Laboratorio di Informatica T (Ch4)

Istruzioni di Iterazione: Cicli `for`

Cosa ci Serve per Programmare?

Il nostro obiettivo è formulare algoritmi, giusto?

Proviamo a capire se possiamo farlo con i costrutti visti finora

Somma degli Elementi di un Vettore

Proviamo a calcolare la somma degli elementi di un vettore

Come fare?

- La somma è inizialmente uguale a 0
- Consideriamo tutti gli elementi successivi
- Aggiungiamoli uno ad uno alla somma

Abbiamo gli strumenti per codificarlo?

Somma degli Elementi di un Vettore

Vediamo come usare queste informazioni per il nostro algoritmo

- Supponiamo che il nostro vettore si chiami `v`

```
v = % il nostro vettore
s = 0;
<facciamo variare una variabile ii da 1 a length(v)>
  s = s + v(ii)
```

- Quando `ii` avrà raggiunto il valore `length(v)`…
- …la variabile `s` conterrà la somma

Ci rimane da capire come gestire l'indice `ii`

P.S: Notate il ";" dopo `s = 0`:

- Aggiungere un ";" alla fine di una riga disabilita l'"eco"
- Matlab non stampa nulla: è comodo per programmi complessi

Istruzioni di Iterazione: Ciclo `for`

Matlab ci permette di gestire `ii` utilizzando una particolare istruzione

Una istruzione è una notazione che, quando viene eseguita, ordina all'elaboratore di fare qualcosa

La definizione è un po' vaga…

- Le istruzioni sono i "passi elementari" del programma
- Un programma è una sequenza di istruzioni

Un esempio: espressione + [INVIO] = una istruzione

Istruzioni di Iterazione: Ciclo `for`

Matlab ci permette di gestire `ii` utilizzando una particolare istruzione

Una istruzione è una notazione che, quando viene eseguita, ordina all'elaboratore di fare qualcosa

Ora vediamo un nuovo tipo di istruzioni:

Istruzioni di iterazione o "cicli": permettono di eseguire ripetutamente una porzione di codice

Istruzioni di Iterazione: Ciclo `for`

La nostra prima istruzione di iterazione si chiama "ciclo `for`"

Sintassi:

```
for <variabile> = <vettore>
  <corpo>
end % o anche endfor
```

- `<corpo>` è una sequenza di istruzioni
- Il corpo viene ripetuto per ogni elemento di `<vettore>`

Ad ogni ripetizione (i.e. iterazione):

- `<variabile>` assume il valore di un elemento di `<vettore>`

Nota: il simbolo "=" in questo caso non ha il solito significato…

Istruzioni di Iterazione: Ciclo `for`

Un esempio semplice:

```
for v = [2, 4, 6]
  v % Risp: ans = 2 (non c'è il ";")
    %       ans = 4
    %       ans = 6
end
```

Di solito, i cicli for compaiono in file di script

Se inserite una istruzione **`for`** nella finestra dei comandi:

- Matlab non esegue subito l'istruzione
- Vi permette di continuare a scrivere e premere [INVIO]
- Finché non inserite la parola chiave **`end`**

Solo allora l'istruzione **`for`** è completa (e quindi eseguibile)

Ciclo `for` e Somma di un Vettore

Il ciclo `for` ci permette di sommare gli elementi di `v`:

```
s = 0;
for ii = 1:length(v)
  s = s + v(ii)
end
```

Oppure, in modo ancora più semplice:

```
s = 0;
for w = v % w assume uno per uno i valori di v
  s = s + w
end
```

C'è una funzione predefinita che fa esattamente questo:

Si chiama `sum` è viene invocata con:

```
sum(<vettore>)
```

Cicli `for` Innestati

Somma di una Matrice

Alziamo un po' il livello di difficoltà

Supponiamo di voler sommare gli elementi di una matrice

Un possibile algoritmo:

```
M = % la nostra matrice
s = 0
<per ogni elemento della matrice>
  s = s + <elemento>
```

Come iterare sugli elementi di una matrice?

- Potremmo provare con un ciclo `for`…
- …del resto, ha funzionato nel caso precedente

Somma di una Matrice

Vediamo cosa succede iterando con un `for` su una matrice:

```
M = [1, 2, 3; 4, 5, 6; 7, 8, 9]; % [1, 2, 3;
                                  %  4, 5, 6;
                                  %  7, 8, 9]
for w = M
  w % giusto per stampare w
end
```

La variabile `w` assume i valori:

- `[1; 4; 7]`, poi `[2; 5; 8]`, poi `[3; 6; 9]`

Ossia le colonne di `M`!

Come possiamo fare per iterare sugli elementi?

Somma di una Matrice

Una soluzione semplice: usare due cicli for

```
M = [1, 2, 3; 4, 5, 6; 7, 8, 9];
for C = M    % itero sulle colonne
  for v = C  % poi sugli elementi della colonna
    v
  end % fine ciclo interno
end % fine ciclo esterno
```

- `for` è una istruzione…
- …Quindi il corpo di un `for` può contenere un altro `for`!

I due cicli si dicono innestati

- `end` si riferisce sempre all'ultimo `for` non ancora chiuso

Somma di una Matrice

Possiamo anche fare iterare due indici:

```
M = % la nostra matrice
s = 0;
[n, m] = size() % Recupero le due dimensioni
for ii = 1:n    % Itero sulle righe
  for jj = 1:m  % Itero sulle colonne
    s = s + M(ii, jj);
  end
end
```

Da notare:

`size(M)` restituisce un vettore `[<nrighe>, <ncolonne>]`…

- Ma se viene chiamato con due variabili a sx di =…
- …Restituisce due valori distinti

Riprenderemo questo comportamento nella prossima lezione

Somma di una Matrice

Possiamo anche fare iterare due indici:

```
M = % la nostra matrice
s = 0;
[n, m] = size() % Recupero le due dimensioni
for ii = 1:n    % Itero sulle righe
  for jj = 1:m  % Itero sulle colonne
    s = s + M(ii, jj);
  end
end
```

Da notare (2):

Vedete che il corpo di ogni `for` è leggermente indentato?

- Non è obbligatorio, ma rende il codice più leggibile
- Consiglio: mantenete sempre il codice ben indentato!

Somma di una Matrice

In alternativa, possiamo usare la funzione `sum`

Ci vuole però qualche accortezza:

```
M = [1, 2; 3, 4] % [1, 2]
                 % [3, 4]
sum(M) % denota [4, 6]
```

- Se applicata ad una matrice…
- …`sum` calcola la somma colonna per colonna

Per avere la somma di tutti gli elementi:

```
sum(sum(M)) % denota 10
```

- La prima applicazione di `sum` ottiene la somma della colonne
- La seconda applicazione ottiene la somma totale

Istruzioni Condizionali

Massimo di un Vettore

Supponiamo di voler calcolare il massimo di un vettore

Cosa riusciamo a scrivere con le istruzioni che conosciamo?

```
V = % il nostro vettore
y = V(1) % primo tentativo: max = il primo elemento
for ii = 2:length(V) % iteriamo sul resto del vettore
  <se V(ii) è maggiore di y, allora y = V(ii)>
end
```

Abbiamo bisogno di un costrutto che:

- Permetta di stabilire se `V(ii)` è maggiore di `y`…
- …Se questo è vero, esegua l'istruzione `y = V(ii)`

Questa funzionalità è fornita dalle istruzioni condizionali

Istruzione Condizionale `if`

Noi utilizzeremo una sola istruzione condizionale, chiamata `if`

Sintassi:

```
if <espressione>
  <corpo1>
else        % opzionale
  <corpo2> % opzionale
end
```

- `<espressione>` viene valutata ed interpretata come valore logico
- `<corpo1>` e `<corpo2>` sono sequenze di istruzioni
- `<corpo1>` esegue se `<espressione>` denota `true` (o `$\neq 0$`)
- `<corpo2>` esegue se `<espressione>` denota `false` (o `$0$`)

Intruzione `if` nel Nostro Algoritmo

Grazie all'istruzione `if` possiamo formulare il nostro algoritmo:

```
V = % il nostro vettore
y = V(1)
for ii = 2:length(V)
  if V(ii) > y
    y = V(ii)
  end
end
```

- L'indentazione non è necessaria, ma è fortemente consigliata

Ricordate:

- Se un programma è ben leggibile…
- …È più difficile fare errori!

Condizioni Complesse

Possiamo esprimere condizioni complesse con gli operatori logici

Per esempio:

```
if (x >= 3) & (x < 5) % true se x è tra 3 e 5
  <corpo>
end
```

E se dobbiamo verificare una condizione su un intero vettore?

- E.g. verificare se `V` contiene almeno uno `0`

Una prima soluzione

```
(V(1) == 0) | (V(2) == 1) | (V(3) == 0) ...
```

- È molto lungo da scrivere :-(
- Funziona solo per vettori di lunghezza fissa :-(

Funzioni `all` e `any`

Seconda soluzione: usare due funzioni predefinite:

```
all(<vettore>) % true se tutti gli elementi sono true
any(<vettore>) % true se almeno un elemento è true
```

- In pratica, sono un grande "and" e un grande "or"

Qualche esempio:

```
a = [1, 2, 3];
all(a < 3) % all([true, true, false]) --> false
any(a < 2) % any([true, false, false]) --> true
```

- I confronti `a < 3` e `a < 2` restituiscono vettori di valori logici
- `all` ed `any` lavorano su di essi

Se applicate a matrici, `all` ed `any` operano colonna per colonna

- Stesso comportamento di `sum`…
- E stessa soluzione (se serve un risultato unico): `any(any(...))`

Istruzione `break`, Programmazione Strutturata

Funzione Zeta di Riemann

Supponiamo di voler calcolare la somma della serie (con $s > 1$):

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

- Si tratta della funzione Zeta di Riemann (per num. reali)

Funzione Zeta di Riemann

Supponiamo di voler calcolare la somma della serie (con $s > 1$):

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

- Si tratta della funzione Zeta di Riemann (per num. reali)

Proviamo ad abbozzare un algoritmo:

```
z = 0;
for n = 1:???? % Il problema è qui
  z = 1 / n.^s
end
```

La serie ha infiniti termini!

- La funzione non è computabile in maniera esatta
- Però possiamo approssimarla!

Funzione Zeta di Riemann

Supponiamo di voler calcolare la somma della serie (con $s > 1$):

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

Un approccio tipico:

- Numero massimo di iterazioni (piuttosto alto)
- Stop quando si raggiunge un certo livello di precisione

Più nel dettaglio:

- Sia $\zeta^{(k)}(s)$ la nostra approssimazione dopo k iterazioni
- Ci possiamo fermare quando $\left|\zeta^{(k)}(s) - \zeta^{(k-1)}(s)\right| < \varepsilon$
- Dove ε è la tolleranza che riteniamo accettabile

Funzione Zeta di Riemann

Vediamo un possibile algoritmo (approssimato):

```
z = 0;                          % val. della somma
old_z = -Inf;                   % vecchio z
for n = 1:1e5                   % 1e5 = iterazioni massime
  z = z + 1 ./ n.^s;
  if abs(z - old_z) < 1e-6      % 1e-6 è la tolleranza
    break                       % Interrompe il ciclo
  end
  old_z = z;                    % rimpiazzo il vecchio z
end
```

- `Inf` è un valore speciale che denota ∞

L'istruzione **`break`**, quando eseguita, interrompe il ciclo corrente

Istruzione `break`

Un altro esempio con `break`: controllare se un vettore `v` contiene `0`

Istruzione `break`

Un altro esempio con `break`: controllare se un vettore `v` contiene `0`

- Un primo metodo (usando funzioni predefinite):

```
any(V == 0)
```

Istruzione `break`

Un altro esempio con `break`: controllare se un vettore `V` contiene `0`

- Un primo metodo (usando funzioni predefinite):

```
any(V == 0)
```

- Un secondo metodo (senza funzioni predefinite):

```
trovato = false;
for w = V
  if w == 0
    trovato = true;
  end
end
```

Istruzione `break`

Possiamo migliorare l'algoritmo:

- Appena troviamo `0`, non serve controllare il resto del vettore!
- Possiamo interrompere subito il ciclo, con `break`

```
trovato = false;
for w = V
  if w == 0
    trovato = true;
    break; % interrompe il ciclo corrente
  end
end
```

- L'istruzione `break` rende il codice un po' più efficiente

Programmazione Strutturata

Le istruzioni condizionali e di iterazione:

- Sono anche note come istruzioni di controllo di flusso
- Sono importanti per un particolare risultato:

> **Teorema del Programma Strutturato**: la possibilità di comporre istruzioni per sequenza, condizione ed iterazione è sufficiente a codificare qualunque algoritmo

- Composizione per sequenza = scrivere le istruzioni una dopo l'altra

Quasi tutti i linguaggi si basano su questi tre metodi di composizione

Esercizio: Prodotto di un Vettore

Esercizio: Prodotto di un Vettore

Estraete lo start-kit per questa lezione:

- Lavorare sul file `my_prod.m`
- Vedrete che gli script costruiscono spesso dati di esempio con:

```
rand(N)      % Matrice quadrata
rand(M, N)   % Matrice MxN
```

La funzione `rand`

- Restituisce matrici di numeri (pseudo) casuali in $]0, 1[$
- Ha la stessa interfaccia di `zeros`, `ones`, etc.

Analogamente, le funzioni:

```
randi(MAX, N)      % Matrice quadrata
randi(MAX, M, N)   % Matrice MxN
```

- Fanno lo stesso, ma con numeri interi tra `1` e `MAX`

Esercizio: Prodotto di un Vettore

Nel file di script `my_prod.m`

Calcolate il prodotto degli elementi del vettore `x`

- Matlab permette di farlo con:

```
prod(X)
```

- Confrontate i vostri risultati con quelli di `prod`

Esercizio: Prodotto Scalare

Esercizio: Prodotto Scalare

Nel file di script `my_dot.m`

Scrivere uno script che calcoli il prodotto scalare:

$$X \cdot Y = \sum_{i=1}^{n} X_i Y_i$$

- Dove n è la lunghezza dei due vettori

Matlab permette di calcolarlo in due modi:

```
X * Y' % Con X e Y vettori riga
dot(X, Y) % funzione dedicata
```

- Confrontate il vostro risultato con quello di Matlab

Esercizio: Identificazione di Numeri Primi

Esercizio: Identificazione di Numeri Primi

Matlab fornisce la funzione:

```
isprime(N)
```

- che individua se il numero **`N`** è primo

Nel file di script `my_isprime.m`:

- Definite un numero intero **`X`** da usare come esempio
- Determinate se il numero sia primo, mediante l'algoritmo seguente:

$$
\begin{aligned}
&\text{prime} = \text{true} \\
&\textbf{for } d = 2 \ldots \sqrt{X} \\
&\quad \textbf{if } x \mod d = 0 \\
&\qquad \text{prime} = \text{false} \\
&\qquad \text{break}
\end{aligned}
$$

- Se al termine dell'esecuzione **`prime`** vale ancora **`true`**…
- Allora il numero è primo

Esercizio: Identificazione di Numeri Primi

In matematica, l'operatore modulo:

$$z = x \mod y$$

- Denota il resto della divisione intera di `$x$` per `$y$`
- `x` è divisibile per `y` se e solo se il resto è `0`
- La divisione intera è quella che abbiamo imparato alle elementari!

Matlab permette di calcolare la divisione intera con:

```
idivide(x, y)  % Restituisce il quoziente
mod(x, y)      % Restituisce il resto
```

Esempio:

```
idivide(5, 2) % denota 2
mod(5, 2) % denota 1
```

Esercizio: Norma 2/Distanza Euclidea

Esercizio: Norma 2/Distanza Euclidea

Nel file di script `my_norm.m`

Scrivere uno script che calcoli la norma:

$$\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

- Dove n è la lunghezza del vettore

Matlab fornisce una funzione per calcolare la norma di un vettore:

```
norm(X, P) % usate help per i dettagli
```

- `X` è il vettore, `P` è l'ordine della norma
- Per `P = 2`, la funzione calcola la distanza euclidea dall'origine

Confrontate il vostro risultato con quello di Matlab

Esercizio: Matrice Diagonale

Esercizio: Matrice Diagonale

Matlab permette di costruire una matrice diagonale con:

```
diag(X)
```

- Restituisce una matrice diagonale
- Gli elementi sulla diagonale sono quelli del vettore **`X`**

$$\begin{pmatrix} x_1 & 0 & \dots & 0 \\ 0 & x_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & x_n \end{pmatrix}$$

Nel file di script `my_diag.m`:

- Costruite una matrice diagonale…
- …Che abbia il vettore **`X`** (fornito) come diagonale principale

Confrontate il vostro risultato con quello di Matlab

- Potere sempre usare **`rand`**/**`randi`** per ottenere dei vettori di test