LABORATORIO DI CALCOLO NUMERICO

Laurea in Statistica e Informatica

Esercitazione di algebra lineare numerica

Prof. Stefano De Marchi

Padova, November 10, 2010

Come fatto finora, presentiamo dapprima alcune utili comandi per lavorare con matrici aventi una certa struttura.

1 Alcune matrici "speciali"

- Matrice di *Toeplitz*. `toeplitz`

$$\begin{bmatrix} c_1 & r_2 & r_3 & \dots & r_n \\ c_2 & c_1 & r_2 & \ddots & r_{n-1} \\ \vdots & c_2 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & r_2 \\ c_n & c_{n-1} & \dots & c_2 & c_1 \end{bmatrix}$$

Si definisce mediante il comando `toeplitz`. Ad esempio

```
>> toeplitz([0,1,2,3],[0,-1,-2,-3])

ans =

     0    -1    -2    -3
     1     0    -1    -2
     2     1     0    -1
     3     2     1     0
```

- Matrice di *Vandermonde*. `vander`

$$\begin{bmatrix} c_1^{n-1} & \dots & c_1^2 & c_1 & 1 \\ c_2^{n-1} & \dots & c_2^2 & c_2 & 1 \\ \vdots & & \ddots & \ddots & \vdots \\ & & & & \\ c_n^{n-1} & \dots & c_n^2 & c_n & 1 \end{bmatrix}$$

```
>> vander([1 2 3 4])

ans =
```

```
1    1    1    1
8    4    2    1
27   9    3    1
64  16    4    1
```

- Altre matrici in

  `http://www.maths.manchester.ac.uk/~higham/mctoolbox/`

2 Il comando `find`

Uno dei comandi più utili in MATLAB® è `find`. `find`

Dato un vettore `v` definito come

```
>> v=10:19
```

Vogliamo sapere quali sono gli elementi del vettore ≥ 15. È sufficiente il seguente comando

```
>> find(v>=15)

ans =

     6     7     8     9    10
```

A questo punto, è possibile eseguire operazioni solo sugli elementi specificati

```
>> index=find(v>=15)

index =

     6     7     8     9    10

>> v(index)

ans =

    15    16    17    18    19

>> v(index)=v(index)-15

v =

    10    11    12    13    14     0     1     2     3     4
```

Nel caso matriciale

```
>> A=[10,11;12,13]

A =

    10    11
    12    13

>> index=find(A<13)

index =

     1
     2
     3
```

Il risultato del comando `find` non è una matrice, come ci si potrebbe aspettare, ma un vettore colonna. Non c'è però alcun problema ad eseguire operazioni solo sugli elementi specificati

```
>> A(index)=0

A =

     0     0
     0    13
```

Per costruire, per esempio, una matrice che abbia elementi pari ad uno solo nelle posizioni specificate, è necessario prima inizializzarla con le dimensioni opportune

```
>> B=zeros(2)

B =

     0     0
     0     0
```

e poi assegnare i valori

```
>> B(index)=1

B =

     1     1
     1     0
```

3 Memorizzazione di matrici in formato sparso in Matlab

Si chiamano matrici *sparse* quelle in cui il numero di elementi *diversi da zero* è proporzionale ad n piuttosto che a n^2 (se n è l'ordine della matrice). Per esempio, una matrice *tridiagonale*

è una matrice con elementi diversi da zero solo in tre diagonali (di solito, la principale e le due adiacenti). Dunque il numero di elementi diversi da zero è $3n$ e quindi la matrice è sparsa. Una matrice triangolare superiore ha circa $n^2/2$ elementi diversi da zero (per la precisione $(n^2 - n)/2 + n$) e dunque **non** è sparsa.

Per matrici sparse è più conveniente memorizzare solo gli elementi diversi da zero e la loro posizione.

ESEMPIO. Invece di memorizzare tutti i 25 elementi della matrice

$$\begin{bmatrix} 10 & 0 & 0 & 0 & 0 \\ 20 & 30 & 0 & 0 & 0 \\ 0 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 0 \\ 0 & 0 & 0 & 0 & 70 \end{bmatrix}$$

si potrebbero memorizzare i tre vettori

$$\begin{aligned} \text{elementi (per colonna):} &\ [10, 20, 30, 40, 50, 60, 70] \\ \text{indici di colonna:} &\ [1, 1, 2, 3, 3, 4, 5] \\ \text{indici di riga:} &\ [1, 2, 2, 3, 4, 4, 5] \end{aligned}$$

Ovviamente, per questo piccolo esempio, le due tecniche di memorizzazione risultano praticamente equivalenti.

MATLAB® utilizza un formato di memorizzazione delle matrici sparse simile a quello elemento/colonna/riga (in realtà, leggermente più efficiente).

- La conversione dalla memorizzazione in formato *pieno* alla memorizzazione in formato sparso avviene con il comando `sparse`. `sparse`

```
>> A=[10,0,0,0,0;20,30,0,0,0;0,0,40,0,0;0,0,50,60,0;0,0,0,0,70]

A =

    10     0     0     0     0
    20    30     0     0     0
     0     0    40     0     0
     0     0    50    60     0
     0     0     0     0    70

>> B = sparse(A)

B =

   (1,1)       10
   (2,1)       20
   (2,2)       30
   (3,3)       40
```

```
   (4,3)       50
   (4,4)       60
   (5,5)       70
```

È possibile operare su una matrice sparsa come al solito, solo l'output ne risulterà eventualmente diverso:

```
>> B(2,1)

ans =

    20

>> B(2,:)

ans =

   (1,1)       20
   (1,2)       30
```

- Il più importante comando per la creazione di matrici direttamente in formato sparso è `spdiags`. Permette di creare matrici specificandone gli elementi e la posizione delle diagonali. Vediamo un esempio.

```
>> a=[1;2;3;4]; b=[0;10;20;30]; c=[100;200;0;0];
>> A=spdiags([c,a,b],[-2,0,1],4,4)

ans =

   (1,1)        1
   (3,1)      100
   (1,2)       10
   (2,2)        2
   (4,2)      200
   (2,3)       20
   (3,3)        3
   (3,4)       30
   (4,4)        4
```

`full`

```
>> full(A)

ans =

     1    10     0     0
     0     2    20     0
   100     0     3    30
     0   200     0     4
```

Oltre ad una maggior efficienza nell'occupazione di memoria, il formato sparso può risultare vantaggioso anche nella velocità di esecuzione, in quanto tutte le operazioni che coinvolgerebbero gli zeri vengono semplificate.

Esercizi proposti

1. Si implementi il metodo di Jacobi con una

   `function [x iter err] = Jacobi(A,b,tol,maxit)`

   che costruisce la matrice di iterazione per mezzo del comando `spdiags`. Lo si testi per la soluzione del sistema lineare $Ax = b$ con `A = toeplitz([4 1 0 0 0 0])` e b scelto in modo che la soluzione esatta sia $x = [2, 2, 2, 2, 2, 2]^{\mathrm{T}}$.

2. Per la matrice e il termine noto dell'esempio precedente, si scriva una *function*

   `function [x iter err] = SOROtt(A,b,tol,maxit,omega)`

   che implementa il metodo SOR con il parametro ottimale ω_0. Nota bene: tale funzione potrà anche utilizzarsi come metodo di Gauss-Seidel quando `omega=1`.

3. Si verifichi, senza far eseguire alcun codice, se il metodo di Jacobi converge nel caso in cui dato `x=[1 1.5 2 2.5 3]`, si consideri il sistema `Va=y`, in cui `V=vander(x)` e termine noto `y=sin(x)`. La soluzione del sistema, se esiste, darebbe i coefficienti del polinomio d'interpolazione di grado 4 della funzione seno sui punti equispaziati `x`. Quale metodo sarebbe applicabile?

4. Si risolva il sistema non lineare

$$\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 = 1 \\ f_2(x_1, x_2) = \sin(\pi x_1/2) + x_2^3 \end{cases}$$

   con il metodo di Newton. Si usi una tolleranza pari a 10^{-6}, un numero massimo di iterazioni pari a 150 e un vettore iniziale $x^{(0)} = [1, 1]^{\mathrm{T}}$.

   Facol. Si risolva lo stesso sistema non lineare usando sempre la matrice Jacobiana relativa al primo passo e aggiornando la matrice Jacobiana ogni r iterazioni, ove r è il più piccolo numero che permette di ottenere la soluzione con la tolleranza richiesta calcolando solo due volte la matrice Jacobiana. La risoluzione dei sistemi lineari deve avvenire con il calcolo della fattorizzazione LU solo quando necessaria.