help nomefunzione
```

In generale uno script in Octave verrà richiamato con il comando:

```
nomefunzione
```

10.6 Esempio di programmazione: bisezione

La seguente funzione esegue il calcolo degli zeri di una funzione con il metodo di bisezione.

```
function [zero,niter]=bisezione(f,a,b,tol,numiter)
%Funzione per trovare gli zeri con il metodo di bisezione
%da usare con la seguente sintassi:
%           [zero,niter]=bisezione(f,a,b,tol,numiter)
%in cui si avrà che:
%f è una stringa che precisa la funzione f
%a è l'estremo inferiore dell'intervallo di ricerca
%b è l'estremo superiore dell'intervallo di ricerca
%tol è la tolleranza desiderata
%numiter è il numero massimo di iterazioni desiderate
%
%la funzione restituisce:
%zero ossia lo zero cercato
%niter ossia il numero di iterazioni necessarie
x=[a,(a+b)/2,b];
fx=eval(f);
n=0;
if sign(fx(1))*sign(fx(3))>0
    disp('Impossibile trovare una soluzione');
    break;
end;
while abs(fx(2))>=tol & n <= numiter
    n=n+1;
    if sign(fx(1))*sign(fx(2))>0
        x(1)=x(2);
    else
        x(3)=x(2);
    end
    x(2)=(x(1)+x(3))/2;
```

```
    fx=eval(f);  
end;  
zero=x(2);  
niter=n;
```

dovrà essere memorizzata nel file bisezione.m

10.7 Esempio di programmazione: ammfranc

La seguente funzione calcola la rata e il piano di ammortamento di un prestito utilizzando l'ammortamento francese.

```
function [rata,piano]=ammfranc(capitale,tasso,periodi)  
%Funzione per trovare la rata di un piano di Ammortamento Francese  
%da usare con la seguente sintassi:  
% [rata,piano]=ammfranc(capitale,tasso,periodi)  
%in cui si avrà che:  
% capitale = il valore del capitale da rimborsare  
% tasso     = il valore del tasso di periodo in valore unitario  
% periodi   = il numero dei periodi di rimborso  
if(nargin<3)  
disp('Argomenti insufficienti');  
else  
numero=(1-(1+tasso)^(-periodi))/tasso;  
rata=capitale/numero;  
end  
debitoresiduo=capitale;  
for i=1:periodi  
piano(i,1)=i;  
piano(i,3)=debitoresiduo*tasso;  
piano(i,2)=rata-piano(i,3);  
piano(i,4)=rata;  
piano(i,6)=debitoresiduo-piano(i,2);  
piano(i,5)=capitale-piano(i,6);  
debitoresiduo=piano(i,6);  
end;  
end;
```

dovrà essere memorizzata nel file ammfranc.m

10.8 Esempio di programmazione: matrice

La seguente funzione permette l'inserimento interattivo di una matrice in Octave.

```
function [y]=matrice(nr,nc)  
%Funzione per inserire una matrice in Octave  
%da usare con la seguente sintassi:  
% [y]=matrice(nr,nc)  
%in cui si avrà che:
```

```
% nr = numero i righe della matrice da inserire
% nc = numero di colonne della matrice da inserire
for i=1:nr
for j=1:nc
y(i,j)=input(sprintf("m[%o,%o]=",i,j));
endfor;
endfor;
```

dovrà essere memorizzata nel file matrice.m

10.9 Esempio di programmazione: zeri

La seguente funzione calcola e visualizza gli zeri complessi di una equazione.

```
function [y]=zeri(z,n)
%Funzione che calcola gli zeri dell'equazione
%          x^n=z
%da usare con la seguente sintassi
% [y]=zeri(z,n)
%in cui si avrà che
% z= il numero complesso
% n= il grado dell'equazione
a=abs(z);
b=angle(z);
for k=0:n-1
    y(k+1)=a^(1/n)*(cos((b+2*k*pi)/n)+i*sin((b+2*k*pi)/n));
end
data=[real(y),imag(y)];
gplot data with points;
```

dovrà essere memorizzata nel file zeri.m

11 Operatori di input e di output

11.1 Introduzione

Molte volte è necessario nella scrittura di programmi procedere anche a visualizzare risultati intermedi o stringhe di caratteri che possono portare ad un uso più interattivo del programam scritto. In questo paragrafo vedremo di analizzare le principali funzioni di input ed output che Octave permette di realizzare.

11.2 Output su video

Le principali funzioni di output su video sono le seguenti:

- `ans` permette di visualizzare il più recente risultato ottenuto
- `disp(x)` permette di visualizzare il valore di `x`
- `disp("stringa")` permette di visualizzare la stringa `stringa`

- `format` permette di fissare il modo di output delle cifre come visto in precedenza

La funzione più utilizzata di output si video è l'istruzione `disp` per questo motivo relativamente a tele funzione presentiamo i seguenti esempi:

- `disp(['gennaio','febbraio'])` stampa i nomi gennaio e febbraio nella stessa riga
- `disp(['gennaio','febbraio'])` stampa i nomi gennaio e febbraio su due righe diverse
- `disp([num2str(3),' febbraio'])` stampa 3 febbraio

Nell'ultimo esempio abbiamo utilizzato la funzione `num2str` per convertire una variabile numeri in stringa. Un modo migliore per effettuare tale operazione è quello di utilizzare la funzione `fprintf` e `sprintf`. Le sintassi di tali istruzioni sono le seguenti:

- `fprintf(formato, variabili)` stampa l'output direttamente sul video
- `stringa=sprintf(formato,variabile)` stampa l'output su di una stringa

in cui `formato` è una stringa di testo che tramite l'uso di caratteri speciali indica il tipo di formato dell'output, mentre `variabili` è una lista opzionale di variabili separate da una virgola e che hanno un corrispondente all'interno della stringa. Possiamo ad esempio scrivere:

```
x=10;y=5.5
```

```
fprintf("x=%d e y=%f",x,y)
```

ottenendo:

```
x=10 e y=5.500000
```

Oppure potremmo scrivere:

```
stringa=sprintf("x=%d e y=%f",x,y)
```

```
disp(stringa)
```

ottenendo lo stesso output di cui sopra.

La stringa di formato contiene codici che specificano il tipo di variabile che deve essere convertita in stringa e rappresentata sullo schermo. Il formato `%d` serve a visualizzare un numero intero, mentre il formato `%f` è utilizzato per i numeri reali. La seguente tabella rappresenta i descrittori di formato che possono essere utilizzati:

Codice di fomato	Azione
<code>%s</code>	formato stringa
<code>%d</code>	formato senza parte frazionaria
<code>%f</code>	formato numero decimale
<code>%e</code>	formato in notazione scientifica
<code>%g</code>	formato in formazione compatta
<code>\n</code>	inserisce carattere di ritorno a capo
<code>\t</code>	inserisce carattere di tabulazione

mentre la seguente tabella rappresenta alcune e semplificazioni:

Valore	<code>%6.3</code>	<code>%6.3e</code>	<code>%6d</code>
2	2.000	2.000e+00	2
0.02	0.020	2.000e-02	2.000000e-02
200	200.000	2.000e+02	200
<code>sqrt(2)</code>	1.414	1.414e+00	1.414214e+00
<code>sqrt(0.02)</code>	0.141	1.414e-01	1.414214e-01

Vogliamo anche porre in evidenza con il seguente esempio, l'uso vettoriale che possiamo fare delle funzioni `fprintf` e `sprintf`. Si ha allora che:

```
n=input("Inserisci il numero di valori: ");
x=linspace(0,pi,n);
c=cos(x);
s=sin(x);
disp("-----");
fprintf("k\t x(k)\t cos(x(k))\t sin(x(k))\n");
disp("-----");
fprintf("%d\t %3.2f\t %6.5f\t %6.5f\n",[1:n;x;c;s]);
```

Se eseguito verrà stampata una tabella avente come intestazione `k,x(k),cos(x(k)),sin(x(k))` e sotto tante righe con i corrispondenti valori in relazione al valore di `n` impostato.

11.3 Input da video

Le principali funzioni di input sono le seguenti:

- `input("stringa")` visualizza la stringa `stringa` e si ferma per aspettare un input numerico
- `input("stringa","s")` visualizza la stringa `stringa` e si ferma per aspettare un input di carattere

Tramite tale funzione possiamo realizzare un input di tipo interattivo. Ad esempio se digitiamo: `n=input("Inserisci un numero ")` il programma si arresterà fino a che non digitiamo un numero da tastiera e premiamo invio, successivamente tale numero sarà assegnato alla variabile `n`.

12 Salvare o caricare dati

12.1 Introduzione

Capiterà spesso di dover salvare risultati ottenuti con Octave affinché essi siano letti da altri programmi o caricare in Octave file prodotti da altri programmi per effettuare opportune elaborazioni. Il programma presenta a riguardo una serie di comandi notevoli. Noi approfondiremo l'uso solamente di due di questi comandi `load` e `save`.

12.2 Salvare i risultati in un file

Il salvataggio dei valori contenuti in una variabile di Octave in un file viene effettuata con il comando:

```
save opzioni file v1 v1 ...
```

in cui:

- `opzioni` sarà sostituito con una delle opzioni di output previste ossia
 - `-ascii`
 - `-binary`
 - `-float-binary`

- -mat-binary
- -save-builtins
- `file` sarà il nome del file in cui salvare le variabili
- `v1,v2,...` saranno le variabili da salvare nel file

Per ulteriori precisazioni si rinvia all’help in linea del comando `save`

12.3 Caricare un file di dati

Il caricamento dei valori contenuti in un file esterno in Octave viene effettuata con il comando:
`load opzioni file v1 v1 ...`

in cui:

- `opzioni` sarà sostituito con una delle opzioni di output previste ossia
 - -force
 - -ascii
 - -binary
 - -mat-binary
- `file` sarà il nome del file da caricare
- `v1,v2,...` saranno le variabili in cui saranno memorizzati i file caricati

Per ulteriori precisazioni si rinvia all’help in linea del comando `load`

12.4 Output ed input formattato

In Octave sono disponibili anche altri comandi che permettono di eseguire una formattazione sia dei valori in input che di quelli in output. Tra le funzioni non analizzate in precedenza ricordiamo:

- `printf` funzione di output
- `fscanf` funzione di input
- `sscanf` funzione di input
- `scanf` funzione di input

Per l’uso di tali funzioni si rimanda all’help in linea.

13 I grafici di funzioni

13.1 Introduzione

Il programma per eseguire i grafici delle funzioni utilizza il programma `gnuplot`. In generale tutte le opzioni che possono essere usate con `gnuplot` possono anche essere usate in Octave con una sintassi leggeremnte diversa. Sarà allora possibile tracciare grafici sia di funzioni bidimensionali che tridimensionali.

13.2 Grafici cartesiani a due dimensioni: primo modo

La sintassi che dobbiamo utilizzare per tracciare i grafici cartesiani a due dimensioni con questo primo procedimento è quella relativa al tracciamento di dati con `gnuplot`. Supponendo quindi di avere creato con Octave una matrice a più colonne contenete i dati da tracciare, il grafico può essere ottenuto con il seguente comando:

```
gplot [a:b][c:d] data using e:f with stilelinea
```

si noti che inserendo alla fine una virgola e ripetendo tutta la stringa ad eccezione di `gplot` possiamo tracciare più grafici in un unico diagramma. Naturalmente si avrà che:

- `a,b` rappresentano il range opzionale della variabile `x`
- `c,d` rappresentano il range opzionale della variabile `y`
- `e,f` sono le colonne della matrice di cui vogliamo tracciare il grafico

inoltre `stilelinea` sarà uno dei seguenti:

- `lines`
- `points`
- `linespoints`
- `impulses`
- `dots`
- `steps`
- `errorbars`
- `boxes`

Si noti che verranno stampati i grafici di colonna contro colonna e quindi in questo modo i dati dovranno essere memorizzati nella matrice.

Come detto Octave per tracciare i grafici utilizza il programma `gnuplot`. Tale programma permette di personalizzare i grafici attraverso la fissazione di opzioni. Tali opzioni possono essere inserite direttamente da octave con il comando:

```
gset opzione
```

in cui `opzione` è una opzione accettata da `gnuplot` (si veda un qualunque manuale di `gnuplot`). Per visualizzare lo stato di una opzione si può utilizzare sempre da octave il comando:

```
gshow opzione
```

in cui `opzione` è sempre una opzione accettata da `gnuplot`.

Come sicuramente sa chi utilizza `gnuplot`, quando in tale programam si fissa una certa opzione, per renderla esecutiva sarà necessario digitare il comando `replot`. Tale fatto si riscontra anche in octave, quindi dopo aver fissato una opzione con il comando `gset` per vederla applicata dovremmo digitare `replot` a meno che non abbiamo impostato la variabile `automatic_replot` ad un valore diverso da 0.

13.3 Grafici cartesiani a due dimensioni: secondo modo

Esiste anche un secondo metodo che permette di tracciare i grafici di funzioni a due dimensioni. La più semplice forma di grafico ottenuto con tale metodo è la seguente:

```
plot(x)
```

in cui x è un vettore colonna comèposto da n elementi. In questo modo viene tracciato un grafico in cui in ascissa troviamo i numeri naturali da 1 ad n in corrispondenza dei quali troviamo il corrispondente valore del vettore x .

Il modo più completo per utilizzare tale comando è il seguente:

```
plot(x,y,fmt='valore',...)
```

in cui per quanto riguarda x ed y si avrà che:

- se x è un vettore colonna mentre y è una matrice, viene tracciato il grafico del vettore x verso ciascuna delle colonne della matrice y .
- se x è una matrice mentre y è un vettore colonna, viene tracciato il grafico di ogni colonna della matrice x verso il vettore y .
- se x ed y sono vettori viene tracciato il grafico di x contro y
- se x ed y sono matrici viene tracciato il grafico delle colonne di y contro le colonne di x
- se x ed y sono scalari è tracciato semplicemente un punto

mentre il valore assunto da `fmt` è il seguente:

- `'-'`
- `'.'`
- `'@'`
- `'-@'`
- `'^'`
- `'L'`
- `'#'`
- `'~'`
- `'#~'`
- `'n'`
- `'nm'`
- `'c'`
- `'+'`
- `'*'`
- `'o'`

- 'x'

Per il significato di tali simboli e il loro uso si veda l'help in linea richiamabile con `help plot`

Si noti che per tracciare più grafici sullo stesso diagramma possiamo utilizzare la sintassi:

```
plot(x,y,fmt='valore',x,y2,fmt='valore')
```

anche più volte o utilizzare il comando `hold on`. In questo modo i grafici tracciati successivamente con `plot` saranno aggiunti a quello visualizzato. Per levare questa caratteristica basterà ovviamente digitare `hold off`.

13.4 Grafici particolari a due dimensioni

Octave permette di ottenere anche dei grafici a due dimensioni differenti da quelli cartesiani. L'elenco è la loro sintassi è la seguente:

- `bar(x,y)` con `x` ed `y` vettori produce un grafico a barre
- `contour(z,n)` produce un contour plot di `n` linee di un grafico tridimensionale descritto da `z`
- `hist(x,y)` produce un istogramma di `x` avente per numero di bin `y` se `y` è scalare
- `loglog(argomenti)` produce un grafico bidimensionale usando la scala logaritmica per entrambi gli assi
- `polar(theta,rho)` produce un grafico bidimensionale polare date le coordinate polari `theta` e `rho`
- `stairs(x,y)` con `x` ed `y` vettori produce uno staircase plot

13.5 Grafici cartesiani tridimensionali

Per creare grafici cartesiani tridimensionali dobbiamo usare la seguente sintassi:

`a=[a:b:c]` per creare il range dell'asse delle `x` con incremento `b`

`b=[a1:b1:c1]` per creare il range dell'asse delle `y` con incremento `b1`

`[x,y]=meshdom(a,b)` per creare la matrice valutativa

`z=espressione funzione con x ed y` per ottenere la matrice da plottare

`mesh(x,y,z)` per ottenere il grafico della funzione voluta

in alternativa per ottenere lo stesso grafico possiamo usare:

`gsplot z`

secondo la sintassi di `gnuplot`.

13.6 Annotazioni nei grafici

Nei grafici ottenuti da octave è possibile inserire delle annotazioni. Esistono varie possibilità:

- tramite il comando `grid` viene inserita una griglia nel grafico
- tramite il comando `title("nometitolo")` viene inserito un titolo al grafico
- tramite il comando `xlabel("sting")` viene specificato un nome per l'asse delle `x`
- tramite il comando `ylabel("sting")` viene specificato un nome per l'asse delle `y`

- tramite il comando `zlabel("string")` viene specificato un nome per l'asse delle z

Ognuno di tali comandi richiede per essere visualizzato il comando `replot`.

13.7 Tracciare più grafici in una pagina

Con octave è anche possibile tracciare più grafici in una stessa pagina. Per fare questo dobbiamo prima di tutto specificare quanti grafici vogliamo ottenere e tale indicazione è data con il comando:

```
multiplot(xn,yn)
```

in cui `xn` rappresenta il numero di righe mentre `yn` quello delle colonne della pagina in cui vogliamo ottenere i grafici, si noti che il numero dei grafici sarà pari ad `xn*yn`.

Successivamente possiamo iniziare a tracciare il primo grafico con i soliti comandi:

```
gplot plot gspot o mesh
```

e modificarne l'aspetto inserendo titolo e altre etichette.

Prima di tracciare il secondo grafico dobbiamo indicare ad Octave che abbiamo finito di modificare il grafico precedente e che vogliamo tracciare il grafico successivo. Ciò è fatto digitando il seguente comando:

```
subplot(xn,yn,n)
```

in cui `n` è il numero del grafico che vogliamo tracciare. Si noti che la numerazione avviene sempre dall'alto verso basso e da sinistra verso destra. Dato il comando potremmo tracciare il grafico utilizzando i soliti comandi.

14 Funzioni applicabili ad una matrice

14.1 Introduzione

Abbiamo visto che Octave opera prevalentemente facendo uso di matrici. Appunto per questo motivo esso dispone di molte funzioni applicabili alle matrici che analizzeremo in questo paragrafo.

14.2 Le funzioni applicabili ad una matrice

Le funzioni che possono essere applicate ad una matrice `x` sono:

- `fliplr(x)` restituisce una matrice con l'ordine delle colonne rovesciato
- `flipud(x)` restituisce una matrice con l'ordine delle righe rovesciato
- `rot90(x,n)` restituisce una matrice ruotata di 90 gradi `n` è opzionale ed indica di quante volte dobbiamo ruotare di 90 gradi la matrice
- `reshape(x,m,n)` ritorna una matrice composta da `m` righe ed `n` colonne i cui elementi sono presi dal vettore `x` in ordine di colonna
- `shift(x,b)`
- `[s,i]=sort(x)`
- `tril(x,k)` ritorna la matrice triangolare inferiore della matrice `x` il valore di `k` è opzionale

- `triu(x,k)` ritorna la matrice triangolare superiore della matrice `x` il valore di `k` è opzionale
- `vec(x)` ritorna un vettore ordinato secondo le colonne dalla matrice `x`
- `eye(x)` ritorna una matrice identità delle stesse dimensioni di `x` se `x` è uno scalare ritorna una matrice identità quadrata di ordine `n`
- `eye(n,m)` ritorna una matrice identità di `n` righe ed `m` colonne
- `ones(x)` ritorna una matrice di tutti uno delle stesse dimensioni di `x` se `x` è uno scalare ritorna una matrice con tutti uno quadrata di ordine `n`
- `ones(n,m)` ritorna una matrice di tutti uno di `n` righe ed `m` colonne
- `zeros(x)` ritorna una matrice di tutti zero delle stesse dimensioni di `x` se `x` è uno scalare ritorna una matrice con tutti zero quadrata di ordine `n`
- `zeros(n,m)` ritorna una matrice di tutti zero di `n` righe ed `m` colonne
- `rand(x)` ritorna una matrice casuale delle stesse dimensioni di `x` se `x` è uno scalare ritorna una matrice con numeri casuali quadrata di ordine `n`
- `rand(n,m)` ritorna una matrice casuale di `n` righe ed `m` colonne
- `rand(serie,x)` ritorna una matrice casuale specificando anche il seme di partenza
- `randn(x)` come `rand` solo che i numeri casuali derivano da una distribuzione normale e non da una distribuzione uniforme
- `randn(n,m)` come `rand` solo che i numeri casuali derivano da una distribuzione normale e non da una distribuzione uniforme
- `randn(serie,x)` come `rand` solo che i numeri casuali derivano da una distribuzione normale e non da una distribuzione uniforme
- `diag(v,k)` ritorna una matrice diagonale avente nella diagonale gli elementi del vettore `v`, l'argomento `k` è opzionale
- `linspace(base,limit,n)` ritorna un vettore riga con `n` elementi spazati linearmente da `base` a `limit`
- `logspace(base,limit,n)` ritorna un vettore riga con `n` elementi spazati logaritmicamente da 10^{base} a 10^{limit}

14.3 Matrici importanti

Con Octave è possibile ottenere matrici importanti per l'analisi matematica. Si dà solo l'elenco in quanto per il loro uso e la sintassi si rimanda all'help in linea. Tali matrici sono:

- `hankel`
- `hilb`
- `invhilb`

- `sylvester_matrix`
- `toeplitz`
- `vander`

15 Funzioni matematiche di base

15.1 Introduzione

Vedremo in questa sezione una serie di funzioni matematiche di base di cui dispone il programma Octave ricordando che tali funzioni possono essere applicate sia a scalari che a vettori o matrici.

15.2 Le funzioni aritmetiche

Le funzioni aritmetiche che possiamo utilizzare in Octave sono le seguenti:

- `ceil(x)`
- `exp(x)`
- `fix(x)`
- `floor(x)`
- `gcd(x, ...)`
- `lcm(x, ...)`
- `log(x)`
- `log10(x)`
- `max(x)`
- `min(x)`
- `nextpow2(x)`
- `pow2(x)`
- `pow2(f, e)`
- `rem(x, y)`
- `round(x)`
- `sign(x)`
- `sqrt(x)`
- `xor(x)`

che non richiedono ulteriori spiegazioni.

15.3 Le funzioni riguardanti i complessi

Le funzioni complesse aritmetiche che possiamo utilizzare in Octave sono le seguenti:

- `abs(z)`
- `arg(z)`
- `angle(z)`
- `conj(z)`
- `imag(z)`
- `real(z)`

con z numero complesso che non richiedono ulteriori spiegazioni.

15.4 Le funzioni trigonometriche

Le funzioni trigonometriche che possiamo utilizzare in Octave sono le seguenti:

- `sin(z)`
- `cos(z)`
- `tan(z)`
- `sec(z)`
- `csc(z)`
- `cot(z)`
- `asin(z)`
- `acos(z)`
- `atan(z)`
- `asec(z)`
- `acsc(z)`
- `acot(z)`
- `sinh(z)`
- `cosh(z)`
- `tanh(z)`
- `sech(z)`
- `csch(z)`
- `coth(z)`

- `asinh(z)`
- `acosh(z)`
- `atanh(z)`
- `asech(z)`
- `acsch(z)`
- `acoth(z)`
- `atan2(x,y)`

con z numero complesso che non richiedono ulteriori spiegazioni.

15.5 Le funzioni di somma e prodotto

Le funzioni di somma e prodotto che possiamo utilizzare in Octave sono le seguenti:

- `sum(x)`
- `prod(x)`
- `cumsum(x)`
- `cumprod(x)`
- `sumsq(x)`

con x vettore che non riedono ulteriori spiegazioni.

15.6 Le funzioni speciali

Le funzioni speciali che possiamo utilizzare in Octave sono le seguenti:

- `besseli(alpha,x)`
- `besselij(alpha,x)`
- `besselk(alpha,x)`
- `bessely(alpha,x)`
- `beta(a,b)`
- `betai(a,b,x)`
- `bincoeff(n,k)`
- `erf(z)`
- `erfc(z)`
- `erfinv(z)`

- `gamma(z)`
- `gammai(a,x)`
- `lgamma(a,x)`
- `gammaln(a,x)`
- `cross(x,y)`
- `commutation_matrix(m,n)`
- `duplication_matrix(n)`

con argomenti scalari reali o complessi o vettori che non riedono ulteriori spiegazioni.

15.7 Le costanti matematiche

Le costanti matematiche che possiamo utilizzare in Octave sono le seguenti:

- `I`
- `J`
- `i`
- `j`
- `Inf`
- `inf`
- `NaN`
- `nan`
- `pi`
- `e`
- `eps`
- `realmax`
- `realmin`

che non riedono ulteriori spiegazioni.

16 Funzioni riguardanti campi specifici della matematica

16.1 Introduzione

Nelle ultime versioni di Octave sono state aggiunte molte funzioni riguardanti campi specifici della matematica. In questo paragrafo verrà precisato solamente in che modo venirne a conoscenza.

16.2 Le funzioni riguardanti i campi specifici della matematica

Per conoscere quali sono tali funzioni disponibili in Octave si veda l'help in linea con:

`help -i`

e si consultino le seguenti sezioni:

- Linear Algebra
- Nonlinear Equations
- Quadrature
- Differential Equations
- Optimization
- Statistics
- Financial Functions
- Sets
- Polynomial Manipulations
- Control Theory
- Signal Processing
- Image Processing
- Audio Processing
- Quaternions

17 Particolarità

17.1 Introduzione

Vogliamo raggruppare in questo paragrafo una serie di particolarità che riguardano Octave e che non trovano collocazione in altre parti del manuale

17.2 Il comando `diary`

Può essere utile a volte disporre dell'elenco dei comandi che sono stati eseguiti in una sessione di lavoro di Octave e dei risultati che sono stati visualizzati nel video. Ciò può essere effettuato tramite la funzione `diary`. Tale funzione prevede innanzitutto di essere inizializzata tramite il comando:

```
diary on
```

successivamente a tale comando tutto quello che digiteremo e che verrà visualizzato a video fino alla digitazione di

```
diary off
```

potrà essere salvato in un file tramite il comando:

```
diary "nomefile.txt"
```

Si noti quindi che per salvare qualcosa sarà necessario inizializzare e terminare l'inizializzazione di tale comando.

Per maggiori informazioni e opzioni si consiglia di visualizzare l'help in linea di tale comando.

17.3 L'uso di history

Octave di default salva tutti i comandi che sono stati digitati in una variabile di nome `history`. Per vedere l'elenco di tutti i comandi digitati si può semplicemente digitare:

```
history
```

e scorrere l'elenco così ottenuto.

Digitando invece il comando:

```
edit_history n
```

verrà richiamato nell'editor predefinito la riga `n` di `history` che potrà essere modificata, all'uscita dall'editor verrà eseguita l'istruzione così modificata. Se invece digitiamo:

```
edit_history n1 n2
```

verranno richiamate nell'editor predefinito le righe da `n1` ad `n2` di `history` che potranno essere modificate, all'uscita dall'editor verranno eseguite le istruzione così modificate.

Il comando `run_history` funziona come `edit_history` ad eccezione del fatto che l'editor non viene richiamato e le istruzioni saranno semplicemente eseguite senza essere visualizzate.

Si noti che il numero di istruzioni memorizzato è predefinito da Octave in 1024 per fissare un'altro valore bisognerà modificare la variabile `OCTAVE_HISTSIZ`.

Per maggiori informazioni e opzioni si consiglia di visualizzare l'help in linea di tali comandi.

Indice

1	Introduzione	1
1.1	Presentazione	1
1.2	Operazioni preliminari	1
1.3	Come ottenere un aiuto su un comando	1
1.4	Come ottenere il manuale in linea	1
1.5	Formato di visualizzazione dei numeri	2
1.6	Memorizzare un valore in una variabile	2
2	I tipi di dati	2
2.1	Introduzione	2
2.2	Diverse tipologie di dati	2
2.3	Funzioni utilizzabili su tutti i tipi di dati	3
3	I Dati numerici	3
3.1	Introduzione	3
3.2	Gli scalari	3
3.3	Le matrici	4
3.4	I vettori	5
3.5	La funzione rand	6
3.6	Predicati per dati numerici	6
4	Le stringhe	6
4.1	Introduzione	6
4.2	Concatenazione di stringhe	6
4.3	Funzioni applicabili alle stringhe	6
4.4	Altre funzioni operanti con stringhe	7
5	Le strutture	8
5.1	Introduzione	8
5.2	Esempio di utilizzo di una struttura	8
6	Le variabili	9
6.1	Introduzione	9
6.2	Assegnazione di valori alle variabili	9
6.3	Funzioni che riguardano le variabili	9
7	Le espressioni	9
7.1	Introduzione	9
7.2	Operatori aritmetici	10
7.3	Operatori di comparazione	11
7.4	Operatori logici	11
7.5	Operatori di assegnazione	11
7.6	Operatori di incremento	12

8	La valutazione	12
8.1	Introduzione	12
8.2	La funzione eval	12
8.3	La funzione feval	12
9	Le strutture di controllo	12
9.1	Introduzione	12
9.2	Le strutture di controllo	13
9.3	La struttura di controllo if	13
9.4	La struttura di controllo switch	13
9.5	La struttura di controllo while	14
9.6	La struttura di controllo for	14
9.7	La struttura di controllo break	14
9.8	La struttura di controllo continue	14
10	Funzioni e script	14
10.1	Introduzione	14
10.2	Definire una funzione	14
10.3	Definire uno script	15
10.4	Richiamare una funzione	15
10.5	Richiamare uno script	16
10.6	Esempio di programmazione: bisezione	16
10.7	Esempio di programmazione: ammfranc	17
10.8	Esempio di programmazione: matrice	17
10.9	Esempio di programmazione: zeri	18
11	Operatori di input e di output	18
11.1	Introduzione	18
11.2	Output su video	18
11.3	Input da video	20
12	Salvare o caricare dati	20
12.1	Introduzione	20
12.2	Salvare i risultati in un file	20
12.3	Caricare un file di dati	21
12.4	Output ed input formattato	21
13	I grafici di funzioni	21
13.1	Introduzione	21
13.2	Grafici cartesiani a due dimensioni: primo modo	22
13.3	Grafici cartesiani a due dimensioni: secondo modo	23
13.4	Grafici particolari a due dimensioni	24
13.5	Grafici cartesiani tridimensionali	24
13.6	Annotazioni nei grafici	24
13.7	Tracciare più grafici in una pagina	25

14 Funzioni applicabili ad una matrice	25
14.1 Introduzione	25
14.2 Le funzioni applicabili ad una matrice	25
14.3 Matrici importanti	26
15 Funzioni matematiche di base	27
15.1 Introduzione	27
15.2 Le funzioni aritmetiche	27
15.3 Le funzioni riguardanti i complessi	28
15.4 Le funzioni trigonometriche	28
15.5 Le funzioni di somma e prodotto	29
15.6 Le funzioni speciali	29
15.7 Le costanti matematiche	30
16 Funzioni riguardanti campi specifici della matematica	30
16.1 Introduzione	30
16.2 Le funzioni riguardanti i campi specifici della matematica	31
17 Particolarità	31
17.1 Introduzione	31
17.2 Il comando diary	31
17.3 L'uso di history	32