PROGETTO E IMPLEMENTAZIONE DEL CODICE “STRAUS DETROYER”

ELEMENTO DI MINDLIN A 8 NODI, MODALITA DI CARICO SUPERFICIALE e VINCOLI

Università degli studi di Padova

Dipartimento di Ingegneria Civile, Edile e Ambientale

Corso di Meccanica Computazionale

PROFESSORI: Mazzucco G., Pomaro B., Demarchi N.

GRUPPO DI LAVORO: Bonato M., Dadduzio A., Gallo G., Guglietta N., Labrag M., Malenza G., Moscardo M., Righetto A., Viero E.

RELAZIONE A CURA DI: Gallo Giuseppe

Sommario

1) INTRODUZIONE AL CODICE

Il codice agli elementi finiti proposto è stato sviluppato in linguaggio matlab facendo uso di una struttura orientata agli oggetti. Nello specifico la struttura del codice si compone di 7 classi principali alcune delle quali richiamano delle classi minori (classi figlie).

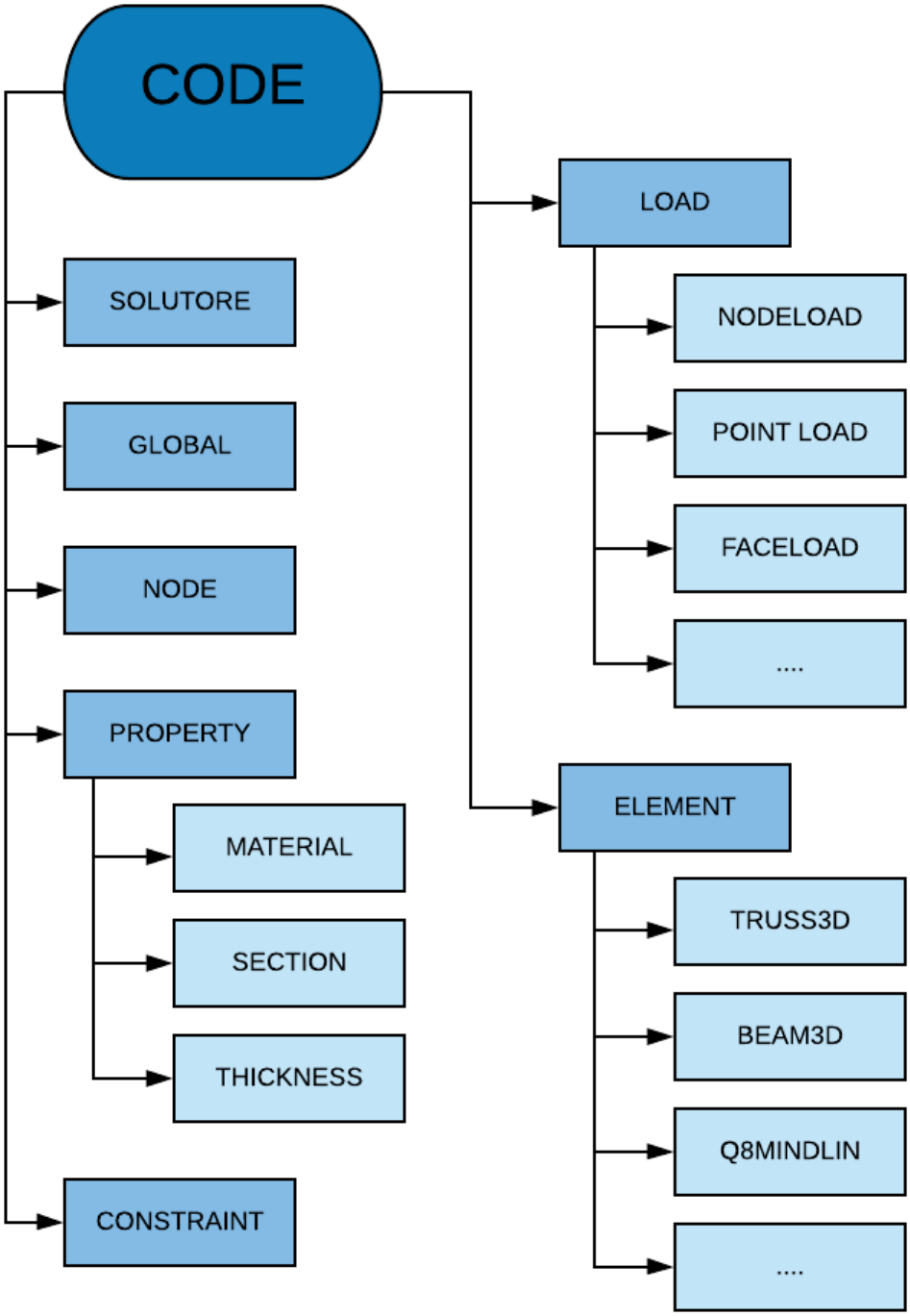


FIGURA 1 STRUTTURA DEL CODICE

1.1) BREVE DESCRIZIONE DELLE CLASSI

- SOLUTORE
  La classe del solutore si occupa di richiamare, assemblare e risolvere il sistema lineare $K\,\vec{u} = \vec{f}$ (con K matrice di rigidezza, $\vec{u}$ vettore degli spostamenti nodali e $\vec{f}$ vettore di carichi equivalenti sui nodi) a partire dai dati forniti dagli oggetti definiti sulle classi dei carichi, degli elementi e dei vincoli.
- GLOBAL
  La classe GLOBAL contiene indicazioni sul sistema di riferimento che si intende adottare come, ad esempio, la dimensione dello spazio e il numero di gradi di libertà che si intende conferire ai nodi. La classe fornisce anche dei metodi che permettono il calcolo della distanza tra due punti e di matrici di rotazione tridimensionale.

- NODE
  La classe dei nodi, molto semplicemente, contiene informazioni sull'indice che si intende assegnare al nodo e sulle sue coordinate spaziali

- PROPERTY
  La classe delle proprietà cerca di racchiudere tutte le proprietà che posso caratterizzare un elemento e lo fa mediante 3 classi figlie:
  - MATERIAL
    Per definire il materiale di cui è composto l'elemento. In MATERIAL sono definiti il coefficiente di Poinsson υ, il modulo di Young E la densità d ecc..
  - SECTION
    La classe SECTION fornisce informazioni sulla sezione di elementi monodimensionali come ad esempio beam e truss. In base alla geometria assegnate, questa classe permette di calcolare parametri essenziali come l'inerzia e l'area della sezione.
  - THICKNESS
    La classe THICKNESS riguarda esclusivamente gli elementi plate e, molto banalmente, definisce lo spessore dell'elemento.

- CONSTRAINT*
  Classe dei vincoli che permette di bloccare o imprimere gli spostamenti sui nodi.

- LOAD
  Classe madre dei carichi che incapsula dentro di se 3 classi figlie ognuna delle quali specializzata in una tipologia di carico:
  - NODELOAD
    Imprime un carico nodale su un nodo
  - POINTLOAD
    Imprime un carico interno ad un elemento monodimensionale
  - FACELOAD*
    Imprime un carico superficiale su di un elemento bidimensionale
- ELEMENTI
  Classe madre degli elementi che racchiude tutte le sue classi figlie ognuna delle quali definisce un elemento diverso:
  - TRUSS3D
    Elemento monodimensionale in grado di assorbire solo sforzi assiali
  - BEAM3D
    Elemento monodimensionale che implementa la teoria di Eulero Bernulli e che prevede la possibilità, oltre che di assorbire sforzo assiale, di inflettersi.
  - Q8MINDLIN*
    Elemento plate ad 8 nodi che implementa la teoria della lastra di Mindlin la quale conferisce all'elemento la possibilità di assorbire deformazione membranale, flessionale e anche tagliante.

*In questa relazione si approfondirà nello specifico le parti di codice relative ai vincoli, e quindi alla classe CONSTRAINT, alla modalità di carico superficiale dei plate (classe FACELOAD e assemblaggio nella classe SOLUTORE) e all'elemento plate di Mindlin (classe Q8MINDLIN).

2) ELEMENTO PLATE DI MINDLIN

2.1) TEORIA DELLA LASTRA DI MINDLIN

La teoria della lastra di Mindlin, essendo una teoria applicata a corpi di sviluppo prevalentemente a due dimensioni, semplifica il problema meccanico del corpo tridimensionale con un problema bidimensionale il quale fa riferimento esclusivamente al piano medio della lastra.

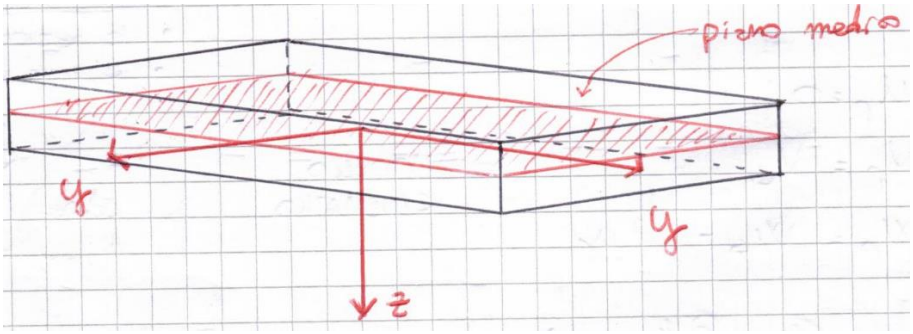


FIGURA 2 SISTEMA DI RIFERIMENTO E PIANO MEDIO

2.1.1) Modello cinematico

Supponendo quindi di definire un sistema di riferimento con assi x e y complanari alla lastra e asse z normale, la teoria di Mindlin prevede due ipotesi:

1) Lo spostamento s_z è indipendente dalla cordinata z

$$\frac{\partial s_z}{\partial z} = 0$$

In altre parole, si sta dicendo che le generatrici della lastra sono inestensibili.

2) La generatrice resta retta anche a deformazione avvenuta. Questa ipotesi è equivalente all'ipotesi della teoria di Eulero-Bernulli per gli elementi monodimensionali in cui si ipotizzava che la sezione rimanesse piana.

Si osservi che non viene detto nulla riguardo al mantenimento della perpendicolarità della generatrice al piano medio. Questo a intendere che, come avviene analogamente nella teoria di Timoshenko per le travi, per assorbire deformazione tagliante si deve prevedere la possibilità di perdere la perpendicolarità.

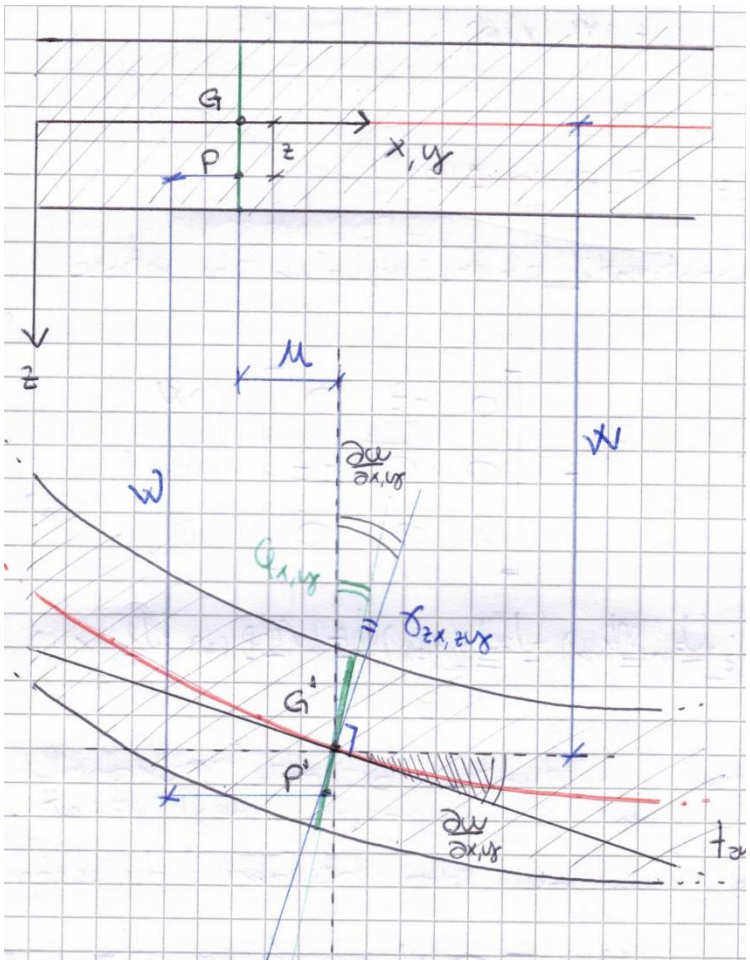


FIGURA 3 SCHEMA DEL MODELLO CINEMATICO DELLA TEORIA DI MINDLIN

A partire da queste due ipotesi si procede ora a illustrare il modello cinematico (figura 3) che permette di passare dai 3 gradi di libertà canonici del problema tridimensionale (x,y,z) a 5 gradi di libertà riferiti al piano medio della lastra (x,y)

$$\vec{s} = (s_x, s_y, s_z)_{(x,y,z)} \quad \xrightarrow{mod.\ cine.} \quad \vec{s} = (u, v, w, \varphi_x, \varphi_y)_{(x,y)}$$

Grazie allo schema in figura, si può visualizzare come vengono composti gli spostamenti $s_x, s_y \; e \; s_z$ attraverso gli spostamenti generalizzati $u, v, w, \varphi_x \; e \; \varphi_y$ (sempre riferiti al piano medio) e formulare le seguenti relazioni.

$$\begin{cases} s_x = u + z\,\varphi_x \\ s_y = v - z\,\varphi_y \\ s_z = w \end{cases}$$

Separando i gradi di libertà legati ad un comportamento deformativo nel piano ($u\ e\ v$) da quelle legate invece ad un comportamento deformativo fuori piano ($w, \varphi_x\ e\ \varphi_y$) e riscrivendo il tutto in forma matriciale, si ottiene:

$$\begin{bmatrix} s_x \\ s_y \\ s_z \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} 0 & 0 & z \\ 0 & -z & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} w \\ \varphi_x \\ \varphi_y \end{bmatrix}$$

$$\vec{s}_{(x,y,z)} = \boldsymbol{n_m} \cdot \vec{U}_{m_{(x,y)}} + \boldsymbol{n_f}_{(z)} \cdot \vec{U}_{f\ (x,y)}$$

2.1.2) Modello deformativo

Derivando il campo degli spostamenti mediante le equazioni di congruenza diretta si ottengono poi le seguenti espressioni:

$$\begin{cases} \varepsilon_x = \dfrac{\partial u}{\partial x} + z\dfrac{\partial \varphi_y}{\partial x} \\ \varepsilon_y = \dfrac{\partial v}{\partial y} - z\dfrac{\partial \varphi_x}{\partial y} \\ \varepsilon_z = 0 \qquad (per\ ipotesi) \\ \gamma_{xy} = \left(\dfrac{\partial v}{\partial x} + \dfrac{\partial u}{\partial y}\right) + z\left(\dfrac{\partial \varphi_y}{\partial y} - \dfrac{\partial \varphi_x}{\partial x}\right) \\ \gamma_{zx} = \dfrac{\partial w}{\partial x} + \varphi_y \\ \gamma_{zy} = \dfrac{\partial w}{\partial y} - \varphi_x \end{cases}$$

A partire da queste equazioni, che esprimono le deformazioni in funzioni degli spostamenti generalizzati, si definiscono tre categorie di deformazioni generalizzate: le deformazioni membranali $\eta_x, \eta_y\ e\ \eta_{xy}$, le deformazioni flessionali $\chi_x, \chi_y\ e\ \chi_{xy}$ e le deformazioni taglianti $\xi_x\ e\ \xi_y$.

Comportamento deformativo nel piano $\vec{q}_m$	**Comportamento deformativo fuori piano** $\vec{q}_f$	
deformazioni membranali generalizzate	deformazioni flessionali generalizzate	Deformazioni taglianti generalizzate
$\eta_x = \frac{\partial u}{\partial x}$ $\eta_y = \frac{\partial v}{\partial y}$ $\eta_{xy} = \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)$	$\chi_x = \frac{\partial \varphi_y}{\partial x}$ $\chi_y = -\frac{\partial \varphi_x}{\partial y}$ $\chi_{xy} = \frac{\partial \varphi_y}{\partial y} - \frac{\partial \varphi_x}{\partial x}$	$\xi_x = \frac{\partial w}{\partial x} + \varphi_y$ $\xi_y = \frac{\partial w}{\partial y} - \varphi_x$

Le equazioni che legano le deformazioni puntuali da quelle generalizzati diventano perciò:

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \\ \gamma_{zx} \\ \gamma_{zy} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \eta_x \\ \eta_y \\ \eta_{xy} \end{bmatrix} + \begin{bmatrix} z & 0 & 0 & 0 & 0 \\ 0 & z & 0 & 0 & 0 \\ 0 & 0 & z & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \chi_x \\ \chi_y \\ \chi_{xy} \\ \xi_x \\ \xi_y \end{bmatrix}$$

$$\vec{\varepsilon}_{(x,y,z)} = \boldsymbol{b_m} \cdot \vec{q}_{m_{(x,y)}} + \boldsymbol{b_f}_{(z)} \cdot \vec{q}_{f\,(x,y)}$$

Le equazioni che invece legano le deformazioni generalizzate con gli spostamenti generalizzati, scritti in forma matriciale, sono:

$$\begin{bmatrix}\eta_x\\ \eta_y\\ \eta_{xy}\end{bmatrix}=\begin{bmatrix}\frac{\partial}{\partial x} & 0\\ 0 & \frac{\partial}{\partial y}\\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x}\end{bmatrix}\cdot\begin{bmatrix}u\\ v\end{bmatrix} \qquad \begin{bmatrix}\chi_x\\ \chi_y\\ \chi_{xy}\\ \xi_x\\ \xi_y\end{bmatrix}=\begin{bmatrix}0 & 0 & \frac{\partial}{\partial x}\\ 0 & -\frac{\partial}{\partial y} & 0\\ 0 & -\frac{\partial}{\partial x} & \frac{\partial}{\partial y}\\ \frac{\partial}{\partial x} & 0 & 1\\ \frac{\partial}{\partial y} & -1 & 0\end{bmatrix}\cdot\begin{bmatrix}w\\ \varphi_x\\ \varphi_y\end{bmatrix}$$

$$\vec{q}_{m_{(x,y)}}=\boldsymbol{L_m}\cdot\vec{U}_{m_{(x,y)}} \qquad \vec{q}_f=\boldsymbol{L_f}\cdot\vec{U}_{f_{(x,y)}}$$

2.1.3) Modello tensionale

Alle corrispettive deformazioni generalizzate $\eta_x,\ \eta_y,\ \eta_{xy},\ \chi_x,\ \chi_y,\ \chi_{xy,}\ \xi_x\ e\ \xi_y$ vengono definite le tensioni generalizzate $N_x,\ N_y,\ N_{xy},\ M_x,\ M_y,\ M_{xy,}\ T_x\ e\ T_y$ ottenute integrando le canoniche tensioni $\sigma_x, \sigma_y, \tau_{xy}, \tau_{zx}\ e\ \tau_{zy}$ lungo la generatrice della lastra.

Posto quindi il sistema di riferimento a metà della generatrice lunga h si definiscono:

Comportamento tensionale nel piano $\vec{\boldsymbol{Q}}_{\boldsymbol{m}}$	**Comportamento tensionale fuori piano** $\vec{\boldsymbol{Q}}_{\boldsymbol{f}}$	
Sforzi membranali	Momenti flettenti e torcenti	Tagli
$N_x=\int_{-h/2}^{h/2}\sigma_x\,dz$	$M_x=\int_{-h/2}^{h/2}z\cdot\sigma_x\,dz$	$T_x=\int_{-h/2}^{h/2}\tau_{zx}\,dz$
$N_y=\int_{-h/2}^{h/2}\sigma_y\,dz$	$M_y=\int_{-h/2}^{h/2}z\cdot\sigma_y\,dz$	$T_y=\int_{-h/2}^{h/2}\tau_{zy}\,dz$
$N_{xy}=\int_{-h/2}^{h/2}\tau_{xy}\,dz$	$M_{xy}=\int_{-h/2}^{h/2}z\cdot\tau_{xy}\,dz$	

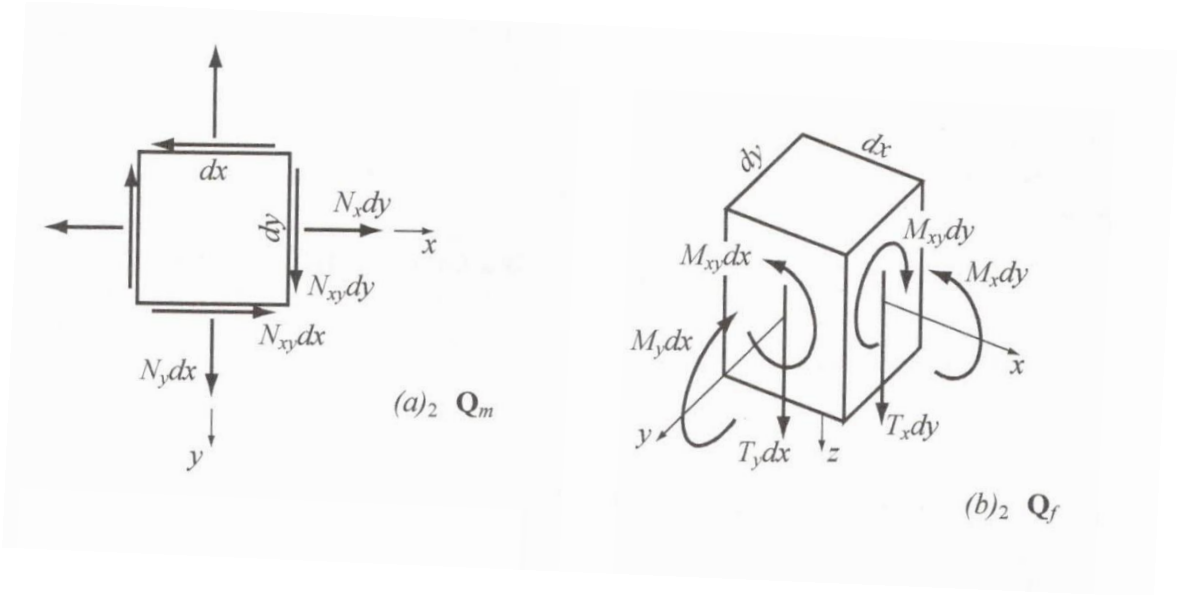


FIGURA 4 VISUALIZZAZIONE DEI PARAMETRI DELLA SOLLECITAZIONE (DELL'ACQUA VOLUME 2)

2.1.4) Legame costitutivo

Arrivati a questo punto sono state definite grandezze generalizzate sia in termini di deformazioni con $\vec{q} = \{\eta_x,\ \eta_y,\ \eta_{xy},\ \chi_x,\ \chi_y,\ \chi_{xy},\ \xi_x, \xi_y\}^T$ che in termini di tensione con $\vec{Q} = \{N_x,\ N_y,\ N_{xy},\ M_x,\ M_y,\ M_{xy},\ T_x, T_y\}^T$ ma non è ancora stato detto nulla sul legame costitutivo che le mette in relazione. Come per le grandezze puntuali si ipotizza che esita una matrice costitutiva **D** tale per cui:

$$\vec{Q} = \boldsymbol{D}\ \vec{q} \qquad legame\ costitutivo\ per\ grandezze\ generalizzate$$

Al fine di determinare tale matrice, è però fondamentale determinare anzitutto la matrice **d** che invece mette in relazione $\vec{\varepsilon} = \{\varepsilon_x, \varepsilon_y, \gamma_{xy}, \gamma_{zx}, \gamma_{zy}\}^T$ e $\vec{\sigma} = \{\sigma_x, \sigma_y, \tau_{xy}, \tau_{zx}\ e\ \tau_{zy}\}^T$.

$$\vec{\sigma} = \boldsymbol{d}\ \vec{\varepsilon} \qquad legame\ costitutivo\ per\ grandezze\ puntuali$$

Essendo $\varepsilon_Z = 0$ per ipotesi cinematica e assumendo $\sigma_z \cong 0$ come ipotesi semplificativa, è infatti necessario prendere in considerazione alcuni accorgimenti a livello costitutivo. Separando le tensioni e le deformazioni nel piano da quelle fuori

piano, pensando di poter considerare disaccoppiati i due comportamenti, scopriamo che la matrice **d** è in realtà composta da due sottomatrici una delle quali può ricondursi alla matrice costitutiva di uno stato piano.

$$\vec{\sigma}_p = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} \qquad \vec{\varepsilon}_p = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} \qquad nel\ piano$$

$$\vec{\tau}_f = \begin{bmatrix} \tau_{zx} \\ \tau_{zy} \end{bmatrix} \qquad \vec{\gamma}_f = \begin{bmatrix} \gamma_{zx} \\ \gamma_{zy} \end{bmatrix} \qquad fuori\ piano$$

$$\begin{bmatrix} \vec{\sigma}_p \\ \vec{\tau}_f \end{bmatrix} = \begin{bmatrix} & \boldsymbol{d_P} & & 0 & 0 \\ & & & 0 & 0 \\ 0 & 0 & 0 & & \\ 0 & 0 & 0 & \boldsymbol{d_G} & \end{bmatrix} \cdot \begin{bmatrix} \vec{\varepsilon}_p \\ \vec{\gamma}_f \end{bmatrix}$$

La sottomatrice $\boldsymbol{d_G}$ governa il legame costitutivo per il taglio e si può dimostrare essere pari a:

$$\boldsymbol{d_G} = \frac{5}{6}\begin{bmatrix} G & 0 \\ 0 & G \end{bmatrix}$$

Per quanto riguarda la sottomatrice $\boldsymbol{d_P}$ è possibile scegliere tra il legame costitutivo per uno stato piano di deformazione o di uno di tensione. Da confronti sperimentali si è però concluso che l'assegnazione di uno stato piano di deformazione, per il comportamento membranale, fornisce una irrealistica rigidezza agli elementi finiti e di conseguenza si predilige l'utilizzo di uno stato piano di tensione.

$$\boldsymbol{d_P} = \frac{E}{1-\upsilon^2}\begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \dfrac{1-\upsilon}{2} \end{bmatrix}$$

$matrice\ costitutiva\ per\ uno\ SPT$

Non dimenticando l'obiettivo di trovare un modo per definire la matrice costitutiva **D**, cominciamo ora con il definire l'energia potenziale di deformazione riferita al piano medio mediante grandezze puntuali e generalizzate:

$$\Omega_{(x,y)} = \frac{1}{2}\,\vec{q}^{\,T}\boldsymbol{D}\,\vec{q}$$

$$\Omega_{(x,y)} = \int_{-h/2}^{h/2} \omega\; dz \;=\; \frac{1}{2}\int_{-h/2}^{h/2} \vec{\varepsilon}^{\,T}\boldsymbol{d}\,\vec{\varepsilon}\; dz$$

Inseriamo ora il legame tra deformazioni puntuale e generalizzate $\vec{\varepsilon} = \boldsymbol{b}\,\vec{q}$ definito al paragrafo 2.1.2 e si ottiene:

$$\Omega_{(x,y)} = \frac{1}{2}\,\vec{q}^{\,T}\left[\int_{-h/2}^{h/2} \boldsymbol{b^T d\, b}\; dz\right]\vec{q}$$

Si conclude dunque che la matrice costitutiva **D** è definita secondo la seguente relazione:

$$\boldsymbol{D} = \int_{-h/2}^{h/2} \boldsymbol{b^T d\, b}\; dz$$

Si ipotizza ora che la lastra sia costituita da un materiale omogeneo e che dunque le caratteristiche costitutive del materiale siano uniformi lungo una generica generatrice.

Sulla base di ciò si può affermare che:

$$\boldsymbol{D} = \begin{bmatrix} & & & 0 & 0 & 0 & 0 & 0 \\ & h\,\boldsymbol{d_p} & & 0 & 0 & 0 & 0 & 0 \\ & & & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & & \frac{h^3}{12}\,\boldsymbol{d_p} & & 0 & 0 \\ 0 & 0 & 0 & & & & 0 & 0 \\ 0 & 0 & 0 & & & & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h\,\boldsymbol{d_G} & \\ 0 & 0 & 0 & 0 & 0 & 0 & & \end{bmatrix}$$

$$\vec{N} = h\,\boldsymbol{d_p}\,\vec{\eta}$$

$$\vec{M} = \frac{h^3}{12}\,\boldsymbol{d_p}\,\vec{\chi}$$

$$\vec{T} = h\,\boldsymbol{d_G}\,\vec{\xi}$$

2.2) IMPLEMENTAZIONE DELLA TEORIA DI MINDLIN AD UN ELEMENTO PLATE AD 8 NODI

L'elemento finito che si vuole andare ad introdurre in questo paragrafo è un elemento di tipo plate (tutti i suoi nodi devono appartenere ad un unico piano) che implementa la teoria di Mindlin. Come già detto, la teoria tiene conto della deformazione a taglio particolarmente rappresentativa di lastre dallo spessore rilevante. L'elemento si compone di 8 nodi e le sue funzioni di forma coincidono con le funzioni interpolanti quadratiche, rendendolo a tutti gli effetti un elemento isoparametrico e parabolico. I gradi di libertà cinematici definiti per ogni nodo riprendono i 5 previsti dalla teoria di Mindlin e sono dunque 3 spostamenti nello spazio (u, v e w) più le 2 rotazioni complanari all'elemento (φ_x e φ_y).

2.2.1) Sistemi di riferimento adottati

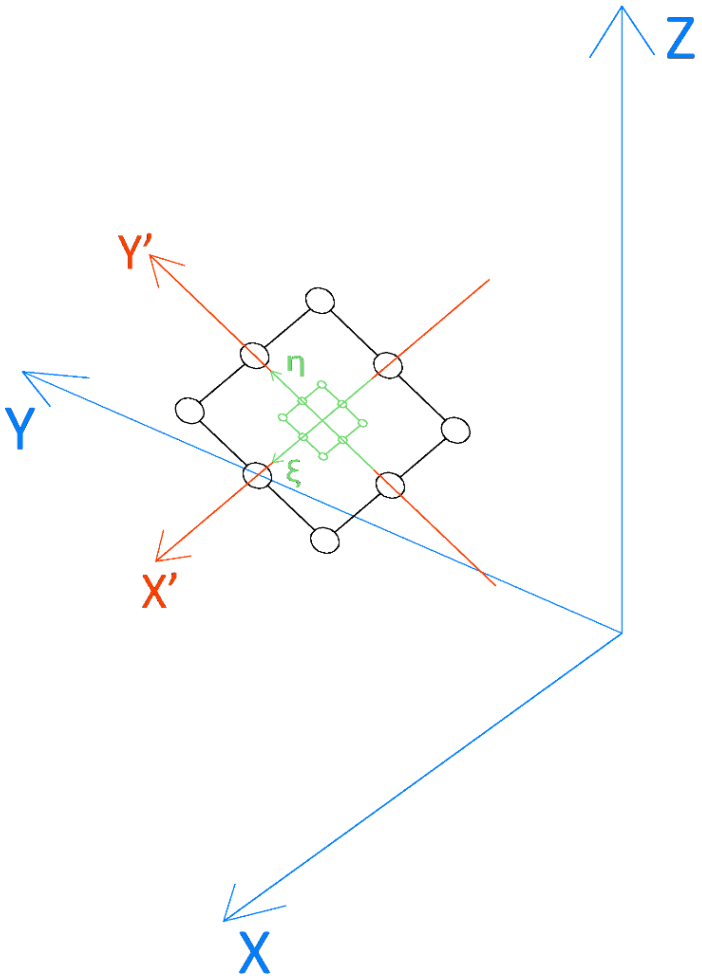


FIGURA 5 SISTEMI DI RIFERIMENTO

Al fine di poter gestire l'elemento nello spazio tridimensionale è necessario anzitutto definire 3 diversi livelli di sistemi di riferimento che sono:

- **Livello globale XYZ**
  Il sistema di riferimento globale è il livello nel quale vengono definite le coordinate (in termini delle variabili *x, y* e *z*) di tutti i nodi in fase di meshing.

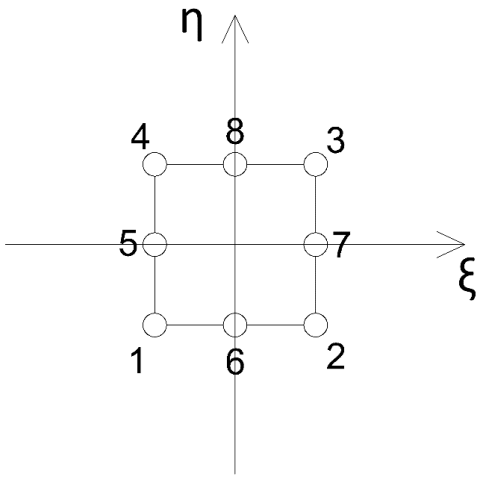
- **Livello locale X' Y'**
  Il sistema di riferimento locale, a differenza del globale, è complanare all'elemento plate ed è costituto da due solo coordinate *x'* e *y'* ottenibili mediante l'applicazione di una matrice di rotazione.

$$\begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = [\vec{n}_x \quad \vec{n}_y \quad \vec{n}_z]^T \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

  $\vec{n}_x$ e $\vec{n}_y$ rapprentano due versori ortogonali e complanari al plate mentre $\vec{n}_z$ rappresenta il versore normale.

- **Livello locale e normalizzato ξ η**
  Il sistema di riferimento locale e normalizzato, come il precedente, è sempre complanare al plate ma con la differenza che le coordinate dei nodi sono già pre-definite. Il cambio di variabile da x' e y ' a ξ e η avviene mediante le funzioni di mapping.

$$\begin{cases} \xi = \xi_{(x',y')} \\ \eta = \eta_{(x',y')} \end{cases} \quad funzioni\ di\ mapping$$



Indice locale del nodo	**ξ**	**η**
1	-1	-1
2	1	-1
3	1	1
4	-1	1
5	-1	0
6	0	-1
7	1	0
8	0	1

2.2.2) Funzioni di forma

Come già introdotto precedentemente, l'elemento plate che si vuole realizzare è di tipo parabolico e isoparametrico. Di conseguenza, si ha che le funzioni di forma $N_{i(\xi,\eta)}$ coincidono con le funzioni interpolanti $h_{i(\xi,\eta)}$ le quali assumono valore unitario nel nodo i, su cui sono definite, e valore nullo in tutti gli altri nodi diversi da i. Essendo in totale 8 i nodi, sono disponibili 8 equazioni per determinare 8 incognite rappresentate dai coefficienti polinomiali $c_{i0}, c_{i1}, \dots, c_{i8}$ della funzione.

$$h_{i(\xi,\eta)} = c_{i0} + c_{i1}\xi + c_{i2}\eta + c_{i3}\xi^2 + c_{i4}\,\xi\eta + c_{i5}\,\eta^2 + c_{i6}\,\xi^2\eta + c_{i7}\,\xi\eta^2$$

$$h_{i(\xi_j,\eta_j)} = 1 \quad se\ (j = i) \qquad e \qquad h_{i(\xi_j,\eta_j)} = 0 \quad \forall\ (j \neq i)$$

In base a questa definizione e alle coordinate assegnate ai nodi nel sistema di riferimento locale e normalizzato, con poche righe di codice è possibile scrivere uno script in grado di calcolare tutti i coefficienti di tutte le funzioni interpolanti.

```
%coordinate locali
local_coord= [-1 -1
               1 -1
               1  1
              -1  1
              -1  0
               0 -1
               1  0
               0  1];


%calcolo dei coefficienti per le funzioni di forma locali
local_coeff_N=zeros(8,8);
BC_elem=zeros(8,8);

for i=1:8
    BC_elem(i,1)=1;
    BC_elem(i,2)=local_coord(i,1);
    BC_elem(i,3)=local_coord(i,2);
    BC_elem(i,4)=local_coord(i,1)^2;
    BC_elem(i,5)=local_coord(i,1)*local_coord(i,2);
    BC_elem(i,6)=local_coord(i,2)^2;
    BC_elem(i,7)=local_coord(i,1)^2*local_coord(i,2);
    BC_elem(i,8)=local_coord(i,1)*local_coord(i,2)^2;
end

for i=1:8
    e=zeros(8,1);
    e(i)=1;
    local_coeff_N(i,:)=(BC_elem\e)';
end
```

I risultati ottenuti sono dunque i seguenti:

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{bmatrix}_{(\xi,\eta)} = \begin{bmatrix} -0.25 & 0 & 0 & 0.25 & 0.25 & 0.25 & -0.25 & -0.25 \\ -0.25 & 0 & 0 & 0.25 & -0.25 & 0.25 & -0.25 & 0.25 \\ -0.25 & 0 & 0 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 \\ -0.25 & 0 & 0 & 0.25 & -0.25 & 0.25 & 0.25 & -0.25 \\ 0.5 & -0.5 & 0 & 0 & 0 & -0.5 & 0 & 0.5 \\ 0.5 & 0 & -0.5 & -0.5 & 0 & 0 & 0.5 & 0 \\ 0.5 & 0.5 & 0 & 0 & 0 & -0.5 & 0 & -0.5 \\ 0.5 & 0 & 0.5 & -0.5 & 0 & 0 & -0.5 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \xi \\ \eta \\ \xi^2 \\ \xi\eta \\ \eta^2 \\ \xi^2\eta \\ \xi\eta^2 \end{bmatrix}$$

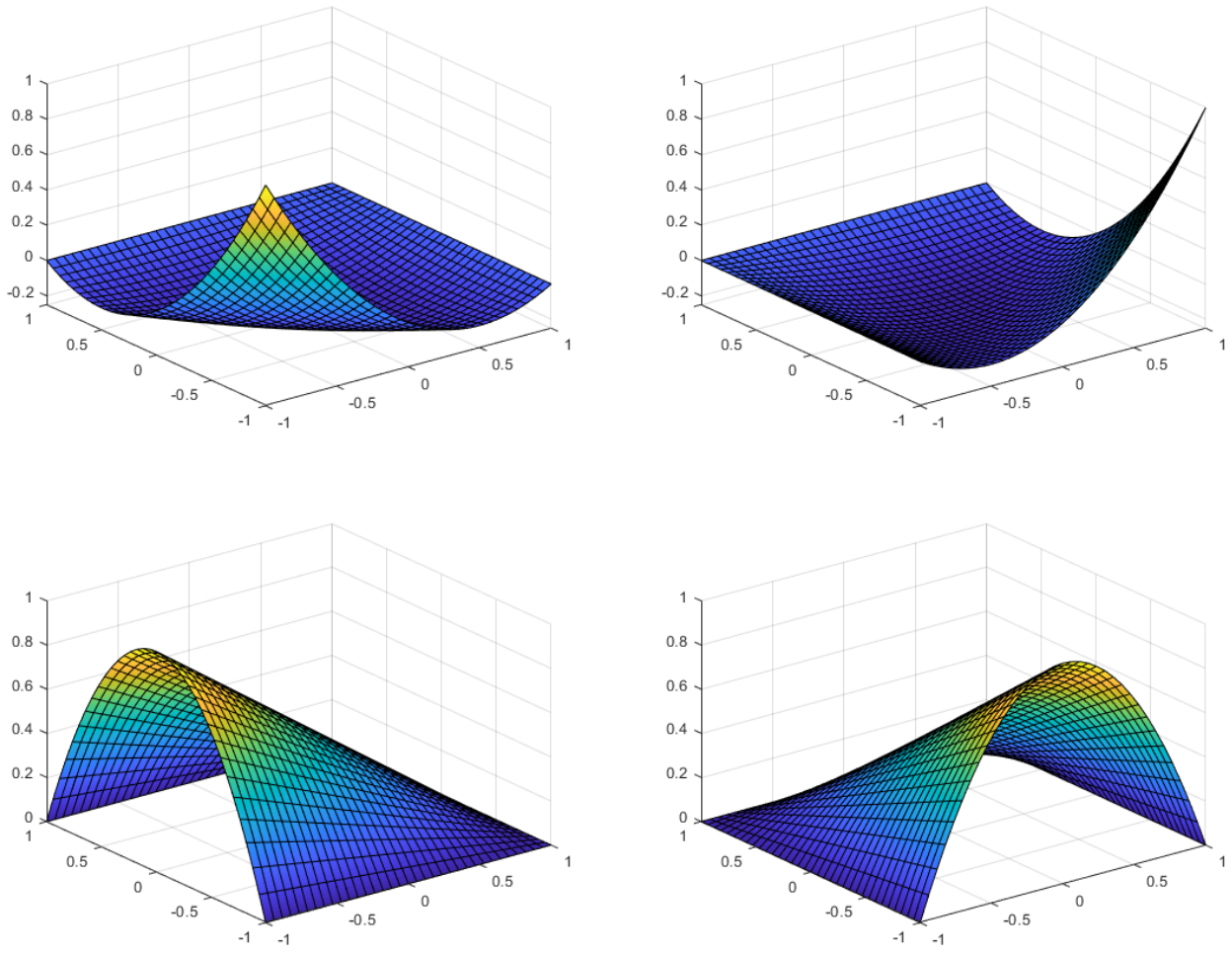


FIGURA 6 FUNZIONI DI FORMA

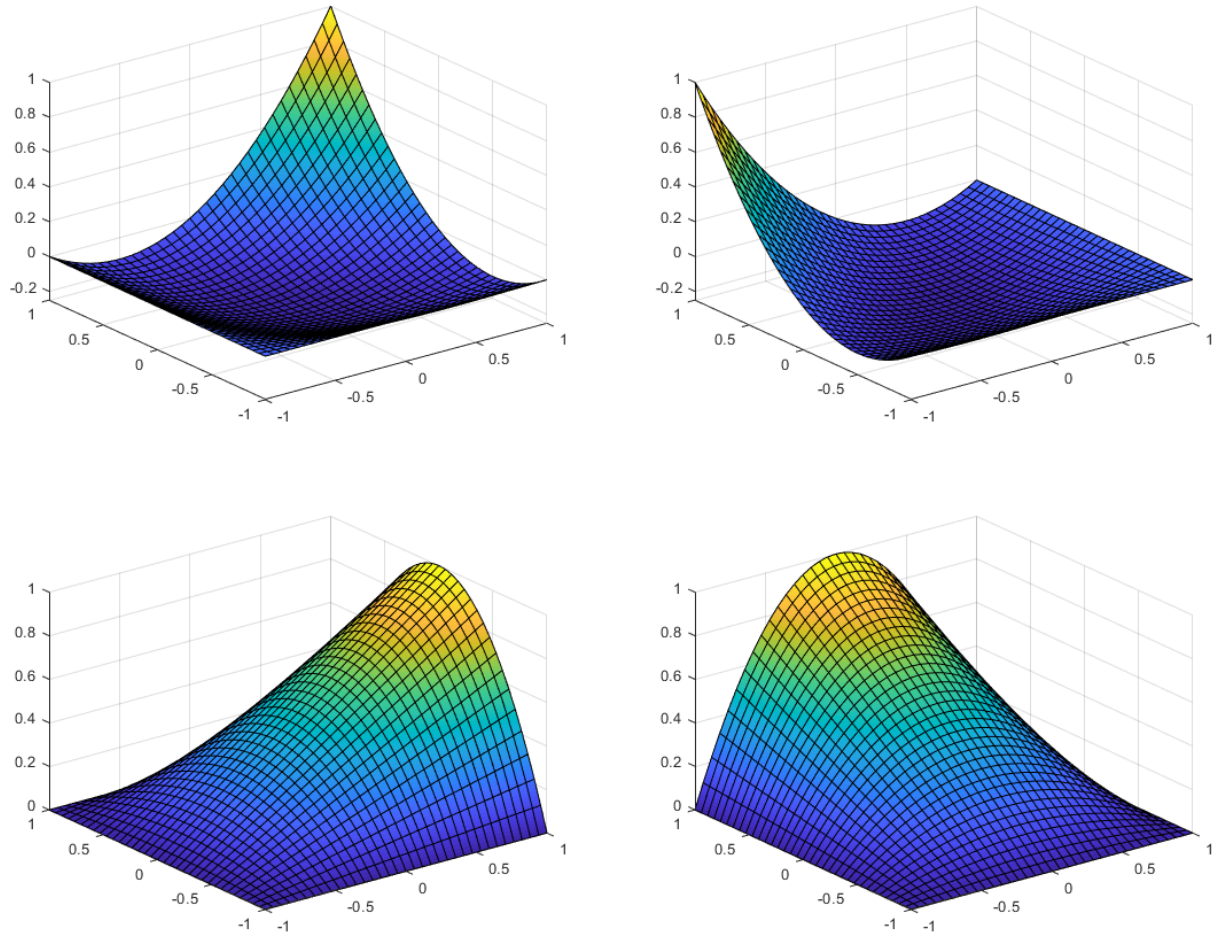


FIGURA 7 FUNZIONI DI FORMA

Note le funzioni interpolanti e le coordinate, è possibile conoscere anche le funzioni inverse di mapping per passare del sistema di riferimento locale normalizzato al sistema di riferimento locale:

$$\begin{cases} x'_{(\xi,\eta)} = h_{1_{(\xi,\eta)}}\, x'_1 + h_{2_{(\xi,\eta)}} x'_2 + \cdots + h_{8_{(\xi,\eta)}} x'_8 \\ y'_{(\xi,\eta)} = h_{1_{(\xi,\eta)}}\, y'_1 + h_{2_{(\xi,\eta)}} y'_2 + \cdots + h_{8_{(\xi,\eta)}} y'_8 \end{cases}$$

2.2.3) Matrice di rigidezza

Sia $\vec{a}$ il vettore contenete tutti e 40 (5 per ognuno degli 8 nodi) gli spostamenti nodali ammissibili:

$$\vec{a} = \begin{bmatrix} \vec{a}_1 \\ \vec{a}_2 \\ \vdots \\ \vec{a}_8 \end{bmatrix} \qquad con \qquad \vec{a}_i = \begin{bmatrix} u_i \\ v_i \\ w_i \\ \varphi_{xi} \\ \varphi_{yi} \end{bmatrix}$$

Sia $\boldsymbol{N}_{(x',y')}$ la matrice delle funzioni di forma:

$$\boldsymbol{N}_{(x',y')} = [\boldsymbol{N_1} \quad \boldsymbol{N_2} \quad \cdots \quad \boldsymbol{N_8}] \quad con \quad \boldsymbol{N}_i = \begin{bmatrix} N_i & 0 & 0 & 0 & 0 \\ 0 & N_i & 0 & 0 & 0 \\ 0 & 0 & N_i & 0 & 0 \\ 0 & 0 & 0 & N_i & 0 \\ 0 & 0 & 0 & 0 & N_i \end{bmatrix}$$

Dagli spostamenti nodali $\vec{a}$, mediante la matrice delle funzioni di forma, possiamo ricondurci ad un campo di spostamento $\vec{u}_{(x',y')}$ continuo in tutto l'elemento finito:

$$\vec{u}_{(x',y')} = \boldsymbol{N}_{(x',y')}\vec{a}$$

Sia ora $\boldsymbol{L}$ la matrice che contiene gli operatori di derivata tali per cui:

$$\vec{q}_{(x',y')} = \boldsymbol{L}\,\vec{u}_{(x',y')}$$

In base alla teoria di Mindlin, esposta al paragrafo 2.1, la matrice $\boldsymbol{L}$ risulta essere:

$$\boldsymbol{L} = [\boldsymbol{L_0} \quad \boldsymbol{L_0} \quad \cdots \quad \boldsymbol{L_0}] \qquad con \qquad \boldsymbol{L_0} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 & 0 & 0 \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial}{\partial x} \\ 0 & 0 & 0 & -\frac{\partial}{\partial y} & 0 \\ 0 & 0 & 0 & -\frac{\partial}{\partial x} & \frac{\partial}{\partial y} \\ 0 & 0 & \frac{\partial}{\partial x} & 0 & 1 \\ 0 & 0 & \frac{\partial}{\partial y} & -1 & 0 \end{bmatrix}$$

Si scopre dunque che:

$$\vec{q}_{(x',y')} = \boldsymbol{L} \cdot \boldsymbol{N}_{(x',y')} \vec{a}$$

Si definisce ora la matrice $\boldsymbol{B}_{(x',y')}$ che contiene le funzioni di forma derivate

$$\boldsymbol{B}_{(x',y')} = \boldsymbol{L} \cdot \boldsymbol{N}_{(x',y')}$$

$$\boldsymbol{B}_{(x',y')} = [\boldsymbol{B_1} \quad \boldsymbol{B_2} \quad \cdots \quad \boldsymbol{B_8}] \quad con \quad \boldsymbol{B_i}_{(x',y')} = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 & 0 & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 & 0 & 0 \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial N_i}{\partial x} \\ 0 & 0 & 0 & -\frac{\partial N_i}{\partial y} & 0 \\ 0 & 0 & 0 & -\frac{\partial N_i}{\partial x} & \frac{\partial N_i}{\partial y} \\ 0 & 0 & \frac{\partial N_i}{\partial x} & 0 & N_i \\ 0 & 0 & \frac{\partial N_i}{\partial y} & -N_i & 0 \end{bmatrix}$$

Applichiamo dunque il teorema dei lavori virtuali:

$$L_i = L_e$$

$$\iint_{A_e} \delta\vec{q}^T \vec{Q} \quad dx'dy' = \delta\vec{a}^T \vec{f}$$

Introduciamo il legame costitutivo di cui si è parlato al 2.1.4 per il quale $\vec{Q} = \boldsymbol{D}\ \vec{q}$

$$\iint_{A_e} \delta\vec{q}^T\ \boldsymbol{D}\ \vec{q}\ \ dx'dy' = \delta\vec{a}^T\ \vec{f}$$

Il vettore delle deformazioni generalizzate è stato definito come $\vec{q}_{(x',y')} = \boldsymbol{B}_{(x',y')}\vec{a}$ quindi:

$$\delta\vec{a}^T \left[\iint_{A_e} \boldsymbol{B}^{\boldsymbol{T}}_{(x',y')}\ \boldsymbol{D}\ \boldsymbol{B}_{(x',y')}\ dx'dy' \right] \vec{a} = \delta\vec{a}^T\ \vec{f}$$

Gli spostamenti nodali virtuali $\delta\vec{a}^T$ si possono semplificare da ambo i membri dell'equazione e, in questo modo, si ottiene il sistema lineare che risolve il problema mediante le incognite degli spostamenti $\vec{a}$.

$$\boldsymbol{K_e}\vec{a} = \vec{f} \quad con \quad \boldsymbol{K_e} = \iint_{A_e} \boldsymbol{B}^{\boldsymbol{T}}_{(x',y')}\ \boldsymbol{D}\ \boldsymbol{B}_{(x',y')}\ dx'dy'\ \ matrice\ di\ rigidezza$$

Una volta calcolato $\boldsymbol{K_e}$ a livello locale x' y', sarà poi necessario applicare una espansione della matrice da 40x40 a 48x48 elementi aggiungendo un grado di libertà φ_z fittizio per ogni nodo. Questo grado di libertà è necessario per rendere l'elemento compatibile con la possibilità di orientarlo nello spazio. A questo punto si può procedere con l'applicazione della rotazione alla matrice di rigidezza:

$$\boldsymbol{K_e^{global}} = \boldsymbol{R_k}\ \boldsymbol{K_e^{local}}\ \boldsymbol{R_k^T}$$

$$\boldsymbol{R_k} = \begin{bmatrix} \boldsymbol{R} & & \\ & \boldsymbol{R} & \\ & & \ddots \\ & & \boldsymbol{R} \end{bmatrix}_{48x48} \quad con \quad \boldsymbol{R} = [\vec{n}_x \quad \vec{n}_y \quad \vec{n}_z]^T$$

2.2.4) integrazione di Gauss

È stato visto, nel paragrafo precedente 2.2.3, che l'assemblaggio della matrice di rigidezza $\boldsymbol{K_e}$, relativa al generico elemento e, richiede una doppia integrazione sull'area dell'elemento. Questa integrazione può essere fatta su due livelli di sdr:

- Il primo è il sistema locale *x' y'*. A questo livello le dimensioni dell'elemento sono preservate in quanto il passaggio dal sistema globale a quello locale richiede solamente una rotazione ossia una trasformazione isometrica che rende inalterate le distanze. Sebbene questo possa essere visto come un vantaggio da un lato, dall'altro renderebbe oneroso il costo computazionale di calcolo delle funzioni di forma, delle loro derivate e della loro integrazione poiché queste funzioni cambiano elemento per elemento in base alla disposizione dei nodi.
- Il secondo, invece, è il sistema locale normalizzato $\xi\ \eta$ nel quale, a differenza del sdr precedente, sono note le coordinate di tutti nodi e, di conseguenza, sono note anche tutte le funzioni di forma. Se sono note le funzioni di forma, allora lo sono anche le loro derivate e il problema si riconduce ad essere unico per qualsiasi elemento si voglia definire. Sul dominio quadrato [-1,1]x[-1,1] è inoltre applicabile un metodo di quadratura numerica che è esatto per le funzioni polinomiali ed è anche molto efficienti. L'unico accorgimento che è necessario prendere è, naturalmente, il cambio di variabili per potersi ricondurre poi ad una matrice di rigidezza definita nel sistema globale.

Si può dunque dimostrare che:

$$\boldsymbol{K_e} = \iint_{A_e} \boldsymbol{B}^{\boldsymbol{T}}_{(x',y')}\, \boldsymbol{D}\, \boldsymbol{B}_{(x',y')}\ dx'dy' = \int_{-1}^{1}\int_{-1}^{1} \boldsymbol{B}^{\boldsymbol{T}}_{(\xi,\eta)}\, \boldsymbol{D}\, \boldsymbol{B}_{(\xi,\eta)} \det\left(\boldsymbol{J}_{(\xi,\eta)}\right) d\xi d\eta$$

$\boldsymbol{J}$ è la matrice Jacobiana 2x2 che permette il passaggio da derivate scritte in termini di x', y' a derivate scritte in termini invece di $\xi\ e\ \eta$.

$$\begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix} = \boldsymbol{J} \cdot \begin{bmatrix} \frac{\partial N_i}{\partial x'} \\ \frac{\partial N_i}{\partial y'} \end{bmatrix} \qquad con\ \ \boldsymbol{J} = \begin{bmatrix} \frac{\partial x'}{\partial \xi} & \frac{\partial y'}{\partial \xi} \\ \frac{\partial x'}{\partial \eta} & \frac{\partial y'}{\partial \eta} \end{bmatrix}$$

$x'_{(\xi,\eta)}$ e $y'_{(\xi,\eta)}$, come già visto nel paragrafo 2.2.2, possono però essere esplicitate mediante le funzioni interpolanti:

$$\begin{cases} x'_{(\xi,\eta)} = h_{1_{(\xi,\eta)}}\, x'_1 + h_{2_{(\xi,\eta)}} x'_2 + \cdots + h_{8_{(\xi,\eta)}} x'_8 \\ y'_{(\xi,\eta)} = h_{1_{(\xi,\eta)}}\, y'_1 + h_{2_{(\xi,\eta)}} y'_2 + \cdots + h_{8_{(\xi,\eta)}} y'_8 \end{cases}$$

Gli elementi della matrice jacobiana possono dunque essere ridefiniti mediante delle sommatorie:

$$\boldsymbol{J} = \begin{bmatrix} \sum_{i=1}^{10} \frac{\partial N_i}{\partial \xi} x'_i & \sum_{i=1}^{10} \frac{\partial N_i}{\partial \xi} y'_i \\ \sum_{i=1}^{10} \frac{\partial N_i}{\partial \eta} x'_i & \sum_{i=1}^{10} \frac{\partial N_i}{\partial \eta} y'_i \end{bmatrix}_{(\xi,\eta)}$$

Capito quindi come definire l'integrale nel sistema di riferimento normalizzato, rimane da vedere come applicare l'integrazione vera e propria. Il metodo di Gauss propone di attuare una sommatoria pesata sui valori che assume la funzione integranda in alcuni specifici punti del suo dominio normalizzato detti "punti Gauss".

$$\int_{-1}^{1}\int_{-1}^{1} g_{(\xi,\eta)}\, d\xi d\eta = \sum_{i=1}^{n}\sum_{j=1}^{n} g_{(\xi_i,\eta_j)}\, W_i\, W_j$$

Il numero di punti gauss dipende dal grado polinomiale della funzione che si intende integrare. In questo caso di elemento finito parabolico, per una integrazione esatta, è suggerito utilizzare una distribuzione di 2x2 punti.

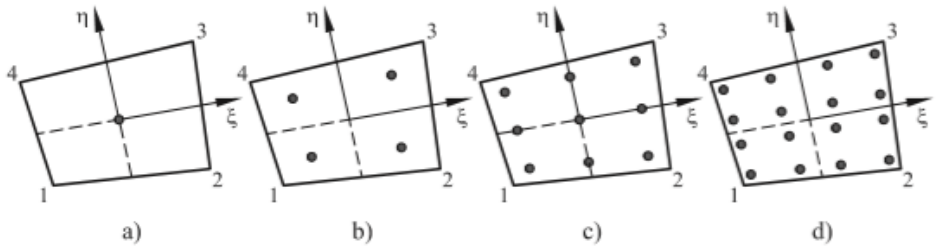


Fig. 6.4 Gauss quadratures over quadrilateral elements, a) 1×1, b) 2×2, c) 3×3, d) 4×4 integration points

FIGURA 8 SCHEMA PRESO DA STRUCTURAL ANALYSIS WITH THE FINITE ELEMENT METHOD, ONATE

Le coordinate e i pesi dei punti Gauss sono di seguito riportati:

	Coordinate punti Gauss
a	$\left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$
b	$\left(-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$
c	$\left(-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right)$
d	$\left(\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right)$

	Pesi
W_1	1.0
W_2	1.0

Considerando il fatto che i pesi sono unitari, il calcolo della matrice di rigidezza diventa:

$$\boldsymbol{K_e} = \int_{-1}^{1}\int_{-1}^{1} \boldsymbol{B}^{\boldsymbol{T}}_{(\xi,\eta)}\, \boldsymbol{D}\, \boldsymbol{B}_{(\xi,\eta)} \det\left(\boldsymbol{J}_{(\xi,\eta)}\right) d\xi d\eta$$

$$= \sum_{k=a,b,c,d} \boldsymbol{B}^{\boldsymbol{T}}_{(\xi_k,\eta_k)}\, \boldsymbol{D}\, \boldsymbol{B}_{(\xi_k,\eta_k)} \det\left(\boldsymbol{J}_{(\xi_k,\eta_k)}\right)$$

Da questa ultima forma per calcolare la matrice di rigidezza si può concludere che è possibile calcolare una volta per tutte due cose: le matrici $\boldsymbol{B}_{(\xi_k,\eta_k)}$ calcolate sui 4 punti gauss e le derivate delle funzioni di forma, necessarie al calcolo del determinante, calcolate sempre sui 4 punti gauss.

Con questo accorgimento, mirato a voler velocizzare l'assemblaggio, sono state calcolate e salvate in una cartella codice 2 matrici:

- B_Q8MINDLIN:
  è una matrice tridimensionale 8x40x4 che contiene le 4 matrici $\boldsymbol{B}$ calcolate sui 4 punti gauss
- dev_N_ptGauss_Q8MINDLIN
  è una matrice tridimensionale 2x8x4 che contiene le derivate delle 8 funzioni di forma secondo le due variabili ξ e η nei 4 punti Gauss.

Lo script con il quale sono state calcolate è il seguente:

```
%coordinate locali
local_coord= [-1 -1
               1 -1
               1  1
              -1  1
              -1  0
               0 -1
               1  0
               0  1];


%calcolo dei coefficienti per le funzioni di forma locali
local_coeff_N=zeros(8,8);
BC_elem=zeros(8,8);

for i=1:8
    BC_elem(i,1)=1;
    BC_elem(i,2)=local_coord(i,1);
    BC_elem(i,3)=local_coord(i,2);
    BC_elem(i,4)=local_coord(i,1)^2;
    BC_elem(i,5)=local_coord(i,1)*local_coord(i,2);
    BC_elem(i,6)=local_coord(i,2)^2;
    BC_elem(i,7)=local_coord(i,1)^2*local_coord(i,2);
    BC_elem(i,8)=local_coord(i,1)*local_coord(i,2)^2;
end

for i=1:8
    e=zeros(8,1);
    e(i)=1;
    local_coeff_N(i,:)=(BC_elem\e)';
end

%calcolo dei coefficienti per le funzioni di forma derivate secondo csi e secondo eta
local_coeff_N_dev_csi=zeros(8,6);
local_coeff_N_dev_eta=zeros(8,6);

for i=1:8
    local_coeff_N_dev_csi(i,1)=local_coeff_N(i,2);
```

```
    local_coeff_N_dev_csi(i,2)=2*local_coeff_N(i,4);
    local_coeff_N_dev_csi(i,3)=local_coeff_N(i,5);

    local_coeff_N_dev_csi(i,5)=2*local_coeff_N(i,7);
    local_coeff_N_dev_csi(i,6)=local_coeff_N(i,8);
end

for i=1:8
    local_coeff_N_dev_eta(i,1)=local_coeff_N(i,3);
    local_coeff_N_dev_eta(i,2)=local_coeff_N(i,5);
    local_coeff_N_dev_eta(i,3)=2*local_coeff_N(i,6);
    local_coeff_N_dev_eta(i,4)=local_coeff_N(i,7);
    local_coeff_N_dev_eta(i,5)=2*local_coeff_N(i,8);

end

%-----------------------quadratura con 4 punti gauss----------------------
eps=0.5773502692;
%coordinate punti gauss
gauss_p=[eps eps
         -eps eps
         -eps -eps
         eps -eps];

B=zeros(8,40,4);
dev_N_ptGauss=zeros(2,8,4);
N_ptGauss=zeros(1,8,4);
for p=1:4
    %.......................termini di N.................................
    gauss_p_term=[1, gauss_p(p,1), gauss_p(p,2), gauss_p(p,1)^2,
gauss_p(p,1)*gauss_p(p,2), gauss_p(p,2)^2, (gauss_p(p,1)^2)*gauss_p(p,2),
gauss_p(p,1)*gauss_p(p,2)^2];
    for i=1:8
        N_ptGauss(1,i,p)=local_coeff_N(i,:)*gauss_p_term';
    end
    %.........................................................................
    %.......................termini di derivata.........................
    clear gauss_p_term
    gauss_p_term=[1, gauss_p(p,1), gauss_p(p,2), gauss_p(p,1)^2,
gauss_p(p,1)*gauss_p(p,2), gauss_p(p,2)^2];
    %derivate su csi
    for i=1:8
        dev_N_ptGauss(1,i,p)=local_coeff_N_dev_csi(i,:)*gauss_p_term';
    end
    %derivate su eta
    for i=1:8
         dev_N_ptGauss(2,i,p)=local_coeff_N_dev_eta(i,:)*gauss_p_term';
    end
    %.........................................................................

end
%assemblaggio delle matrici B
for p=1:4
    for i=1:8
        a=dev_N_ptGauss(1,i,p);
        b=dev_N_ptGauss(2,i,p);
```

```
        c=N_ptGauss(1,i,p);
        B(:,(5*(i-1)+1):5*i,p)=[a  0  0  0  0
                                0  b  0  0  0
                                b  a  0  0  0
                                0  0  0  0  a
                                0  0  0 -b  0
                                0  0  0 -a  b
                                0  0  a  0  c
                                0  0  b -c  0];
    end
end
%-------------------------------------------------------------------------

save('B_Q8MINDLIN.mat','B');
save('dev_N_ptGauss_Q8MINDLIN.mat', 'dev_N_ptGauss');
```

2.3) COMMENTO ALLA CLASSE Q8MINDLIN

Come è ben visibile dallo schema flow chart in figura 9, la classe dedicata all'elemento Q8MINDLIN è strutturata secondo 4 proprietà (di cui 2 costanti) e 2 metodi.

PROPRIETA':

- DIMDOF
  Contiene la dimensione dei gradi di libertà totali dell'elemento. In linea teoria DIMDOF dovrebbe essere pari a 40 in quanto la teoria di Mindlin prevede 5 gradi di libertà cinematici che, moltiplicati per gli 8 nodi, fa per l'appunto 40. Una osservazione del genere si potrebbe fare però se l'elemento fosse definito in uno spazio bidimensionale ma, essendo il codice creato per poter interfacciare diversi elementi nello spazio tridimensionale, è necessario aggiungere un grado di libertà fittizio anche se non esprime alcuna rigidezza per rotazioni normali al piano. Ecco dunque spiegato il motivo per cui DIMDOF=48.
- TOTNODES
  Contiene il numero di nodi che compongono l'elemento, in questo caso 8.
- nodes_id
  È un vettore che contiene gli indici dei nodi che compongono l'elemento.
- prop_id
  Contiene l'indice della proprietà che caratterizza l'elemento plate.

METODI:

- Costruttore
  Il costruttore è un metodo che è sempre necessario definire all'interno della classe poiché serve a inizializzare tutte le proprietà. In questo caso il costruttore assegna i valori al vettore nodes_id e alla variabile prop_id.
- localstifness
  Il metodo localstifness è il cuore della classe poiché svolge l'importante compito di assemblare la matrice di rigidezza locale dell'elemento. Essa prende in input la proprietà indicata da prop_id e le coordinate dei nodi indicate da nodes_id e, in base ai procedimenti spiegati al paragrafo 2.2, crea la matrice K_e che successivamente verrà smembrata e poi riassemblata all'interno della grande matrice globale K^{global}.

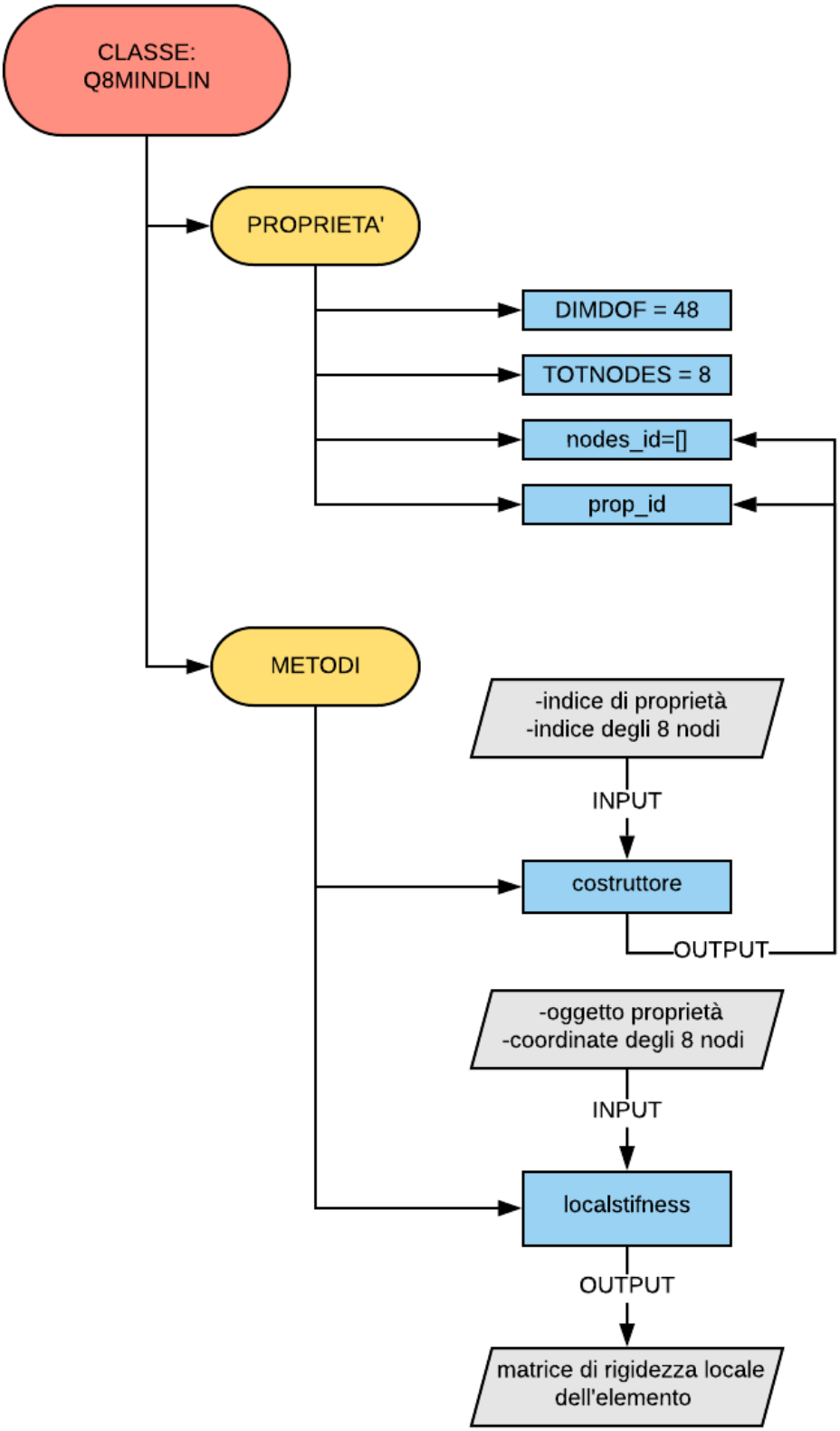


FIGURA 9 STRUTTURA DELLA CLASSE Q8MINDLIN

2.3.1) Listato del codice relativo alla classe Q8MINDLIN

```
classdef Q8MINDLIN

%.........................PROPRIETA'.....................................
    properties(Constant)
        DIMDOF = 48;    %dimensione della matrice di rigidezza
        TOTNODES = 8;   %totale di nodi che compongono l'elemento
    end

    properties
        nodes_id=[];    %id dei nodi che compongono l'elemento (NB la disposizione segue
la convenzione di )
        prop_id;
    end
%.........................................................................

%.............................METODI.....................................
    methods
        %-------------------------COSTRUTTORE-----------------------------
        function this=Q8MINDLIN(nodes_id,property_id)
            this.nodes_id=nodes_id;
            this.prop_id=property_id;
        end
        %-------------------------------------------------------------------

        %-------------ASSEMBLAGGIO MATRICE DI RIGIDEZZA--------------------
        function [kglobal]=localstiffnes(this, prop, X)

            %matrice di rotazione sdr globale->locale
                %versore di z' (normale al plate)
                nz=-cross(X(4,:)'-X(1,:)', X(2,:)'-X(1,:)');
                nz=nz*(1/sqrt(nz(1)^2+nz(2)^2+nz(3)^2));

                %cerco il baricentro dell'elemento per definire il sdr
                O=[0;0;0];
                for i=1:8
                    O(1)=O(1)+X(i,1);
                    O(2)=O(2)+X(i,2);
                    O(3)=O(3)+X(i,3);
                end
                O=O*(1/8);

                %versore di y' (complanare al plate)
                ny=X(8,:)'-O;
                ny=ny*(1/sqrt(ny(1)^2+ny(2)^2+ny(3)^2));

                %versore di x' (complanare al plate)
                nx=cross(ny,nz);

                %matrice di rotazione
                R=[nx, ny, nz];
                Rk=zeros(48,48);
                for i=1:16
                    t1=(i-1)*3+1;
                    t2=i*3;
```

```
                Rk(t1:t2,t1:t2)=R;
            end

        %cambio del sistema di riferimento
            R=R';
            for i=1:8
                x(i)=R(1,:)*X(i,:)';
                y(i)=R(2,:)*X(i,:)';
            end

        %assemblaggio matrice costitutiva
            E = prop(this.prop_id).mat.E;
            ni = prop(this.prop_id).mat.nu;
            G = E /(2 * (1-ni));
            h = prop(this.prop_id).thickness;
            dp=E/(1-ni^2)*[1 ni 0; ni 1 0; 0 0 (1-ni)/2];
            dG=(5/6)*G*[1 0; 0 1];
            D=zeros(8,8);
            D(1:3,1:3)=h*dp;
            D(4:6,4:6)=(h^3/12)*dp;
            D(7:8,7:8)=h*dG;

        %assemblaggio matrice di rigidezza
            %chiamata delle matrici B e dev_N_ptGauss
            cd 'Q8_mindlin'
            L = matfile('B_Q8MINDLIN.mat');
            B = L.B;
            L = matfile('dev_N_ptGauss_Q8MINDLIN.mat');
            dev_N_ptGauss = L.dev_N_ptGauss;
            cd ..\

            %calcolo dei jacobiani
            det_J=zeros(4,1);
            for p=1:4
                a=dev_N_ptGauss(1,:,p)*x';
                b=dev_N_ptGauss(1,:,p)*y';
                c=dev_N_ptGauss(2,:,p)*x';
                d=dev_N_ptGauss(2,:,p)*y';
                J=[a b; c d];
                det_J(p)=det(J);
            end

            %assemblaggio finale
            klocal=zeros(40,40);  %matrice di rigidezza nel sistema di riferimento
locale con 5gdl per nodo
            for p=1:4
                klocal=klocal+B(:,:,p)'*D*B(:,:,p)/det_J(p);
            end

            %ritorno al sdr globale
            kglobal=eye(48,48);    %matrice di rigidezza nel sistema di riferimento
globale con 6gdl per nodo
            for i=1:8
                for j=1:8
                    %intervalli della matrice di rigidezza nel sdr locale da copiare
                    i1=(i-1)*5+1;
```

```
                        i2=i*5;
                        j1=(j-1)*5+1;
                        j2=j*5;
                        %intervalli della matrice di rigidezza nel sdr globale su cui
incollare
                        I1=(i-1)*6+1;
                        I2=i*6-1;
                        J1=(j-1)*6+1;
                        J2=j*6-1;
                        %banalmente copia e incolla dei blocchetti di
                        %matrice
                        kglobal(I1:I2,J1:J2)=klocal(i1:i2,j1:j2);
                    end
                end
                %rotazione applicata alla matrice di rigidezza
                kglobal=Rk*kglobal*Rk';
        end
        %-----------------------------------------------------------------------

    end
%.....................................................................

end
```

3) CARICO SUPERFICIALE

La modalità di carico superficiale permette di studiare il comportamento di un elemento plate soggetto ad una pressione uniforme $\vec{q} = [q_x \quad q_y \quad q_z]$ applicata alla sua superficie.

3.1) APPLICAZIONE DI UN CARICO SUPERFICIALE AD UN ELEMENTO FINITO

Per poter assegnare una pressione $\vec{q} = [q_x \quad q_y \quad q_z]^T$ ad un elemento palte dobbiamo anzitutto riorientare il carico dal sistema di riferimento globale a quello locale applicando una rotazione

$$\begin{bmatrix} q_{x'} \\ q_{y'} \\ q_{z'} \end{bmatrix} = R^T \cdot \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \qquad con \quad R = [\vec{n}_x \quad \vec{n}_y \quad \vec{n}_z]$$

Una volta ottenuto il carico a livello locale, per ottenere il vettore dei carichi equivalenti, è necessario attuare una integrazione delle funzioni di forma sulla superfice dell'elemento.

$$\vec{F}_e^{local} = \iint_{A_e} \boldsymbol{N}^{\boldsymbol{T}}_{(x',y')}\, \vec{q}\; dx'dy' \qquad con \qquad \boldsymbol{N}_{(x',y')} = [\boldsymbol{N_1} \quad \boldsymbol{N_2} \quad \cdots \quad \boldsymbol{N_8}]$$

$$N_i = \begin{bmatrix} \boldsymbol{N_i} & 0 & 0 \\ 0 & \boldsymbol{N_i} & 0 \\ 0 & 0 & \boldsymbol{N_i} \end{bmatrix}$$

Come già visto nel paragrafo 2.2.4, integrare a livello non normalizzato non conviene mai e si predilige, piuttosto, applicare una quadratura di Gauss nel seguente modo:

$$\vec{F}_e^{local} = \iint_{A_e} \boldsymbol{N}^{\boldsymbol{T}}_{(x',y')}\, \vec{q}\; dx'dy' = \int_{-1}^{1}\int_{-1}^{1} \boldsymbol{N}^{\boldsymbol{T}}_{(\xi,\eta)}\, \vec{q}\; det\big(\boldsymbol{J}_{(\xi,\eta)}\big)\, d\xi d\eta$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{N}^{\boldsymbol{T}}_{(\xi_i,\eta_j)}\, \vec{q}\; det\left(\boldsymbol{J}_{(\xi_i,\eta_j)}\right) W_i W_j$$

Si ricorda che per una quadratura adatta a funzioni polinomiali paraboliche sono richiesti almeno 4 punti Gauss con pesi unitari:

	Coordinate punti Gauss
a	$\left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$
b	$\left(-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$
c	$\left(-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right)$
d	$\left(\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}\right)$

	Pesi
W_1	1.0
W_2	1.0

L'integrazione diventa quindi:

$$\vec{F}_e^{local} = \sum_{k=a,b,c,d} \boldsymbol{N}^{\boldsymbol{T}}_{(\xi_k,\eta_k)}\, \vec{q}\; det\left(\boldsymbol{J}_{(\xi_k,\eta_k)}\right)$$

A questo punto non rimane che riportare il vettore dei carichi equivalenti al sistema di riferimento globale tramite una rotazione:

$$\vec{F}_e^{global} = \boldsymbol{R} \cdot \vec{F}_e^{local}$$

3.2) COMMENTO ALLA CLASSE FACELOAD e ALL'ASSEMBLAGGIO

La classe FACELOAD, relativa alla modalità di carico discussa in questo capitolo, rientra tra le classi figlie della classe LOAD che raccoglie tutte le classi dei carichi. Di fatto FACELOAD svolge unicamente il compito di definire la modalità con cui il vettore dei carichi equivalenti verrà assemblato all'interno del metodo dedicato assemblyF() all'interno della classe SOLUTORE.

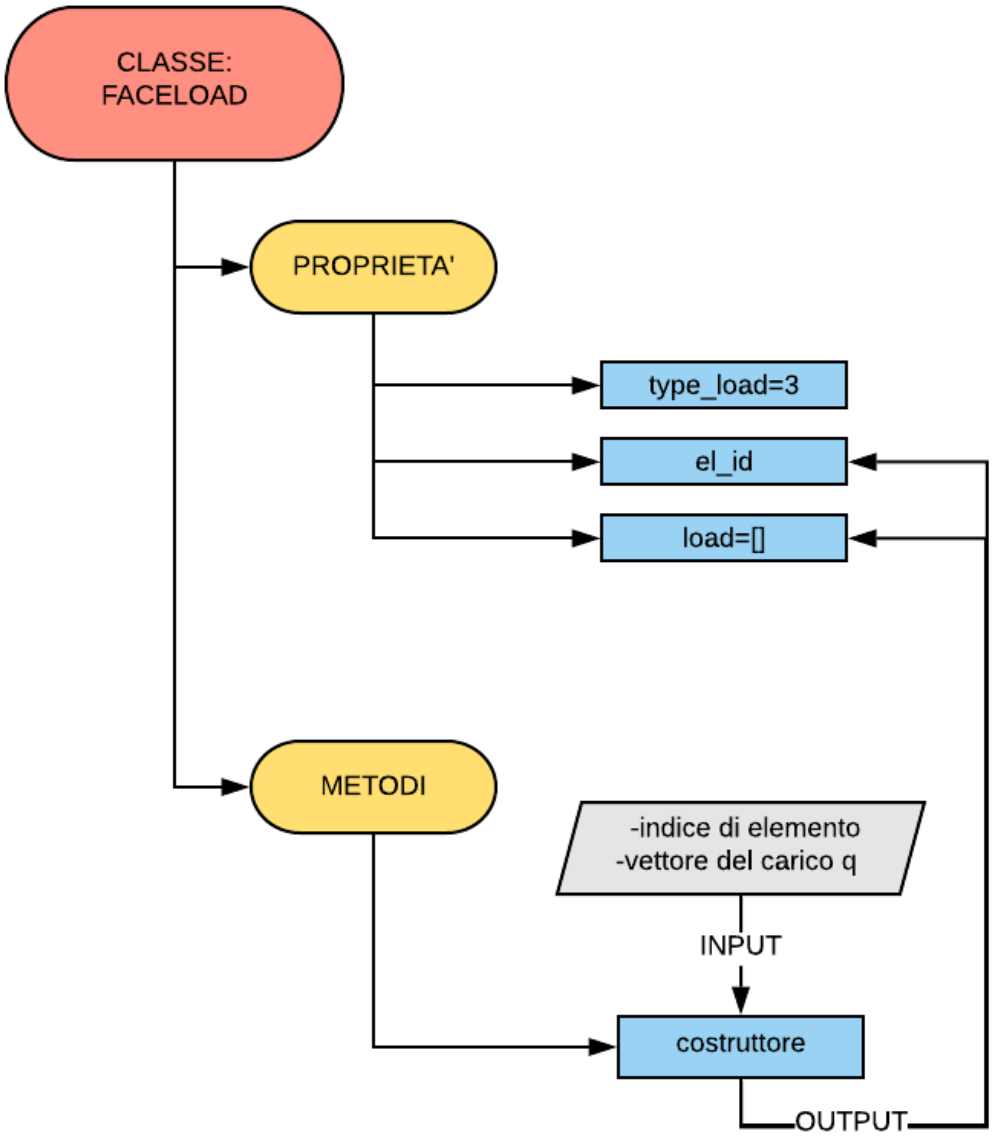


FIGURA 10 CLASSE FACELOAD

La struttura di FACELOAD, come tutte le classi, si compone di proprietà e metodi strutturati nel modo qui di seguito spiegato:

PROPRIETA':

- type_load
  Contiene l'indice della tipologia del carico superficiale che vale 3.
- el_id
  Indice dell'elemento plate a cui è stato assegnato il carico.
- load[]
  Vettore che contiene esprime la direzione e l'intensità della pressione applicata.

METODI:

- costruttore
  Il costruttore inizializza i valori di el_id e di load in base alle variabili ricevute alla creazione dell'oggetto del carico.

3.2.1) Listato del codice relativo alla classe FACELOAD

```
classdef FACELOAD
    % classe figlia per un carico superficiale da applicare ad un plate
    %-------------------------------------------------------------------------
    properties
      el_id;        %numero dell' elemento a cui è applicato il carico
      load = [];    %vettore contenete le componenti della pressione lungo x,y,z
      type_load=3;  %tipo di carico (in questo caso 3)
    end
    %-------------------------------------------------------------------------
    methods
    %-----------------------COSTRUTTORE-------------------------------------
        function this = FACELOAD(load,el_id)
            this.el_id = el_id;
            this.load = load;
        end
    %-------------------------------------------------------------------------

    end
end
```

3.2.2) Listato della porzione di codice relativa all'assemblaggio all'interno del metodo assemblyF() della classe SOLUTORE

```
%--------------------APPLICAZIONE CARICHI-------------------------
function [F_ext]=assemblyF(this)
    totloads=length(this.loads);
    F_ext = zeros(this.dim,1);
    for l=1:totloads
        f_ext = this.loads(l).getload();
        %.....................................................
        %carico nodale
        if this.loads(l).type_load == 1
        [...]
        %.....................................................
        %carico puntuale sull'elemento
        elseif this.loads(l).type_load == 2
        [...]
        %.....................................................
        %carico superficiale sull'elemento palte
        elseif this.loads(l).type_load == 3
            %ricavo le coordinate globali
            el_id = this.loads(l).getelement_id();
            nodes_id = this.elements(el_id).el.nodes_id;
            X=zeros(length(nodes_id),3);
            for i=1:length(nodes_id)
                X(i,:)=this.nodes(nodes_id(i)).x;
            end
            %ricavo le coordinati locali
            [x,y,R]=this.elements(el_id).el.rotation_plate(X);

            %calcolo dei jacobiani
            cd 'Q8_mindlin'
            M = matfile('N_ptGauss_Q8MINDLIN.mat');
            N_ptGauss = M.N_ptGauss;
            L = matfile('dev_N_ptGauss_Q8MINDLIN.mat');
            dev_N_ptGauss = L.dev_N_ptGauss;
            cd ..\
            det_J=zeros(4,1);
            for p=1:4
                a=dev_N_ptGauss(1,:,p)*x';
                b=dev_N_ptGauss(1,:,p)*y';
                c=dev_N_ptGauss(2,:,p)*x';
                d=dev_N_ptGauss(2,:,p)*y';
                J=[a b; c d];
                det_J(p)=det(J);
            end

            q=this.loads(l).load';
            %porto il carico dal sdr globale a quello locale
            q=R'*q;

            F_nodes=zeros(3,8);
            for j=1:8

                %integrazione di gauss
```

```
                    for p=1:4
                        for i=1:3
                           F_nodes(i,j)=F_nodes(i,j)+ N_ptGauss(1,j,p)*q(i)*det_J(p);
                        end
                    end

                    %riporto i carichi equivalente dal sdr locale a quello globale
                    F_nodes(:,j)=R*F_nodes(:,j);
                end

                %riporto i carichi nodali sul vettore complessivo dei carichi
                for j=1:length(nodes_id)
                    ind1=(nodes_id(j)-1)*this.gdlmax+1;
                    ind2=ind1+2;
                    F_ext(ind1:ind2)=F_ext(ind1:ind2)+F_nodes(:,j);
                end

            end
            %.....................................................
        end
    end
    %---------------------------------------------------------------------
```

4) VINCOLI

L'aspetto relativo ai vincoli, insieme ai carichi, costituisce una condizione al contorno al nostro problema meccanico e, di conseguenza, risulta essere di fondamentale importanza al fine di determinare la soluzione.

4.1) APPLICAZIONE DELLE CONDIZIONI AL CONTORNO CINEMATICHE IN UN PROBLEMA MECCANICO AGLI ELEMENTI FINITI

Supposto di conoscere la serie di nodi e i rispettivi gradi di libertà che si vogliono bloccare o fissare ad un determinato valore, esistono due metodi per condizionare il sistema lineare affinché tenga conto di questi vincoli: il metodo di eliminazione riga-colonna della matrice di rigidezza e il metodo penalty. Entrambi i metodi sono implementati nel solutore lineare.

Eliminazione riga-colonna:

Dei due è il metodo più esatto ma al tempo stesso il più dispendioso dal punto di vista computazionale. Infatti, l'eliminazione di righe e colonne all'interno di una matrice, che può raggiungere dimensioni anche dell'ordine di un milione per un milione di celle, richiede lo spostamento di una quantità di dati che può essere notevole. In ogni caso, per modelli relativamente piccoli, questo metodo offre una valida strategia.
A titolo di esempio si immagina di avere una matrice di rigidezza $\boldsymbol{K}$ 4x4 e un vettore dei carichi equivalenti $\boldsymbol{f}$**.** Si suppone di voler assegnare agli spostamenti $u_1 = \overline{u_1}$ e a $u_4 = \overline{u_4}$ lasciando incogniti solo gli spostamenti u_2 e u_3.

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

Essendo questo un sistema a 4 equazioni e 2 incognite vuol dire che è possibile ignorare due equazioni riducendo le reali dimensioni del problema. Si riportano dunque i termini noti ottenuti dagli spostamenti imposti $\overline{u_1}$ e $\overline{u_4}$ moltiplicati per la colonna 1 e la colonna 4 della matrice e si eliminano le righe e le colonne che incrociano gli elementi della matrice in posizione 11 e in posizione 44.

$$\begin{bmatrix} \cancel{k_{11}} & \cancel{k_{12}} & \cancel{k_{13}} & \cancel{k_{14}} \\ \cancel{k_{21}} & k_{22} & k_{23} & \cancel{k_{24}} \\ \cancel{k_{31}} & k_{32} & k_{33} & \cancel{k_{34}} \\ \cancel{k_{41}} & \cancel{k_{42}} & \cancel{k_{43}} & \cancel{k_{44}} \end{bmatrix} \cdot \begin{bmatrix} \cancel{u_1} \\ u_2 \\ u_3 \\ \cancel{u_4} \end{bmatrix} = \begin{bmatrix} \cancel{f_1 - \overline{u}_1 k_{11} - \overline{u}_4 k_{14}} \\ f_2 - \overline{u}_1 k_{21} - \overline{u}_4 k_{24} \\ f_3 - \overline{u}_1 k_{31} - \overline{u}_4 k_{34} \\ \cancel{f_4 - \overline{u}_1 k_{41} - \overline{u}_4 k_{44}} \end{bmatrix}$$

Il sistema da risolvere diventa dunque:

$$\begin{bmatrix} k_{22} & k_{23} \\ k_{32} & k_{33} \end{bmatrix} \cdot \begin{bmatrix} u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} f_2 - \overline{u}_1 k_{21} - \overline{u}_4 k_{24} \\ f_3 - \overline{u}_1 k_{31} - \overline{u}_4 k_{34} \end{bmatrix}$$

Metodo penalty:

Il metodo penalty, contrariamente al precedente, è molto efficiente anche su sistemi con molti gradi di libertà ma ha lo svantaggio di restituire una soluzione affetta da un certo grado di approssimazione. Facendo fede all'esempio precedentemente usato, la strategia consiste, da un lato, nel sostituire ai termini diagonali della matrice di rigidezza, in corrispondenza dei gradi di libertà vincolati, un numero molto grande (R) detto coefficiente penalty e dall'altro nel sostituire al carico equivalente corrispondente il coefficiente penalty moltiplicato per lo spostamento imposto. In questo modo si costringe il sistema lineare a pesare enormemente il grado di libertà vincolato facendolo convergere a dei valori molto prossimi a quelli imposti.

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \Rightarrow \begin{bmatrix} R & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & R \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} R\,\overline{u_1} \\ f_2 \\ f_3 \\ R\,\overline{u_4} \end{bmatrix}$$

4.2) COMMENTO ALLA CLASSE CONTRAINT e ALL'ASSEMBLAGGIO

A monte dei metodi per l'applicazione delle condizioni cinematiche, è necessario che il codice sia in grado di raccogliere le tutte le informazioni riguardanti i vincoli, definiti in fase di pre-processamento, e assemblare dei vettori che indichino quali spostamenti sono incogniti, quali no e che valore devono assumere questi ultimi. A questo scopo contribuiscono la classe dedicata ai vincoli CONTRAINT e il metodo assemblyU() nella classe del SOLUTORE.

Anche la classe CONTRAINT, come tutte le altre, si compone di proprietà e metodi:

PROPRIETA':

- gdlmax
  è una variabile che contiene il numero di gradi di libertà che possiede un nodo. Questa informazione viene ottenuta chiamando la classe GLOBAL
- node_id
  contiene l'indice di nodo su cui si vuole applicare il vincolo
- gdl_fixed [ ]*
  vettore lungo gdlmax contenente elementi non nulli (ad esempio 1) solo in corrispondenza del grado di libertà bloccato o imposto in quel nodo.
- u_fixed [ ]*
  vettore lungo gdlmax contenente gli spostamenti che si vogliono imporre (0 se si vuole bloccare lo spostamento). Si faccia attenzione al fatto che le componenti del vettore u_fixed che non hanno un corrispondente non nullo nel vettore gdl_fixed verranno ignorate.
- ind_u_vinc [ ]
  vettore contenente unicamente gli indice di riga, con riferimento al sistema lineare $\boldsymbol{K}\,\vec{u} = \vec{f}$, degli spostamenti vincolati.
- u_vinc [ ]
  vettore contenente i valori di spostamento bloccate o imposte indicati da ind_u_vinc

*i vettori gdl_fixed e u_fixed sono gli stessi vettori che vengono passati in input in fase di creazione dell'oggetto CONTRAINT. Essi infatti, insieme alla variabile node_id, contengono l'informazione di vincolo ancora in uno stato che si può dire grezzo e non interpretabile dal SOLUTORE. Il vettore ind_u_vinc, invece, reinterpreta l'informazione fornendo direttamente gli indici riga del sistema lineare da fixare.

METODI:

- costruttore
  In questa classe, a differenza delle altre, il costruttore non svolge solo il compito di passare le variabili di input alle proprietà ma si occupa anche di calcolare le proprietà ind_u_vinc[ ] e u_vinc[ ] che rappresentano un vero e proprio output necessario alle fasi successive che prenderanno luogo nel solutore.

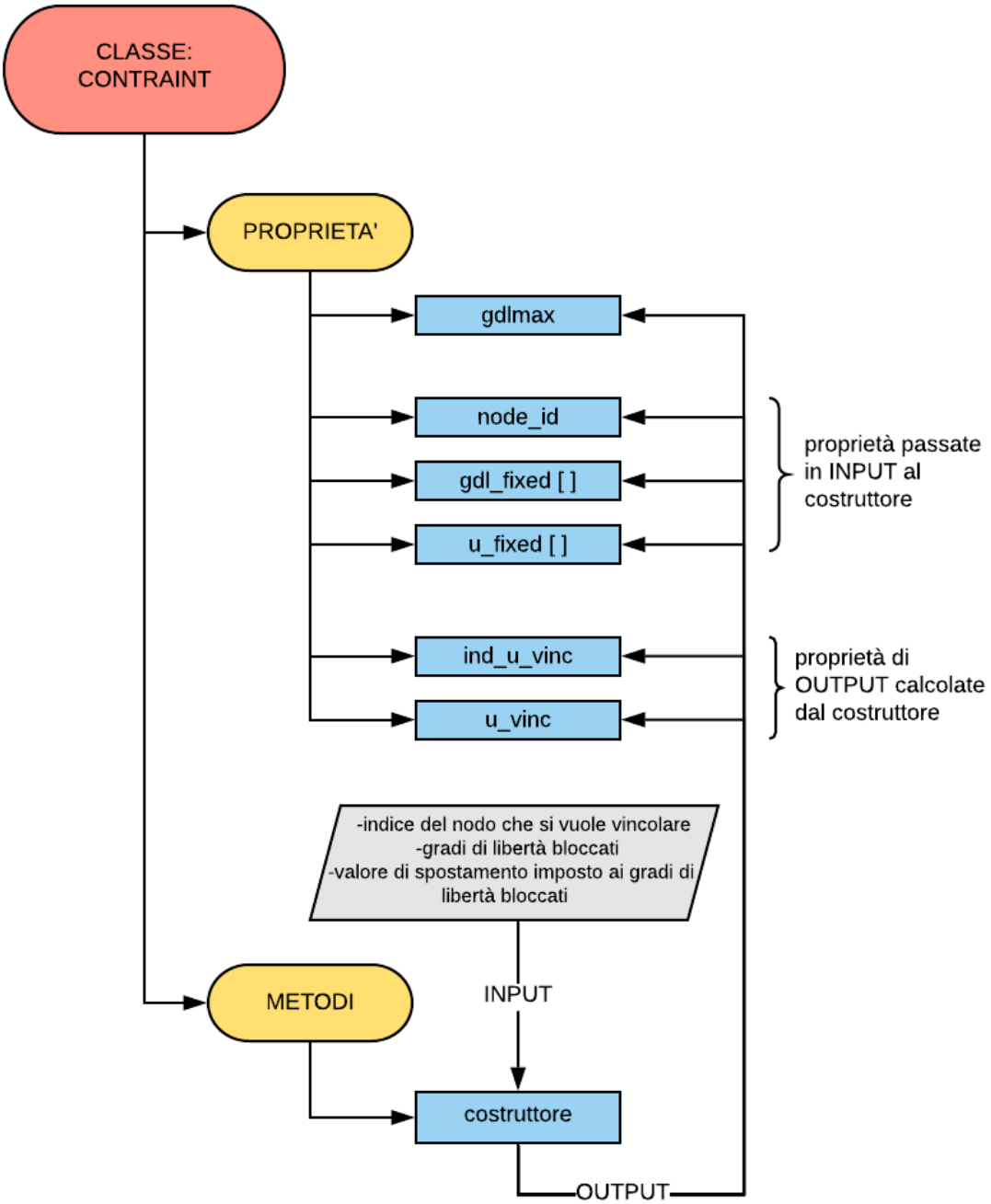


FIGURA 11 CLASSE DEI VINCOLI

4.2.1) Listato del codice relativo alla classe CONSTRAINT

```
classdef CONSTRAINT

    %.....................................................................
    properties
        %INPUT
        node_id;        %indice di nodo
```

```
        gdl_fixed=[];  %vettore lungo gdlmax contenete elementi non nulli
                       %    (ad esempio 1) solo in corrispondenza del grado
                       %     di libertà bloccato o imposto in quel nodo
        u_fixed=[];    %vettore lungo gdlmax contenente gli spostamenti
                       %    che si vogliono imporre (0 se si vuole bloccare
                       %    lo spostamento)

                       % ATTENZIONE: le componenti del vettore u_fixed che
                       % non hanno un corrispondente non nullo nel vettore
                       % gdl_fixed verranno ignorate!!

        gdlmax;        %variabile che contiene il numero di gradi di libertà che può
                       %    assumere un nodo

        %OUTPUT
        ind_u_vinc;  %vettore contenente gli indici di posizione, nel
                     %    sistema lineare Ku=f, degli spostamenti vincolati

        u_vinc;      %vettore contente le componenti di spostamento bloccate
                     %    o imposte
    end
    %.....................................................................

    %.....................................................................
    methods
        %--------------------COSTRUTTORE-----------------------------------
        function this = CONSTRAINT(node_id, gdl_fixed, u_fixed)
            this.node_id=node_id;
            this.gdl_fixed=gdl_fixed;
            this.u_fixed=u_fixed;
            gl=GLOBAL();
            this.gdlmax=gl.TOTDOF;

            %check su possibili errori di INPUT
            size_a=length(gdl_fixed);
            size_u=length(u_fixed);
            if (size_a~=this.gdlmax)||(size_u~=this.gdlmax)
                fprintf('ATTENZIONE!! errore sui gdlmax  \n')
            end

            %costruzione dei vettore ind_u_vinc e u_vinc
            cont=0; % variabile contatore per gli spostamenti vincolati
            for j=1:this.gdlmax
                if(this.gdl_fixed(j)~=0)
                    cont=cont+1;
                    this.ind_u_vinc(cont)=(this.node_id-1)*this.gdlmax+j;
                    this.u_vinc(cont)=this.u_fixed(j);
                end
            end
        end
        %------------------------------------------------------------------
    end
    %.....................................................................
end
```

4.2.2) Listato del codice relativa all'assemblaggio all'interno del metodo assemblyU() della classe SOLUTORE

```
%------------------------------VINCOLI SUGLI SPOSTAMENTI---------------------------------
        function [ind_u_inc, ind_u_vinc, u_vinc]=assemblyU(this)
           %OUTPUT
           % ind_u_inc:  vettore contenente gli indici di posizione, nel sistema
           %             lineare Ku=f, di tutti gli spostamenti incogniti
           % ind_u_vinc: vettore contenente gli indici di posizione, nel
           %             sistema lineare Ku=f, di tutti gli spostamenti
           %             vincolati
           % u_vinc:     vettore di dimensione dim contente le componenti di spostamento
           %             bloccate o imposte

           %pre allocazione di memoria per ind_u_vinc e u_vinc
           long=0;
           for i=1:length(this.constraint)
               long=long+length(this.constraint(i).u_vinc);
           end
           ind_u_vinc=zeros(long,1);
           u_vinc=zeros(long,1);

           %assemblaggio di ind_u_vinc e di u_vinc
           cont_start=0; %variabile temporanea per l'assemblaggio
           cont_end=0;   %variabile temporanea per l'assemblaggio
           for i=1:length(this.constraint)
               cont_end=cont_start+length(this.constraint(i).u_vinc);
               ind_u_vinc(cont_start+1:cont_end)=this.constraint(i).ind_u_vinc;
               u_vinc(cont_start+1:cont_end)=this.constraint(i).u_vinc;
               cont_start=cont_end;
           end

           ind_u_inc=zeros(this.dim-length(ind_u_vinc),1);   %inizializzazione ind_u_inc
           temp=zeros(this.dim,1);                           %vettore temporaneo per
                                                             %memorizzare le posizioni
                                                             %vincolate

           temp(ind_u_vinc)=1;                               %temp è uguale a 1 nelle
                                                             %posizioni di spostamento
                                                             %vincolato
           cont=0;
           for i=1:this.dim
               if temp(i)==0
                   cont=cont+1;
                   ind_u_inc(cont)=i;
               end
           end
        end
%-------------------------------------------------------------------------------------------
```

5) BENCHMARKS

5.1) PIASTRA IN SEMPLICE APPOGGIO SOGGETTA A CARICO UNIFORME

In questo test si vuole andare a testare due cose:

- l'effettiva capacità dell'elemento di adattarsi ad un generico sistema di riferimento globale che non coincide con quello locale
- dimostrare che l'elemento è in grado di rispondere correttamente ad un carico superficiale

Con l'obiettivo di mettere alla prova questi due aspetti si è quindi creata una mesh quadrata, rappresentativa di una piastra 2m x 2m, costituita da 6x6 elementi Q8MINDLIN soggetti ad un carico normale di 100 Mpa. Agli elementi sono stati sono stati assegnati 10 cm di spessore e delle caratteristiche di materiali pari a quelle di un acciaio da costruzione con quindi E=210000 Mpa e υ=0,3. Ai bordi della piastra quadrata sono stati messi dei vincoli sugli spostamenti traslazionali costituendo quindi l'effettiva situazione di una piastra in appoggio semplice. La piastra è stata inoltre orientata con un versore normale pari a:

$$n_z = [-0.1871 \quad 0.7485 \quad 0.6362]^T$$

Il risultato della deformata ottenuta (rete blu), confrontata con la mesh indeformata, (rete nera) è il seguente:

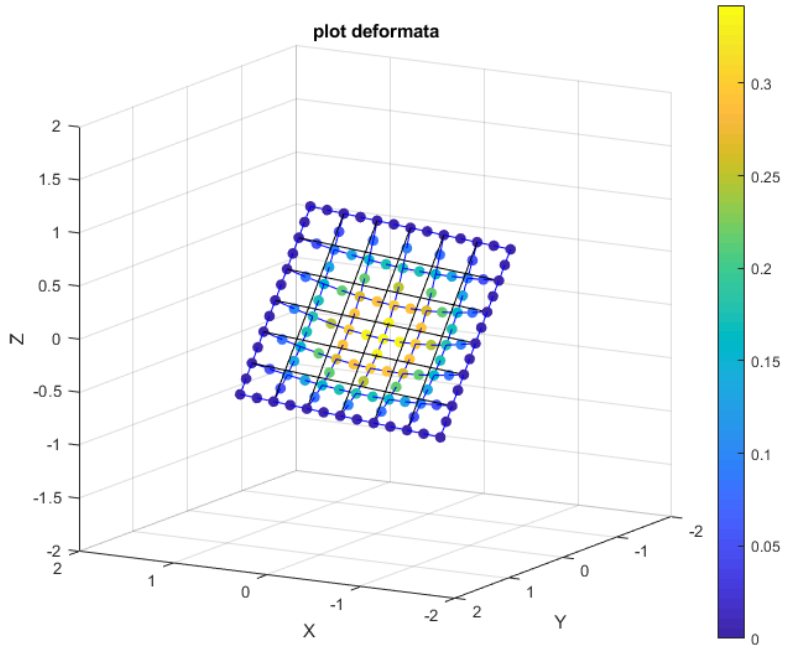


FIGURA 12 RISULTATI OTTENUTI DAL TEST "PIASTRA IN SEMPLICE APPOGGIO"

I nodi sono stati colorati secondo una scala che rappresenta gli spostamenti in modulo e, in base a ciò, si può subito vedere che la deformazione è compatibile con i vincoli. È evidente, infatti, che non ci sono spostamenti in corrispondenza del bordo vincolato e il massimo è situato al centro della piastra. Il nodo centrale riceve, secondo i risultati ottenuti, uno spostamento pari a 0,3422 m che è molto vicino a quello ottenuto da un modello costruito su Sraus7 con elementi quad8 in cui lo spostamento massimo trovato è pari a 0,3512 m.

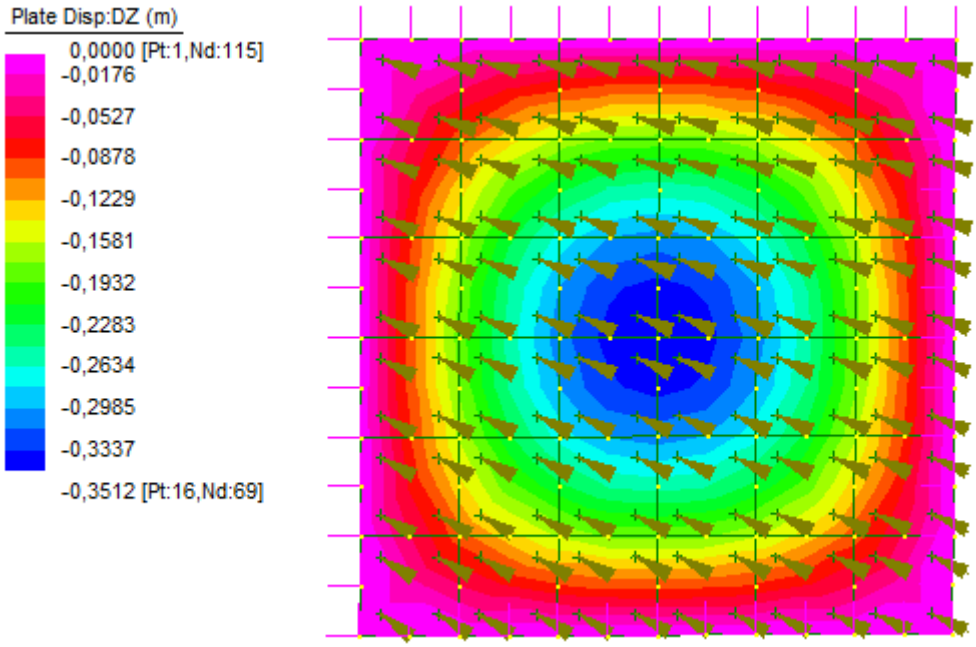


FIGURA 13 RISULTATI OTTENUTI CON STRAUS 7

5.2) STAFFA INCASTRATA SOGGETTA A VINCOLO CEDEVOLE

In questo test si vuole testare un modello che si sviluppa in 3 dimensioni e che contiene una connessione angolosa tra elementi plate. A questo scopo è stata modellata una geometria che ricorda la forma di una staffa a forma di L (2m x 2m x 2m) a cui sono state imposte due linee di vincoli: una alla base che blocca sia le rotazioni che le traslazioni (un incastro) e una con un vincolo cedevole che impone uno spostamento verticale u di 0,3 m.

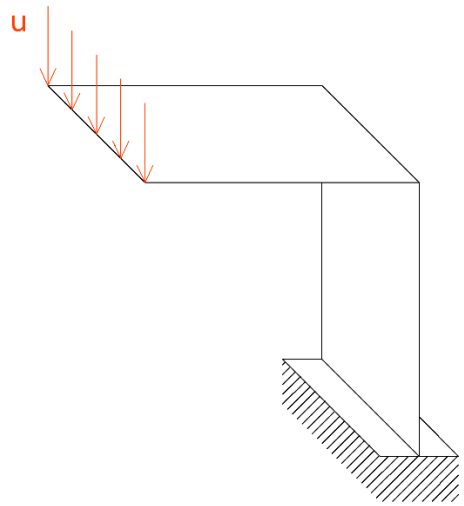
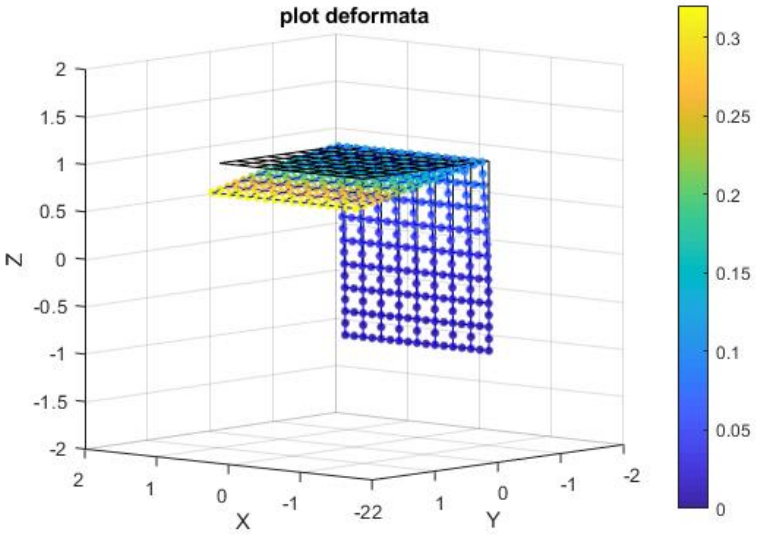


FIGURA 14 SCHEMA DELLA GEOMETRIA CHE SI INTENDE MODELLARE

Per quanto riguarda le proprietà degli elementi si fa riferimento alle stesse adottate nel test precedente (spessore h=10cm, E=210000 Mpa, υ=0,3). Il risultato ottenuto è riportato nelle seguenti figure:



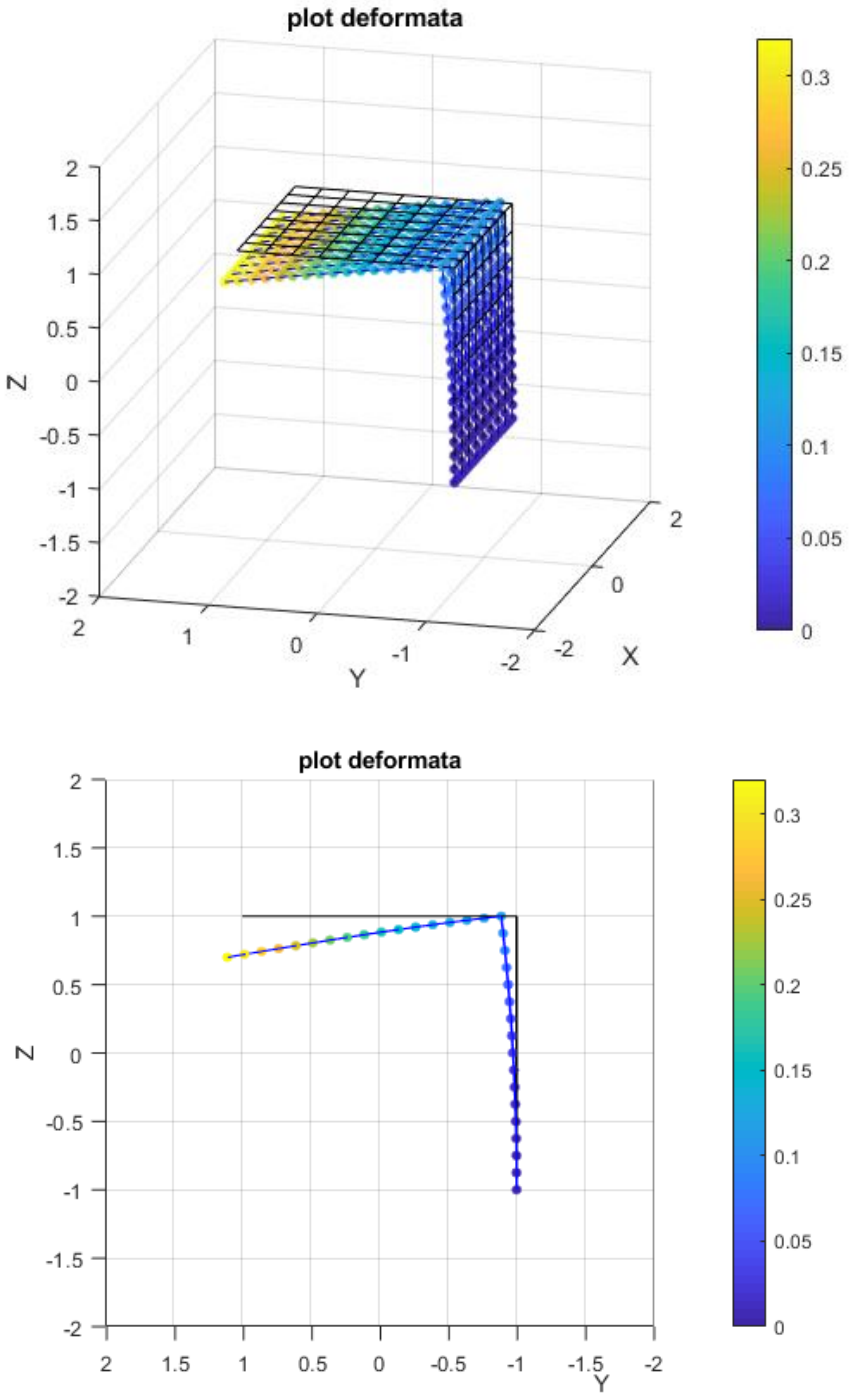


FIGURA 15 DEFORMATA DELLA STAFFA SOGGETTA A VINCOLO CEDEVOLE

Oltre al pieno rispetto delle condizioni di vincolo imposte sui due bordi, si nota come l'angolo di 90° tra i due piatti della staffa si preserva anche a deformazione avvenuta, indicando il fatto che i gradi di libertà sono stati definiti in modo corretto.

5.3) LASTRA SOGGETTA A SFORZO MEMBRANALE

Si vuole ora testare la capacità, dell'elemento plate, di rispondere correttamente ai carichi membranali simulando una lastra rettangolare (1m x 2m) vincolata sui due bordi corti. Su un bordo inferiore vengono bloccati tutti gli spostamenti mentre sull'altro vengono bloccati gli spostamenti orizzontali ma imposti a 0,3 metri quelli verticali.

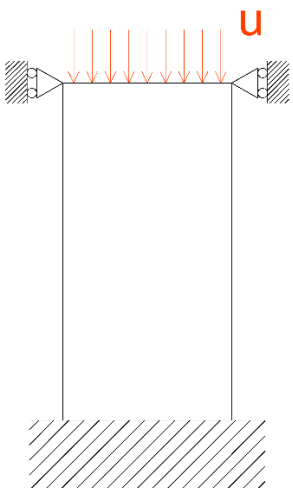
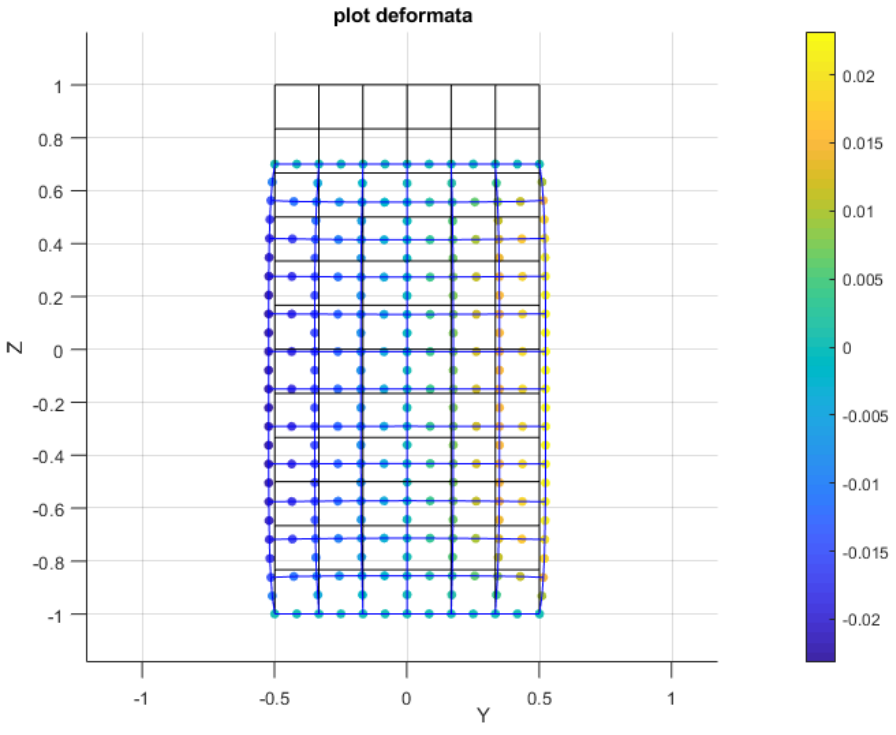


FIGURA 16 TEST DEFORMAZIONI MEMBRANALI

Ancora una volta le proprietà degli elementi si mantengono a spessore h=10cm, E=210000 Mpa, υ=0,3. Il risultato ottenuto da questo test è il seguente:



In questo caso la scala dei colori è tarata in modo tale da rappresentare gli spostamenti orizzontali causati dall'effetto Poisson. Lo spostamento massimo registrato è pari a 0,0232 m ed è perfettamente in linea con i risultati ottenuti dal modello costruito su straus7.

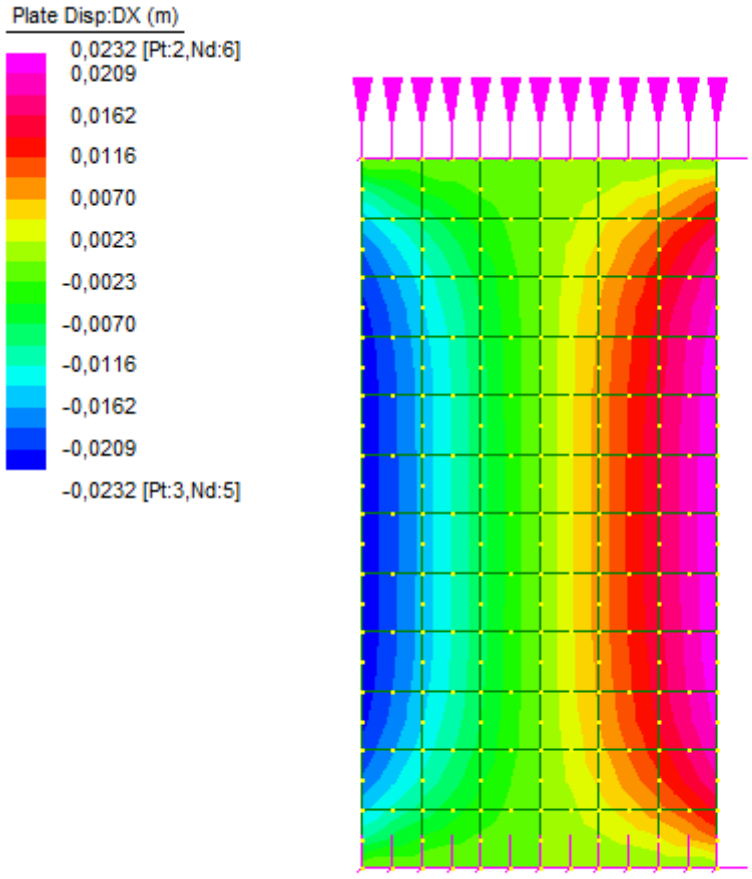


FIGURA 17 MODELLO TEST MEMBRANALE COSTRUITO SU STRAUS7

Ancora una volta l'elemento Q8MIDLIN dà prova di rispondere correttamente alle sollecitazioni imposte.