Esercitazioni sulla complessità degli algoritmi

28 novembre 2007

1 Esercitazioni sulla complessità degli algoritmi

1.1 Calcolo del determinante di una matrice

Il calcolo di un determinante di una matrice quadrata $A = (a_{i,j})$ di ordine n può essere eseguito usando la regola di Laplace. Sfortunatamente, come è già noto manualmente per matrici di piccole dimensioni, il numero di operazioni additive e moltiplicative da eseguire è particolarmente elevato. Ha quindi senso cercare delle procedure alternative che ovvino a questo problema. Vediamone di seguito un esempio.

1.1.1 Calcolo del determinante di una matrice via eliminazione gaussiana

Si implementi una function per il metodo dell'eliminazione gaussiana con pivoting parziale (per righe) per il calcolo del determinante di una matrice. La function deve restituire in output il determinante e la matrice triangolare superiore che si ottiene come risultato del metodo; deve saper gestire anche il caso del pivoting nullo (matrice singolare). La si applichi a tre matrici di numeri casuali di ordine rispettivamente 10, 25 e 50 e la si confronti con l'apposita funzione di Matlab/Octave per il calcolo del determinante.

1.1.2 Risoluzione

Per generare una matrice di numeri casuali di ordine n si usa il comando

```
A=rand(n);
```

Salviamo la function richiesta in un file `meg.m`

```
function [U, deter]=meg(A)
n=size(A,1);
U=A;       % INIZIALIZZAZIONI.
deter=1;
for k=1:1:n-1
    [piv, j]=max(abs(A(k:n,k))); % PIVOTING.
```

```
    if (piv == 0)    % CASO DETERMINANTE 0.
       deter=0;
       return
    end

    if (j ~=1)       % SCAMBIO RIGHE.
       temp=A(j+k-1,:);
       A(j+k-1,:)=A(k,:);
       A(k,:)=temp;
       deter=-deter;
    end

    % deter=deter*A(k,k);

    % ELIMINAZIONE GAUSSIANA.

    for index=k+1:1:n
        m(index,k)=A(index,k)/A(k,k);
        A(index,k)=0;
        for j=k+1:1:n
            A(index,j)=A(index,j)-m(index,k)*A(k,j);
        end
    end

end

U=A;
det_U=prod(diag(U)); % CALCOLO DETERMINANTE
                          % MATRICE FINALE "U".
deter=deter*det_U;
```

Qualche commento sul codice:

1. Nel ciclo for `for k=1:1:n-1`, il codice calcola l'elemento $a_{j,k}$ con $j \geq k$ che sia più grande in modulo. Facciamo un esempio sull'uso della funzione `max` in Matlab:

```
>> u=[100; 20; 100; 30]
u =
   100
    20
   100
    30
>> [s,t]=max(u)
s =
   100
t =
     1
>>
```

La variabile s descrive il massimo valore tra le componenti del vettore u, mentre t dice in quale indice del vettore viene assunto. Osserviamo che u ha il suo massimo nella prima e nella terza componente di u, ma che di default, in Matlab/Octave viene scelto quale indice t il più piccolo intero positivo per cui tale massimo viene assunto (nell'esempio $t = \min(1,3) = 1$).

Questa considerazione sulla funzione `max` di Matlab/Octave ha dei riflessi sull'algoritmo `meg.m`. Qualora il massimo di $|a_{j,k}|$ con $j \geq k$ sia assunto in più indici, tra questi viene scelto il minore.

2. Nella porzione di codice

```
if (piv == 0)
   deter=0;
   return
end
```

si stabilisce che se il massimo $|a_{j,k}|$ con $j \geq k$ è uguale a 0, allora il determinante di A è 0. Il comando `return` blocca immediatamente la funzione e assegna ad `U` e `deter` i valori fino allora assegnati. Per avere un'idea perchè ciò succeda facciamo un esempio:

```
>> A=[3 4 5 6 7; 0 8 9 1 2; 0 0 0 1 6; 0 0 0 4 9; 0 0 0 5 2]
A =
     3     4     5     6     7
     0     8     9     1     2
     0     0     0     1     6
     0     0     0     4     9
     0     0     0     5     2
>> det(A)
ans =
     0
>>
```

Le prime 3 colonne generano un sottospazio di $\mathbb{R}^5$ di dimensione 2. Quindi i vettori $(3,0,0,0,0)$, $(4,8,0,0,0)$, $(5,9,0,0,0)$ sono linearmente *dipendenti* e conseguentemente il determinante della matrice A è nullo. Questa idea si estende al caso generale. Se tutte le componenti $a_{j,k}$ con $j \geq k$ sono nulle, le prime k colonne generano un sottospazio di dimensione $k-1$ e quindi sono linearmente dipendenti. Di conseguenza il determinante di a è 0.

3. Nel blocco

```
if (j ~=1)
   temp=A(j+k-1,:);
   A(j+k-1,:)=A(k,:);
   A(k,:)=temp;
   deter=-deter;
end
```

si nota che $a_{j+k-1,k} \geq a_{s,k}$ per $s = k,\ldots,n$ e quindi si scambiano la riga $j+k-1$-sima con la k-sima, tenendo in mente che lo scambio di righe produce una matrice A' il cui determinante ha valore assoluto uguale a quello di A ma segno opposto. Vediamone un esempio:

```
>> A=[1 2; 3 4]
A =
     1     2
     3     4
>> B=A([2 1],[1 2])
B =
     3     4
     1     2
>> det(A)
ans =
    -2
>> det(B)
ans =
     2
>>
```

4. La porzione di codice

```
U=A;
det_U=prod(diag(U)); % CALCOLO DETERMINANTE
                           % MATRICE FINALE "U".
deter=deter*det_U;
```

è più complicata di quello che si creda. Vediamo su un esempio cosa succede di A, al variare di k.

```
>> A=[1 4 2 6; 3 2 5 7; 1 3 8 6; 1 3 5 6]
A =
     1     4     2     6
     3     2     5     7
     1     3     8     6
     1     3     5     6
>> [U, deter]=meg(A)
k =
     1
A =
    3.0000    2.0000    5.0000    7.0000
         0    3.3333    0.3333    3.6667
         0    2.3333    6.3333    3.6667
         0    2.3333    3.3333    3.6667
k =
     2
```

```
A =
    3.0000    2.0000    5.0000    7.0000
         0    3.3333    0.3333    3.6667
         0         0    6.1000    1.1000
         0         0    3.1000    1.1000
k =
     3
A =
    3.0000    2.0000    5.0000    7.0000
         0    3.3333    0.3333    3.6667
         0         0    6.1000    1.1000
         0         0         0    0.5410
U =
    3.0000    2.0000    5.0000    7.0000
         0    3.3333    0.3333    3.6667
         0         0    6.1000    1.1000
         0         0         0    0.5410
deter =
  -33.0000
>>
```

La matrice U ha un determinante uguale a quello di A a meno del segno. Visto che la differenza di segno tra A ed U è tenuta *sotto controllo* nella parte relativa al pivoting (controllare!), non resta che calcolare il determinante di U.

Ricordando che `diag` applicato a una matrice $A = (a_{i,j})$ fornisce un vettore $u = (u_k)$ tale che $u_k = a_{k,k}$, e che `prod` applicato ad un vettore u esegue $\prod_k u_k$, visto che il determinante di una matrice triangolare superiore coincide per la regola di Laplace al prodotto degli elementi diagonali, si ha che in effetti il determinante di U è dato da `prod(diag(U))`.

Alternativamente si poteva togliere il blocco sopramenzionato

```
U=A;
det_U=prod(diag(U)); % CALCOLO DETERMINANTE
                           % MATRICE FINALE "U".
deter=deter*det_U;
```

e inserire

```
deter=deter*A(k,k);
```

tra lo scambio di righe e l'eliminazione gaussiana. Vediamo perchè . Nell'esempio fatto poco fa, per $k = 1$ si esegue il pivoting per colonne, mentre negli altri casi la strategia non comporta scambi di righe (perchè ?). A partire dalla matrice iniziale A, si determinano delle matrici $A^{(k)}$, il cui determinante coincide a meno del segno con quello di A. Notiamo che fissato $\bar{k}$, la matrice $A^{(\bar{k},\bar{k})} =$

$(A^{(\bar{k})}_{i,j})$ con $i = 1,\ldots,\bar{k}$, $j = 1,\ldots,\bar{k}$ non viene più modificata nei *passi successivi* in cui $k > \bar{k}$ (perchè ?) ed è triangolare superiore. Quindi il determinante di $A^{(\bar{k},\bar{k})}$ è uguale a quello di $A^{(\bar{k}-1,\bar{k}-1)}$ moltiplicato per $A^{(\bar{k})}_{\bar{k},\bar{k}} = A^{(\bar{k},\bar{k})}_{\bar{k},\bar{k}}$. Alla fine del processo, la matrice $A^* = A^{(n)}$ ottenuta dalle varie traformazioni, è triangolare superiore ed ha quali elementi diagonali proprio $A^{(k)}_{k,k}$ con $k = 1,\ldots,n$. Quindi il suo determinante è

$$\prod_{k=1}^{n} A^{(k)}_{k,k} = \prod_{k=1}^{n} A^{(k,k)}_{k,k}$$

che a meno del segno coincide con il determinante della matrice A.

5. L'ultimo blocco

```
for index=k+1:1:n
    m(index,k)=A(index,k)/A(k,k);
    A(index,k)=0;
    for j=k+1:1:n
        A(index,j)=A(index,j)-m(index,k)*A(k,j);
    end
end
```

   esegue le operazioni richieste dalla eliminazione di Gauss [6].

6. Osserviamo che la chiamata della function `meg(A)` non calcola il determinante. Per convincerci digitiamo nella shell di Matlab/Octave:

```
>> A=rand(3)
A =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214
>> meg(A)
ans =
    0.9501    0.4860    0.4565
         0    0.7731   -0.0925
         0         0    0.5839
>>
```

   Il fatto è che la function `meg` ha due variabili di output, e dalla chiamata `meg(A)` nessuna di queste viene specificata. Il corretto utilizzo è quindi

```
>> [U,deter]=meg(A)
U =
    0.9501    0.4860    0.4565
         0    0.7731   -0.0925
         0         0    0.5839
```

```
deter =
    0.4289
>> det(A)
ans =
    0.4289
>>
```

Nota. Osserviamo che in algebra lineare è naturale usare indici quali i, j, k eccetera. Purtroppo bisogna tener conto che le ultime release di Matlab non permettono l'uso di i quale indice in quanto per default i corrisponde con il relativo numero complesso.

Esercizio per casa. Si assegni `A=rand(n)` per $n = 10, 25, 50$ e quindi si digiti sulla shell di Matlab/Octave

```
[U,deter]=meg(A); s=deter;
```

Di seguito si confronti il valore di `s` con quello di `t=det(A)` (ricordiamo che `det` è un comando Matlab che calcola il determinante di una matrice). Quante cifre hanno `s` e `t` in comune? Qual'è l'errore relativo $e_{\text{rel}} := |s-t|/|t|$? E quello assoluto $e_{\text{abs}} := |s-t|$?

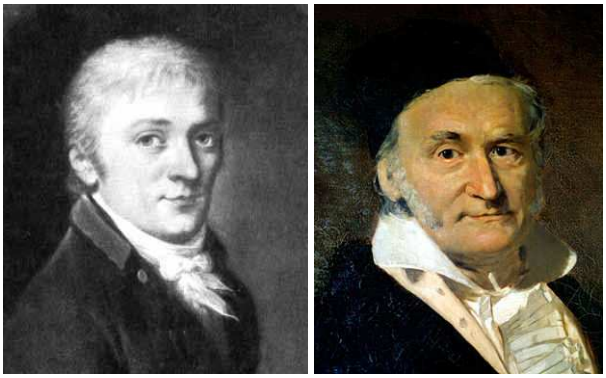

Figura 1: Carl Friedrich Gauss (1777-1855).

1.2 Calcolo della potenza di una matrice

Per calcolare la potenza p-esima di una matrice quadrata A di ordine n cioè

$$A^p := \underbrace{A * \ldots * A}_{p \text{ volte}}$$

senza usare l'operatore di elevamento a potenza $\wedge$, si può implementare il seguente algoritmo (pseudocodice)

```
B=I;
for i=1:p
    B=B*A;
end
```

in cui I è la matrice identica di ordine n e $*$ è il classico prodotto tra matrici.

Alternativamente (in maniera più stabile ed efficiente) si può decomporre p come

$$p = \sum_{i=0}^{M} c_i 2^i$$

ove $M = \lfloor \log_2 p \rfloor$ e $c_i = 0$ oppure $c_i = 1$. Si osserva facilmente che questa non è altro che la classica rappresentazione di p in base 2. Usando la proprietà della potenze

$$B = A^p = A^{\sum_{i=0}^{M} c_i 2^i} = \prod_{i=0}^{M} \left(A^{2^i}\right)^{c_i}$$

ove ogni termine A^{2^i} può essere calcolato come $A^{2^{i-1}} A^{2^{i-1}}$.

Confrontiamo i due metodi per $p = 6$. Nel primo si calcola A^6 come

$$A^6 = A * A * A * A * A * A$$

e quindi sono necessari 5 prodotti tra matrici. Nel secondo caso essendo $6 = 0 * 2^0 + 1 * 2^1 + 1 * 2^2$ si ha

$$A^6 = (A^2) * (A^4).$$

Calcolati $A^2 = A * A$ ed in seguito $A^4 = (A^2) * (A^2)$, abbiamo finalmente A^6 con solo 3 prodotti tra matrici ma con lo storage addizionale di alcune matrici in memoria.

Esercizio per casa. Si implementino i due algoritmi proposti per il calcolo della potenza di matrice tramite due functions (senza usare l'operatore $\wedge$) e si calcoli l'errore relativo in norma infinito rispetto all'elevamento a potenza di Matlab o GNU Octave per diverse matrici e potenze ($n = 5, 10, 15, \ldots, 50$ e $p = 5, 10, 15, \ldots, 50$). Si confrontino poi i tempi di esecuzione delle due functions per il calcolo di A^{100}, con A matrice di numeri casuali di ordine 200.

Suggerimento: ci si aiuti col seguente pseudocodice:

```
p=100; n=200;
c=trasforma_in_binario(p);
A=rand(n);
B=eye(n);
C=A;
M=floor(log2(p));

% B contiene la potenza di A finora calcolata.
% C contiene la potenza A^(2^index)) finora calcolata.
```

```
for index=0:M
   j=index+1;
   if c(j) == 1
      B=B*C;
   end
   C=C*C;
end
```

1.3 Commenti all'esercitazione

1. Il comando che produce la matrice identità di ordine n è `eye(n)`. Se in particolare si desidera produrre una matrice identica dello stesso ordine di una matrice A si usi il comando Matlab `eye(size(A))`.

2. Un codice che produce la decomposizione in potenze di 2 di un numero p è il seguente:

   ```
   q=p
   M=floor(log2(p))+1;
   c=[];
   for i=1:1:M
       c(i)=mod(q,2);
       q=floor(q/2);
   end
   ```

   Osserviamo la presenza del comando `c=[]`, in quanto, qualora si testino i metodi con p diversi, può restare in memoria un precedente vettore c che compromette la correttezza del codice. Si testi tale codice per la rappresentazione di 22 quale numero binario.

3. La norma infinito della differenza di due matrici A e B si ottiene con comando

   `norm(A-B,inf)`

   Ricordiamo che se $A = \{a_{i,j}\}$ allora la norma infinito di A si definisce come

   $$\|A\|_\infty = \max_i \sum_j |a_{i,j}|.$$

   L'utilità di introdurre la norma infinito è legato alla verifica dei metodi implementati. Sia $S = A^p$ e supponiamo A sia una matrice di ordine piccolo (ad esempio 7) e similmente l'esponente p non sia troppo grande (ad esempio 6). Se B è l'approssimazione fornita dal metodo allora si valuta la norma $\|S-B\|_\infty$. Se questa è piccola allora l'implementazione è corretta. Vediamo un esempio in Matlab:

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

```
A =
     1     2
     3     4
>> E=rand(2)*10^(-5);
>> B=A+E;
>> format long
>> B
B =
   1.00000950129285   2.00000606842584
   3.00000231138514   4.00000485982469
>> norm(B-A,inf)
ans =
  1.5570e-005
```

Quindi si due matrici A e B sono *vicine* allora

$$\|B - A\|_\infty$$

è piccola. Citiamo di seguito le norme più comuni di un vettore $x = (x_i)_{i=1,\dots,n}$ e una matrice $A = (a_{i,j})_{i,j=1,\dots,n}$ (cf. [1, p.21, p.25]):

- **Norma 1**: $\|x\|_1 = \sum_{i=1}^n |x_i|$, $\|A\|_1 = \max_j \sum_{i=1}^n |a_{i,j}|$;
- **Norma 2**: $\|x\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2}$, $\|A\|_2 = \sqrt{\rho(A^T * A)}$ dove $\rho(A)$ è il massimo modulo di un autovalore di A;
- **Norma** ∞: $\|x\|_\infty = \max_i |x_i|$, $\|A\|_\infty = \max_i \sum_{j=1}^n |a_{i,j}|$;

4. Per capire il funzionamento del secondo metodo, introduciamo un esempio esplicativo. Sia A una matrice di ordine 7 e $p = 22$. Il valore assunto da c è quindi [0 1 1 0 1] poiché, come si legge dalla rappresentazione binaria (da sinistra verso destra),

$$22 = 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4.$$

Notiamo che il vettore c ha lunghezza $M+1 = \texttt{floor}(log2(22))+1 = 5$. Vogliamo quindi calcolare

$$B = (A)^0 * (A^2)^1 * (A^4)^1 * (A^8)^0 * (A^{16})^1. \tag{1}$$

Vediamo il processo passo passo.

1. Al primo passo, dobbiamo calcolare $B = (A)^0$. Poniamo $C = A$. Come suggerito, se inizialmente $B = I_7$, a questo punto non c'è nulla da fare poichè A ha ordine 7 e $A^0 = I_7$. A questa altezza abbiamo immagazzinato B, C ed A.

2. Al secondo passo, ricordando (1), dobbiamo moltiplicare il valore di B ottenuto al passo precedente per $(A^2)^1$. Essendo $C = A$ osserviamo che $A^2 = A * A = C * C$ e quindi posto il nuovo C uguale a $C * C$ (per cui $C = A^2$) si ha $B = B * C$. A questa altezza abbiamo immagazzinato B, C ed A.

3. Al terzo passo, ricordando (1), dobbiamo moltiplicare il valore di B ottenuto al passo precedente per $(A^4)^1$. Essendo $C = A^2$ osserviamo che $A^4 = A^2 * A^2 = C * C$ e quindi posto il nuovo C uguale a $C * C$ (per cui $C = A^4$) si ha $B = B * C$. A questa altezza abbiamo immagazzinato B, C ed A.

4. Al quarto passo, ricordando (1), dobbiamo moltiplicare il valore di B ottenuto al passo precedente per $(A^8)^0$. Essendo $C = A^4$ osserviamo che $A^8 = A^4 * A^4 = C * C$. Poniamo il nuovo C uguale a $C * C$ (e quindi $C = A^8$). A questo punto non c'è nulla da fare poichè $(A^8)^0 = I_7$ e quindi $B = B * I_7$. A questa altezza abbiamo immagazzinato B, C ed A.

5. Al passo $M = 5$, ricordando (1), dobbiamo moltiplicare il valore di B ottenuto al passo precedente per $(A^{16})^1$. Essendo $C = A^8$ osserviamo che $A^{16} = A^8 * A^8 = C * C$ e quindi posto il nuovo C uguale a $C * C$ (per cui $C = A^{16}$) si ha $B = B * C$. A questa altezza abbiamo immagazzinato B, C ed A.

Notiamo, che una volta finito il processo, A^{16} è immagazzinata in B.

Infine, per calcolare il tempo macchina di esecuzione, ci si può aiutare con l'`help` del comando `tic` o `cputime`.

2 Materiale disponibile online

Si può trovare online materiale relativo a questa lezione e sue integrazioni. Citiamo in particolare [7] e quali argomenti correlati, ma non trattati direttamente in questa lezioni [5], [8]. Per quanto riguarda la biografia di Gauss un ottimo link in italiano è [6], in inglese [4].

3 Frasi celebri

1. When a philosopher says something that is true then it is trivial. When he says something that is not trivial then it is false. (Gauss)

2. ... through systematic, palpable experimentation [dopo avergli chiesto come scopriva i suoi teoremi]. (Gauss)

3. I mean the word proof not in the sense of the lawyers, who set two half proofs equal to a whole one, but in the sense of a mathematician,where half proof is nothing, and it is demanded for proof that every doubt becomes impossible. (Gauss)

4. Ask her to wait a moment. I am almost done. [avvisato della moglie morente mentre lui scopriva un teorema]. (Gauss)

5. Theory attracts practice as the magnet attracts iron. (Gauss)

6. The total number of Dirichlet's publications is not large: jewels are not weighed on a grocery scale. (Gauss)

7. You know that I write slowly. This is chiefly because I am never satisfied until I have said as much as possible in a few words, and writing briefly takes far more time than writing at length. (Gauss)

References

[1] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.

[2] The MathWorks Inc., *Numerical Computing with Matlab*, `http://www.mathworks.com/moler`.

[3] A. Quarteroni e F. Saleri, *Introduzione al calcolo scientifico*, Springer Verlag, 2006.

[4] MacTutor History of Mathematics archive, `http://www-history.mcs.st-andrews.ac.uk/Biographies/Gauss.html`.

[5] Wikipedia: `http://it.wikipedia.org/wiki/Fattorizzazione_LU`.

[6] Wikipedia: `http://it.wikipedia.org/wiki/Gauss`.

[7] Wikipedia: `http://it.wikipedia.org/wiki/Algoritmo_di_Gauss-Jordan`.

[8] Wikipedia, `http://it.wikipedia.org/wiki/Matrice_permutativa`.