Laboratorio computazionale numerico
Lezione 4

Federico Poloni <f.poloni@sns.it>

2009-11-04

1 Sottomatrici e determinanti

Utilizzando l'operatore :, in Octave è possibile selezionare un'intera sottomatrice di una matrice:

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

   1   2   3
   4   5   6
   7   8   9

octave:2> A(2:3,2:3)
ans =

   5   6
   8   9

octave:3> A(2,:)
ans =

   4   5   6

octave:4> A(:,:)
ans =

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

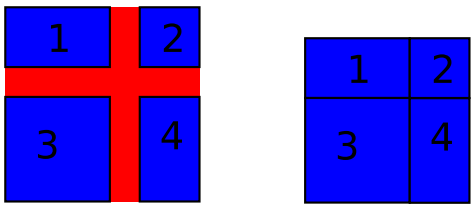
La sintassi a:b seleziona tutte le righe/colonne comprese tra a e b (estremi inclusi); utilizzare semplicemente : come indice di riga/colonna seleziona l'intera riga/colonna. Possiamo anche assegnare un valore a una sottomatrice selezionata in questo modo:

```
octave:5> A(1:2,1:2)=eye(2)
A =

   1   0   3
   0   1   6
   7   8   9
```

(ovviamente le dimensioni devono essere compatibili: non posso selezionare una sottomatrice 2×2 e assegnarle il valore **eye**(3)!)

La seguente function ritorna la *matrice minore* di (i, j) in A, cioè la matrice che si ottiene eliminando la i-esima riga e la j-esima colonna di A.




```
function B=minor(A,i,j)
  n=size(A,1);
  B=zeros(n-1,n-1);
  X=A(1:i-1,1:j-1);
  Y=A(1:i-1,j+1:n);
  Z=A(i+1:n,1:j-1);
  W=A(i+1:n,j+1:n);
  B=[X Y; Z W];
endfunction
```

Abbiamo già visto che se X,Y,Z,W sono numeri, la riga di codice B=[X Y; Z W] crea la matrice

$$\begin{bmatrix} X & Y \\ Z & W \end{bmatrix};$$

ora vediamo che la stessa sintassi funziona anche se X,Y,Z,W sono matrici di dimensioni "compatibili" e crea la matrice formata accostando i quattro blocchi.

Esercizio 1. Creare una funzione **function** d=mydet(A) che calcoli il determinante di una matrice A utilizzando la formula di Laplace sulla prima riga, cioè

$$\det(A) = A_{11} \det A^{(11)} - A_{12} \det A^{(12)} + A_{13} \det A^{(13)} - \cdots + (-1)^{n+1} A_{1n} \det A^{(1n)}$$

dove A_{ij} è l'elemento di A nella posizione (i, j) e $A^{(ij)}$ è la matrice minore di A rispetto a (i, j). (hint: la funzione dovrà essere *ricorsiva*, cioè chiamare sé stessa al suo interno). Testare sulla matrice laplacian(5).

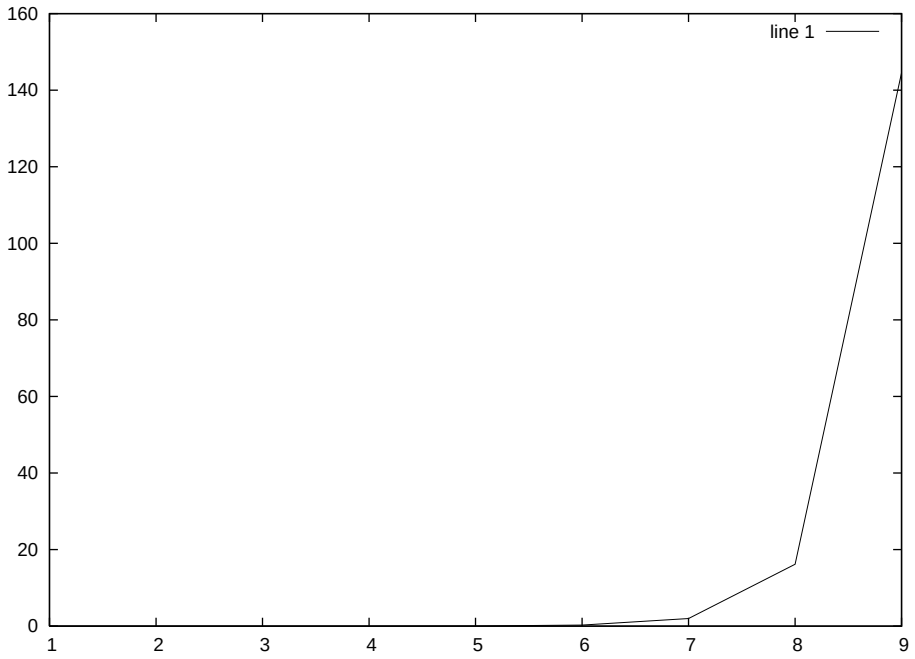
(non date alla funzione il nome **det** perché in Octave c'è già una funzione che si chiama **det**...).

2 Tempi di calcolo

La seguente funzione disegna un grafico del tempo impiegato per calcolare i determinanti delle matrici laplacian(n) con n=1:7.

```
function plottimes();
  n=7;
  tempi=zeros(n,1); %prepara il vettore dei tempi
  for i=1:n
    A=laplacian(i); %la matrice viene generata prima del ''tic''
    tic;
```

```
    mydet(A);
    tempi(i)=toc;
  endfor
  plot(1:n,tempi);
endfunction
```



I tempi di calcolo crescono molto velocemente: difatti, con questo algoritmo, per calcolare un determinante di dimensione n dobbiamo calcolarne n di dimensione $n-1$; quindi vale

$$tempo(n) \approx n \cdot tempo(n-1)$$

da cui $tempo(n) \approx n! \cdot tempo(1)$. Il nostro algoritmo quindi non è adatto a calcolare il determinante in modo efficiente.

Se vi state annoiando.... Provare a misurare nello stesso modo i tempi della funzione **det** di Octave. *(purtroppo i risultati non sono altrettanto chiari in quanto i risultati dipendono da molti dettagli degli algoritmi e dell'architettura del calcolatore)*

Vedremo presto un algoritmo migliore di questo per calcolare il determinante di una matrice.

3 Fattorizzazione LU ed eliminazione di Gauss

3.1 Fattorizzazione LU

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ * & 1 & 0 & 0 & 0 \\ * & 0 & 1 & 0 & 0 \\ * & 0 & 0 & 1 & 0 \\ * & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ 0 & U_{22} & * & * & * \\ 0 & U_{32} & * & * & * \\ 0 & U_{42} & * & * & * \\ 0 & U_{52} & * & * & * \end{pmatrix}$$

Passo 2: al posto degli elementi rossi di L, sostituiamo $L_{i2} = \frac{U_{i2}}{U_{22}}$. Al posto degli elementi rossi di U, sostituiamo $U_{ij} = U_{ij} - L_{i2}U_{2j}$. Nota che in questo modo gli elementi rossi sulla seconda colonna (U_{i2}) diventano 0.

La seguente funzione calcola la fattorizzazione LU di una matrice quadrata (non possiamo chiamarla **lu** perché c'è già una funzione di Octave che si chiama così). [Non state a copiarla, tra poco ve ne passo una versione già scritta. Piuttosto cercate di capire cosa fa mentre la scrivo al proiettore.]

```
function [L,U]=my_lu(A)
  n=size(A,1);
  L=eye(n);
  U=A;
  for k=1:n
    pivot=U(k,k);
    L(k+1:n,k)=U(k+1:n,k)/pivot;
    colonnaL=L(k+1:n,k);
    rigaU=U(k,k+1:n);
    U(k+1:n,k+1:n)=U(k+1:n,k+1:n)-colonnaL*rigaU;
    U(k+1:n,k)=0;
  endfor
endfunction
```

```
octave:47> M=4*eye(5)+ones(5,5)
M =

   5   1   1   1   1
   1   5   1   1   1
   1   1   5   1   1
   1   1   1   5   1
   1   1   1   1   5

octave:48> [L,U]=mylu(M)
octave:49> L*U-M
ans =

 Columns 1 through 3:

    0.00000000000000e+00    0.00000000000000e+00    0.00000000000000e+00
    0.00000000000000e+00    0.00000000000000e+00    0.00000000000000e+00
    0.00000000000000e+00    0.00000000000000e+00   -8.88178419700125e-16
    0.00000000000000e+00    0.00000000000000e+00    2.22044604925031e-16
    0.00000000000000e+00    0.00000000000000e+00    2.22044604925031e-16

 Columns 4 and 5:
```

```
   0.00000000000000e+00    0.00000000000000e+00
   0.00000000000000e+00    0.00000000000000e+00
   0.00000000000000e+00    0.00000000000000e+00
   0.00000000000000e+00    2.22044604925031e-16
   2.22044604925031e-16    0.00000000000000e+00
```

3.2 Sistemi triangolari

Le seguenti funzioni risolvono un sistema triangolare inferiore/superiore. Sono diverse da quelle della lezione scorsa.

```
function x=inf_solve(L,b)
%risolve un sistema con L triangolare inferiore
  n=size(L,1);
  x=b; %x "vettore di accumulatori"
  for i=1:n
    x(i)=x(i)/L(i,i);
    x(i+1:n)=x(i+1:n) - L(i+1:n,i)*x(i);
  endfor
endfunction
```

```
function x=sup_solve(U,b)
%risolve un sistema con U triangolare superiore
  n=size(U,1);
  x=b; %x "vettore di accumulatori"
  for i=n:-1:1
    x(i)=x(i)/U(i,i);
    x(1:i-1)=x(1:i-1) - U(1:i-1,i)*x(i);
  endfor
endfunction
```

Se vi state annoiando.... Capire come (e perché) funziona questa versione di inf_solve. Notate che i calcoli che esegue non sono gli stessi della versione della scorsa lezione! Ricordarsi che la formula è

$$x_i = \frac{b_i - \sum_{j=0}^{i-1} L_{ij} x_j}{L_{ii}}, \quad x = 1 \dots n.$$

4 Esperimenti numerici

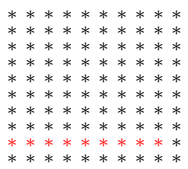
Esercizio 2. Testare i seguenti metodi di soluzione di un sistema lineare $Ax = b$:

- L'istruzione x=sys_solve(A,b), che utilizza la funzione contenuta nel file `http://poisson.dm.unipi.it/~poloni/dida/lcn09/lezione%204/sys_solve.m` (da scaricare!), che contiene gli algoritmi della sezione precedente (fattorizzazione LU senza pivoting + soluzione di due sistemi triangolari).
- Il comando di Octave x=**inv**(A)*b, che calcola la matrice inversa e la moltiplica per b.

- Il comando di Octave x=A\b: il comando \ (barra rovesciata) serve proprio per risolvere sistemi lineari, ed è basato sulla fattorizzazione LU con pivoting parziale[1].

Per testarli, utilizzate le seguenti matrici:

- La matrice M1=9∗**eye**(10)+ones(10), che è dominante diagonale.
- La matrice M2=**rand**(10), che è una matrice con elementi casuali — può essere abbastanza mal condizionata! Potete controllare il condizionamento con il comando **cond**(M2).
- La matrice data da M3=M1;M3(9,1:9)=0, che ha una riga quasi tutta di zeri che rende la sottomatrice principale 9×9 singolare (e quindi non ammette fattorizzazione LU).
- La matrice data da M4=M2;M4(9,1:9)=**sum**(M4(1:8,1:9)):



  gli elementi in rosso sono ognuno la somma degli otto elementi che stanno direttamente sopra di esso; quindi la sottomatrice principale 9×9 è singolare. (Ma lavorando con i numeri floating point...)
- La matrice data da M5=M2;M5(10,1:10)=**sum**(M5(1:9,1:10)): l'ultima riga è la somma delle 9 precedenti, quindi la matrice è singolare. (Ma lavorando con i numeri floating point...)

Ponete v=transpose(1:10) (vettore conentente i numeri da 1 a 10 in ordine); per ognuna di queste matrici, calcolate bi=v (per $i = 1, \ldots, 4$), e andate a risolvere il sistema Mi∗x=bi. La soluzione esatta di questo sistema è v; di quanto si discosta la soluzione calcolata?

Se vi state annoiando... Guardate la fattorizzazione LU (senza pivoting) di M4. U(9,9) è molto piccolo; perché? U(10,10) è molto grande; perché?

[1]Non vi ho fatto scrivere una fattorizzazione LU con pivoting parziale per mancanza di tempo, ma non è niente di particolarmente complicato.