Laboratorio computazionale numerico
Lezione 3

f.poloni&sns.it

2008-10-29

Esercizio 1 (di riscaldamento). Creare una funzione **function** M=laplacian(n) che crea la matrice di dimensione $n \times n$ che ha 2 sulla diagonale e -1 sulla sopra- e sotto-diagonale:

```
octave:1> laplacian(5)
ans =

   2  -1   0   0   0
  -1   2  -1   0   0
   0  -1   2  -1   0
   0   0  -1   2  -1
   0   0   0  -1   2
```

1 Soluzione di sistemi triangolari

1.1 Sistemi triangolari inferiori

$$\begin{pmatrix} L_{11} & 0 & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} & 0 \\ L_{51} & L_{52} & L_{53} & L_{54} & L_{55} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}$$

Possiamo risolverlo per sostituzione: a ogni passo, supponendo di avere calcolato $x_1, \dots, x_{i-1}$ si ha

$$x_i = \frac{b_i - \sum_{j=0}^{i-1} L_{ij} x_j}{L_{ii}}$$

```
function x=inf_solve(L,b)
  n=size(L,1); %size(L,1) e' il numero di righe di L
  x=zeros(n,1); %riserva spazio in memoria per x

  for i=1:size(L,1)
    p=b(i); % accumulatore
    for j=1:i-1
      p=p-L(i,j)*x(j);
    endfor
    x(i)=p/L(i,i);
```

```
  endfor
endfunction
```

Testiamo la funzione con la matrice L=**tril**(laplacian(5)) e il termine noto L∗y, dove y è un vettore semplice.

```
octave:4> y=[1:5]'
y =

   1
   2
   3
   4
   5

octave:5> y=[1:5]'
y =

   1
   2
   3
   4
   5

octave:6> L=tril(laplacian(5))
L =

    2    0    0    0    0
   -1    2    0    0    0
    0   -1    2    0    0
    0    0   -1    2    0
    0    0    0   -1    2

octave:7> inf_solve(L,L*y)
ans =

   1
   2
   3
   4
   5
```

Esercizio 2. Scrivere una function sup_solve(U,b) che risolva un sistema $Ux = b$ con U triangolare superiore (hint: sostituire a partire dall'ultima riga). Testare su U=**triu**(laplacian(5)), b=U∗y (con y vettore opportuno).

1.2 Soluzione di sistemi con la matrice di Laplace

Il comando [L,U]=**lu**(laplacian(5)) restituisce due matrici, una L triangolare inferiore, e una U triangolare superiore, tali che L∗U=laplacian(5).

```
octave:9> [L,U]=lu(laplacian(5))
L =

   1.00000   0.00000   0.00000   0.00000   0.00000
```

```
   -0.50000    1.00000    0.00000    0.00000    0.00000
    0.00000   -0.66667    1.00000    0.00000    0.00000
    0.00000    0.00000   -0.75000    1.00000    0.00000
    0.00000    0.00000    0.00000   -0.80000    1.00000

U =

    2.00000   -1.00000    0.00000    0.00000    0.00000
    0.00000    1.50000   -1.00000    0.00000    0.00000
    0.00000    0.00000    1.33333   -1.00000    0.00000
    0.00000    0.00000    0.00000    1.25000   -1.00000
    0.00000    0.00000    0.00000    0.00000    1.20000

octave:10> L*U-laplacian(5)
ans =

  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
```

Utilizzando questa fattorizzazione e le funzioni inf_solve e sup_solve possiamo scrivere una funzione che risolve un sistema $Ax = b$ con A=laplacian(5): infatti,

$$A^{-1}b = (LU)^{-1}b = U^{-1}(L^{-1}b)$$

```
function x=lap_solve(n,b)
  [L,U]=lu(laplacian(n));
  y=inf_solve(L,b)
  x=sup_solve(U,y);
endfunction
```

Testiamo:

```
octave:20> A=laplacian(5); y=[1:5]';
octave:21> lap_solve(5,A*y)
ans =

  1.0000
  2.0000
  3.0000
  4.0000
  5.0000
```

Possiamo testarlo anche su matrici molto più grandi: per $n = 1000$ sui computer del laboratorio dovrebbe impiegare meno di 20 secondi.

Notiamo però che le matrici L e U per le matrici laplacian(n) hanno una struttura particolare: tutti gli elementi sono nulli tranne quelli sulla diagonale principale e sulla sottodiagonale (per L) o sopradiagonale (per U). L e U sono dette *bidiagonali*. Possiamo sfruttare esplicitamente questo fatto nei nostri algoritmi.

Esercizio 3. Scrivere due funzioni inf_solve2 e sup_solve2 che risolvano i sistemi $Ly = b$ e $Ux = y$ per L e U bidiagonali (facendo meno calcoli di inf_solve e supsolve!). Poi scrivere la funzione lap_solve2 analoga a lap_solve ma che utilizza le due funzioni appena scritte.

1.3 Misurazione dei tempi

In Octave possiamo cronometrare quanto tempo viene utilizzato per eseguire una o più istruzioni con i comandi **tic** e **toc**. Il primo fa partire il cronometro; il secondo lo ferma e restituisce la quantità di tempo (in secondi) che è passata. Quindi con le istruzioni

```
tic;
lap_solve(A,b);
tempo=toc;
```

scriviamo nella variabile "tempo" il numero di secondi necessari al computer per eseguire la funzione lap_solve(A,b). Utilizzando queste due funzioni, possiamo confrontare i tempi di esecuzione di lap_solve e lap_solve2.

```
octave:27> b=laplacian(1000)*[1:1000]';
octave:28> tic;lap_solve(1000,b); toc
ans = 11.669
octave:29> tic;lap_solve2(1000,b); toc
ans = 2.1535
```

Notare che:

- **tic**, l'istruzione e **toc** sono stati messi sulla stessa riga perché vengano eseguiti uno dopo l'altro; poiché alla fine dell'ultima istruzione non c'è un punto e virgola, Octave scrive il risultato di **toc** sullo schermo
- l'istruzione che calcola b è al di fuori della coppia di comandi **tic** ... **toc**: altrimenti anche il suo tempo di esecuzione verrebbe conteggiato.

Morale della favola: quando una matrice ha una struttura dobbiamo sempre cercare di sfruttarla nei calcoli; il guadagno di tempo (e a volte anche di precisione dei risultati) può essere notevole.

2 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
```

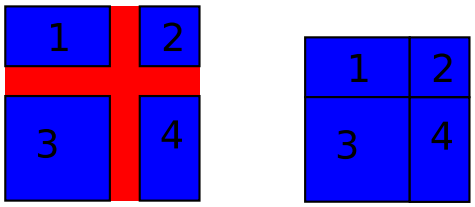
La sintassi a:b seleziona tutte le righe/colonne comprese tra a e b (estremi inclusi); utilizzare semplicemente : come indice di riga/colonna seleziona l'intera riga/colonna. Possiamo anche assegnare un valore a una sottomatrice selezionata in questo modo:

```
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);
  B(1:i-1,1:j-1)=A(1:i-1,1:j-1);
  B(1:i-1,j:n-1)=A(1:i-1,j+1:n);
  B(i:n-1,1:j-1)=A(i+1:n,1:j-1);
  B(i:n-1,j:n-1)=A(i+1:n,j+1:n);
endfunction
```

Una versione un po' più semplice da scrivere si può ottenere così:

```
function B=minor2(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 4 (*). Creare una funzione **function** d=mydet(A) che calcoli il determinante di una matrice A utilizzando la formula di Laplace sulla prima riga, cioè

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

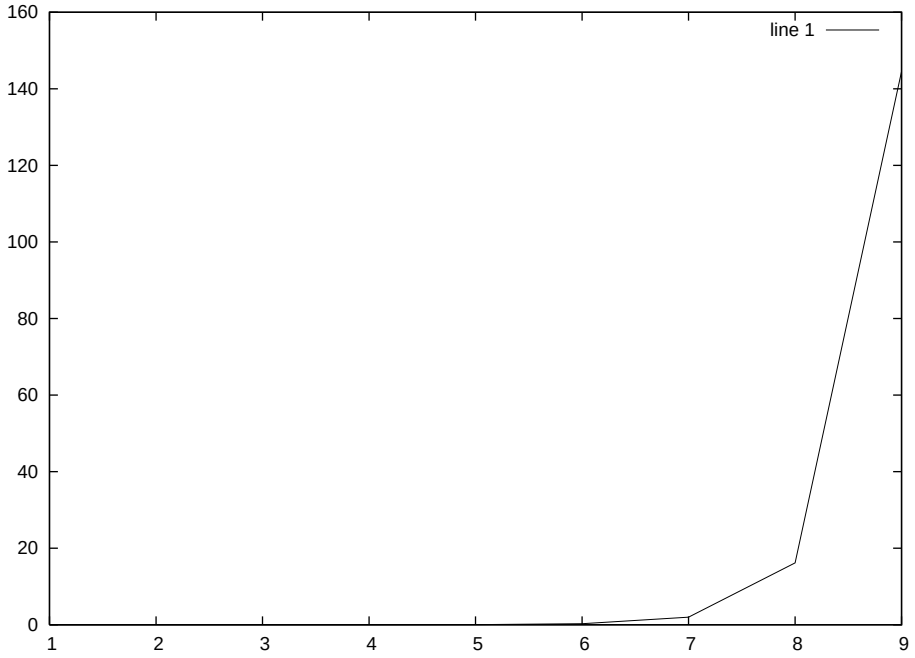
dove A_{ij} è l'elemento di A nella posizione (i, j) e $A^{(ij)}$ è la matrice minore di A rispetto a (i, j). (hint: la funzione dovrà essere *ricorsiva*, cioè chiamare sé stessa al suo interno). Testare sulla matrice laplacian(5).

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

3 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.

Esercizio 5. 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...