CAP. 3 – ASSEMBLAGGIO DELLE MATRICI DI RIGIDEZZA ELEMENTARI

3.1 Introduzione

Il processo di assemblaggio sarà descritto facendo riferimento alla struttura reticolare. Comunque, l'intera procedura si applica a qualsiasi elemento, indipendentemente dal suo tipo, forma e numero di nodi. I concetti importanti sono:

a) Ci sono due sistemi di numerazione, uno per la struttura ed un altro assegnato all'elemento in fase di assemblaggio;

b) La relazione tra i due sistemi di numerazione determina la posizione entro $[K]$ in cui andrà posizionata la matrice elementare $[k_e]$.

Consideriamo come la matrice di rigidezza della struttura dell'eq. 2.2.5 possa essere formata assemblando le tre matrici di rigidezza elementari. Per iniziare scriviamo le matrici di rigidezza dei tre elementi. Poiché le proprietà dell'elemento rimangono invariate se lo si ruota di 180°, i nodi i e j della Fig. 2.3.1 possono essere assegnati in modo arbitrario. Così, usando le eq. 2.3.1 e 2.3.3 abbiamo:

elemento	nodo	Angolo θ	Seno	Coseno	Matrice di rigidezza elementare	Vettore degli spostamenti
1	2 3	90°	1	0	$k_1\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 1 & 0 & -1\\ 0 & 0 & 0 & 0\\ 0 & -1 & 0 & 1\end{bmatrix}$	$\begin{Bmatrix}u_2\\ v_2\\ u_3\\ v_3\end{Bmatrix}$
2	3 1	315°	$-\sqrt{2}/2$	$\sqrt{2}/2$	$\frac{k_2}{2}\begin{bmatrix}1 & -1 & -1 & 1\\ -1 & 1 & 1 & -1\\ -1 & 1 & 1 & -1\\ 1 & -1 & -1 & 1\end{bmatrix}$	$\begin{Bmatrix}u_3\\ v_3\\ u_1\\ v_1\end{Bmatrix}$
3	1 2	180°	0	-1	$k_3\begin{bmatrix}1 & 0 & -1 & 0\\ 0 & 0 & 0 & 0\\ -1 & 0 & 1 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}$	$\begin{Bmatrix}u_1\\ v_1\\ u_2\\ v_2\end{Bmatrix}$

dove gli spostamenti nell'ultima colonna sono stati indicati semplicemente per identificare i gradi di libertà associati con ogni elemento.

Possiamo immaginare che la struttura sia costruita sommando un elemento alla volta, posizionando ogni elemento in modo prestabilito. Come gli elementi vengono sommati alla struttura, si aggiungono dei contributi alla matrice di rigidezza globale. Per ottenere la matrice di rigidezza della struttura possiamo infatti sommare le matrici di rigidezza elementari, purché abbiano le stesse dimensioni della matrice globale e agiscano sugli stessi vettori di spostamento. L'espansione verso la dimensione della struttura si realizza semplicemente aggiungendo delle righe e delle colonne di zeri, in modo che $[k_e]$ diventi grande quanto $[K]$.

Per esempio, per l'elemento n.1 del nostro esempio, l'espansione dà luogo alla seguente matrice:

$$\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0\\ 0 & k_1 & 0 & -k_1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & -k_1 & 0 & k_1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$ a cui è associato il vettore spostamento $$\begin{Bmatrix}u_2\\ v_2\\ u_3\\ v_3\\ u_1\\ v_1\end{Bmatrix}$$

Questa espansione può essere giustificata nel modo seguente. Non è possibile che uno spostamento arbitrario $\{u_2 \quad v_2 \quad u_3 \quad u_3\}^T$ dei nodi dell'elemento n.1 possa generare una forza sul nodo 1: ciò giustifica le ultime due righe piene di zeri. Inoltre, lo spostamento del nodo 1 non può fare in modo che l'elemento n.1 applichi una forza sui propri nodi (il 2 ed il 3): ciò giustifica le ultime due colonne piene di zeri.

Prima della somma, ogni matrice elementare $[k_e]$ deve essere riorganizzata in modo da operare sullo stesso vettore di spostamenti nodali. La riorganizzazione si ottiene scambiando righe e colonne; forse è più

semplice capire questo punto scrivendo la relazione $[k_e]\{d\} = \{f\}$ come un sistema di equazioni, mettendo in ordine i vari coefficienti e riscrivendo il risultato in forma matriciale. Questa operazione di riordino applicata alla matrice del primo elemento conduce al seguente risultato:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1 & 0 & -k_1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -k_1 & 0 & k_1 \end{bmatrix}$$ a cui è associato il vettore spostamento $$\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix}$$

Si può facilmente verificare che la semplice somma delle matrici di rigidezza espanse e riordinate produce la matrice di rigidezza della struttura dell'eq.2.2.5.

Può essere utile una spiegazione alternativa dell'assemblaggio delle matrici di rigidezza. La Fig.3.1 mostra una porzione di una struttura reticolare arbitraria. Il nodo n.5 della struttura è comune agli elementi n.1, 2 e 3. Gli indici i e j di ogni elemento sono stati assegnati arbitrariamente. Per ogni elemento abbiamo:

$$[k_e]_n \cdot \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \end{Bmatrix}_n = \begin{Bmatrix} U_i \\ V_i \\ U_j \\ V_j \end{Bmatrix}_n \qquad [3.1.1]$$

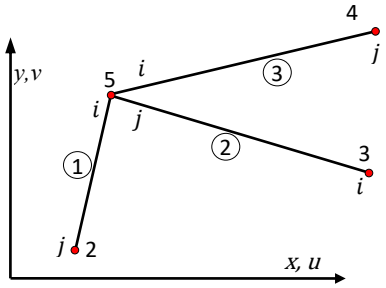


Fig.3.1

in cui $n = 1, 2, 3$. Consideriamo adesso, per esempio, la forza orizzontale U_5 che deve essere applicata al nodo n.5 per mantenere l'equilibrio sotto un arbitrario insieme di spostamenti nodali. Ognuno dei tre elementi che concorrono nel nodo n. 5 contribuiscono alla formazione di U_5:

$$U_5 = (U_i)_1 + \left(U_j\right)_2 + (U_i)_3 \qquad [3.1.2]$$

Sostituendo i valori di $(U_i)_1$, $\left(U_j\right)_2$ e $(U_i)_3$ che si ottengono dall'eq.3.1.1 abbiamo:

$$\begin{aligned} U_5 = {} & (k_{11})_1 \cdot u_5 + (k_{12})_1 \cdot v_5 + (k_{13})_1 \cdot u_2 + (k_{14})_1 \cdot v_2 + \\ & + (k_{31})_2 \cdot u_3 + (k_{32})_2 \cdot v_3 + (k_{33})_2 \cdot u_5 + (k_{34})_2 \cdot v_5 + \\ & + (k_{11})_3 \cdot u_5 + (k_{12})_3 \cdot v_5 + (k_{13})_3 \cdot u_4 + (k_{14})_3 \cdot v_4 \end{aligned} \qquad [3.1.3]$$

Raggruppando i vari contributi, abbiamo:

$$\begin{aligned} U_5 = {} & [(k_{11})_1 + (k_{33})_2 + (k_{11})_3] \cdot u_5 + [(k_{12})_1 + (k_{34})_2 + (k_{12})_3] \cdot v_5 + \\ & + (termini\ che\ coivolgono\ gli\ spostamenti\ u_2, v_2, u_3, v_3, u_4 v_4) \end{aligned} \qquad [3.1.4]$$

$$U_5 = \left(K_{9,9}\right) \cdot u_5 + \left(K_{9,10}\right) \cdot v_5 + (\cdots)$$

I coefficienti della matrice di rigidezza hanno gli indici (9,9) e (9,10) perché gli spostamenti u_5 e v_5 sono i coefficienti in nona e decima posizione nel vettore degli spostamenti della struttura $\{D\}$.

Ancora una volta, l'eq. 3.1.4 mostra che la matrice di rigidezza è costruita sommando i termini delle matrici di rigidezza elementari. Essa mostra anche che la posizione entro $[K]$ in cui vengono assegnati i coefficienti di un elemento $[k_e]$ dipendono solo dal modo in cui sono stati numerati i nodi della struttura globale. Così vediamo che gli indici i e j di un elemento sono usati solo per convenienza nella formulazione delle proprietà dell'elemento; avremo ugualmente potuto usare dei numeri o delle lettere.

3.2 Assemblaggio delle equazioni dell'equilibrio

In pratica è conveniente assemblare la matrice di rigidezza della struttura e le equazioni dell'equilibrio nodale in un'unica operazione. Prima di spiegare questa procedura introduciamo ulteriori carichi nodali.

Consideriamo questi carichi come se fossero applicati sui nodi dagli elementi. I nodi applicheranno sugli elementi delle forze uguali e contrarie.

Ipotizzando che la forza di gravità agisca in direzione verticale $-y$, un'asta prismatica di peso specifico pari a γ, applica una forza $\{f_\gamma\}$ ai nodi pari a:

$$\{f_\gamma\} = \frac{\gamma \cdot A \cdot L}{2} \cdot \begin{Bmatrix} 0 \\ -1 \\ 0 \\ -1 \end{Bmatrix} \qquad [3.2.1]$$

La (3.2.1) trascura la piccola variazione di sforzi agenti su ogni asta causata dal peso proprio.

Se gli spostamenti nodali sono impediti, il riscaldamento di T gradi centigradi produce una forza assiale di compressione pari a $\alpha \cdot E \cdot A \cdot T$, in cui α indica il coefficiente di dilatazione termica lineare. Di conseguenza ai nodi sono applicate le seguenti forze:

$$\{f_T\} = \alpha \cdot E \cdot A \cdot T \cdot \begin{Bmatrix} -c \\ -s \\ c \\ s \end{Bmatrix} \qquad [3.2.2]$$

in cui, come prima, $c = cos(\vartheta)$ e $s = sin(\vartheta)$ (vedi Fig.2.3.2).

La stessa forza $\{f_T\}$ sarebbe causata dal montaggio forzato di un elemento troppo lungo, la cui lunghezza eccede quella corretta della quantità $\alpha \cdot L \cdot \Delta T$.

Le forze $[k_e]\{d\}$ causate dagli spostamenti nodali $\{d\}$ sono state definite come forze applicate sull'elemento. Di conseguenza la forza complessiva $\{f\}$ applicata sui nodi da un elemento è

$$\{f\} = -[k_e]\{d\} + \{f_\gamma\} + \{f_T\} \qquad [3.2.3]$$

Le equazioni strutturali sono un insieme di equazioni di equilibrio ai nodi. Possono essere formate isolando i nodi da considerare liberi di spostarsi nel piano e scrivendo per ognuno di essi le equazioni dell'equilibrio statico. Da un punto di vista fisico, i nodi di una struttura reticolare possono essere immaginati come dei perni che uniscono le bielle (tiranti/puntoni). I carichi nodali consistono nelle forze esterne $\{P\}$ e nelle forze $\{f\}$ applicate da ognuno degli elementi. Di conseguenza l'intero insieme di equazioni di equilibrio, due per nodo perché stiamo considerando una struttura piana priva di momenti flettenti, è:

$$\{P\} + \sum_{n=1}^{m} \{f\}_n = 0 \qquad [3.2.4]$$

dove per ogni elemento $\{f\}$ è data dall'eq.3.2.3 e la sommatoria si estende su tutti gli elementi m della struttura. Sostituendo l'eq.3.2.3 nell'eq.3.2.4 otteniamo:

$$\left(\sum_{n=1}^{m} [k_e]_n\right) \cdot \{D\} = \{P\} + \sum_{n=1}^{m} \left(\{f_\gamma\}_n + \{f_T\}_n\right) \qquad [3.2.5]$$

o anche

$$[K]\{D\} = \{F\}$$

Realizzando questa sostituzione abbiamo supposto che le matrici elementari $[k_e]_n$, $\{f_\gamma\}_n$ e $\{f_T\}_n$ siano state espanse alla dimensione della struttura e che i loro termini siano stati ordinati in modo che il vettore degli spostamenti $\{d\}$ di ogni elemento sia stato sostituito dal vettore degli spostamenti della struttura $\{D\}$.

L'implementazione in un software per il processo di assemblaggio segue una procedura che si discosta significativamente dal metodo "*espandi e somma*" descritto precedentemente. Le differenze principali sono le seguenti:

a) Le matrici di rigidezza elementari non vengono espanse alla dimensione della matrice globale. I loro coefficienti sono sommati direttamente nella matrice globale $[K]$ per mezzo di un vettore di puntatori ai gradi di libertà;

b) La matrice di rigidezza globale $[K]$ viene salvata in memoria in un formato che sfrutta al meglio la sua struttura, cioè sparsezza e simmetria (metodo a banda, metodo skyline oppure a matrice sparsa).

In questa fase, per semplicità, ipotizzeremo che la matrice quadrata e simmetrica $[K]$ sia salvata completamente in memoria. In un secondo momento vedremo le altre tecniche di memorizzazione.

Nel caso di una struttura bidimensionale costituita da elementi tirante/puntone (*truss* o *rod* nella letteratura anglosassone, *link* in ANSYS), in cui abbiamo due gradi di libertà per nodo, i puntatori alla matrice di rigidezza $[K]$ possono essere calcolati nel modo seguente:

$$K_{pq} = \sum_{e=1}^{Nelem} K_{ij}^{e} \qquad \begin{matrix} per & i = 1,\cdots,4 & p = gdl^{e}(i) \\ per & j = 1,\cdots,4 & q = gdl^{e}(j) \end{matrix} \qquad [3.2.6]$$

Qui K_{ij}^{e} indica il coefficiente della matrice elementare di dimensione (4×4) nella posizione locale (i,j); gdl^{e} indica la lista dei gradi di libertà dell'elemento e. Per l'esempio esposto nel 2° capitolo questi vettori sono:

$$gdl^{(1)} = \{3,4,5,6\} \quad ; \quad gdl^{(2)} = \{1,2,5,6\} \quad ; \quad gdl^{(3)} = \{3,4,1,2\} \qquad [3.2.7]$$

Fisicamente questi vettori mappano gli indici dei gradi di libertà locali su quelli globali. Per esempio il terzo grado di libertà locale del secondo elemento è u_{x3}, che corrisponde al numero 5 nel sistema globale; di conseguenza $gdl^{(2)}(3) = 5$. Bisogna notare che in un programma automatico di assemblaggio, la (3.2.6) comporta tre cicli annidati: uno (il più esterno) sugli elementi, gli altri sui gradi di libertà elementari i e j. L'ordine di questi due ultimi cicli è irrilevante. Per risparmiare tempo di calcolo, si può sfruttare la simmetria della matrice di rigidezza globale per assemblarne solo la metà, per esempio il suo triangolo superiore destro, da copiare al termine dell'assemblaggio, nell'altra metà.

Le istruzioni in MATLAB della Fig.3.2.1 eseguono l'assemblaggio di una matrice di rigidezza elementare $[k_e]$ in una matrice di rigidezza globale $[K]$. *Nnodi* indica il numero di nodi di cui è composto l'elemento (nel nostro esempio *Nnodi*=2). Il vettore *Nodi* contiene la lista di nodi di cui è composto l'elemento (nel nostro esempio i e j). *Ngdln* indica il numero di gradi di libertà per nodo (nel nostro esempio, trattandosi di una struttura reticolare piana, *Ngdln* = 2, lo spostamento orizzontale e verticale). La matrice di rigidezza dell'elemento deve essere calcolata precedentemente e fornita alla routine per l'assemblaggio.

```
function AssemblaRigidezzaPiena (ke, Ngdln, Nnodi, Nodi)
%AssemblaRigidezzaPiena Matrice quadrata piena.
%        ke:    matrice di rigidezza elementare
%        Ngdln: numero di gradi di libertà per nodo: per esempio nei ROD_2D Ngdln=2
%        Nnodi: numero di nodi dell'elemento: per esempio nei ROD Nnodi=2
%        Nodi:  vettore dei nodi che costituiscono l'elemento
%=========================================================
global MatriceGlobale
ngdle = Nnodi*Ngdln;
ij(1:ngdle)=0;
i = 0;
for n = 1:Nnodi
    last  = Ngdln*Nodi(n);
    first = last - Ngdln + 1;
    for gdl = first:last
        i = i + 1;
        ij(i) = gdl;
    end
end
for i = 1:ngdle
    I = ij(i);
    for j = 1:ngdle
        J = ij(j);
        MatriceGlobale(I,J) = MatriceGlobale(I,J) + ke(i,j);
    end
end
```

Fig. 3.2.1 – Codice MATLAB per l'assemblaggio della matrice di rigidezza in forma piena.

E' importante analizzare come il programma MATLAB esegue le procedure codificate negli M-file. Le procedure utilizzano uno spazio di lavoro locale (Workspaces) dove vengono momentaneamente copiate solo le variabili d'input che saranno modificate durante la procedura. Per esempio, nella `function AssemblaRigidezzaPiena` precedentemente indicata nella Fig.3.2.1, la matrice di rigidezza elementare `ke` non viene modificata, ma semplicemente sommata alla matrice globale `MatriceGlobale`, e quindi non viene copiata nello spazio di lavoro locale. Viceversa se la matrice di rigidezza globale `K` fosse passata alla procedura attraverso i suoi parametri d'input, prima di essere aggiornata verrebbe copiata nello spazio di lavoro locale. Poiché la matrice `K` può avere dimensioni considerevoli, la procedura richiederebbe molto tempo, perso a copiare la matrice nello spazio di lavoro locale. Per accelerare il processo di assemblaggio, è

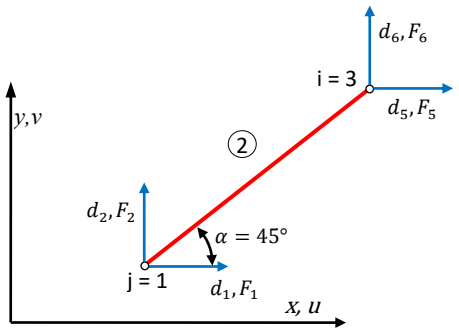
sufficiente dichiarare che la matrice di rigidezza globale `K` è una variabile globale ed eliminarla dalla lista dei parametri d'input. La nuova versione del codice è stata messa a confronto con la vecchia (che passava la matrice globale `K` tra i parametri d'input) per stimare la riduzione del tempo necessario all'assemblaggio: assemblare 1000 matrici di rigidezza ognuna di dimensione 4×4 in una matrice globale piena di dimensione 1000×1000 è risultato 100 più rapido con la nuova versione del software rispetto alla vecchia.

Il codice per l'assemblaggio delle forze esterne è molto simile, vedi Fig. 3.2.2.

```
function y = AssemblaForze (f, Ngdln, NNE, Nodi, F )
neq = Ngdln*NNE;
ij = zeros(neq,1);
i = 0;
for n = 1:NNE
    nodo = Nodi(n);
    for gdl = 1:Ngdln
        i = i + 1;
        ij(i) = Ngdln*(nodo-1) + gdl;
    end
end
for i = 1:neq
    I = ij(i);
    F(I) = F(I) + f(i);
end
y = F;
end
```

Fig. 3.2.2 – Codice per l'assemblaggio del vettore delle forze.

Abbiamo visto che l'espansione della matrice elementare alla dimensione della struttura è puramente simbolica; la vera operazione consiste nel collocare opportunamente i coefficienti per mezzo degli indici. L'uso degli indici può risultare un po' complesso perché ad ogni nodo sono associati più gradi di libertà (in questo esempio due). Per chiarire il problema, consideriamo l'elemento n.2 della struttura reticolare a tre aste mostrata nella Fig.3.2.2. Gli indici mostrati nella figura sono stati calcolati dal codice MATLAB precedente. Questi indici identificano la riga e la colonna nelle equazioni strutturali in cui bisognerà inserire i coefficienti degli elementi (la variabile `ij` del codice MATLAB). Così, la procedura descritta nel paragrafo 3.1 può realizzarsi senza espandere le singole matrici alla dimensione della struttura e senza la necessità di modificarne l'ordine.



$$2 \cdot i - 1 = 2 \cdot (i-1) + 1 = 5$$
$$2 \cdot i \quad = 2 \cdot (i-1) + 2 = 6$$
$$2 \cdot j - 1 = 2 \cdot (j-1) + 1 = 1$$
$$2 \cdot j \quad = 2 \cdot (j-1) + 2 = 2$$

$$[k]_2 = \frac{k_2}{2} \cdot \begin{bmatrix} 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \cdot \begin{Bmatrix} d_5 \\ d_6 \\ d_1 \\ d_2 \end{Bmatrix} = \begin{Bmatrix} F_5 \\ F_6 \\ F_1 \\ F_2 \end{Bmatrix}$$

Fig.3.2.3 Schema per indicizzare i gradi di libertà nodali

3.3 Numerazione dei nodi ed assemblaggio a banda

I nodi di una struttura possono essere numerati in modo arbitrario. Per una data struttura, tutti gli schemi di numerazione conducono alla stessa dimensione della matrice di rigidezza $[K]$, allo stesso numero di coefficienti diversi da zero, ma ad una loro diversa disposizione all'interno della matrice. E' auspicabile ottenere una disposizione nota come "*matrice a banda*", poiché in tal caso possiamo utilizzare delle tecniche di memorizzazione semplici ed efficienti della matrice $[K]$ e degli algoritmi di soluzione particolarmente veloci.

La fig.3.3.1 mostra una piccola struttura reticolare i cui nodi sono stati numerati in modo ottimale. Nella Fig.3.3.2 è mostrata la matrice di rigidezza globale, dove è stato utilizzato il simbolo X per individuare i

coefficienti diversi da zero. La matrice è sparsa ed ha tutti i suoi coefficienti diversi da zero concentrati in una banda disposta intorno alla diagonale. Questa banda include, in ogni riga, i coefficienti appartenenti ad entrambe i lati della diagonale. Poiché abbiamo a che fare con matrici simmetriche, è conveniente indicare con il simbolo B la "larghezza della semi banda" **che comprende anche il termine diagonale**. La larghezza totale della banda è quindi pari a $2B-1$. In una struttura reticolare con molti gradi di libertà, è evidente quanto sia sparsa la matrice e quanto sia stretta la sua banda.

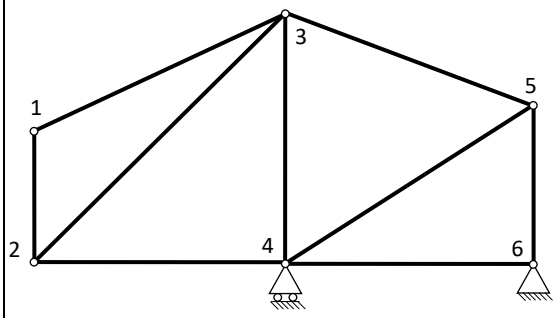


Fig. 3.3.1 - Esempio di numerazione dei nodi

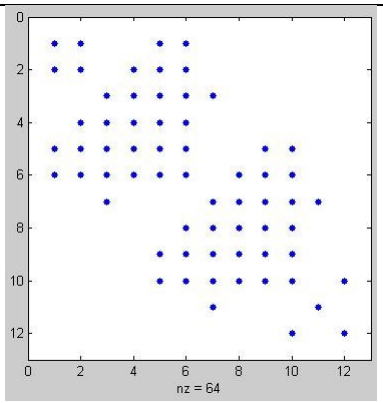


Fig.3.3.2 – Disposizione dei 64 coefficienti diversi da zero presenti nella matrice di rigidezza globale $[K]_{12x12}$ relativa alla struttura a lato.

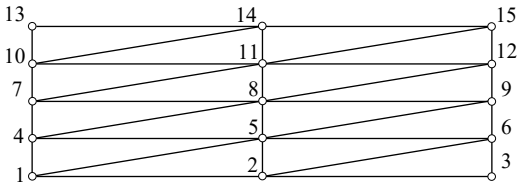
$K_{1,1}$	x	0	0	x	x
$K_{2,2}$	0	x	x	x	0
$K_{3,3}$	x	x	x	x	0
$K_{4,4}$	x	x	0	0	0
$K_{5,5}$	x	0	0	x	x
$K_{6,6}$	0	x	x	x	0
$K_{7,7}$	x	x	x	x	0
$K_{8,8}$	x	x	0	0	0
$K_{9,9}$	x	0	0	0	0
$K_{10,10}$	0	x	0	0	0
$K_{11,11}$	0	0	0	0	0
$K_{12,12}$	0	0	0	0	0

Fig.3.3.3 – Memorizzazione a banda; gli zeri rossi sono sempre nulli perché esterni alla matrice di partenza.

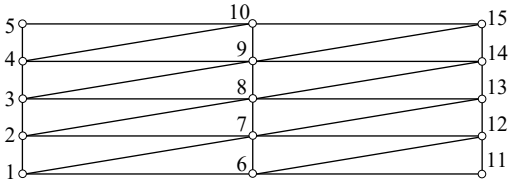
Grazie alla simmetria ed al blocco triangolare di zeri evidenti nella Fig.3.3.2, per memorizzare l'intera informazione contenuta nella matrice, in ogni riga i dobbiamo conservare solo i coefficienti appartenenti alle colonne che vanno dalla i-esima fino alla $(i+5)$-esima. Un modo adeguato per sfruttare questa osservazione è indicata nella Fig.3.3.3, dove ogni colonna j della riga i-esima della Fig.3.3.2 è stata spostata di $i-1$ posizioni verso sinistra. Così nella Fig.3.3.3 tutti i coefficienti della diagonale della matrice reale sono stati memorizzati nella prima colonna della tabella.

Il metodo di memorizzazione a banda della matrice $[K]$ richiede solo $N \cdot B$ posizioni rispetto alle $N^2/2$ posizioni richieste per memorizzare il triangolo superiore o inferiore dell'intera matrice simmetrica di ordine N. E' anche importante il risparmio di tempo richiesto per risolvere il sistema di equazioni. In uno dei seguenti paragrafi sarà descritto un solutore di Gauss adattato alle matrici a banda. Da un semplice conto delle operazioni richieste per eseguire l'algoritmo, si può osservare che il tempo è approssimativamente proporzionale a $N \cdot B^2/2$, rispetto al tempo necessario per eseguire le $N^3/6$ operazioni necessarie per risolvere l'intera matrice simmetrica. E' chiaro quindi che, nei casi pratici in cui spesso il numero di equazioni N è superiore a venti volte la larghezza di banda, il vantaggio di memorizzare la matrice in forma di banda è molto grande.

Normalmente si ottiene una piccola larghezza di banda numerando i nodi lungo la dimensione più corta della struttura, come in fig.3.3.1. In questo contesto, "*dimensione più corta*" va inteso in senso topologico e non geometrico: per esempio, nella fig.3.3.4 la dimensione "*topologicamente*" più corta è quella orizzontale.



(a)



(b)

Fig.3.3.4 – Due diverse numerazioni: nella soluzione (a), dove i nodi sono stati numerati nella direzione topologicamente più corta, $B = 2(\Delta N_{max} + 1) = 2(4+1) = 10$; nella soluzione (b), $\Delta N_{max} = 6$ e $B = 14$.

Indicando con *NGdLN* il numero di gradi di libertà per nodo (2 nel caso di strutture reticolari piane), la larghezza della semi banda vale:

$$B = NGdLN \cdot (\Delta N_{max} + 1)$$

dove ΔN_{max} indica la massima differenza tra i nodi di un elemento.

Nelle strutture ad anello (come un cilindro o un telaio) la numerazione dei nodi può condurre a larghezze di banda considerevoli. Esaminiamo il semplice anello costituito da 6 elementi monodimensionali con due gradi di libertà per nodo e cerchiamo la struttura che assume la matrice di rigidezza nel caso di due numerazioni alternative.

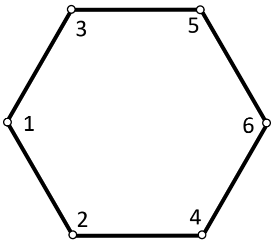


Fig. 3.3.5a - Esempio n.1

x	x	x	x	x	x	0	0	0	0	0	0
x	x	x	x	x	x	0	0	0	0	0	0
x	x	x	x	0	0	x	x	0	0	0	0
x	x	x	x	0	0	x	x	0	0	0	0
x	x	0	0	x	x	0	0	x	x	0	0
x	x	0	0	x	x	0	0	x	x	0	0
0	0	x	x	0	0	x	x	0	0	x	x
0	0	x	x	0	0	x	x	0	0	x	x
0	0	0	0	x	x	0	0	x	x	x	x
0	0	0	0	x	x	0	0	x	x	x	x
0	0	0	0	0	0	x	x	x	x	x	x
0	0	0	0	0	0	x	x	x	x	x	x

B = 6

x	x	x	x	x	x
x	x	x	x	x	0
x	x	0	0	x	x
x	0	0	x	x	0
x	x	0	0	x	x
x	0	0	x	x	0
x	x	0	0	x	x
x	0	0	x	x	0
x	x	x	x	0	0
x	x	x	0	0	0
x	x	0	0	0	0
x	0	0	0	0	0

Matrice a banda

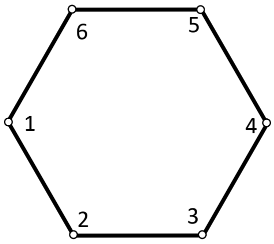


Fig. 3.3.5b - Esempio n.2

x	x	x	x	0	0	0	0	0	0	x	x
x	x	x	x	0	0	0	0	0	0	x	x
x	x	x	x	x	x	0	0	0	0	0	0
x	x	x	x	x	x	0	0	0	0	0	0
0	0	x	x	x	x	x	x	0	0	0	0
0	0	x	x	x	x	x	x	0	0	0	0
0	0	0	0	x	x	x	x	x	x	0	0
0	0	0	0	x	x	x	x	x	x	0	0
0	0	0	0	0	0	x	x	x	x	x	x
0	0	0	0	0	0	x	x	x	x	x	x
x	x	0	0	0	0	0	0	x	x	x	x
x	x	0	0	0	0	0	0	x	x	x	x

B = 12

x	x	x	x	0	0	0	0	0	0	x	x
x	x	x	0	0	0	0	0	0	x	x	0
x	x	x	x	0	0	0	0	0	0	0	0
x	x	x	0	0	0	0	0	0	0	0	0
x	x	x	x	0	0	0	0	0	0	0	0
x	x	x	0	0	0	0	0	0	0	0	0
x	x	x	x	0	0	0	0	0	0	0	0
x	x	x	0	0	0	0	0	0	0	0	0
x	x	x	x	0	0	0	0	0	0	0	0
x	x	x	0	0	0	0	0	0	0	0	0
x	x	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	0	0	0

Matrice a banda

Nel primo caso nella matrice a banda sono presenti 42 coefficienti diversi da zero, la semi banda è pari a $B = NGdLN \cdot (\Delta N_{max} + 1) = 2 \cdot (2 + 1) = 6$ e l'occupazione di memoria è pari a $NB = 12 \cdot 6 = 72$.

Nel secondo caso abbiamo 42 coefficienti diversi da zero (come nel caso precedente), la semi banda è pari a $B = NGdLN \cdot (\Delta N_{max} + 1) = 2 \cdot (5 + 1) = 12$ e l'occupazione di memoria è pari a $NB = 12 \cdot 12 = 144$. Di conseguenza aumenta anche il tempo di elaborazione.

La situazione può diventare intollerabile nel caso di strutture tridimensionali in quanto l'occupazione di memoria può diventare tale da impedire la soluzione del problema. Vediamo il semplice caso di un parallelepipedo diviso in elementi a 8 nodi, ma con una diversa numerazione di nodi.

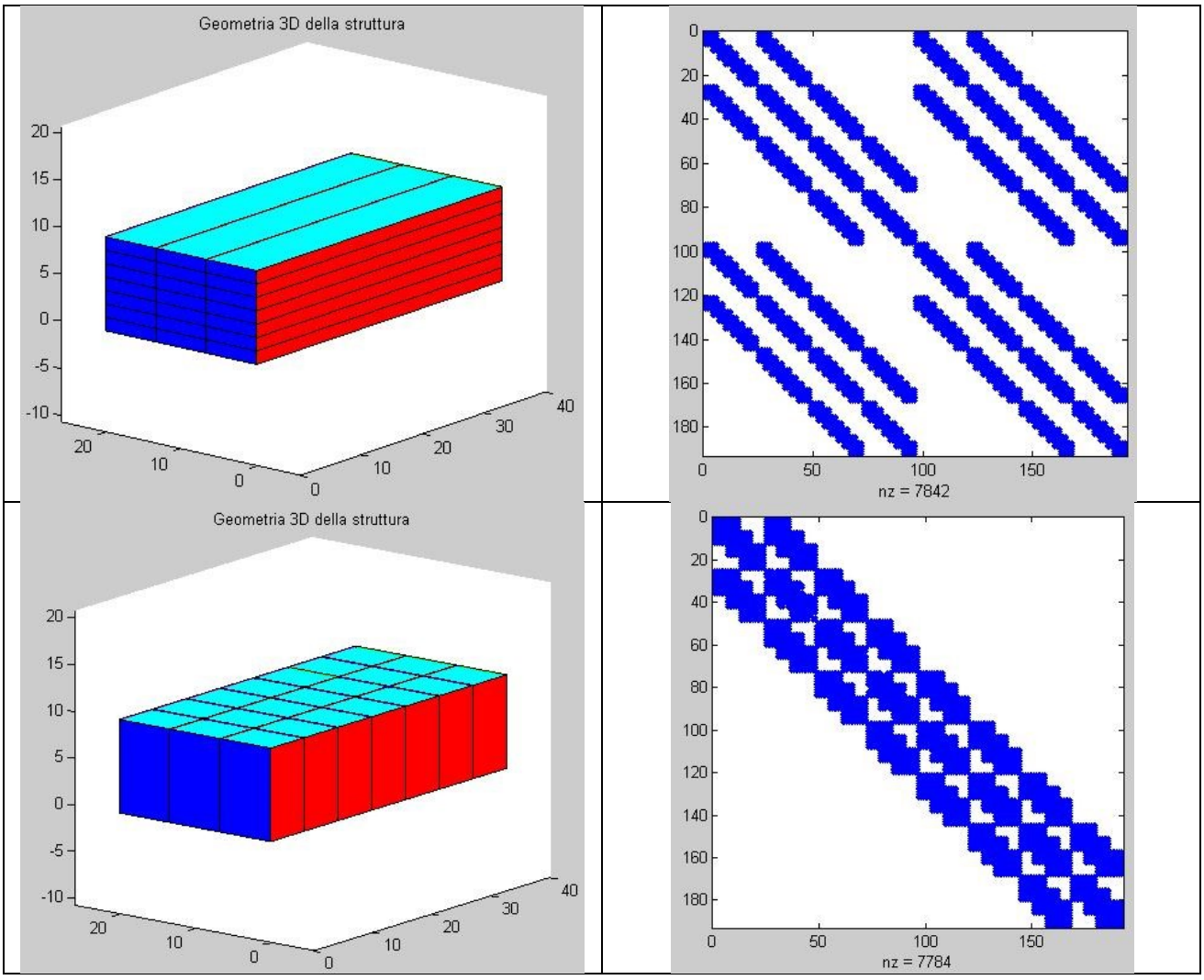
Il numero complessivo di nodi in entrambe i casi è pari a $N_x \times N_y \times N_z = 2 \times 4 \times 8 = 64$, e poiché i gradi di libertà per nodo sono 3, il numero complessivo di equazioni è pari a 192. L'occupazione complessiva di memoria per la sola matrice di rigidezza globale è pari a 36864 coefficienti da 8 bytes ciascuno, cioè 288 kbyte, mentre il numero di coefficienti diversi da zero è inferiore a 8000. Il tempo per risolvere il sistema di equazioni è proporzionale a $N^3/6 = 1179648$.

Nel primo esempio la numerazione procede prima in direzione z (la direzione topologicamente meno favorevole visto che $N_z = 8$), poi y e finalmente x. La semi larghezza di banda risulta quindi pari a $B = 126$, la memoria necessaria scende a 189 kbyte ed il tempo di calcolo diventa proporzionale a $N \cdot B^2/2 = 1524096$, cioè aumenta rispetto al tempo necessario per risolvere il sistema memorizzato in forma piena !!

Nel secondo esempio la numerazione procede prima in direzione x, poi y e finalmente z. La semi larghezza di banda risulta pari a $B = 36$, la memoria necessaria scende a 54 kbyte ed il tempo di calcolo diventa proporzionale a $N \cdot B^2/2 = 124416$, cioè è necessario un tempo dieci volte inferiore al primo caso.

I codici di calcolo FEM utilizzano degli algoritmi di riordino dei nodi che hanno lo scopo di ridurre la banda della matrice di rigidezza globale, riducendo di conseguenza la quantità di memoria necessaria ed i tempi di calcolo. Uno degli algoritmi più noti è il "*Symmetric reverse Cuthill-McKee permutation*" che è stato inserito nella libreria delle procedure di MATLAB.

Per programmare l'assemblaggio di $[K]$ in forma di banda, è necessario modificare il codice MATLAB prima indicato, ma è necessario anche utilizzare un solutore capace di operare su una tale disposizione dei coefficienti.




```
function AssemblaRigidezzaBanda (ke, Ngdln, Nnodi, Nodi)
%AssemblaRigidezzaBanda Assembla la matrice globale K in forma di banda
%       (triangolo superiore destro e diagonale nella prima colonna)
%=========================================================================
global MatriceGlobale
ngdle = Ngdln*Nnodi;    % Nnodi indica il numero di nodi di un elemento
ij(1:ngdle) = 0;
i = 0;
for n = 1:Nnodi
    last  = Ngdln*Nodi(n);
    first = last - Ngdln + 1;
    for gdl = first:last
        i = i + 1;
        ij(i) = gdl;
    end
end
for i = 1:ngdle
    I = ij(i);
    for j = 1:ngdle
        J = ij(j);
        if J >= I   % Assembla il triangolo superiore destro
           JB = J - I + 1;
           MatriceGlobale(I,JB) = MatriceGlobale(I,JB) + ke(i,j);
        end
    end
end
```

Fig. 3.3.6 – Codice MATLAB per l'assemblaggio della matrice di rigidezza in forma di semi banda.

Soluzione diretta del sistema di equazioni lineari

I metodi di assemblaggio più adatti dipendono dagli algoritmi che poi verranno utilizzati per la soluzione del sistema di equazioni lineari. Per esempio quando le matrici sono di grandissime dimensioni, ma sono anche sparse, può diventare conveniente memorizzarne solo i coefficienti diversi da zero. In queste circostanze diventano interessanti le tecniche di soluzione iterative (come l'algoritmo di Gauss-Seidel o quello del Gradiente Coniugato) che normalmente vengono ignorate perché la loro convergenza è lenta.

Qui di seguito verrà descritto brevemente l'algoritmo di Gauss (su cui ritorneremo con maggiori dettagli nel cap.5) per la soluzione dei sistemi di equazioni lineari del tipo $[K]\{D\} = \{F\}$ con la matrice $[K]$ simmetrica: il suo funzionamento giustifica il metodo di assemblaggio skyline, il più usato nei metodi FEM, che verrà descritto successivamente.

In base a questo metodo, per prima cosa si esprime il valore dell'incognita D_1 in funzione delle altre variabili e la si sostituisce nelle successive equazioni. Quindi si procede nell'elaborare nello stesso modo la seconda equazione, quindi la terza, etc. Seguendo questo processo di riduzione "*in avanti*", la matrice $[K]$ assume la forma triangolare superiore, con valori unitari sulla diagonale e viene modificato anche il termine noto $\{F\}$. Il valore delle incognite si trova con una "*sostituzione all'indietro*" e di conseguenza l'incognita D_1 è l'ultima ad essere calcolata.

Come si può dedurre da ciò che è stato detto, si usa l'i-esima equazione per eliminare l'i-esima incognita; cioè tutti i pivot sono i coefficienti della diagonale di $[K]$, il che consente di evitare la ricerca del pivot più grande in ogni eliminazione. Questo metodo è adeguato quando la matrice dei coefficienti $[K]$ è la matrice di rigidezza, perché in essa i termini sulla diagonale non sono mai troppo piccoli, a meno che la struttura risulti quasi instabile o nel caso in cui sia stata modellata in modo non adeguato.

Per un esempio dell'eliminazione gaussiana, consideriamo il seguente sistema di tre equazioni:

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \cdot \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix} \qquad [3.4.1]$$

Dividiamo la prima equazione per k_{11}, quindi dalla seconda equazione sottraiamo la prima dopo averla moltiplicata per k_{21}. Si ripete la procedura per tutte le righe successive.

1° passo:
$$\begin{bmatrix} 1 & k_{12}/k_{11} & k_{13}/k_{11} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \cdot \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} F_1/k_{11} \\ F_2 \\ F_3 \end{Bmatrix} \qquad [3.4.2]$$

2° passo:
$$\begin{bmatrix} 1 & k_{12}/k_{11} & k_{13}/k_{11} \\ 0 & k_{22} - k_{21} \cdot k_{12}/k_{11} & k_{23} - k_{21} \cdot k_{13}/k_{11} \\ 0 & k_{32} - k_{31} \cdot k_{12}/k_{11} & k_{33} - k_{31} \cdot k_{13}/k_{11} \end{bmatrix} \cdot \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} F_1/k_{11} \\ F_2 - k_{21} \cdot F_1/k_{11} \\ F_3 - k_{31} \cdot F_1/k_{11} \end{Bmatrix} \qquad [3.4.3]$$

Nelle spiegazioni che seguono, per risparmiare spazio utilizzeremo i seguenti simboli, legati in modo evidente all'ultima espressione [3.4.3]:

$$\begin{bmatrix} 1 & k_{12}^* & k_{13}^* \\ 0 & k_{22}^* & k_{23}^* \\ 0 & k_{32}^* & k_{33}^* \end{bmatrix} \cdot \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} F_1^* \\ F_2^* \\ F_3^* \end{Bmatrix} \qquad [3.4.4]$$

Ripetiamo i passi precedenti sul sistema 3.4.4:

$$\begin{bmatrix} 1 & k_{12}^* & k_{13}^* \\ 0 & 1 & k_{23}^*/k_{22}^* \\ 0 & 0 & k_{33}^* - k_{32}^* \cdot k_{23}^*/k_{22}^* \end{bmatrix} \cdot \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} F_1^* \\ F_2^*/k_{22}^* \\ F_3^* - k_{32}^* \cdot F_2^*/k_{22}^* \end{Bmatrix} \qquad [3.4.5]$$

Ultimo passo in avanti:

$$\begin{bmatrix} 1 & k_{12}^* & k_{13}^* \\ 0 & 1 & k_{23}^*/k_{22}^* \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \end{Bmatrix} = \begin{Bmatrix} F_1^* \\ F_2^*/k_{22}^* \\ \frac{F_3^* - k_{32}^* \cdot F_2^*/k_{22}^*}{k_{33}^* - k_{32}^* \cdot k_{23}^*/k_{22}^*} \end{Bmatrix} \qquad [3.4.6]$$

Arrivati a questo punto, la matrice $[K]$ risulta triangolare superiore ed è possibile procedere al calcolo delle incognite per mezzo della "*sostituzione all'indietro*":

$$D_3 = \frac{F_3^* - k_{32}^* \cdot F_2^*/k_{22}^*}{k_{33}^* - k_{32}^* \cdot k_{23}^*/k_{22}^*} \qquad [3.4.7]$$

$$D_2 = F_2^*/k_{22}^* - (k_{23}^*/k_{22}^*) \cdot D_3 \qquad [3.4.8]$$

$$D_1 = F_1^* - k_{13}^* \cdot D_3 - k_{12}^* \cdot D_2 \qquad [3.4.9]$$

Sono possibili le seguenti osservazioni:

- Dopo ogni riduzione, le rimanenti equazioni rimangono simmetriche e a banda: infatti possiamo osservare nell'eq. [3.4.4] che $k_{23}^* = k_{32}^*$;
- Se la larghezza della semi banda è pari a B, ogni riduzione coinvolge solo B righe e non più di B coefficienti in ogni riga (nell'esempio che precede $B = 2$);
- Ad ogni passo del processo di riduzione il vettore delle incognite $\{D\}$ non cambia.

L'analisi del metodo indica che la riduzione del termine noto $\{F\}$ può essere condotta dopo che la matrice $[K]$ è stata ridotta nella forma di matrice triangolare superiore. L'informazione necessaria per realizzare questa operazione consiste nel termine diagonale k_{ii}^* poco prima dell'eliminazione dell'i-esima riga oltre ai restanti coefficienti ridotti k_{ij}^* ,con $j > i$ nella matrice triangolarizzata [K]. Poiché sappiamo che il valore corretto del termine diagonale è unitario, per risparmiare spazio possiamo conservare i coefficienti ridotti k_{ii}^* sulla diagonale.

Le operazioni di riduzione sono divise in due parti per consentire l'analisi efficiente di due o più condizioni di carico applicate alla stessa struttura. **La riduzione in avanti della matrice $[K]$ deve essere effettuata una sola volta**, mentre quella del carico $\{F\}$ e la sostituzione all'indietro sono ripetute tante volte quante sono le condizioni di carico. In questo modo circa $B/2$ vettori di carico possono essere ridotti al costo di una singola riduzione di $[K]$, dove come al solito, B indica la larghezza della semi banda.

Per risolvere i sistemi di equazioni lineari è possibile utilizzare la procedura `linsolve(K,F,opts)` di MATLAB, che in base al valore assegnato al parametro `opts`, sceglie l'algoritmo diretto più adatto. Qui di seguito, solo a fini didattici, viene riportata un'altra implementazione dell'algoritmo di Gauss: MATLAB è ottimizzato per lavorare su matrici e vettori ed il codice qui descritto dovrebbe essere modificato per essere adattato a questo linguaggio di programmazione.

```
function [K, F] = Gauss_solver (K, F, iopt)
%Gauss_solver Solutore di Gauss per matrici quadrate, simmetriche e definite positive
%             if iopt = 0 --> Solo la Forward reduction
%             if iopt = 1 --> Solo la Back-Substitution
%             if iopt = 2 --> Forward & Back-Substitution
%             F(Neq,Nrhs) in input  è il vettore dei termini noti;
%             F(Neq,Nrhs) in output è il vettore delle soluzioni
% ================================================================
 Neq = size(K,1);
 if iopt == 0 || iopt == 2   % Forward reduction
    for irow = 1:Neq-1
        for i = irow+1:Neq
            if K(i,irow) ~= 0.
               K(i,irow) = K(i,irow)/K(irow,irow);
               K(i,irow+1:Neq) = K(i,irow+1:Neq) - K(i,irow)*K(irow,irow+1:Neq);
            end
        end
    end
 end
 if iopt == 1 || iopt == 2   % Forward reduction dei termini noti
    NRhS = size(F,2);
    for j = 1:NRhS
        for irow = 1:Neq-1
            for i = irow+1:Neq
                if K(i,irow) ~= 0.
                   F(i,j) = F(i,j) - K(i,irow)*F(irow,j);
                end
            end
        end
```

```
        F(Neq,j) = F(Neq,j)/K(Neq,Neq);  % Back-substitution: Calcola le soluzioni
        for irow = Neq-1:-1:1
            for k = irow+1:Neq
                F(irow,j) = F(irow,j) - K(irow,k)*F(k,j);
            end
            F(irow,j) = F(irow,j)/K(irow,irow);
        end
    end
 end
 end
```

Fig. 3.4.1 – Codice MATLAB per la soluzione del sistema di equazioni lineari con NRhS vettori destri (delle forze).

La libreria di MATLAB non prevede matrici memorizzate a banda ed in formato skyline. Per questo motivo qui di seguito vengono riportati i codici per l'applicazione dell'algoritmo di Gauss su questo tipo di matrici.

```
function [K,F] = BandGaussSolver(K, F, iopt)
% BandGaussSolver   Solutore di Gauss per matrici a banda
%     if iopt = 0 --> Solo la Forward reduction
%     if iopt = 1 --> Solo la Back-Substitution
%     if iopt = 2 --> Forward & Back-Substitution
%     K(Neq,Mband) è una matrice simmetrica, a banda, definita positiva;
%                  La diagonale occupa la prima colonna. La matrice K viene modificata.
%     F(Neq,Nforze) in input  è il vettore dei termini noti;
%     F(Neq,Nforze) in output è il vettore delle soluzioni
% ================================================================================
% Forward reduction of Matrix (Gauss elimination)
[Neq, Mband] = size(K);
Nforze = size(F,2);
if iopt == 0 || iopt == 2
    for n = 1: Neq
        for L = 2: Mband;
            if K(n,L) ~= 0.
                i = n + L - 1;
                coef = K(n,L)/K(n,1);
                K(i,1:Mband-L+1) = K(i,1:Mband-L+1) - coef*K(n,L:Mband);
                K(n,L) = coef;
            end
        end
    end
end
% Forward reduction of constants (Gauss elimination)
if iopt == 1 || iopt == 2
    for irhs = 1: Nforze
        for n = 1: Neq
            for L = 2: Mband;
                if K(n,L) ~= 0.
                    i = n + L - 1;
                    F(i,irhs) = F(i,irhs) - K(n,L)*F(n,irhs);
                end
            end
            F(n,irhs) = F(n,irhs)/K(n,1);
        end
%       Solve for unknowns by back-substitution
        for m = 2: Neq;
           n = Neq + 1 - m;
           for L = 2: Mband;
               if K(n,L) ~= 0.
                  k = n + L - 1;
                  F(n,irhs) = F(n,irhs) - K(n,L)*F(k,irhs);
               end
           end
        end
    end
end
```

Fig. 3.4.2 – Codice MATLAB per la soluzione del sistema di equazioni lineari con Nforze vettori destri.
Metodo di Gauss e matrice di rigidezza in forma di banda.

3.4 Assemblaggio delle equazioni in forma skyline

L'esame dell'eq.3.4.3 mette in evidenza una caratteristica importante del solutore di Gauss che può condurre ad un assemblaggio più efficiente delle matrici di rigidezza rispetto a quello a banda. Osserviamo infatti che, per esempio: $k_{23}^{*} = k_{23} - k_{21} \cdot k_{13}/k_{11}$. Ciò significa che se $k_{13} = 0$ allora $k_{23}^{*} = k_{23}$ e quindi il coefficiente originale non cambia. Generalizzando, possiamo dire che il coefficiente k_{ij} non viene

modificato dall'algoritmo se tutti i coefficienti della colonna *j* in posizione più alta (cioè dalla prima riga fino alla riga immediatamente precedente) sono nulli. Questa osservazione ha dato origine al metodo *skyline*, cioè "*linee di cielo*": vengono memorizzati in ordine, dall'alto verso il basso, tutti i coefficienti della colonna a partire dal primo coefficiente diverso da zero fino alla diagonale. Tornando all'esempio della mesh ad anello osserviamo che è stato necessario memorizzare complessivamente 54 coefficienti contro i 72 del metodo a banda. Bisogna però tenere conto del fatto che è necessaria una struttura dati più complessa: la matrice di rigidezza viene memorizzata in forma vettoriale ed è necessario un puntatore alle linee di cielo, che per ogni termine sulla diagonale indichi l'altezza della colonna. Naturalmente le matrici elementari devono essere assemblate direttamente in forma skyline senza passare dalla matrice quadrata ed il solutore di Gauss deve essere opportunamente modificato.

Nel caso dell'esempio precedente, il puntatore alle linee di cielo è il seguente:

$$Skyline = \{1 \quad 1 \quad 1 \quad 1 \quad 3 \quad 3 \quad 5 \quad 5 \quad 7 \quad 7 \quad 1 \quad 1\}$$

I coefficienti della matrice $[K]$ possono essere messi in memoria in due modi alternativi: o, come detto, vengono memorizzati in ordine, dall'alto verso il basso, tutti i coefficienti della colonna a partire dal primo coefficiente diverso da zero fino alla diagonale; nel nostro esempio i primi 15 coefficienti del vettore sarebbero i seguenti:

$$K = \{K_{11} \quad K_{12} \quad K_{22} \quad K_{13} \quad K_{23} \quad K_{33} \quad K_{14} \quad K_{24} \quad K_{34} \quad K_{44} \quad K_{35} \quad K_{45} \quad K_{55} \quad K_{36} \quad K_{46} \quad \cdots\}$$

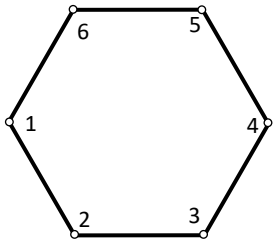


Fig. 3.3.5c - Esempio n.2

x	x	x	x	0	0	0	0	0	0	x	x
x	x	x	x	0	0	0	0	0	0	x	x
x	x	x	x	x	x	0	0	0	0	0	0
x	x	x	x	x	x	0	0	0	0	0	0
0	0	x	x	x	x	x	x	0	0	0	0
0	0	x	x	x	x	x	x	0	0	0	0
0	0	0	0	x	x	x	x	x	x	0	0
0	0	0	0	x	x	x	x	x	x	0	0
0	0	0	0	0	0	x	x	x	x	x	x
0	0	0	0	0	0	x	x	x	x	x	x
x	x	0	0	0	0	0	0	x	x	x	x
x	x	0	0	0	0	0	0	x	x	x	x

Matrice completa: in blu i coefficienti da salvare in forma skyline.

oppure, al contrario, vengono memorizzati dal basso verso l'alto, dalla diagonale fino all'ultimo coefficiente diverso da zero appartenente alla colonna; nel nostro esempio i primi coefficienti sarebbero i seguenti:

$$K = \{K_{11} \quad K_{22} \quad K_{12} \quad K_{33} \quad K_{23} \quad K_{13} \quad K_{44} \quad K_{34} \quad K_{24} \quad K_{14} \quad K_{55} \quad K_{45} \quad K_{35} \quad K_{66} \quad K_{56} \quad \cdots\}$$

Usando il vettore $Skyline$, è possibile costruire un puntatore alla posizione occupata dai coefficienti diagonali di $[K]$ nel vettore che contiene la matrice di rigidezza. Usando il primo sistema di memorizzazione il puntatore è il seguente:

$$Kadrs = \{1 \quad 3 \quad 6 \quad 10 \quad 13 \quad 17 \quad 20 \quad 24 \quad 27 \quad 31 \quad 42 \quad 54 \quad 55\}$$

nel secondo caso abbiamo invece:

$$Kadrs = \{1 \quad 2 \quad 4 \quad 7 \quad 11 \quad 14 \quad 18 \quad 21 \quad 25 \quad 28 \quad 32 \quad 43 \quad 55\}$$

La procedura che segue utilizza il primo metodo di memorizzazione.

```
function AssemblaRigidezzaSkyline (ke, Ngdln, Nnodi, Nodi, Skyline, Kadrs)
%AssemblaRigidezzaSkyline Assembla la matrice globale K in forma Skyline
% Skyline: Per ogni colonna, prima riga del coeff. diverso da zero
% Kadrs:   Per ogni colonna, indirizzo al primo coeff. di K diverso da zero
% Ngdln:   N. gradi di libertà per nodo (costante sulla struttura)
%=====================================================================
global MatriceGlobale
ngdle = Nnodi*Ngdln;
ij(1:ngdle)=0;
i = 0;
```

```
for n = 1:Nnodi
    last  = Ngdln*Nodi(n);
    first = last - Ngdln + 1;
    for gdl = first:last
        i = i + 1;
        ij(i) = gdl;
    end
end
for i = 1:ngdle
    I = ij(i);
    for j = 1:ngdle
        J = ij(j);
        if J >= I
           iadr = Kadrs(J) + I - Skyline(J);
           MatriceGlobale(iadr) = MatriceGlobale(iadr) + ke(i,j);
        end
    end
end
```

Fig. 3.5.1 – Codice per l'assemblaggio della matrice di rigidezza in forma di skyline.

```
function [K, V, ISTAT] = SkyLDLTGaussSolver (K, V, Kadrs, KEX)
%SkyLDLTGaussSolver: Solutore di Gauss per matrici simmetriche in forma skyline
%                    Fattorizza la matrice di rigidezza nella forma L*D*Lt
%  NEQ  = Numero di equazioni
%    K  = Vettore che in ingresso contiene i coefficienti della matrice di rigidezza
%    V  = Vettore che in ingresso contiene il vettore dei carichi
%  Kadrs = Vettore che contiene l'indirizzo degli elementi diagonali della matrice K
%  if KEX = 1   Fattorizzazione della matrice K
%  if KEX = 2   Calcolo degli spostamenti
%  In uscita:
%     K = D and L - Fattori della matrice K
%     V = Vettore della soluzione.
% ====================================================================
ISTAT = 0;
Neq = size(Kadrs,1) - 1;
Kadrs = cast(Kadrs,'uint32');
if KEX == 1   %  REDUCE COEFFICIENT MATRIX (A)
   for n = 2:Neq
       ku = Kadrs(n-1) + 1;
       kn = Kadrs(n);
       kl = kn - 1;
       kh = kl - ku;
       if kh > 0
          k = n - kh;
          ic = 0;
          klt = ku;
          for j = 1:kh
              ic = ic + 1;
              klt = klt + 1;
              ki = Kadrs(k);
              nd = ki - Kadrs(k-1) - 1;
              if nd > 0
                 kk = min(ic,nd);
                 c = 0.0;
                 for l = 1:kk
                     c = c + K(ki-l)*K(klt-l);
                 end
                 K(klt) = K(klt) - c;
              end
              k = k + 1;
          end
       end
       if kh >= 0
          k = n;
          b = 0.0;
          for kk = kl:-1:ku
              k = k - 1;
              ki = Kadrs(k);
              c = K(kk)/K(ki);
              b = b + c*K(kk);
              K(kk) = c;
          end
          K(kn) = K(kn) - b;
       end
       if K(kn) <= 0
          fprintf('\nWARNING - Matrice di rigidezza non Definita Positiva\n');
```

```
            fprintf('Pivot non positivo per l''equazione n.%u\nPivot = %20.12e\n',n,K(kn));
            ISTAT = 1;
        end
    end
else
 % ==================================
 % Reduce Right-Hand-Side Load Vector
 % ==================================
    for n = 2:Neq
        kl = Kadrs(n) - 1;
        ku = Kadrs(n-1) + 1;
        if kl >= ku
            k = n;
            c = 0.0;
            for kk = kl:-1:ku
                k = k - 1;
                c = c + K(kk)*V(k);
            end
            V(n) = V(n) - c;
        end
    end
 % ===============
 % Back-Substitute
 % ===============
    for n = 1:Neq
        k = Kadrs(n);
        V(n) = V(n)/K(k);
    end
    if Neq == 1
       return
    end
    n = Neq;
    for l = 2:Neq
        kl = Kadrs(n) - 1;
        ku = Kadrs(n-1) + 1;
        if kl >= ku
            k = n;
            for kk = kl:-1:ku
                k = k - 1;
                V(k) = V(k) - K(kk)*V(n);
            end
        end
        n = n - 1;
    end
end
```

Fig. 3.5.2 – Codice per la soluzione del sistema di equazioni lineari. Metodo di Gauss e matrice di rigidezza in forma skyline.

3.6 Memorizzazione delle matrici in forma sparsa

Un sistema di equazioni lineari è detto *sparso* se solo un numero relativamente piccolo di coefficienti k_{ij} è diverso da zero. Usare i metodi di fattorizzazione su questi sistemi non è efficiente perché la maggior parte delle $\emptyset(N^3)$ operazioni aritmetiche necessarie a risolvere il sistema di equazioni si applica a dei coefficienti nulli. Inoltre capita di dover risolvere problemi talmente grandi da occupare tutta la memoria disponibile e non ha senso allocare memoria per registrare dei coefficienti nulli. Ci sono due obiettivi distinti (e non sempre compatibili) per ogni metodo a matrice sparsa: ridurre il tempo di calcolo e/o risparmiare spazio di memoria.

Abbiamo già osservato che le matrici di rigidezza possono essere salvate in un formato ridotto (a banda o in forma skyline) allocando solo gli elementi che possono risultare diversi da zero, il che semplifica la loro localizzazione: questi metodi comportano talvolta lo spreco di alcune locazioni di memoria che saranno occupate da coefficienti nulli (si vedano per esempio le ultime due colonne della matrice di rigidezza memorizzata in forma skyline relative all'esempio Fig. 3.3.5c). Cosa si può dire sulle matrici sparse di forma più generale? Quando una matrice sparsa di dimensioni $N \times N$ contiene solo pochi elementi diversi da zero, è sicuramente inefficiente – e spesso fisicamente impossibile – allocare la memoria per tutti gli N^2 coefficienti. Anche se fosse possibile, sarebbe inefficiente o proibitivo in termini di tempo macchina cercare in tale spazio i coefficienti diversi da zero per una loro elaborazione. Chiaramente è necessaria una adeguata struttura dei dati che preveda, oltre alla registrazione dei coefficienti diversi da zero, anche un modo efficiente per individuarli e rendere più rapide le abituali operazioni che si applicano sulle matrici.

Sfortunatamente non esiste uno schema standard di uso generale. Knuth [1] ha descritto un metodo; le librerie matematiche "Yale Sparse Matrix Package" [2], ITPACK[3] e PCGPACK [4] ne descrivono altri.

Nel seguito verrà descritto il modo in cui vengono memorizzate le matrici sparse con la libreria di MATLAB. Quando una matrice è sparsa, MATLAB registra solo i suoi coefficienti diversi da zero e i loro indici usando il così detto formato Harwell-Boeing o a colonna compressa che prevede tre vettori. Consideriamo una matrice rettangolare formata da M righe ed N colonne che contenga NNZ coefficienti diversi da zero memorizzati in un vettore di lunghezza $NZmax$. Il primo vettore contiene tutti i coefficienti diversi da zero: la sua lunghezza è pari a $NZmax$. Il secondo vettore contiene l'indice di riga dei corrispondenti coefficienti diversi da zero, memorizzati nelle prime NNZ posizioni, ma la lunghezza del vettore è pari a $NZmax$. Il terzo vettore contiene N valori interi che indicano il numero di coefficienti diversi da zero presenti in ogni colonna della matrice, oltre ad un ulteriore intero che indica il numero NNZ di coefficienti diversi da zero: la sua lunghezza è quindi pari a $N+1$. Questo tipo di memorizzazione richiede lo spazio necessario per contenere $NZmax$ coefficienti di numeri reali in doppia precisione (8 bytes) oltre a $NZmax + N + 1$ numeri interi (4 bytes). Si può notare come la quantità di memoria necessaria dipenda da $NZmax$ e dal numero di colonne N. Per riservare la memoria necessaria a contenere una matrice sparsa formata da M righe ed N colonne il cui numero di coefficienti diversi da zero è pari a $NZmax$ si usa la seguente routine:

S = spalloc (M, N, $NZmax$)

La memoria richiesta per memorizzare una matrice sparsa costituita da un grande numero di righe e da poche colonne è molto inferiore alla memoria necessaria a memorizzare la sua trasposta.

S1 = spalloc(2^20,2,1);

S2 = spalloc(2,2^20,1);

Nel primo esempio abbiamo M =1048576 righe, N = 2 colonne e $NZmax = 1$ coefficienti diversi da zero. Lo spazio di memoria necessario è pari a 8 bytes per l'unico coefficiente diverso da zero della matrice S oltre a $4 \cdot (NZmax + N + 1) = 4 \cdot (1 + 2 + 1) = 16$ bytes per il vettore degli indirizzi.

Nel secondo esempio abbiamo M = 2 righe, N = 1048576 colonne e $NZmax = 1$. In questo caso lo spazio riservato sarà pari a 8 bytes per l'unico coefficiente diverso da zero della matrice S oltre a $4 \cdot (NZmax + N + 1) = 4 \cdot (1 + 1048576 + 1) = 4194312$ bytes per il vettore degli indirizzi.

Data la matrice $[A]$ memorizzata in forma piena, è possibile convertirla in forma sparsa con il seguente commando:

S = sparse(A)

Per esempio, data la seguente matrice quadrata 4×4:

$$[A] = \begin{bmatrix} 0 & 0 & 0 & 5 \\ 0 & 2 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 0 & 0 & 4 & 0 \end{bmatrix}$$

il commando S = sparse(A) produce la seguente stampa:

```
S =
   (3,1)       1
   (2,2)       2
   (3,2)       3
   (4,3)       4
   (1,4)       5
```

La struttura dati interna è la seguente:

$S = \{1 \quad 2 \quad 3 \quad 4 \quad 5\}$ $\quad NNZ = 5$

$riga = \{3 \quad 2 \quad 3 \quad 4 \quad 1\}$ $\quad NNZ = 5$

$Ncoef = \{1 \quad 2 \quad 1 \quad 1 \quad 5\}$ $\quad Lunghezza = N + 1 = 5$

dove N indica il numero di colonne della matrice A.

Usando il commando:

S = sparse(i,j,s,m,n)

è possibile creare una matrice sparsa a partire dalla lista dei suoi coefficienti diversi da zero. i e j sono vettori che rispettivamente indicano gli indici delle righe e delle colonne dei coefficienti della matrice diversi da zero; s è il vettore dei coefficienti diversi da zero i cui indici sono indicati dalle coppie (i,j); m indica il numero di righe ed n il numero delle colonne.
La matrice S indicata nell'esempio precedente può essere generata con il seguente commando:

S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)

Quando la matrice è simmetrica è naturalmente possibile registrarne solo la triangolare superiore; d'altra parte le attuali procedure di MATLAB per la soluzione diretta dei sistemi di equazioni lineari prevedono che la matrice sia stata registrata completamente. La seguente procedura assembla l'intera matrice di rigidezza in forma sparsa.

```
function AssemblaRigidezzaSparsa (ke, Ngdln, Nnodi, Nodi)
%AssemblaRigidezzaSparsa K è una matrice sparsa.
%=========================================================================
global MatriceGlobale
Neq = size(MatriceGlobale,1);
ngdle = Nnodi*Ngdln;    % Nnodi indica il numero di nodi di un elemento
ij = zeros(ngdle,1);
i = 0;
for n = 1:Nnodi
    last  = Ngdln*Nodi(n);
    first = last - Ngdln + 1;
    for gdl = first:last
        i = i + 1;
        ij(i) = gdl;
    end
end
I = zeros(ngdle*ngdle,1);
J = I;
v = I;
k = 0;
for i = 1:ngdle
    for j = 1:ngdle
        k = k + 1;
        I(k) = ij(i);
        J(k) = ij(j);
        v(k) = ke(i,j);
    end
end
MatriceGlobale = MatriceGlobale + sparse(I,J,v,Neq,Neq);
end
```

Fig. 3.6.1 – Codice per l'assemblaggio della matrice di rigidezza in forma sparsa.

Nel caso in cui si decida di risolvere il sistema di equazioni lineari con un metodo iterativo, l'unica operazione che si effettua sulla matrice è il suo prodotto per un vettore. In questo caso, se della matrice simmetrica ne è stata registrata solo la metà, è necessario prevedere una procedura adeguata per la sua moltiplicazione per un vettore. Usando le matrici sparse di MATLAB sono necessari tre passi:

1) Moltiplicare la matrice per il vettore (dove la matrice contiene solo il triangolo superiore)
2) Moltiplicare la trasposta della matrice per il vettore e sommare il risultato al primo vettore
3) Sottrare il prodotto della diagonale della matrice per il vettore (che altrimenti comparirebbe due volte).

```
v = A*V + A'*V - diag(A).*V;
```

Bibliografia

[1] D.E. Knuth – "Fundamental Algorithms", vol.1 of The Art of Computer Programming, 1968.
[2] S.C. Eisenstat, M.C.Gursky, M.H.Schultz, A.H.Sherman - "Yale Sparse Matrix Package" Technical Reports 112 e 114 (Yale University Department of Computer Science), 1977.
[3] D.R.Kincaid, J.R.Respess, D.M.Young, R.G.Grimes – ACM Transactions on Mathematical Software, vol.8, pp.302-322.
[4] PCGPACK User's Guide (New Haven: Scientific Computing Associate, Inc.)