Fondamenti di Informatica, A.A. 2011-2012

24/09/2012

Esercizio 1

È dato il frammento di codice Matlab

```
n = 3;
v = zeros(n,1);
w = zeros(1,n);
a = zeros(n,n);
for i=1:n
   v(i) = 2;
   w(i) = i;
   for j=1:n
      a(i,j) = (j-1)*n+i;
   end
end
a = a-v*w;
disp(a);
```

Si chiede cosa viene visualizzato dall'interprete Matlab.

Soluzione

La matrice `a` viene riempita per colonne con i numeri da 1 a 9:

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

Il prodotto tra il vettore colonna `v` ed il vettore riga `w` produce tre righe identiche, essendo `v` costante

```
   v*w
ans =

     2     4     6
     2     4     6
     2     4     6
```

Infine la sottrazione tra le due matrici produce il risultato finale:

```
   disp(a);
    -1     0     1
     0     1     2
     1     2     3
```

Esercizio 2

Discutere delle principali caratteristiche degli operatori su array di Matlab.

Soluzione

Gli operatori su array di Matlab seguono di norma le stesse regole delle operazioni matriciali in algebra lineare; pertanto tutti gli operatori relazionali, logici e gli operatori `+` e `-` operano elemento per elemento ed i loro operandi devono pertanto avere le stesse dimensioni. A parziale eccezione di questa regola, è sempre possibile usare come operando uno scalare insieme ad una matrice, ed in questo caso l'operazione viene intesa applicata a ciascun elemento della matrice rispetto allo scalare dato. Sono inoltre disponibili gli operatori di prodotto, divisione ed esponenziazione termine a termine, evidenziati dalla presenza del carattere `.` come prefisso, ad esempio

```
A .* B
```

effettuerà un prodotto termine a termine; anche in questo caso valgono le regole precendenti di coerenza delle dimensioni (ovvero di operando scalare).

L'operatore di prodotto invece segue le regole del prodotto di matrici, pertanto in una espressione

```
A * B
```

il numero di colonne di `A` deve essere uguale al numero di righe di `B`. Anche l'operatore di esponenziazione segue le regole dell'algebra lineare e quindi può essere applicato solo a matrici quadrate, in quanto si ottiene per prodotti ripetuti.

L'operatore di divisione a sinistra

```
A\B
```

corrisponde alla soluzione di un sistema lineare o ai minimi quadrati avente `A` come matrice dei coefficienti e le colonne di `B` come termini noti; pertanto il vincolo è che il numero di righe di `A` e `B` sia lo stesso; nei casi piu' comuni `A` è quadrata.

L'operatore di divisione a destra è definito utilizzando l'operatore di trasposizione `'` con l'usuale significato matematico, e per definizione vale

```
B/A = (A'\B')'
```

Esercizio 3

In una prova di esame di informatica viene richiesto di trovare, in un vettore contenente solo zeri ed uni, la dimensione della più lunga sequenza di zeri. Uno studente propone il seguente codice

```
count=0;
maxl=0;
for i=1:length(v)
    if (v(i)==1)
        if (count > maxl)
            maxl=count;
        end
        count=0;
    else
        count=count+1;
    end
end
disp("La massima lunghezza è");
disp(maxl);
```

Si chiede di trovare l'errore presente nel codice e di proporre una correzione adeguata.

Soluzione

Il problema del codice è che il riconoscimento della terminazione di una sottosequenza di zeri avviene quando si incontra un elemento di valore uno; questa strategia ignora il caso in cui l'ultimo elemento del vettore valga zero. Nel caso limite in cui tutti gli elementi del vettore siano zeri infatti il codice risponde che la massima lunghezza è zero. Una correzione possibile è la seguente:

```
count=0;
maxl=0;
for i=1:length(v)
    if (v(i)==1)
        if (count > maxl)
            maxl=count;
        end
        count=0;
    else
        count=count+1;
    end
end
if (v(length(v))==0)
    if (count > maxl)
        maxl=count;
    end
end
disp("La massima lunghezza è");
disp(maxl);
```

Esercizio 4

È data la seguente funzione Matlab

```
function [x]=jacobi(a,x,b,thr,imax)
  n=length(x);
  if ((size(a,1) ~= n)||(size(a,2) ~= n))
     disp('size mismatch');
     return
  end
  d=diag(a);
  f=d.\b;
  m=-(a-diag(diag(a)));
  r=b-a*x;
  i=0;
  while ((i<imax)&&(norm(r)>thr))
     x=d.\(m*x)+f;
     r=b-a*x;
     i=i+1;
  end
end
```

Si stimi il numero di operazioni aritmetiche eseguite, assumendo come parametro il numero di iterazioni compiute nel ciclo `while`; si assuma inoltre che l'operatore `norm` su un vettore di lunghezza n abbia costo $2n$. Si ricorda che l'operatore `diag` applicato ad una matrice estrae la diagonale, invece applicato ad un vettore costruisce una marice avente il vettore dato come diagonale e zeri altrove.

Soluzione

Secondo le indicazioni del testo, ipotizziamo che il ciclo `while` venga eseguito per k volte; al costo del ciclo occorrerà aggiungere il costo della preparazione dei dati. In termini di operazioni aritmetiche avremo

```
f=d.\b;
```

che comporta la esecuzione di n operazioni (divisione elemento per elemento tra vettori),

```
m=-(a-diag(diag(a)));
```

che comporta n^2 operazioni (sottrazione elemento per elemento tra matrici), e

```
r=b-a*x;
```

di costo $2n^2$, in quanto composto da un prodotto matrice-vettore di costo $2n^2 - n$ e una sottrazione di vettori elemento per elemento di costo n. Il costo totale della preparazione dei dati è quindi $3n^2 + n$. All'interno del ciclo vengono eseguite le seguenti operazioni:

```
x=d.\(m*x)+f;
```

consta di un prodotto matrice-vettore `m*x` di costo $2n^2 - n$, una divisione elemento per elemento `d.\(` di costo n ed una addizione elemento per elemento `+f` di costo n; il calcolo di

```
r=b-a*x;
```

come già visto costa $2n^2$. Al costo del corpo del ciclo va aggiunto il costo $2n$ della istruzione `norm` impiegata per controllare la condizione di uscita. In definitiva una iterazione del ciclo costa $4n^2 + 3n$; quindi il costo totale dell'algoritmo è

$$k(4n^2 + 3n) + 3n^2 + n \approx (4k + 3)n^2.$$

Esercizio 5

È dato il frammento di codice Matlab

```
i=1;
so=0;
se=0;
while (i<=length(v))
    if (mod(i,2)==1)
        so = so + v(i)^2;
    else
        se = se + v(i)^3;
    end
    i = i + 1;
end
```

Riscrivere il codice facendo uso del ciclo `for` e senza usare la istruzione `if`. Bonus: riscrivere usando solo operazioni su array.

Soluzione

Il codice proposto utilizza il ciclo `while` per scorrere gli elementi del vettore `v` separando gli elementi di posto pari da quelli di posto dispari. Pertanto per rispondere alla richiesta occorrerà utilizzare un ciclo `for` sugli elementi dispari ed uno sui pari, come segue

```
so=0;
for i=1:2:n
    so = so + v(i)^2;
end
se=0;
for i=2:2:n
    se = se + v(i)^3;
end
```

Lo stesso risultato può essere ottenuto con le espressioni:

```
so=sum(v(1:2:n).^2);
se=sum(v(2:2:n).^3);
```