Un’introduzione a Octave

Di Sonia Gigli e Mariangela Marinelli

Capitolo I

Octave: introduzione

Il programma GNU Octave è un programma di calcolo numerico. Inizialmente, GNU Octave nacque presso l'Università del Texas nel 1988 come programma di calcolo per l'ingegneria chimica; il nome 'Octave' è il nome del docente di uno dei corsi presso i quali John W. Eaton, il principale autore del programma, ne iniziò lo sviluppo nel 1992. Poi fu esteso a coprire ambiti matematici più generali, come l'algebra lineare e le equazioni differenziali, infatti la prima alpha rilasciata è datata 4 gennaio 1993 e il 17 febbraio 1994 fu rilasciata la versione 1.0.

Attualmente GNU Octave è un programma mantenuto e sviluppato dal progetto GNU, rilasciato con licenza GPL (è, quindi, software libero) ed è incluso nelle principali distribuzioni GNU/Linux, ma riceve contributi

da un gran numero di sviluppatori indipendenti, spesso ricercatori e docenti universitari.

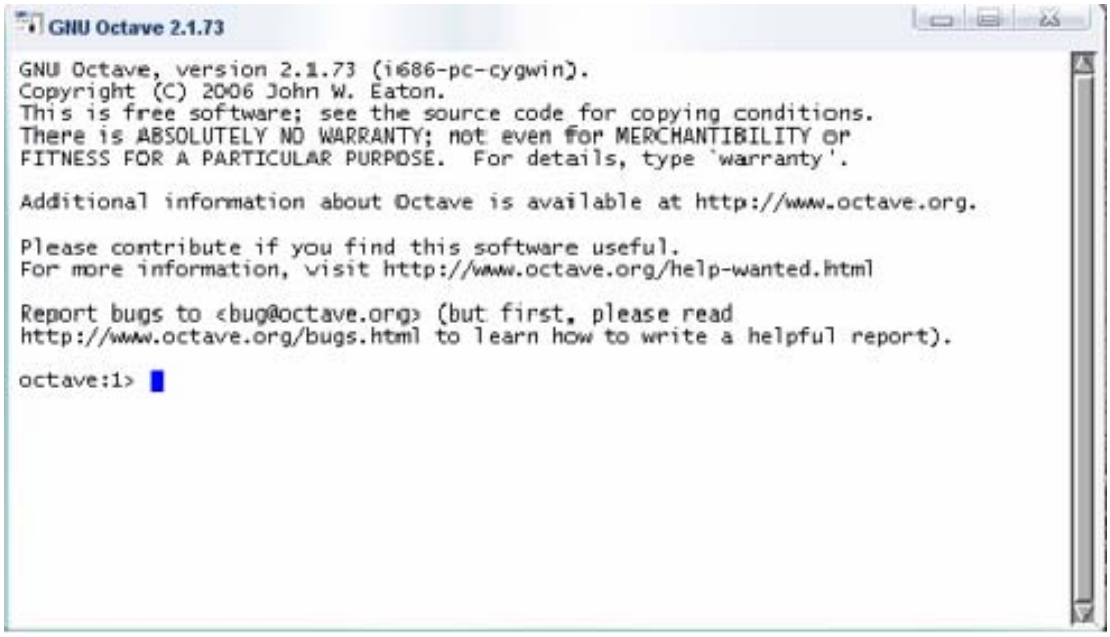
Tra gli ambiti di utilizzo di GNU Octave (Octave per semplicità, d'ora in poi) si trovano: l'aritmetica di base, l'algebra lineare, il calcolo matriciale, le equazioni algebriche e differenziali, la statistica, la ricerca operativa, la matematica finanziaria, la teoria dei controlli, l'analisi delle immagini, l'analisi dei segnali, l'analisi del suono, e molti altri.

È semplicemente estendibile e caratterizzabile con funzioni definite dall'utente stesso, scritte nel linguaggio di *Octave* o usando moduli scritti in *C++ usando le librerie STL*, *C*, *Fortran* o altri linguaggi.

Octave è liberamente scaricabile dal sito www. Octave.org sia per piattaforma Linux, sia per Windows ed è compreso in gran parte delle distribuzioni Linux. Le sue funzionalità sono molti simili a quelle di Matlab. In ambito ingegneristico Matlab è molto usato ma Octave si presenta come una valida alternativa open source, infatti gli sviluppatori stanno cercando di implementare le stesse funzioni con lo stesso nome di Matlab in Octave.

E’ un programma open source mantenuto e sviluppato dal progetto GNU liberamente scaricabile dal sito www.octave.org. Nella sezione download, occorre scegliere la versione adatta al sistema operativo utilizzato (nel nostro caso windows) e cliccare su Octave forge-

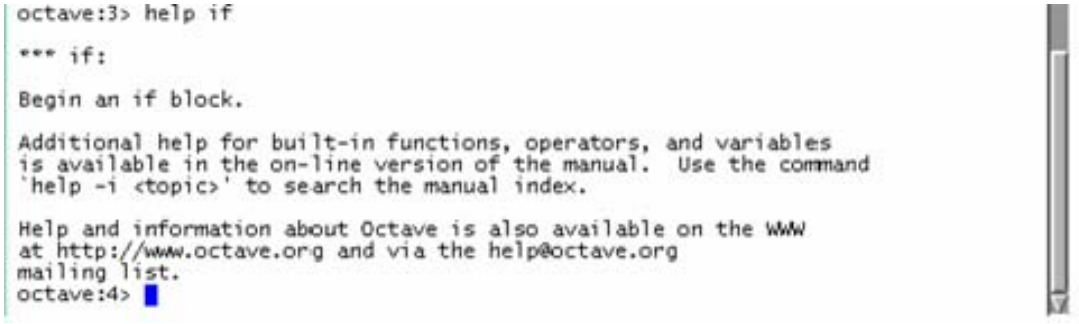
windows installer per installare il programma (la versione da noi utilizzata è octave 2.1.73). In seguito all’installazione verrà creato un collegamento sul desktop, basterà cliccarci su due volte per lanciare il programma.



Help

La documentazione di supporto è disponibile sul prompt di Octave, fornisce informazioni sul comando che si vuole utilizzare semplicemente scrivendo il nome del comando dopo la parola “help”.

Esempio:



Se richiamato senza alcuna voce l’help stampa una lista di tutti gli operatori e funzioni disponibili.

Per esempio il comando “help help” descrive il comando help.

Octave Forge e la compatibilità con Matlab

:. Vantaggi

Il programma presenta notevoli vantaggi. Ne elenchiamo i principali.

- È facilissimo da usare per chi conosce Matlab
- E' uno strumento affidabile perché utilizza, per il calcolo numerico, le librerie del sito http://netlib.org. Esse implementano i principali algoritmi per l'analisi numerica sviluppati dalla comunità mondiale di esperti in calcolo scientifico, e sono disponibili a tutti secondo l'etica scientifica. Tali librerie, sviluppate principalmente in Fortran e C, sono

soggette ad un continuo processo di ottimizzazione ed aggiornamento, e costituiscono lo standard di riferimento del calcolo numerico.

- E' un ottimo sussidio didattico: a differenza dei programmi commerciali, permette di illustrare agli studenti anche il funzionamento del programma stesso. Si immagini l'utilità in corsi di informatica o di matematica applicata.

:. *Svantaggi*

E' parzialmente compatibile con il linguaggio di Matlab, questo perché in Octave mancano diversi tools presenti in Matlab per utilizzi particolari. Quest'ultimo programma è diffusamente utilizzato in tutto il mondo per scopi didattici e di ricerca. Ma poiché è proprietario, è impossibile studiare i dettagli dell'implementazione, e ciò diseduca gli allievi a chiedersi che cosa ci sia sotto il cofano degli strumenti che utilizzano, e getta un'ombra sulla verificabilità dei risultati scientifici con esso ottenuti. La transizione da Matlab a Octave è particolarmente agevole.

Capitolo II

Operazioni con Octave

Come salvare un file

Una sequenza di comandi, di chiamate a funzioni e di istruzioni di controllo di flusso di Octave può essere raccolta in un singolo *file* di testo con estensione . *m* in una *directory* accessibile da *Octave* e devono essere richiamate tramite

[VARIABILI DI OUTPUT]=NOME_FUNZIONE(VARIABILI DI INPUT)

Da quel momento in poi, è sufficiente digitare il nome del *file* (senza estensione) all'interno dell'ambiente Octave per eseguire la sequenza contenuta nel *file*. In questo modo è possibile costruire semplici programmi *script* per automatizzare operazioni ripetitive. Il comando

path mostra l'elenco delle cartelle in cui Octave cerca i *file script. m* in una *directory* accessibile da *Octave* e devono essere richiamate tramite

[VARIABILI DI OUTPUT]=NOME_FUNZIONE(VARIABILI DI INPUT)

La *directory* accessibile dal programma è quella in cui si sta lavorando oppure quella specificata tramite la definizione di un percorso a cui il programma farà riferimento per attingere funzioni e script. La sintassi

LOADPATH=[:"PERCORSO"];

può essere impostata direttamente nel file del programma che si sta eseguendo oppure può essere memorizzata nel file */usr/share/octave/site/m/startup/octaverc*, in modo da specificare una *directory* di default. Nel caso in cui si stia lavorando con file provenienti da *Matlab*, si può lasciare la sintassi usata da quest'ultimo, ovvero:

path(path,'PERCORSO);

il che risulterà di sicuro molto comodo. Infine, il linguaggio Octave mette a disposizione la possibilità di definire funzioni utente,che si comportano in tutto e per tutto come le funzioni matematiche predefinite. La sintassi è semplice e sufficientemente efficace, e la illustriamo con un esempio. Una funzione che calcola i numeri di Fibonacci fino all' N-esimo è la seguente:

```
## -- Numeri di Fibonacci: f = fibonacci(N)
## Restituisce nel vettore f i primi N numeri
## di Fibonacci
function f = fibonacci(N)
f=ones(1,N);
for i=3:N
f(i)=f(i-1)+f(i-2);
end
endfunction
```

Per mantenere la compatibilità con Matlab, vi è la curiosa limitazione di dover chiamare il *file* in cui si salva la funzione col nome della funzione stessa. Nel nostro caso il nome del *file* è, obbligatoriamente, fibonacci.m. Per il medesimo motivo, ogni funzione deve essere definita in un *file* a parte (è

possibile definire più funzioni all'interno di uno *script*, ma esse saranno poi accessibili solo all'interno dello *script*). I commenti prima della definizione della funzione (le linee precedute da ##) integrano il manuale *on-line* di Octave: digitando help fibonacci essi saranno mostrati sullo schermo. Infine, le funzioni di Octave possono essere ricorsive (cioè possono chiamare se stesse), e possono avere un numero variabile di argomenti sia in ingresso che in uscita.

Octave permette di attingere i dati da un file creato apposta o di stampare i risultati, anziché a video, in un file di testo. Per aprire un file in modalità scrittura o lettura si usa il comando

```
fopen("NOME_FILE","MODO","ARCH")
```

Se la differenza con *Matlab* in relazione alle virgolette è irrilevante (" o ' sono utilizzabili, in molti casi, indifferentemente senza produrre errori in *Octave*, al contrario di *Matlab* che usa la seconda scrittura), in *Octave* è necessario affiancare al *"MODO"*, una

```
"t"
```

per aprire il file in modalità testuale o da una

"b"

per aprirlo in modalità binaria (che è, tra l'altro, impostata di default). Quindi avremo, ad esempio:

"wt"

per la scrittura in modalità testuale,

"rb"

per la lettura in modalità binaria. Se si vuole richiedere il nome del file da cui prelevare i dati da tastiera, è necessario farsi dare il nome attribuendolo ad una variabile stringa e, tramite il comando

eval(VARIABILE)

farla eseguire come se fosse un programma:

```
VARIABILE=input('Nome file dati (senza estensione)=','s');eval(VARIABILE);
```

Inserire i commenti

Il commento è un testo che compare nel programma per spiegare cosa fa e come lavora per una maggiore comprensione del lettore.

Nel linguaggio Octave il commento è preceduto dal carattere “#” o dal simbolo “%” e continua fino alla fine della linea.

Esempio:

```
# descrizione variabile utilizzata
```

Comandi

Sono speciali classi di funzioni.

Un comando può essere chiamato come una funzione ordinaria, ma può anche essere richiamato senza parentesi. Esempio

my_command hello world

equivale a

my_command(“hello”,”world”)

Una funzione può essere usata come comando se accetta stringhe come argomenti di input. Per far ciò, la funzione deve essere definita come un comando, che può essere fatto con il comando “mark_as_command”.

mark_as_command name

dove name è la funzione che deve essere chiamata come un comando.

Una difficoltà si ha quando le stringhe di argomenti di input sono memorizzate in una variabile.

Siccome octave non può dire la differenza tra il nome di una variabile e una stringa ordinaria, non è possibile passare una variabile come input al comando.

Operazioni di base

I programmi scritti con octave sono degli script o liste di chiamate a funzioni. Il linguaggio di scripting dispone di diversi tipi di dati, oltre quello numerico esistono anche un tipo logico (true, false), uno di tipo stringa e strutture dati analoghe alle strutture del linguaggio c. esiste anche un altro tipo di dato molto potente ed è la matrice che consente molte operazioni di tipo matriciale.

La sintassi di base di Octave è molto simile a quella di Matlab. I commenti iniziano con il carattere %, ed il punto e virgola a fine comando inibisce la stampa dell’output di Octave in risposta al comando stesso.

Operazioni sulle matrici

Creare una Matrice

Il comando per creare una matrice e per salvarla è:

```
>> A = [1,2,12; 4,5,6; 7,8,9]
```

Le virgole separano gli elementi sulla stessa riga (colonne), mentre il “;”indica il passaggio alla riga successiva.

Octave risponderà visualizzando la matrice nella forma tradizionale (righe x

colonne)

A =

1 2 12

4 5 6

7 8 9

Nel linguaggio Octave la dimensione della matrice è determinata automaticamente quindi non è necessario esplicitarla. Inserendo per esempio una matrice:

```
a=[12
   34]
```

L’espressione

```
>>[a,a]
```

Produce la matrice:

ans

>>[1212

3434]

Bisogna stare attenti al modo in cui vengono inseriti i dati nelle matrici in quanto, per esempio, l’espressione:

[1-1]

Viene vista come differenza dando come risultato 0, mentre:

[1 -1]

Viene interpretata come un vettore [1,-1].

Se i valori della matrice sono molto grandi o molto piccoli attraverso la funzione

>>fixed_point_format

Possono essere visualizzati in un formato fisso. Questa funzione, però, non è raccomandabile in quanto può produrre un risultato che potrebbe essere mal interpretato.

Matrici vuote

Una matrice potrebbe avere una o entrambe le dimensioni (righe e colonne) pari a 0; in questo caso nel programma Octave sarà visualizzato:

[ ]

La funzione

>>val=print_empty_dimensions()

Controlla appunto tale comportamento.

Per esempio l’espressione

Zeros (3,0)

Stamperà

>>ans=[ ] (3*0)

La matrice vuota potrebbe essere anche un modo conveniente per cancellare

righe o colonne di matrici.

Calcolo Matriciale

Octave permette di effettuare operazioni su matrici. I simboli da usare sono i seguenti.

Matrice per uno scalare >>2*A

Moltiplicare 2 matrici A e B >>A*B

Matrice trasposta >>A’

Matrice inversa di A >>inv(A)

Determinante >>det(A)

Risolvere un’equazione lineare

Un sistema lineare in n incognite è così fatto

a11 x1 + a12 x2 + ... + a1n xn = b1 a21 x1 + a22 x2 + ... + a2n xn = b2 ... an1

x1 + an2 x2 + ... + ann xn = bn

E può essere scritto come Ax=B

Così che x=A-1*B

Risolvere il suddetto sistema in Octave è semplicissimo basta utilizzare l'operatore '\'.

```
>> A=[2,3,4;1,7,9;3,2,1]

A =

2 3 4

1 7 9

3 2 1

>> B=[1;7;3]

B =

1
```

7

3

>> A\B

ans =

-1.05000

4.30000

-2.45000

Dati numerici

All’interno del linguaggio Octave un numero decimale può essere espresso attraverso diverse notazioni.

Esempi:

>>105

>>1.05 e+2

>>1050 e-1

Per i numeri complessi l’espressione è:

>>3+4i

>>3.0+4.0e-1i

L’unità immaginaria i va scritta senza lasciare spazi fra il numero e la i.

La funzione >>double(x) converte x in un tipo a doppia precisione

La funzione >>single(val) converte il valore numerico val in un tipo a singola precisione

La funzione >>complex(val) e >>complex(re,im) trasformano un valore reale in numero complesso.

Range

L’espressione Range è definita dal valore del primo elemento nel range, dal valore dell’incremento tra gli elementi (se omesso è assunto pari a 1) e il massimo valore che gli elementi del range non eccederanno. Questi tre valori sono separati da colonne (il carattere “:”).

Esempio:

>>1:5 (incremento omesso)

>>ans [1,2,3,4,5,]

Esempio:

>>1:3:5

>>ans [1,4]

Valori logici

Il linguaggio di programmazione Octave permette di utilizzare valori logici come true o false il cui risultato sarà un valore logico che dipenderà dalla comparazione.

Altri operatori logici sono:

&= congiunzione logica

|= or

!= negazione logica

La funzione

>>logical (arg)

converte l’argomento in un valore logico.

Capitolo III

Le Variabili

Le variabili permettono di dare un nome ai valori e richiamarli dopo. Il nome di una variabile può essere una sequenza di lettere, di numeri o barre. Octave non dà un limite alla lunghezza dei nomi delle variabili, ma di solito non è utile avere variabili con nomi lunghi più di 30 caratteri.

Esempi di nomi di variabili validi sono: x, x15, _foo_bar_baz,….

Tuttavia nomi come foobarbaz che cominciano e terminano con l’undersore (_) possono essere intesi come riservati all’uso interno di Octave.

NB: in Octave i simboli a e A sono variabili distinte.

Il nome di una variabile è un’espressione valida da sé e rappresenta il valore corrente della variabile.

Ci sono poi variabili che hanno un significato a sé nel programma.

Per esempio:

ans tiene la directory di lavoro corrente pi indica il raggio di un cerchio Alcuni di questi simboli inseriti in Octave sono costanti e non possono essere modificati. Gli altri possono essere usati e assegnati come tutte le altre variabili ma i loro valori sono anche usati e modificati automaticamente da Octave.

Le variabili di octave non sono fisse, quindi è possibile prima memorizzare un valore numerico in una variabile e dopo usare lo stesso nome per mettere il valore nella stringa nello stesso programma. Le variabili non possono essere usate prima che gli sia stato assegnato un valore se no dà errore.

Esempio di variabile reimpostata:

>>isvarname(nome) #dà vero se nome è un nome di variabile valido

3.1 Variabili globali

Operatori aritmetici

X + Y addizione (se x e y sono matrici devono avere lo stesso numero di righe e di colonne)

X**.**+ Y addizione elemento per elemento

X-Y sottrazione (se x e y sono matrici devono averelo stesso numerodirighee di colonne)

X.- Y sottrazione elemento per elemento

X*Y moltiplicazione (se x e y sono matrici devono avere lo stesso numero di righe e di colonne)

X.* Y moltiplicazione elemento per elemento

X/Y divisione (concettualmente uguale a (inverse(Y’)*X’))

X./Y divisione elemento per elemento

X\Y divisione sinistra (concettualmente uguale a(inverse(X)*Y))

X.\Y divisione elemento per elemento

X^Y potenza tra matrici o scalari

X**Y

X.^Y potenza elemento per elemento

X.**Y

-X negazione

X’ trasposta

Operatori di comparazione

Gli operatori di comparazione danno come risultato 1 se la comparazione è vera, 0 se è falsa. Per le matrici gli operatori lavorano elemento per elemento.

Per esempio:

>> [1,2;3,4]= =[1,3;2,4]

>> ans 1 0

0 1

X<Y vero se x è minore di y

X<=Y vero se x è minore o uguale a y

X= = Y vero se x è uguale a y

X>=Y vero se x è maggiore o uguale a y

X>Y vero se x è maggiore di y

X !=Y vero se x è diverso da y

X<>Y vero se x è diverso da y

La funzione *isequal (x1, x2, …)* dà vero se tutti *x1, x2, …* sono uguali.

Operatori booleani elemento per elemento

Sono espressioni di comparazione che usano gli operatori booleani *or*, *and* (&), *not*(!). Un valore è falso se è 0, vero altrimenti (se è1).

Operatori di assegnazione

Un’assegnazione è un’espressione che memorizza un nuovo valore in una variabile.

Per esempio:

```
>>  z=1
```

Assegna il valore 1 alla variabile z.

“=” è chiamato operatore di assegnazione. L’assegnazione può assegnare anche stringhe.

Esempio:

```
>>  thing=”food”

>>  predicate=”good”
```

>> message=[“this”,thing,”is”,predicate]

Assegna il valore “this food is good” alla variabile *message* (qui è illustrate anche la concatenazione di stringhe). La parte sinistra di un’assegnazione non può essere una variabile. Può anche essere un elemento di una matrice o una lista di valori di output. Questi sono tutti chiamati *IVALUES* e significa che possono apparire sulla parte sinistra dell’operatore di assegnazione. Se viene assegnato uno scalare ad un amatrice indicizzata questo pone tutti gli elementi chiamati dall’indice uguali al valore dello scalare.

Per esempio, se a è una matrice con almeno 2 colonne:

>> a(:,2)=5

Pone tutti gli elementi nella seconda colonna di a uguali a 5. Assegnare una matrice vuota “[ ]” permette di cancellare righe o colonne della matrice o del vettore.

Per esempio, data una matrice A 4*5, l’assegnazione:

>> A(3,:)=[ ]

Cancella la terza riga di a e l’assegnazione:

```
>>  A(:,1:2:5)=[]
```

Cancella la prima, terza e quinta colonna.

Un’assegnazione è un’espressione quindi ha un valore. Infatti ponendo

```
>>  z=1
```

È un’espressione che ha valore 1. Una conseguenza di ciò è che è possibile fare assegnazioni multiple contemporaneamente.

Esempio:

```
>>  x=y=z=0
```

Assegna il valore 0 in tutte e tre le variabili.

Si può anche assegnare una lista di valori:

```
>>  [a,b,c]=[u,s,v]=svd (a)
```

Che è equivalente a:

```
>>  [u,s,v]=svd (a)
```

```
>>  a=u

>>  b=s

>>  c=v
```

Il numero di valori sul lato sinistro dell’espressione non può eccedere il numero di valori sul lato destro. Per esempio la seguente espressine produce errore:

```
>>  [a,b,c,d]=[u,s,v]=svd (a)

>>  error: element number 4 undefined in return list

>>  error: evaluating assignment expression near line 8, coloumn 15
```

Un’assegnazione comune è quella di incrementare una variabile esistente con un valore dato, come:

```
>>  a=a+2
```

Che può essere scritto in modo più compatto con l’operatore +=

```
>> a+=2
```

Operatori simili esistono anche per sottrazione, moltiplicazioni e divisioni.

Un'espressione della forma

>> expr1 op= expr2

È valutata come:

>> expr1=(expr1) op (expr2)

Dove *op* può essere un altro operatore come +, -, *, /.

Operatori di incremento

Incrementano o decrementano il valore di una variabile di 1. L’operatore di incremento è scritto “++”. Può incrementare la variabile prima o dopo che gli sia stato assegnato il valore. Per esempio, per preincrementare la variabile si scrive:

>> ++x

Aggiunge1 a x e dà il nuovo valore di x come risultato dell’espressione. E’ uguale all’espressione

>> x0x+1

Per postincrementare una variabile si scrive:

>> x++

Aggiunge 1 alla variabile x, ma da il valore che x aveva prima dell’incremento.

Per esempio, se x=2, il risultato dell’espressione x++ è 2 e il nuovo valore di x è 3.

Per gli elementi della matrice e del vettore, gli operatori di incremento e decremento lavorano su ogni elemento dell’operando. Espressioni di incremento e decremento:

++x incrementa la variabile x. Il valore dell’espressione è il nuovo valore di x. E’equivalente a x=x+1

--x decrementa la variabile x. E’ equivalente a x=x-1

X++ porta la x a essere incrementata. Il valore dell’ espressione è il vecchio valore di x

x-- porta la x a essere decrementata. Il valore dell’espressione è il vecchio

valore di x.

Operatori di precedenza

Determinano come gli operatori sono raggruppati, quando operatori diversi compaiono nella stessa espressione. Per esempio, * ha precedenza rispetto a 2+. Quindi l’espressione

```
>>  a+b*c
```

Significa moltiplicare b*c e aggiungere a al prodotto (a+(b*c)).

E’ comunque opportuno utilizzare le parentesi quando c’è una combinazione inusuale di operatori, in quanto chi legge il programma potrebbe non ricordare qual è la precedenza.

Capitolo IV

Gli statements

Possono essere una semplice espressione costante o una lista complicata di cicli e condizioni. Gli statements *di controllo* come *IF*, *WHILE* controllano il flusso di esecuzione dei programmi in Octave. Tutti gli statements di controllo iniziano con speciali parole chiave come *IF* e *WHILE* per distinguerli dalle espressioni semplici. Molti statements di controllo contengono altri statements; per esempio lo statement *if* contiene un altro statement che può o non può essere eseguito. Ogni statement di controllo ha un corrispondente *end* che segue la fine del controllo. Per esempio *endif* segue la fine dello statement *if* e *endwhile* segue la fine dello statement *while.* La lista degli statements presente fra if o while e il loro corrispondente end è chiamato il *corpo* dello statement di controllo.

Statement if

Prende le decisioni in octave. Ci sono 3 forme base di if.

Forma più semplice:

```
>>  if (condizione)

>>    Body

>>  endif
```

Condizione è un’espressione che controlla cosa fare nel resto dello statement. Il *body* è eseguito solo se la *condizione* è vera. La condizione in uno statement *if* è considerata vera se il suo valore è diverso da zero e falsa se il suo valore è zero. Se la condizione è un vettore o matrice è considerata vera solo se non è vuoto/a e tutti gli elementi sono diversi da zero.

Seconda forma:

```
>>  if  (condizione)

>>     body
```

```
>>  else
>>     else-body
>>  endif
```

Se *Condizione* è vera, *body* viene eseguito altrimenti è eseguito *elsebody*.

Esempio:

```
>>  if (rem(x,2)= =0)
>>      printf (“x is even\n”)
>>  else
>>      printf (“x is odd\n”)
>>  endif
```

In questo esempio se l’espressione rem (x,2)=0 è vera (quindi x è divisibile per 2), viene eseguito il primo *printf* altrimenti viene eseguito il secondo *printf*.

La terza forma di *if* permette di combinare decisioni multiple in un singolo statement.

Esempio:

```
>> if  (condizione)
>>      body
>> elseif (condizione)
>>    else-body
>> endif
```

Si possono inserire un qualsiasi numero di *elseif*. Ogni condizione è testata a turno e di quella considerata vera viene eseguito il *body*. Se nessuna condizione è vera e c’è l’*else* viene eseguito il suo *body*. Ci può essere solo un *else* e deve trovarsi nell’ultima parte dello statement. Tra *else* e *if* non ci devono essere spazi se no Octave lo tratta come un nuovo *if*.

Statement switch

Valuta differenti azioni che dipendono dal valore di una variabile. E’ possibile

utilizzando l’affermazione *if* nel seguente modo:

```
>>  if (x= =1)
>>     fai_ualcosa( );
>>  elseif (x= =2)
>>     fai_qualcos’_altro;
>>  else
>>    fai_qualcosa_diversa;
>>  endif
```

Questo codice può essere molto ingombrante sia da scrivere che da mantenere.

Per superare questo problema Octave prevede lo statement *switch* .

L’esempio sopra diventa:

```
>>  switch (x)
>>   case1
```

```
>>      fai_qualcosa;
>>   case 2
>>      fai_qualcos’_altro;
>>   otherwise
>>      fai_qualcosa_diversa;
>>  endswitch
```

Con questo codice la struttura ripetitiva del problema è più esplicita e il codice è più facile da leggere e da mantenere. Anche se x dovesse cambiare nome dovrebbe essere cambiata solo una linea.

La forma generale di *switch* è:

```
>>  switch espressione
>>      case label
>>              lista_comandi
>>      case label
```

```
>>            lista_comandi

>>….

>>      otherwise

>>            lista_comandi

>>  endswitch
```

Dove *label* può essere qualsiasi espressione. Uno dei vantaggi nell’usare *switch* invece di *If* è che le *labels* possono essere stringhe. Con if non si può scrivere:

```
>>  if (x= = “stringa”)  # non è valido
```

Con switch invece si può scrivere:

```
>>  switch (x)

>>     case “stringa”

>>                fai_qualcosa;

>>…..
```

>> endswitch

Statement while

Nella programmazione un ciclo è una parte di programma che è (o può essere eseguita) 2 o più volte in successione. *While* è lo statement di ciclo più semplice in Octave. Come con l’*if* , con *while* il ciclo viene eseguito fin quando la condizione è vera (valore diverso da zero).

La sua forma è:

```
>> while (condizione)
>>       body
>> endwhile
```

La prima cosa che fa *while* è testare la condizione. Se è vera esegue il *body*. Dopo che l’ha eseguito, la condizione è testata di nuovo e se è ancora vera il body è eseguito di nuovo. Questo processo è eseguito finché la condizione è vera. Se inizialmente la condizione è falsa il body non è mai eseguito.

L’esempio sotto crea una variabile *fib* che contiene i primi 10 elementi della sequenza di Fibonacci:

```
>>  fib=ones (1,10);

>>  i=3;

>>  while (i<=10)

>>        fib(i)=fib (i-1)+fib(i-2);

>>        i++;

>>  endwhile
```

Qui il corpo del ciclo contiene 2 statement. Il ciclo lavora così: prima i è posto uguale a 3. Poi while testa se è minore o uguale a 10. In questo caso i è uguale a 3 quindi il primo elemento di fib è dato dalla somma dei precedenti 2 valori nella sequenza. Poi i++ incrementa il valore di i e il ciclo si ripete. Termina quando i=11.

Statement do-until

E’ simile alchile solo che questo esegue rapidamente lo statement fino a che la condizione non è vera e il test della condizione è alla fine del ciclo, così il corpo del ciclo è sempre eseguito almeno una volta. Come con if la condizione è vera se diversa da zero e falsa se è uguale a zero.

L’espressione di *do-until* è :

```
>>  do

>>          body

>>  until (condizione)
```

Qui il body è uno statement o una lista di statements che sono chiamati *corpo del ciclo* e la condizione è l’espressione che controlla quante volte il ciclo sarà eseguito.

Questo esempio crea una variabile *fib* che contiene i primi 10 elementi della sequenza di Fibonacci:

```
>>  fib =ones (1,10);
```

```
>>  i=2;
>>  do
>>     i++;
>>     fib(i)=fib(i-1)+fib(i-2);
>>  until (i= =10)
```

Statement for

Il *for* è più utile in quanto può contenere iterazioni di cicli.

La forma generale è:

```
>>  for var=espressione
>>           body
>>  endfor
```

Dove *body* può essere uno statement o una lista di statements, *espressione* è

qualsiasi espressione valida e *var* può avere diverse forme. Di solito è un nome di variabile semplice o una variabile indicizzata. Se *espressione* è una struttura, *var* può anche essere un vettore con 2 elementi. L’assegnazione nel for lavora un po’ diversamente dalle normali assegnazioni in Octave. Invece di assegnare il risultato completo dell’espressione, assegna ogni colonna dell’espressione al *var* a turno. Se *espressione* è un range o un vettore riga o uno scalare, il valore di var sarà uno scalare ogni volta che il corpo del ciclo sarà eseguito. Se var è un vettore colonna o una matrice, var sarà un vettore colonna ogni volta che il corpo del ciclo è eseguito.

Il seguente esempio mostra un altro modo per creare un vettore che contenga i primi 10 elementi della sequenza di Fibonacci:

```
>>  fib=ones (1,10);

>>    for i=3:10

>>            fib(i)=fib(i-1)+fib(i-2);

>>    endfor
```

Questo codice analizza prima l’espressione 3:10, poi produce un range di

valori da 3 a 10 incluso. Dopo che la variabile i è assegnata al primo elemento del range il corpo del ciclo eseguito una volta. Raggiunta la fine del ciclo è assegnato un altro valore alla variabile i e il ciclo è eseguito di nuovo. Questo processo continua fin quando non ci sono più elementi da assegnare. Sebbene è possibile riscrivere tutti i cicli for con i cicli while il linguaggio Octave ha entrambi gli statements perché spesso il ciclo for permette di scrivere meno e viene più naturale pensarlo.

Statement break

Si trova dentro i cicli for o while e può essere usato solo nel corpo del ciclo. Il seguente esempio mostra il divisore più piccolo di un numero intero dato e identifica anche i numeri primi:

```
>>  num=103;

>>  div=2;

>>    while (div*div<=num)
```

```
>>          if (rem(num,div)= =0)
>>             Break;
>>          endif
>>  div++;
>>  if (rem(num,div)= =0)
>>      printf(“il più piccolo divisore di %d è %d/n”,num,div)
>>  else
>>      printf(“%d è prime\n”,num);
>>  endif
```

Quando il resto è zero nel primo while Octave *break* (interrompe) il ciclo e procede allo statement seguente e il processo continua. Questo è diverso da *exit* che ferma l’intero programma.

Statement continue

Come *break* è usato solo nei cicli for e while. Questo statement salta il resto del corpo del ciclo passando all’altro ciclo che inizia subito (invece break salta tutto il ciclo).

Esempio:

```
>>  # stampa gli elementi di un vettore casuale formato da numeri interi

>>  # crea un vettore di 10 numeri interi casuali con valori da 0 a 100

>>  vec=round(rand(1,10)*100);

>>  # stampa ciò che noi vogliamo

>>  for x=vec

>>              if (rem(x,2) !=0)

>>                  continue;

>>              endif
```

```
>>              printf(“%d\n”,x);
>>  endfor
```

Se uno degli elementi di vec è un numero dispari questo esempio salta la stampa dell’elemento e ritorna al primo statement nel ciclo.

Statement unwind_protect

Questo statement gestisce le eccezioni.

Forma generale:

```
>>  unwind_protect
>>       body
>>  unwind_protect_cleanup
>>       cleanup
>>  end_unwind_protect
```

Dove nody e cleanup sono entrambi opzionali e possono contenere qualsiasi espressione o commando di Octave. Gli statements in cleanup sono sicuramente eseguiti indipendentemente dall’esito del controllo di body. Questo è utile per proteggere cambiamenti temporanei della variabile globale da possibili errori. Per esempio, il seguente codice mostra la rassegnazione del valore originario alla variabile globale frobnositcate anche se si verifica un errore mentre si esegue l’operazione indicizzata.

```
>>  save_frobnositcate=frobnositcate;

>>    unwind_protect

>>            frobnositcate=true;

>>….

>>    unwind-protect_clenup

>>             frobnositcate=save_frobnositcate;

>>  end_unwind_protect
```

Senza unwind_protect il valore di frobnositcate non sarebbe riassegnato se si

verificasse un errore mentre si esegue l’operazione indicizzata perchè l’analisi si fermerebbe al punto dell’errore e lo statement riassegnerebbe il valore non eseguito.

Statement try

Questo statement gestisce le eccezioni.

Forma generale:

```
>>  try
>>     body
>>  catch
>>     cleanup
>>  end_try_catch
```

Dove body e cleanup sono entrambi opzionali e possono contenere espressioni e comandi di Octave. Lo statement in cleanup sarà eseguito solo se si verifica

un errore in body. Non sarà stampato nessun avviso o messaggio di errore mentre il body eseguito. Se riverifica un errore durante l’esecuzione del corpo, cleanup può usare la funzione *lasterr* per accedere al testo del messaggio che è stato stampato.

Linee di continuazione

Nel linguaggio Octave molti statments finiscono con un carattere di nuova linea e bisogna dire ad Octave di ignorarle per continuare lo statement da quella linea al testo. Le linee che finiscono con i caratteri “…” o “\” sono unite con la linea seguente prima che siano divise dal decodificatore di Octave.

Per esempio, le linee:

```
>>  x=long_variable_nome

>>    +longer_variable_nome\

>>  -42
```

Forma uno statement singolo. Il carattere backslash (\)sulla seconda linea è

interpretato come un carattere di continuazione non come un operatore di divisione.

Capitolo V

Analisi della Statistica descrittiva su Octave

Statistica descrittiva con Octave

Questo capitolo affronta le problematiche relative al calcolo della statistica descrittiva, sempre attraverso Octave.

Questo programma ha una serie di funzioni predefinite per gestire l'analisi statistiche anche se è un campo in pieno sviluppo.

E' comunque possibile effettuare analisi statistiche di base con un ampia gamma di funzioni di seguito elencate:

• **mean(x, opt, dim)**

Se x è un vettore, il calcolo della media degli elementi di x è il seguente:

mean (x) = SUM_i x(i) / N

Se x è una matrice, si calcola la media per ogni colonna e il risultato è un vettore riga.

L'argomento opzionale opt serve per determinare quale media calcolare, opt può assumere i seguenti valori:

a serie aritmetica

g serie geometrica

h serie armonica

L’argomento opzionale dim serve per definire la dimensione sulla quale si lavora.

• **median(x)**

Se x è un vettore, median(x) calcola il valore mediano degli elementi di x. Se gli elementi di x sono ordinati, la mediana è definita come:

$$\text{median}(x) = \begin{cases} x(\text{ceil}(N/2)), & N \text{ odd} \\ (x(N/2) + x((N/2)+1))/2, & N \text{ even} \end{cases}$$

• **std(x,opt)**

Se x è un vettore calcola la deviazione standard degli elementi di x

```
std (x) = sqrt (sumsq (x - mean (x)) / (n - 1))
```

Se x è una matrice calcola la deviazione standard per ogni colonna e il risultato lo restituisce in un vettore riga.

L’argomento opt determina il tipo di normalizzazione da usare. I valori validi sono:

- 0: normalizza con N-1, calcola la radice quadrata del migliore stimatore della varianza;
- 1: normalizza con N, calcola la radice quadrata del secondo momento intorno alla media.

• **var(x)**

Se x è un vettore, var calcola la varianza dei valori.

Se x è una matrice, dà come risultato un vettore riga contenente la varianza per ogni colonna.

L’argomento opt determina il tipo di normalizzazione da usare. I valori validi sono:

- 0: normalizza con N-1, calcola la radice quadrata del migliore stimatore della varianza;
- 1: normalizza con N, calcola la radice quadrata del secondo momento intorno alla media.

• **cov(x,y)**

Se ogni riga di x di y è una osservazione e ogni colonna è una variabile, la $\mathrm{cov}(x_i, y_j)$ è la covarianza tra l’i-esima variabile in x e la j-esima variabile in y.

Esempio

```
octave:1> x=[1  1  2  3  4  3  5  5  2  2  1]
x =
1  1  2  3  4  3  5  5  2  2  1

octave:2> mean(x)

ans = 2.6364
```

octave:3> median(x)

ans = 2

octave:4> std(x)

ans = 1.5015

• **corrcoef(x,y)**

Se ogni riga di x di y è una osservazione e ogni colonna è una variabile, la $corrcoef(x_i, y_j)$ è la correlazione tra l’i-esima variabile in x e la j-esima variabile in y.

corrcoef(x,y) = cov(x,y)/(std(x)*std(y))

octave:8> corrcoef(x,y)

ans = 0.56974

• **Kurtosis (x)**

Se x è un vettore di lunghezza N, l’indice di Curtosi si calcola nel seguente modo:

```
kurtosis (x) = N^(-1) std(x)^(-4) sum ((x mean(x)).^4)- 3
```

• **Skewness (x)**

Se x è un vettore di lunghezza N, l’indice di skewness si calcola nel seguente modo:

```
skewness (x) = N^(-1) std(x)^(-3) sum ((x - mean(x)).^3)
```

Capitolo VI

Grafici con Octave

Grafici di dati

Accade molte volte di avere a disposizione dei dati derivanti ad esempio da una analisi statistica o da un esperimento e di dover predisporre un grafico di questi dati. Supporremmo che tali dati siano memorizzati nel file dati. dat raggiungibile da gnuplot, in caso contrario nel richiamare il file dovremmo specificare la directory in cui esso è memorizzato. Si noti che il file dati. dat può essere composto da più colonne e il separatore di colonna deve essere tab.

Plotting bidimensionale

I comandi di base del plotting sono

>>axes()

>>axes(proprietà, valore, ..)

>>axes(h)

>>plot(args)

Questa funzione produce grafici bidimensionali. Sono possibili diverse combinazioni di argomenti. La più semplice è:

>>plot(y)

Dove l’argomento indica le coordinate x e y prese come indici degli elementi. Per salvare un grafico si usa il comando print. Se c’è più di un argomento è scritto:

>>plot(x,y,fmt,…)

Oppure

>>plot(x,y, proprietà, valore, ….)

Dove x, y, proprietà, valore sono opzionali e può esserci qualsiasi numero di argomenti.

I valori x e y sono interpretati come:

Se c’è un solo argomento le coordinate x e y sono considerate indici di elementi che iniziano con 1;

Se x è un vettore e y una matrice, le colonne (o righe) di y sono rappresentate rispetto a x;

Se y è un vettore e x una matrice, y è rappresentato rispetto alle colonne (o righe) di x;

Se entrambi gli argomenti sono vettori, gli elementi di y sono rappresentati rispetto a quelli di x;

Se entrambi gli argomenti sono matrici le colonne di y sono rappresentate rispetto a quelle di x. In questo caso entrambe le matrici devono avere lo stesso numero di righe e colonne.

Se entrambi gli argomenti sono scalari viene rappresentato un singolo punto.

Alcuni esempi di grafici a due dimensioni

Grafico a barre a due vettori:

```
> y=10 * rand (1,10)  dove rand rappresenta l’ampiezza delle classi

> x = (1:10)     dove 10 è il numero delle osservazioni

 x =

    1  2  3  4  5  6  7  8  9  10

> bar (x,y).
```

Grafico esponenziale

```
> x = (1:10)

> y = x.^2

> plot (x,y)
```

Grafici tridimensionali

- functionfile: plot3(args)

questa funzione produce grafici tridimensionali. Sono possibili diverse combinazioni di argomenti. La forma più semplice è:

```
>>plot3(x,y,z)
```

Dove gli argomenti rappresentano i vertici dei punti disegnati in 3 dimensioni.

Se tutti gli argomenti sono vettori della stessa lunghezza viene rappresentata una singola linea continua. Se sono tutti matrici, ogni colonna della matrice è trattata come una linea separata. Possono essere dati solo due argomenti:

>>plot3(x,c)

Dove le parti reale e immaginaria del secondo argomento sono considerate rispettivamente come le coordinate y e z.

Se è dato un solo argomento:

>>plot3(c)

Dove le parti reali e immaginarie dell’argomento sono considerati rispettivamente come valori y e z.

Per stampare un grafico si usa il comando print.

Per fare il grafico di gruppi di due o un argomento, si separa ciascun gruppo con una stringa vuota:

>>plot3(x1,c1,’’,c2,’’,….)

Inserire la griglia nel grafico

Per inserire la griglia nel grafico si usa il comando:

set grid

per eliminarla il comando

set nogrid

Aggiungere un titolo

Per aggiungere un titolo ad un grafico si usa il comando:

set title “titolodelgrafico“

Aggiungere etichette al grafico

Una delle potenzialità più elevate di gnuplot è quella di permettere di aggiungere delle etichette in qualunque punto del grafico. Ciò può essere effettuato con il comando:

set label n “etichetta“ at a,b,c

in cui:

• n rappresenta il numero dell’etichetta utile per il comando nolabel

• a,b,c sono le coordinate in cui vogliamo che l’etichetta sia inserita. Per levare una etichetta da un grafico si usa il comando:

set nolabel n

in cui n `e il numero dell’etichetta stampata in precedenza.

Manipolare grafici esistenti

-functionfile: axis(limits)

L’argomento “limits”dovrebbe essere un vettore di due, quattro o sei elementi.

Il primo e il secondo elemento indicano i limti inferiore e superiore per l’asse x. Il terzo e il quarto elemento per l’assey. Il quinto e il sesto per l’asse z.

Senza gli argomenti axis assegna in automatico l’ampiezza. Con un argomento

x=axis dà gli assi correnti. L’argomento limits è opzionale:possono essere usate stringhe nell’argomento per specificare le varie proprietà degli assi. Esempio:

>>axis([1,2,3,4],“square”);

Stampare il grafico

functiofile: print(filename,options)

stampa un grafico o lo stampa in un file.

Filename definisce il nome del file da stampare. Se non è specificato nessun file, l’output è stampato.