Laboratorio di Calcolo Numerico
Lezione 4

1 Matrici di permutazione e riducibilità

Ogni permutazione sull'insieme $\{1, \ldots, n\}$ si può rappresentare tramite un vettore $p \in \mathbb{N}^n$ (fatto con i numeri interi da 1 a n nell'ordine che segue la permutazione). Ad esempio, nel caso $n = 4$, per rappresentare la permutazione che manda $j + 1$ in j e 1 in 4 avremo

```
p = [2 3 4 1];
```

Sia $\Pi \in \mathbb{R}^n$ un certa matrice di permutazione ottenuta applicando la permutazione p alle righe dell'identità. In Matlab moltiplicare una matrice A a sinistra per Π è particolarmente semplice, in quanto basta applicare la permutazione direttamente agli indici di riga di A. Ad esempio per calcolare $\Pi \cdot A$ si può procedere come segue

```
A = [ 1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16];
>> A

A =

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

A(p, :)

ans =

5 6 7 8
9 10 11 12
13 14 15 16
1 2 3 4
```

Moltiplicare a destra per Π (ovvero calcolare $A \cdot \Pi$) è un filo più complicato perchè la permutazione Π rappresenta, se vista per colonne, la permutazione inversa rispetto a p. Quindi è necessario trovare la permutazione inversa di p; questo si può fare come segue

```
n = length(p);
ip = zeros(1, n); % calcolo la permutazione inversa come quella
ip(p) = 1:n;      % che composta con p da 1:n
A(:, ip)

ans =

4 1 2 3
8 5 6 7
12 9 10 11
16 13 14 15
```

Infine, dato che Π^T rappresenta la permutazione inversa rispetto a Π, si ha che $\Pi A \Pi^T$ si trova semplicemente con il comando

```
A(p, p)

ans =

6 7 8 5
10 11 12 9
14 15 16 13
2 3 4 1
```

Esercizio 1. Sia

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 2 & 5 & 0 & -3 \\ 0 & 1 & 3 & 6 \\ 2 & 0 & 0 & -1 \end{bmatrix}.$$

si trovi una matrice di permutazione Π tale per cui $\Pi A \Pi^T$ risulta triangolare superiore a blocchi. Si verifichi la correttezza del risultato trovato con Matlab.

2 Soluzione di sistemi triangolari

Consideriamo il seguente sistema triangolare inferiore

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

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

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

In particolare $x_1 = \frac{b_1}{L_{11}}$.

```
function x=inf_solve(L,b)
s=size(L);
n=s(1);
x=zeros(n,1);
for i=1:n
       p=b(i); % accumulatore
       for j=1:i-1
              p=p-L(i,j)*x(j);
       end
       x(i)=p/L(i,i);
end
end
```

Il comando `tril` estrae la parte triangolare inferiore di una matrice fornita come input; analogamente `triu` estrae la parte triangolare superiore. Testiamo la funzione `inf_solve` con `L=tril(randn(5))` e termine noto `L*y`, dove `y=[1:5]'` (in questo modo sappiamo che la soluzione del sistema è proprio y).

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

1
2
3
4
5

>> L=tril(randn(5))
L =

0.5377   0       0     0     0
1.8339 -0.4336   0     0     0
-2.2588 0.3426 0.7254  0     0
0.8622 3.5784 -0.0631 1.4090 0
0.3188 2.7694 0.7147 1.4172 0.4889

>> inf_solve(L,L*y)
ans =
```

```
1.0000
2.0000
3.0000
4.0000
5.0000
```

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

Esercizio 3. Modificare `inf_solve` e `sup_solve` in modo da sostituire il ciclo `for` più interno con operazioni su vettori. Chiamate queste nuove funzioni rispettivamente `inf_solve2` e `sup_solve2` e confrontate le performance su matrici grandi (ad esempio 1000×1000).

3 Eliminazione di Gauss

Ricordate come funziona il passo k-esimo dell'algoritmo di eliminazione di Gauss (nell'immagine $k = 3$):

$$\left(A^{(k)} \mid b^{(k)}\right) = \left(\begin{matrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & A_{33}^{(k)} & * & * & * \\ 0 & 0 & A_{43}^{(k)} & * & * & * \\ 0 & 0 & A_{53}^{(k)} & * & * & * \\ 0 & 0 & A_{63}^{(k)} & * & * & * \end{matrix}\right| \left.\begin{matrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(3)} \\ b_5^{(3)} \\ b_6^{(3)} \end{matrix}\right.$$

dovete modificare $A^{(k)}|b^{(k)}$ sommando alle sue righe da $k+1$ ad n multipli della riga k in modo che gli elementi in posizione $(k+1,k), (k+2,k), \ldots, (n,k)$ diventino zero.

Esercizio 4. Scrivete una `function [U, c] = my_gauss(A, b)` che applichi l'eliminazione di Gauss al sistema lineare $Ax = b$ restituendo la matrice in forma triangolare U ed il vettore dei termini noti c, tale che il sistema $Ux = c$ sia equivalente a quello di partenza.

Esercizio 5. Scrivete una `function x = sys_solve(A,b)` che risolva un sistema lineare generico utilizzando la fattorizzazione l'eliminazione di Gauss, ed il metodo implementato in `sup_solve`. Si testi il metodo usando matrici e vettori generati casualmente con la function handle `randn`, ad esempio eseguendo:

```
n=5;
A=randn(n);
x=randn(n, 1);
b=A*x;
y= sys_solve(A,b);
x-y
```

In caso di implementazione corretta ci aspettiamo che il vettore $x - y$ nostrato a schermo abbia tutte le entrate vicine a 0.