Parametric Design Optimization of Multiple-repair Composite Structures

Politecnico di Bari

Department of Mechanics, Mathematics and Management
MECHANICAL AND MANAGEMENT ENGINEERING
Ph.D. Program
SSD: ING-IND/14– PROGETTAZIONE MECCANICA E COSTRUZIONE DI MACCHINE

Final Dissertation

Parametric Design Optimization of Multiple-repair Composite Structures

---

by
Ficarella Elisa:

Supervisor:

Prof. Ing. Lamberti Luciano

Coordinator of Ph.D. Program:
Prof. Ing. Demelio Giuseppe Pompeo

Course n°31, 01/11/2015-31/10/2018

Politecnico di Bari

Department of Mechanics, Mathematics and Management
MECHANICAL AND MANAGEMENT ENGINEERING
Ph.D. Program
SSD: ING-IND/14– PROGETTAZIONE MECCANICA E COSTRUZIONE DI MACCHINE

Final Dissertation

Parametric Design Optimization of Multiple-repair Composite Structures

---

by
Ficarella Elisa:
Elisa Ficarella

Referees:
Ing. Gambino Benedetto
Ing. Barile Claudia

Supervisor:
Prof. Ing. Lamberti Luciano
Luciano Lamberti

Coordinator of Ph.D Program:
Prof. Ing. Demelio Giuseppe Pompeo

Course n°31, 01/11/2015-31/10/2018

Sommario

INTRODUZIONE 3

Capitolo 1 I MATERIALI COMPOSITI 5

1.1 Introduzione ai materiali compositi 5

1.2 Macro-Meccanica della Lamina 6

1.2.1 Ortotropia speciale e ortotropia generale 6

1.2.2 Cambio del sistema di riferimento 11

1.3 Macro-Meccanica del Laminato 16

Capitolo 2 IL SOFTWARE ABAQUS 24

2.1 Introduzione 24

2.2 Modulo Part 26

2.3 Modulo Property 28

2.4 Modulo Assembly 29

2.5 Modulo Step 30

2.6 Modulo Interaction 31

2.7 Modulo Load 32

2.8 Modulo Mesh 33

2.9 Modulo Job 35

2.10 Modulo Visualization 36

Capitolo 3 MODELLO NUMERICO DI UN PANNELLO CON DIFETTI MULTIPLI 38

3.1 Introduzione al modello 38

3.1.1 Materiali e lay-up 39

3.1.2 Parametrizzazione 40

3.2 Creazione delle parti 46

3.2.1 Skin 46

3.2.2 Stringer 48

3.2.3 Rib 49

3.2.4 Stringer criccato 52

3.2.5 Piastra di riparazione dello skin 53

3.2.6 Piastra di riparazione dello stringer 54

3.3 Definizione delle proprietà 55

3.3.1 Assegnazione dei materiali 55

3.3.2 Assegnazione delle Sezioni 57
3.4 Creazione della mesh 64
3.4.1 Skin 64
3.4.2 Stringer e Rib 70
3.4.3 Stringer criccati 71
3.4.4 Piastre di riparazione 74
3.5 Creazione dell'Assembly 76
3.6 Interazioni tra istanze 78
3.6.1 Vincolo Tie 78
3.6.2 Vincolo di contatto Surface-to-Surface 79
3.6.3 Fastener 81
3.7 Condizioni al contorno e di risoluzione 84
3.7.1 Creazione Step 84
3.7.2 Applicazione dei vincoli 84
3.7.3 Applicazione dei carichi 85
Capitolo 4 SIMULAZIONI E CASI DI STUDIO 87
4.1 Configurazioni 87
4.2 Configurazione A: due cricche sul medesimo stringer 87
4.3 Configurazione B: due cricche su due stringer 98
4.4 Confronto tra le configurazioni A e B 106
Capitolo 5 ANALISI DI UN PANNELLO IN PRESENZA DI UN DIFETTO 109
5.1 Ottimizzazione strutturale 109
5.1.1 Ottimizzazione Multi-objective 110
5.1.2 Algoritmo genetico 110
5.2 Modello numerico 112
5.2.1 Pannello danneggiato non riparato 112
5.2.2 Pannello danneggiato con riparazione 116
5.3 Interfaccia Abaqus – MATLAB 117
Capitolo 6 CONCLUSIONI E SVILUPPI FUTURI 120

BIBLIOGRAFIA ... 122

CODICI IN PYTHON ... 124

INTRODUZIONE

La riparazione di pannelli in materiale composito ha assunto sempre maggiore importanza a causa dell'aumento del loro utilizzo in ambito aeronautico; infatti, compagnie come Airbus e Boeing hanno realizzato aerei con fusoliera in materiale composito [1].

Esistono diverse tecniche di riparazione attualmente usate, a seconda di diversi parametri quali per esempio la dimensione e lo spessore del danneggiamento. La riparazione mediante patch in metallo bullonata viene utilizzata per strutture in materiale composito spesse e molto sollecitate, oppure qualora sia necessaria una riparazione in tempi brevi di un danneggiamento di modesta entità [2].

L'ottimizzazione strutturale della riparazione, in termini per esempio di dimensione della patch, numero di bulloni e così via, acquisisce importanza in quanto può permettere la riduzione del peso della riparazione a parità di carico sopportato dalla struttura riparata [3]. L'obiettivo principale di questo lavoro di tesi è stato realizzare un modello parametrico di un pannello in materiale composito che presenta stringer e rib; il danneggiamento interessa lo stringer e possono essere presenti più danneggiamenti contemporaneamente. Con questo modello parametrico è stato possibile realizzare una pseudo-ottimizzazione per studiare l'interazione di due riparazioni su due stringer adiacenti o su un unico stringer. Successivamente si è lavorato sull'ottimizzazione di un pannello simile, con due stringer e due rib, che presenti il difetto non riparato, in modo da analizzare l'influenza dei parametri di progettazione, quali layup, dimensioni degli stringer e così via, sulla capacità di carico residua.

I modelli numerici agli elementi finiti (FEM) sono stati realizzati con il software commerciale Abaqus versione 6.13 [4]; sono stati scritti degli script in linguaggio Python [5] che generano i modelli FEM all'interno di Abaqus. Con gli script è possibile variare i

parametri di progettazione quali geometria del pannello, numero di danneggiamenti, condizioni di carico e così via.

Nel capitolo 1 sono presentati i materiali compositi e il loro comportamento meccanico, in termini di macromeccanica della lamina e del laminato; nel capitolo 2 si descrive il software Abaqus utilizzato per l'analisi FEM. Nel capitolo 3 si descrive in dettaglio il modello numerico di un pannello rinforzato con rib e stringer; il modello è parametrico nella geometria, nel numero di rinforzi, nei layup e così via. Il danneggiamento è modellato come una cricca passante che interessa lo stringer; il numero e la posizione dei danneggiamenti sono parametrici. I danneggiamenti sono riparati con una piastra metallica bullonata.

Nel capitolo 4 si presentano i risultati delle simulazioni per due diverse configurazioni di danneggiamento: configurazione A, dove vi sono due cricche, una fissa e l'altra mobile, sullo stesso stringer; configurazione B, dove vi sono due cricche ognuna su uno stringer diverso, di cui una sola è mobile. Per ognuna di queste due configurazioni si analizzano tre posizioni diverse della cricca mobile e si confrontano i risultati numerici.

Nel capitolo 5 si presenta il modello FEM di un pannello con due stringer e due rib e un difetto, senza riparazione; vengono descritti il codice Python e il codice Matlab utilizzati per l'ottimizzazione tramite algoritmo genetico.

Nell'ultimo capitolo sono presentate le conclusioni e gli sviluppi futuri di questo lavoro.

Capitolo 1 I MATERIALI COMPOSITI

1.1 Introduzione ai materiali compositi

I materiali compositi nascono dalla combinazione di due o più materiali differenti, con lo scopo di migliorare le caratteristiche del singolo materiale, nonché, a volte, di generare proprietà non presenti nei costituenti [6]. Dalla definizione stessa di materiale composito è facile intuire come la fase di progettazione sia fondamentale per la caratterizzazione finale del materiale stesso.

Nella trattazione seguente si approfondirà la risposta meccanica dei materiali compositi sotto forma di laminati, essendo tale tipo di composito uno dei maggiormente utilizzati.

Un laminato è composto da più strati, definiti lamine, ognuna delle quali è costituita da due fasi differenti:

- la matrice, che funge da collante
- le fibre, che fungono da rinforzo

Nella canonica caratterizzazione dei compositi in laminato si distinguono tre principali campi di interesse:

- micro-meccanica della lamina, in cui si studia l'interazione tra matrice e rinforzo; ciò che si vuole conoscere è la risposta complessiva della lamina, in termini di proprietà elastiche e resistenza, relazionate alle singole risposte dei costituenti
- macro-meccanica della lamina, in cui si studia la risposta del materiale composito in termini di proprietà medie, ovvero viene trattato come se fosse omogeneo
- macro-meccanica del laminato, in cui si studia la risposta complessiva del materiale, ottenuto dalla sovrapposizione delle singole lamine.

Nel seguito non verrà approfondita la micro-meccanica della lamina, essendo di poco interesse in relazione al lavoro di tesi: le proprietà assunte per i materiali sono state ricavate da letteratura tecnica. Invece sia la macro-meccanica della lamina che la macro-meccanica del laminato sono di interesse nella progettazione FEM, ragion per cui saranno approfondite nel seguito.

1.2 Macro-Meccanica della Lamina

1.2.1 Ortotropia speciale e ortotropia generale

La macro-meccanica della lamina studia il comportamento della singola lamina, considerando proprietà medie e dunque supponendo un andamento omogeneo all'interno del materiale.

L'ipotesi alla base è quella di stato piano di sollecitazione, definito "plane stress", che si può esprimere come:

$$\sigma_3 = 0\,,\tau_{23} = 0\,,\tau_{13} = 0 \tag{1.1}$$

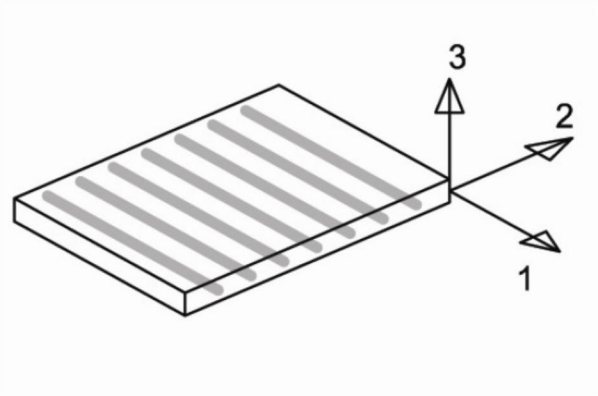


Figura 1-1 Lamina in composito con fibre [7]

La notazione utilizzata, come si può osservare in [Fig 1.1], indica con:

- 1 la direzione delle fibre

- 2 la direzione ortogonale alle fibre
- 3 la direzione ortogonale al piano della lamina

Poiché la lamina è sottile e viene caricata del suo stesso piano l'ipotesi di stato piano di tensione è ampiamente soddisfatta.

Solitamente il comportamento di una lamina di composito è definito ortotropo. Le possibili caratterizzazioni del comportamento meccanico [Fig 1.2], che si possono dare ad un materiale sono:

- isotropo, se le caratteristiche meccaniche rimangono uguali qualsiasi sia la direzione osservata, il punto considerato o il sistema di riferimento adottato: ovvero la risposta è indipendente dalla direzione; appartengono a tale classe la maggior parte dei materiali metallici
- anisotropo, se le caratteristiche meccaniche variano da punto a punto, in funzione anche del sistema di riferimento adottato. Sono necessarie 81 costanti elastiche per caratterizzarlo
- ortotropo, in cui le relazioni costitutive risultano uguali solo in tre piani di simmetria ortogonali

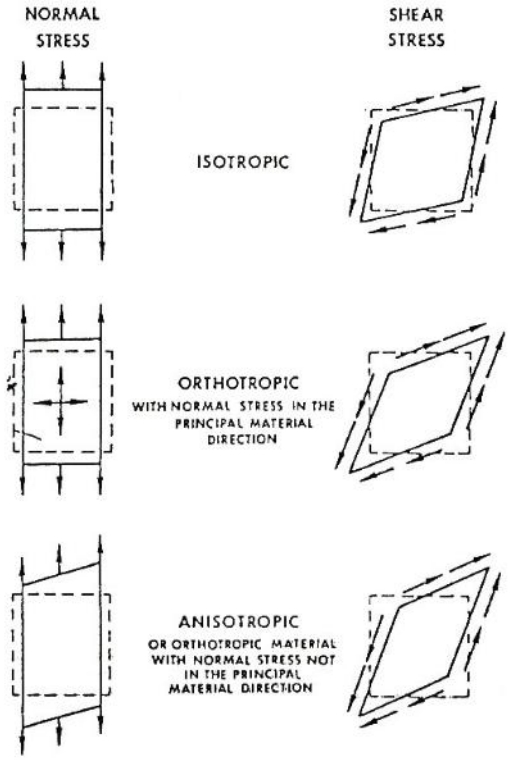


Figura 1-2 Diverse risposte dei materiali isotropi, ortotropi e anisotropi

La condizione di ortotropia può a sua volta essere distinta in:

- ortotropia generale
- ortotropia speciale

Per comprendere tale distinzione si consideri la relazione tra stress e deformazioni per un composito ortotropo generale:

$$\{\sigma\} = [\mathrm{Q}]\{\varepsilon\} \tag{1.2}$$

con:

- σ , tensore degli stress
- Q , matrice di rigidezza

- ε , tensore delle deformazioni

In forma estesa tale relazione diventa:

$$\begin{Bmatrix}\sigma_1\\ \sigma_2\\ \tau_{12}\end{Bmatrix}=\begin{bmatrix}Q_{11} & Q_{12} & Q_{16}\\ Q_{12} & Q_{22} & Q_{26}\\ Q_{16} & Q_{26} & Q_{66}\end{bmatrix}\begin{Bmatrix}\varepsilon_1\\ \varepsilon_2\\ \gamma_{12}\end{Bmatrix} \tag{1.3}$$

La presenza dei termini Q_{16} e Q_{26} fa si che la lamina si comporti come un materiale perfettamente anisotropo: una sollecitazione monodirezionale causa un allungamento , ma anche una distorsione, con conseguente variazione di forma della lamina.

Invece nel caso di ortotropia speciale tali termini sono nulli:

$$\begin{Bmatrix}\sigma_1\\ \sigma_2\\ \tau_{12}\end{Bmatrix}=\begin{bmatrix}Q_{11} & Q_{12} & 0\\ Q_{12} & Q_{22} & 0\\ 0 & 0 & Q_{66}\end{bmatrix}\begin{Bmatrix}\varepsilon_1\\ \varepsilon_2\\ \gamma_{12}\end{Bmatrix} \tag{1.4}$$

Tale condizione si ottiene sollecitando la lamina lungo una delle direzioni principali di ortotropia ("1", "2", "3") e fa si che il composito si comporti come un materiale isotropo: una sollecitazione monodirezionale causa esclusivamente un allungamento.

Nell'equazione considerata e in caso di comportamento isotropo si possono esplicitare i singoli termini come segue:

$$Q_{11}=\frac{E_{11}}{1-\nu_{12}\nu_{21}} \qquad Q_{22}=\frac{E_{22}}{1-\nu_{12}\nu_{21}} \qquad Q_{12}=\frac{\nu_{21}E_{11}}{1-\nu_{12}\nu_{21}} \qquad Q_{66}=G_{12} \tag{1.5}$$

Invertendo l'equazione (1.4) si ottiene:

$$\begin{Bmatrix}\varepsilon_1\\ \varepsilon_2\\ \gamma_{12}\end{Bmatrix}=\begin{bmatrix}S_{11} & S_{12} & S_{16}\\ S_{12} & S_{22} & S_{26}\\ S_{16} & S_{26} & S_{66}\end{bmatrix}\begin{Bmatrix}\sigma_1\\ \sigma_2\\ \tau_{12}\end{Bmatrix} \tag{1.6}$$

dove:

$$S_{11}=\frac{1}{E_1} \qquad S_{22}=\frac{1}{E_2} \qquad S_{12}=-\frac{\nu_{12}}{E_1}=-\frac{\nu_{21}}{E_2} \qquad S_{66}=\frac{1}{G_{12}} \tag{1.7}$$

essendo E_1 , E_2 i moduli di elasticità longitudinale lungo le direzioni principali di

isotropia.

Come evidente per considerazioni sulla simmetria deve essere soddisfatta la seguente relazione:

$$E_{11}\nu_{21} = E_{22}\nu_{12} \tag{1.8}$$

Per quanto riguarda le costanti elastiche della matrice di rigidezza è necessario imporre che i termini della diagonale principale siano positivi:

1. $S_{11}, S_{22}, S_{66} > 0 \;\Rightarrow\; E_1, E_2, G_{12} > 0$
2. $Q_{11}, Q_{22}, Q_{66} > 0 \;\Rightarrow 1 - (\nu_{12}\nu_{21}) > 0$

La seconda condizione, in particolare, porta a scrivere:

$$\nu_{12} < \frac{1}{\nu_{21}} \;\Rightarrow\; \nu_{12} < \frac{E_{11}}{\nu_{12}E_{22}} \;\Rightarrow\; \nu_{12} < \left(\frac{E_{11}}{E_{22}}\right)^{\frac{1}{2}} \tag{1.9}$$

allo stesso modo:

$$\nu_{21} < \frac{1}{\nu_{12}} \;\Rightarrow\; \nu_{21} < \frac{E_{22}}{\nu_{21}E_{11}} \;\Rightarrow\; \nu_{21} < \left(\frac{E_{22}}{E_{11}}\right)^{\frac{1}{2}} \tag{1.10}$$

Poiché per materiale isotropo vale la seguente relazione:

$$G = \frac{E}{2(1+\nu)} > 0 \;\Rightarrow\; (1+\nu) > 0 \;\Rightarrow\; \nu > -1 \tag{1.11}$$

Ed invece per materiale sferico:

$$G > 0 \;\Rightarrow\; (1-2\nu) > 0 \;\Rightarrow\; \nu < \frac{1}{2} \tag{1.12}$$

In definitiva il coefficiente di Poisson è limitato nel range:

$$-1 < \nu < \frac{1}{2} \tag{1.13}$$

1.2.2 Cambio del sistema di riferimento

La risoluzione di problemi numerici risulta molto più complessa per l'ortotropia generale, ragion per cui ha senso effettuare una variazione del sistema di riferimento considerato, passando dal sistema di riferimento della lamina al sistema di riferimento del laminato tramite l'utilizzo di una apposita matrice di rotazione [8].

La relazione che consente di passare dal sistema di riferimento principale della lamina ad un sistema di riferimento qualsiasi è del tipo puramente geometrico:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \begin{bmatrix} cos^2\theta & sin^2\theta & -2cos\theta sin\theta \\ sin^2\theta & cos^2\theta & 2cos\theta sin\theta \\ cos\theta sin\theta & -cos\theta sin\theta & cos^2\theta - sin^2\theta \end{bmatrix} \begin{Bmatrix} \sigma_1 \\ \sigma_2 \\ \tau_{12} \end{Bmatrix} \tag{1.14}$$

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \frac{\gamma_{xy}}{2} \end{Bmatrix} = \begin{bmatrix} cos^2\theta & sin^2\theta & -2cos\theta sin\theta \\ sin^2\theta & cos^2\theta & 2cos\theta sin\theta \\ cos\theta sin\theta & -cos\theta sin\theta & cos^2\theta - sin^2\theta \end{bmatrix} \begin{Bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \frac{\gamma_{12}}{2} \end{Bmatrix} \tag{1.15}$$

dove:

- 1 e 2 sono gli assi principali della lamina
- x e y quelli di un generico sistema di riferimento
- θ l'angolo che l'asse 1 forma con x [Fig 1.3]
- $\frac{\gamma_{xy}}{2} = \varepsilon_{xy}$ e $\frac{\gamma_{12}}{2} = \varepsilon_{12}$

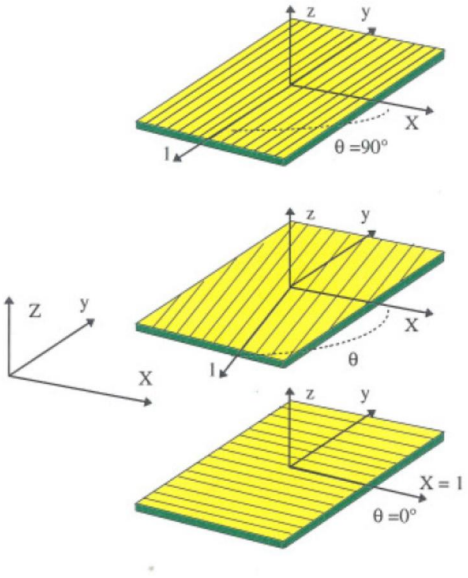


Figura 1-3 Caratterizzazione dell'angolo di direzione delle fibre rispetto al sistema di riferimento globale

Introducendo la matrice di trasformazione [T]:

$$[T]^{-1} = \begin{bmatrix} cos^2\theta & sin^2\theta & -2cos\theta sin\theta \\ sin^2\theta & cos^2\theta & 2cos\theta sin\theta \\ cos\theta sin\theta & -cos\theta sin\theta & cos^2\theta - sin^2\theta \end{bmatrix} \tag{1.16}$$

si avrà:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = [T]^{-1} \begin{Bmatrix} \sigma_1 \\ \sigma_2 \\ \tau_{12} \end{Bmatrix} \qquad \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \frac{\gamma_{xy}}{2} \end{Bmatrix} = [T]^{-1} \begin{Bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \frac{\gamma_{12}}{2} \end{Bmatrix} \tag{1.17}$$

Ricordando le relazioni costitutive (1.3):

$$\begin{Bmatrix} \sigma_1 \\ \sigma_2 \\ \tau_{12} \end{Bmatrix} = [\mathrm{Q}] \begin{Bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \gamma_{12} \end{Bmatrix}$$

e introducendo una matrice di comodo:

$$[R] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

si può scrivere:

$$\begin{Bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{Bmatrix} = [T]^{-1}[Q][R][T][R]^{-1}\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{Bmatrix} \tag{1.18}$$

Poiché vale:

$$[R][T][T]^{-1} = [T]^{-T} \tag{1.19}$$

ponendo:

$$[T]^{-1}[Q][T]^{-T} = [\bar{Q}] \tag{1.20}$$

si ottiene, dall'unione della (1.18), (1.19) e (1.20):

$$\begin{Bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{Bmatrix} = [\bar{Q}]\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{Bmatrix} \tag{1.21}$$

dove le componenti di $\bar{Q}$ sono dipendenti dall'angolo θ e dalle componenti della matrice di partenza Q.

Invertendo la relazione (1.21) si ottiene:

$$\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{Bmatrix} = [\bar{S}]\begin{Bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{Bmatrix} \tag{1.22}$$

Data l'indipendenza delle caratteristiche dalla direzione, per la lamina ortotropa speciale si può scrivere:

$$\begin{Bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{Bmatrix} = \begin{Bmatrix}\sigma_1\\ \sigma_2\\ \tau_{12}\end{Bmatrix} = \begin{bmatrix}Q_{11} & Q_{12} & 0\\ Q_{12} & Q_{22} & 0\\ 0 & 0 & Q_{66}\end{bmatrix}\begin{Bmatrix}\varepsilon_1\\ \varepsilon_2\\ \gamma_{12}\end{Bmatrix} \tag{1.23}$$

Per la lamina ortotropa generale, invece:

$$\begin{Bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{Bmatrix} = \begin{bmatrix}\bar{Q}_{11} & \bar{Q}_{12} & \bar{Q}_{16}\\ \bar{Q}_{12} & \bar{Q}_{22} & \bar{Q}_{26}\\ \bar{Q}_{16} & \bar{Q}_{26} & \bar{Q}_{66}\end{bmatrix}\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{Bmatrix} \tag{1.24}$$

Si osserva dunque che non ci sono differenze tra la risoluzione di una lamina ortotropa speciale e quella di una lamina perfettamente anisotropa, a causa della presenza dei termini $\overline{Q}_{16}$ e $\overline{Q}_{26}$.

Da tali considerazioni è possibile ricavare le costanti ingegneristiche apparenti E_{xx}, E_{yy}, n_{xy}, G_{xy}.

Nel seguito è possibile osservare l'andamento delle costanti ingegneristiche apparenti, per una lamina con matrice epossidica e fibre di rinforzo in vetro, in funzione dell'angolo θ [fig 1.4].

$$\frac{1}{E_x} = \frac{1}{E_1} \cdot \cos^4\theta + \left(\frac{1}{G_{12}} + \frac{2\nu_{12}}{E_1}\right)\sin^2\theta\cos^2\theta + \frac{1}{E_2}\sin^4\theta \tag{1.25}$$

$$\frac{1}{E_y} = \frac{1}{E_1} \cdot \sin^4\theta + \left(\frac{1}{G_{12}} + \frac{2\nu_{12}}{E_1}\right)\sin^2\theta\cos^2\theta + \frac{1}{E_2}\cos^4\theta \tag{1.26}$$

$$\nu_{xy} = E_x\left[\frac{\nu_{12}}{E_1}(\sin^4\theta + \cos^4\theta) - \left(\frac{1}{E_1} + \frac{1}{E_2} - \frac{1}{G_{12}}\right)\sin^2\theta\cos^2\theta\right] \tag{1.27}$$

$$\frac{1}{G_{xy}} = 2\left(\frac{2}{E_1} + \frac{2}{E_2} + 4\frac{\nu_{12}}{E_1} - \frac{1}{G_{12}}\right)\sin^2\theta\cos^2\theta + \frac{1}{G_{12}}(\sin^4\theta + \cos^4\theta) \tag{1.28}$$

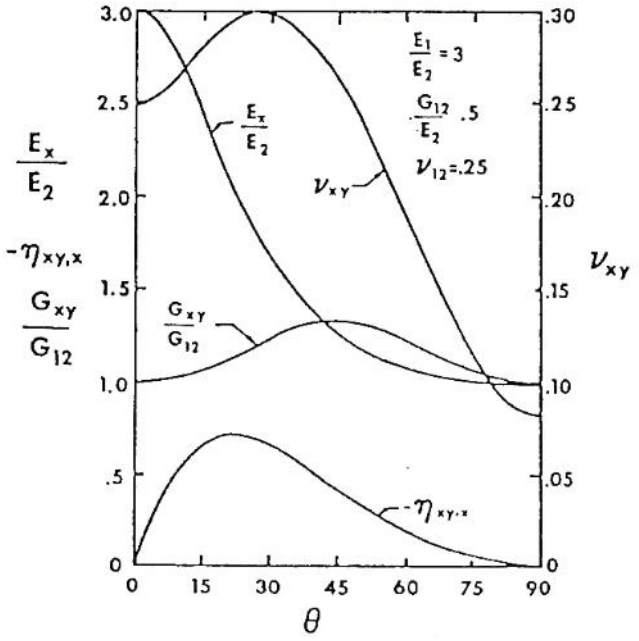


Figura 1-4 andamento delle costanti ingegneristiche in funzione dell'anomalia θ per una lamina con matrice epossidica

Dall'osservazione della [Fig. 1.4] è immediato notare che le caratteristiche meccaniche dipendono dall'orientamento tra fibre e carico. In particolare il modulo di elasticità è massimo se direzione di applicazione del carico e direzione delle fibre coincidono, viceversa è minimo se il carico è applicato in direzione ortogonale a quella delle fibre.

Tali considerazioni comportano un'accurata attenzione nella fase di progettazione: si può decidere infatti di direzionare le fibre nella stessa direzione del carico o anche creare laminati con comportamento complessivo quasi ortotropo.

A questo proposito si distinguono laminati di differente tipo [Fig. 1.5]:

- Laminato Cross-ply:in cui l'angolo della fibra di ogni lamina è normale o parallela a quello delle altre lamine (per esempio contiene sono lamine 0-90°)
- Laminato Angle-ply: in cui l'angolo della fibra non ha alcun limite di parallelismo

o ortogonalità alle altre fibre del laminato

- Laminato simmetrico, con le fibre simmetriche rispetto al piano medio del laminato
- Laminato anti-simmetrico, con le fibre anti-simmetriche ripsetto al piano medio del laminato
- Laminato bilanciato: per ogni ply con fibre in una determinata direzione esiste un ply con le fibre nella direzione dello stesso angolo con segno opposto

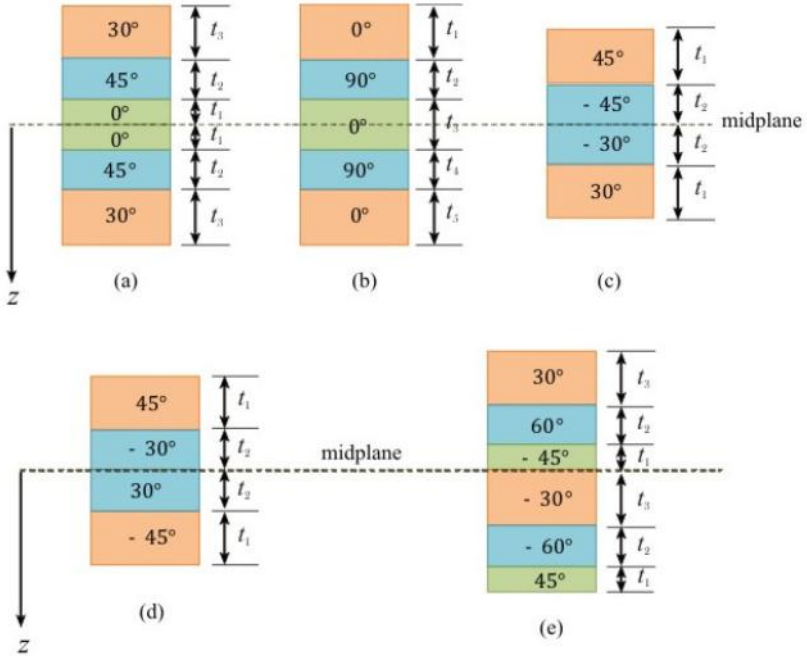


Figura 1-5 Esempi di laminati. (a) laminato simmetrico, (b) laminato cross-ply, (c) laminato angle-ply, (d) laminato anti-simmetrico, (e) laminato bilanciato

1.3 Macro-Meccanica del Laminato

Nella macro-meccanica del laminato si ricavano le proprietà dell'intero laminato considerando le proprietà della singola lamina; l'approccio utilizzato è simile a quello della

teoria delle piastre.

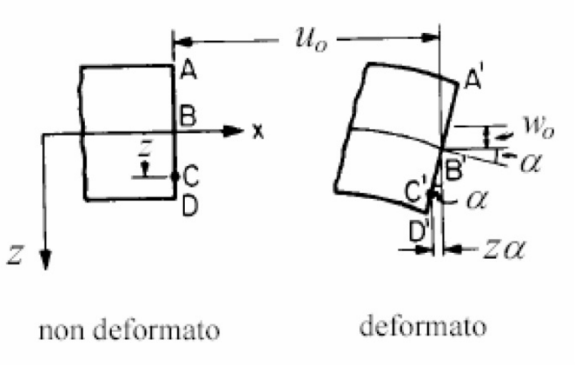


Figura 1-6 Rappresentazione della lamina in caso di deformazione per effetto delle forze out-of-plane

Si possono infatti ipotizzare le seguenti condizioni [Fig 1.6]:

- il campo di spostamento in-plane "u" e "v", associato ad una deformazione del laminato, può essere espresso in funzione di due componenti: lo spostamento del punto considerato relativo al piano medio e un secondo contributo, funzione della distanza tra piano medio e punto analizzato, indicato con "z"
- lo spostamento fuori dal piano "w" è indipendente dalla coordinata "z"
- assenza di deformazione in direzione normale al piano medio :

$$\varepsilon_z = 0 \qquad \gamma_{xz} = 0 \qquad \gamma_{yz} = 0 \tag{1.29}$$

In particolare con riferimento alla prima condizione:

$$u(x,y,z) = u_0(x,y) - z\frac{\partial w_0}{\partial x} \qquad v(x,y,z) = v_0(x,y) - z\frac{\partial w_0}{\partial y} \tag{1.30}$$

Le deformazioni associate al campo di spostamenti saranno:

$$\varepsilon_x = \frac{\partial u}{\partial x} \qquad \varepsilon_y = \frac{\partial v}{\partial y} \qquad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{1.31}$$

Sostituendo la (1.30) nella (1.31) si ottiene:

$$\varepsilon_x = \frac{\partial u^0}{\partial x} - z\frac{\partial^2 w^0}{\partial x^2} \qquad \varepsilon_y = \frac{\partial v^0}{\partial y} - z\frac{\partial^2 w^0}{\partial y^2} \qquad \gamma_{xy} = \frac{\partial u^0}{\partial y} + \frac{\partial v^0}{\partial y} - 2z\frac{\partial^2 w^0}{\partial x \partial y} \tag{1.32}$$

Definendo le deformazioni nel piano medio rispettivamente con:

$$\varepsilon_x^0 = \frac{\partial u^0}{\partial x} \qquad \varepsilon_y^0 = \frac{\partial v^0}{\partial y} \qquad \gamma_{xy}^0 = \frac{\partial u^0}{\partial y} + \frac{\partial v^0}{\partial x} \tag{1.33}$$

e indicando i cambi di curvatura come segue:

- $\frac{\partial^2 w^0}{\partial x^2} = K_x$ (cambio di curvatura nel piano *xz)*
- $\frac{\partial^2 w^0}{\partial y^2} = K_y$ (cambio di curvatura nel piano *yz)*
- $2z\frac{\partial^2 w^0}{\partial x \partial y} = K_{xy}$ (cambio di curvatura dovuto alla torsione)

si ricava:

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} {\varepsilon_x}^0 \\ {\varepsilon_y}^0 \\ {\gamma_{xy}}^0 \end{Bmatrix} + \begin{Bmatrix} K_x \\ K_y \\ K_{xy} \end{Bmatrix} \tag{1.34}$$

Tale relazione indica come la deformazione sia funzione della posizione della lamina rispetto all'asse baricentrico.

Considerando la lamina K-esima:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix}_K = \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} & \bar{Q}_{16} \\ \bar{Q}_{12} & \bar{Q}_{22} & \bar{Q}_{26} \\ \bar{Q}_{16} & \bar{Q}_{26} & \bar{Q}_{66} \end{bmatrix}_K \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix}_K \tag{1.35}$$

da cui sostituendo con l'equazione (1.34) appena ricavata:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix}_K = \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} & \bar{Q}_{16} \\ \bar{Q}_{12} & \bar{Q}_{22} & \bar{Q}_{26} \\ \bar{Q}_{16} & \bar{Q}_{26} & \bar{Q}_{66} \end{bmatrix}_K \left\{ \begin{Bmatrix} {\varepsilon_x}^0 \\ {\varepsilon_y}^0 \\ {\gamma_{xy}}^0 \end{Bmatrix} + \begin{Bmatrix} K_x \\ K_y \\ K_{xy} \end{Bmatrix} \right\}_K \tag{1.36}$$

Dunque le fibre in direzione perpendicolare si deformano di più, essendo meno rigide, mentre quelle in direzione longitudinale si deformano di meno, essendo più rigide: la tensione sarà maggiore nelle fibre a 0°, a parità di deformazione.

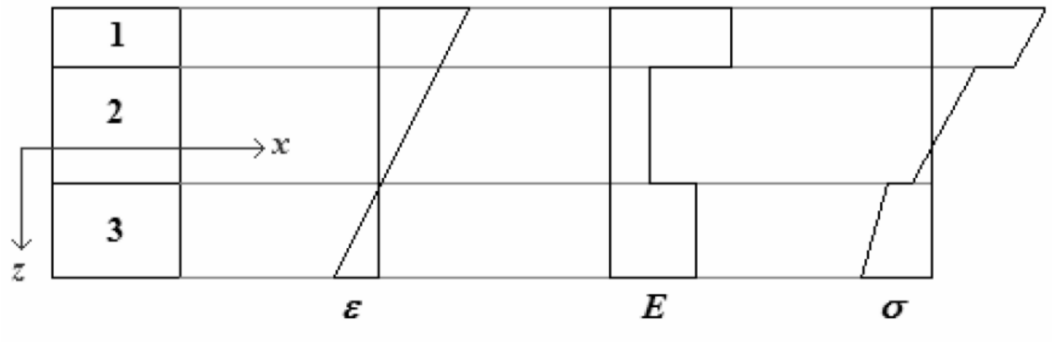


Figura 1-7 Discontinuità della deformazione e della tensione lungo lo spessore del laminato, in funzione della rigidezza dei ply

La differenza di tensione tra le diverse lamine causa discontinuità all'interfaccia [Fig. 1.7], che a sua volta genera taglio interlaminare. Indicando con *h* lo spessore del laminato e con *n* il numero delle lamine, le forze agenti sul laminato [Fig 1.8] saranno date dalle seguenti espressioni:

$$\begin{Bmatrix} N_x \\ N_y \\ N_{xy} \end{Bmatrix} = \int_{-h/2}^{h/2} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} dz = \sum_{i=1}^{h_k} \int_{h_{(k-1)}}^{h_k} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} dz \tag{1.37}$$

$$\begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} = \int_{-h/2}^{h/2} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} z dz = \sum_{i=1}^{h_k} \int_{h_{(k-1)}}^{h_k} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} z dz \tag{1.38}$$

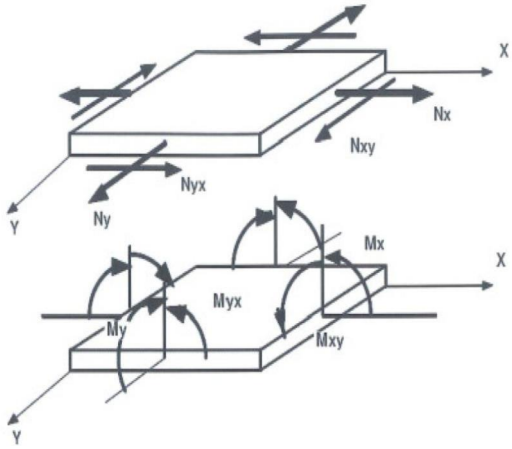


Figura 1-8 Rappresentazione delle forze e dei momenti agenti sulla lamina

Sostituendo le equazioni precedentemente ricavate (1.36), si ottengono le relazioni tra sollecitazioni e deformazioni [Fig 1.9]:

$$\begin{Bmatrix} N_x \\ N_y \\ N_{xy} \end{Bmatrix} = \sum_{k=1}^{N} \begin{bmatrix} \bar{Q_{11}} & \bar{Q_{12}} & \bar{Q_{16}} \\ \bar{Q_{21}} & \bar{Q_{22}} & \bar{Q_{26}} \\ \bar{Q_{61}} & \bar{Q_{62}} & \bar{Q_{66}} \end{bmatrix}_k \left\{ \int_{h_k-1}^{h_k} \begin{Bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{Bmatrix} dz + \int_{h_k-1}^{h_k} \begin{Bmatrix} K_x \\ K_y \\ K_z \end{Bmatrix} z dz \right\} \tag{1.39}$$

$$\begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} = \sum_{k=1}^{N} \begin{bmatrix} \bar{Q_{11}} & \bar{Q_{12}} & \bar{Q_{16}} \\ \bar{Q_{21}} & \bar{Q_{22}} & \bar{Q_{26}} \\ \bar{Q_{61}} & \bar{Q_{62}} & \bar{Q_{66}} \end{bmatrix}_k \left\{ \int_{h_k-1}^{h_k} \begin{Bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{Bmatrix} z dz + \int_{h_k-1}^{h_k} \begin{Bmatrix} K_x \\ K_y \\ K_z \end{Bmatrix} z^2 dz \right\} \tag{1.40}$$

Poiché $\varepsilon_x^0, \varepsilon_y^0, \gamma_{xy}^0, K_x, K_y, K_{xy}$ non dipendono da *z* possono essere portate fuori dal termine di integrale:

$$\begin{Bmatrix} N_x \\ N_y \\ N_{xy} \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{16} \\ A_{21} & A_{22} & A_{26} \\ A_{61} & A_{62} & A_{66} \end{bmatrix} \begin{Bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{Bmatrix} + \begin{bmatrix} B_{11} & B_{12} & B_{16} \\ B_{21} & B_{22} & B_{26} \\ B_{61} & B_{62} & B_{66} \end{bmatrix} \begin{Bmatrix} K_x \\ K_y \\ K_{xy} \end{Bmatrix} \begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} =$$

$$\begin{bmatrix} B_{11} & B_{12} & B_{16} \\ B_{21} & B_{22} & B_{26} \\ B_{61} & B_{62} & B_{66} \end{bmatrix} \begin{Bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{Bmatrix} + \begin{bmatrix} B_{11} & D_{12} & D_{16} \\ D_{21} & D_{22} & D_{26} \\ D_{61} & D_{62} & D_{66} \end{bmatrix} \begin{Bmatrix} K_x \\ K_y \\ K_{xy} \end{Bmatrix} \tag{1.41}$$

Dove:

- $A_{ij} = \sum_{k=1}^{N} (\bar{Q_{ij}})_k (h_k - h_{(k-1)})$
- $B_{ij} = \frac{1}{2} \sum_{k=1}^{N} (\bar{Q_{ij}})_k (h_k^2 - h_{(k-1)}^2)$
- $D_{ij} = \frac{1}{3} \sum_{k=1}^{N} (\bar{Q_{ij}})_k (h_k^3 - h_{(k-1)}^3)$

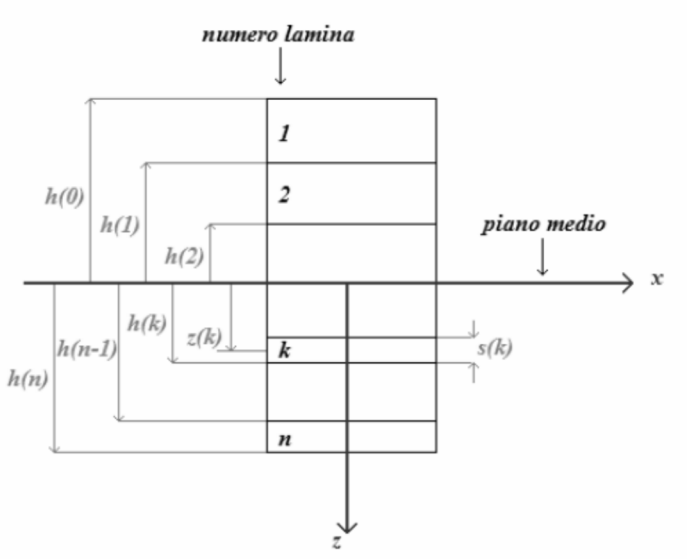


Figura 1-9 Geometria del laminato e notazione utilizzata

In sintesi:

$$\begin{Bmatrix} N \\ \dots \\ M \end{Bmatrix} \begin{bmatrix} A & \vdots & B \\ \dots & \dots & \dots \\ B & \vdots & D \end{bmatrix} \begin{Bmatrix} \varepsilon^0 \\ \dots \\ K \end{Bmatrix} \tag{1.42}$$

in cui:

- $\begin{bmatrix} A & \vdots & B \\ \dots & \dots & \dots \\ B & \vdots & D \end{bmatrix}$ è la matrice di rigidezza del laminato
- $[A]$ è la matrice di rigidezza assiale
- $[D]$ è la matrice di rigidezza flessionale

- $[B]$ è la matrice di rigidezza di accoppiamento (lega la flessione allo sforzo normale); se il laminato è bilanciato e simmetrico $[B]$=0.

Si possono fare alcune semplici considerazioni:

- il laminato se soggetto a solo sforzo normale si deforma anche flessionalmente, infatti espandendo il calcolo matriciale:

$$N = A\varepsilon^0 + BK$$

- se il laminato è soggetto solo a momento flettente si deforma anche assialmente, perchè:

$$M = B\varepsilon^0 + DK$$

Dalla prima delle due equazioni ricavate:

$$\varepsilon^0 = A^{-1}N - A^{-1}BK$$

e sostituendo nella seconda:

$$M = BA^{-1}N + (-BA^{-1}B + D)K$$

quindi:

$$\begin{Bmatrix}\varepsilon^0\\ \dots\\ M\end{Bmatrix}\begin{bmatrix}A^{-1} & \vdots & -A^{-1}B\\ \dots & \dots & \dots\\ BA^{-1} & \vdots & D - BA^{-1}B\end{bmatrix}\begin{Bmatrix}N\\ \dots\\ K\end{Bmatrix} \tag{1.43}$$

cioè:

$$\varepsilon^0 = A^*N - B^*K \qquad M = H^*N - D^*K \tag{1.44}$$

dove:

$$A^* = A^{-1} \qquad B^* = -A^{-1}B \qquad H^* = BA^{-1} \qquad D^* = D - BA^{-1}B$$

risolvendo per *K*:

$$K = D^{*-1}M - D^{*-1}H^*N \tag{1.45}$$

Per sostituzione della (1.45) nella (1.44):

$$\varepsilon^0 = B^* D^{*-1} M + (A^* - B^* D^{*-1} H^*) N \qquad (1.46)$$

e quindi, in forma matriciale:

$$\begin{Bmatrix} \varepsilon^0 \\ \cdots \\ K \end{Bmatrix} = \begin{bmatrix} A^* - B^{*-1} H^* & \vdots & B^* D^{*-1} \\ \cdots & \vdots & \cdots \\ -D^{*-1} H^* & \vdots & D^{*-1} \end{bmatrix} \begin{Bmatrix} N \\ \cdots \\ M \end{Bmatrix} \qquad (1.47)$$

Raccogliendo i termini:

$$\begin{Bmatrix} \varepsilon^0 \\ \cdots \\ K \end{Bmatrix} = \begin{bmatrix} A' & \vdots & B' \\ \cdots & \vdots & \cdots \\ H' & \vdots & D' \end{bmatrix} \begin{Bmatrix} N \\ \cdots \\ M \end{Bmatrix} \qquad (1.48)$$

dove:

$$A' = A^* - B^{*-1} H^* \quad B' = B^* D^{*-1} \quad H' = -D^{*-1} H^* \quad D' = D^{*-1}$$

Si può inoltre dimostrare che *H' = B'* in virtù della simmetria di *A*, *B*, *D* e delle definizioni di *A'*, *B'*, *D'*, A^*, B^*, D^*.

Capitolo 2 IL SOFTWARE ABAQUS

2.1 Introduzione

Il software utilizzato in questo lavoro è il software commerciale di modellazione Abaqus FEA (Finite Element Analysis) versione 6.13. Questi si basa sul metodo degli elementi finiti, che consiste nel discretizzare il dominio di calcolo tramite degli elementi; a ciascuno degli elementi si assegnano delle proprietà ed un comportamento meccanico [9]. Grazie alla discretizzazione, le derivate con cui si descrive il problema fisico vengono approssimate con delle equazioni algebriche. Il metodo FEM segue tre fasi [10]: impostazione matematica del problema; risoluzione del problema matematico; analisi dei risultati.

Abaqus permette di risolvere numericamente problemi di natura differente utilizzando sezioni del programma diverse:

- Abaqus/Standard, in cui si risolvono problemi di carattere generale
- Abaqus/Explicit, in cui si risolvono problemi di natura dinamica in modo esplicito
- Abaqus/CAE, in cui è possibile progettare e rifinire modelli ad elementi finiti, risolvere le equazioni che lo caratterizzano, valutare i risultati
- Abaqus/Viewer, in cui si possono visualizzare i risultati ottenuti

Nel caso specifico di questa trattazione è stato utilizzato il modulo Abaqus/CAE, motivo per cui nel seguito sarà l'unico analizzato. La sezione Abaqus/CAE è composta da differenti moduli:

- modulo Part, in cui viene costruito il modello da un punto di vista geometrico

- modulo Property, in cui sono assegnate le proprietà alle singole entità create nel modulo Part
- modulo Assembly, in cui si definiscono le parti presenti, nonché la posizione reciproca delle stesse
- modulo Step, in cui viene impostato il metodo di risoluzione e la durata dei singoli intervalli temporali
- modulo Interaction, in cui si stabiliscono le influenze reciproche tra le entità definite nel modulo Assembly
- modulo Load, in cui vengono definiti carichi e vincoli del modello
- modulo Mesh, in cui si procede alla costruzione della mesh delle singole parti
- modulo Optimization, in cui, eventualmente, si effettua l'ottimizzazione
- modulo Job, che è il modulo di risoluzione del problema
- modulo Visualization, in cui si visualizzano i risultati ottenuti
- modulo Sketch, in cui si possono creare disegni utili nella caratterizzazione dei vari pezzi

In [Fig 2.1] si possono osservare tali moduli nell'apposita finestra a tendina, insieme all'interfaccia utente che si presenta all'apertura del programma. L'interfaccia varia passando da modulo a modulo, ad esempio con variazione dei comandi presenti nella Toolbox Area.

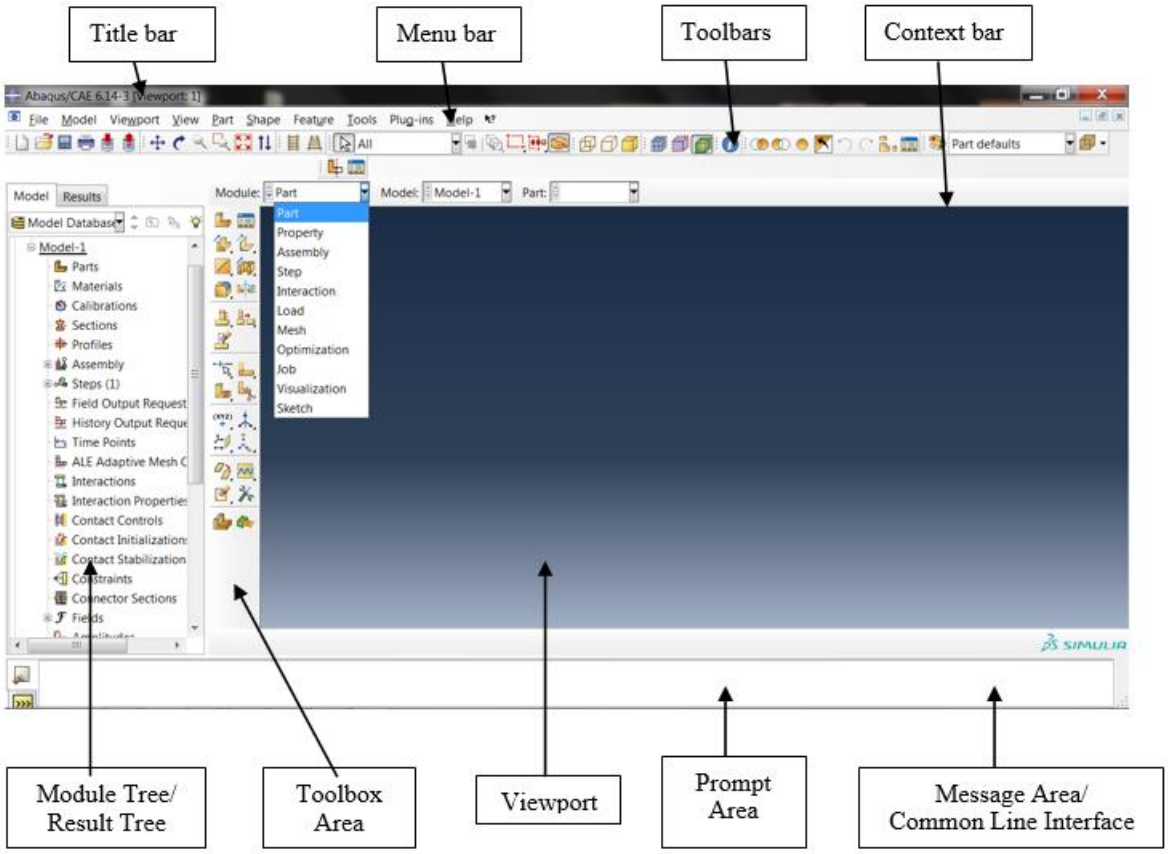


Fig 2.1 Interfaccia Base del Software,
con in evidenza i differenti moduli disponibili

2.2 Modulo Part

Nel modulo Part si procede alla costruzione del modello nelle sue singole componenti; tale procedura può essere effettuata in maniera differente:

- partendo dalla geometria e creando la mesh nella fase successiva
- partendo dalla mesh e importandola nel modulo Part

Si può effettuare il suddetto procedimento sia sul software stesso, che tramite l'importazione di appositi file dall'esterno.

La parte creata utilizzando il modulo Part è definita "*native part*", ovvero parte d'origine, nel senso che le sue caratteristiche saranno utilizzate come informazioni nelle fasi successive: vengono trattate come vere e proprie regole di comportamento della geometria.

All'interno del modulo Part è possibile [Fig 2.2]:

- creare e modificare parti di varia natura: deformabili, rigide in modo discreto, rigide in modo analitico o parti Euleriane
- generare le caratteristiche che definiscono la geometria, ad esempio scegliere se effettuare tagli o raccordature o caratterizzare la parte come solido o come piastra
- effettuare modifiche sulla parte, tramite l'apposita barra di "*Feature Manipulation*", letteralmente manipolazione delle caratteristiche
- definire geometrie di riferimento, quali punti, piani o sistemi di riferimento
- generare partizioni all'interno della parte, intendendo con partizione una sottoregione della parte. Tale definizione risulterà molto utile nella costruzione della mesh del modello, come si vedrà in seguito
- utilizzare il modulo di Sketch per creare, modificare e gestire disegni bidimensionali, da cui ottenere il profilo della parte stessa. A seconda della geometria finale il profilo può essere estruso, utilizzato direttamente, nel caso di elementi bidimensionali, utilizzato come primitiva di rivoluzione, ecc.

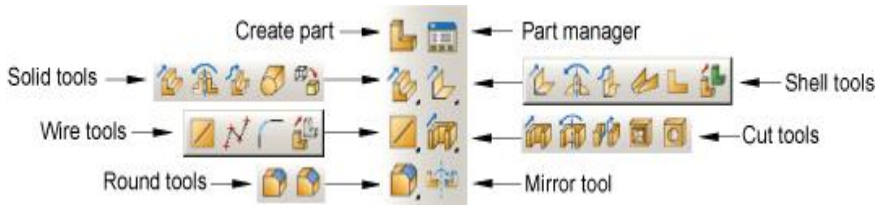


Fig 2.2 Toolbox Area nel modulo Part

2.3 Modulo Property

Il modulo Property caratterizza le proprietà della parte, in particolare è possibile [Fig 2.3]:

- definire i materiali
- caratterizzare le sezioni ed assegnarle alle parti
- definire la sezione trasversale e caratterizzarla tramite un apposito sistema di riferimento
- definite gli strati di un materiale composito
- definire eventuali rinforzi della parte
- definire l'inerzia, ovvero assegnare il centro di massa, l'inerzia alla rotazione, ecc.
- definire elementi elastici e smorzatori tra due punti o tra un punto ed il terreno
- definire, come nel modulo Part, partizioni o punti e piani caratteristici

Nella definizione dei materiali si può scegliere di utilizzare i materiali già schedati nella libreria di Abaqus, piuttosto che crearne di nuovi e caratterizzarli.

Fig 2.3 Toolbox Area nel modulo Property

2.4 Modulo Assembly

Il modulo Assembly consente di creare e modificare l'insieme delle parti, definito appunto "*assembly*". Ogni modello contiene un assembly principale, composto a sua volta da varie istanze di parti, che possono appartenere al modello stesso o essere parte di altri modelli. Si osservi che la singola parte può essere presente nell'assieme in più istanze.

Nella creazione del singolo elemento, ogni parte ha un suo proprio sistema di riferimento, differente dal sistema di riferimento globale dell'assieme. Per questa ragione è necessario definire la posizione relativa del singolo elemento rispetto al sistema di riferimento globale.

Per posizionare l'elemento si definiscono una serie di vincoli di posizione, quali possono essere allineamento delle facce di varie parti, rotazione di una parte rispetto ad un asse o altri.

Nel dettaglio i comandi disponibili in tale modulo consentono [Fig 2.4]:

- la creazione di un pattern regolare di parti, sia di tipo lineare che di tipo circolare
- la traslazione o la rotazione di una istanza
- la definizione di vincoli relativi di posizionamento, quali contatto tra due superfici, due linee o due punti, distanza tra due superfici, due linee o due punti, coassialità tra due istanze, coincidenza del sistema di riferimento relativo

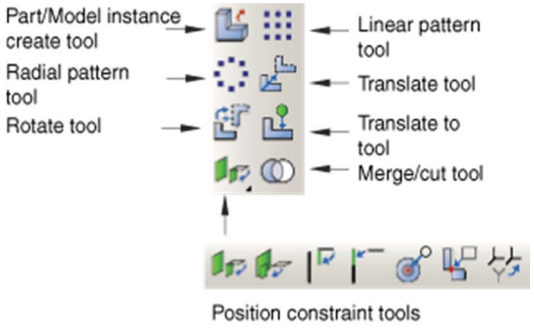


Fig 2.4 Toolbox Area nel modulo Assembly

2.5 Modulo Step

Un concetto basilare nella costruzione del modello in Abaqus è la definizione di "*steps*", ovvero intervalli, in cui suddividere la storia del problema. Nel singoli step viene definita la procedura di risoluzione dello stesso, con possibili variazioni da step a step

Il modulo step viene utilizzato per definire e caratterizzare questi diversi intervalli temporali del problema; in particolare si può [Fig 2.5]:

- creare un'analisi a gradini, ovvero definire i vari step ed essere in grado di stimare le variazioni tra un momento temporale ed il successivo: ad esempio si possono modificare, tra uno step e l'altro, le condizioni al contorno, le forze applicate, la presenza di alcune parti o le interazioni tra i corpi. Inoltre si può variare tipo di analisi, le richieste in output e altri vari controlli
- specificare le richieste di output, definendole in uno step specifico, in seguito al quale verranno propagate in automatico
- definire e specificare il controllo su mesh "*adaptive*", ovvero mesh variabili, a prescindere dalla geometria su cui sono costruite
- definire le regole generali di risoluzione del problema

In generale si distinguono due principali tipologie di step:

- uno step iniziale, in cui sono caratterizzate le condizioni al contorno e le interazioni che si propagano nel resto dell'analisi
- gli step di analisi, che possono essere in numero maggiore di uno. Ad ogni step è associata una specifica procedura che definisce il tipo di analisi da eseguire; tale scelta è effettuata di step in step, garantendo grande flessibilità nel corso dell'analisi. I risultati dello step precedente influenzano la storia dello step successivo.

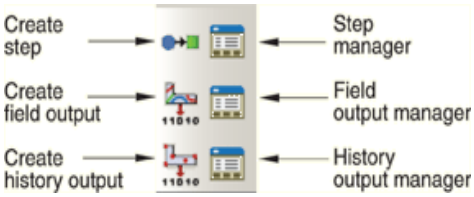


Fig 2.5 Toolbox Area nel modulo Step

2.6 Modulo Interaction

Nel modulo Interaction sono definite le interazioni che intercorrono tra le varie istanze, in modo dipendente dallo step, nel senso che tali interrelazioni sono presenti in uno step specifico e si può decidere se propagarle o meno negli step successivi.

Tra le varie interazioni che possono essere definite vi sono:

- interazioni di contatto
- supporto elastico
- radiazioni ambientali e verso l'ambiente
- condizioni termiche al contorno
- vincoli di rigidezza
- penetrazioni
- vincoli di accoppiamento
- equazioni di accoppiamento solido-piastra
- fratture
- molle e smorzatori
- altre

La necessità di definire le interazioni è dovuta al fatto che Abaqus non riconosce il contatto tra due parti a meno che questo non sia definito tramite apposito comando in tale modulo: corpi in contatto geometrico, ma non in contatto analitico, non saranno considerati in relazione.

In generale si distingue tra [Fig 2.6]:

- interaction, se si definiscono condizioni di contatto, di penetrazione, ecc tra due o più corpi; è necessario caratterizzare tali interazioni tramite opportuna definizione di proprietà
- costraint, se si impongono dei limiti sui gradi di libertà. A differenza che nel modulo Assembly tali vincoli non sono definiti solo come posizioni iniziali delle istanze, ma possono essere generate in step specifici

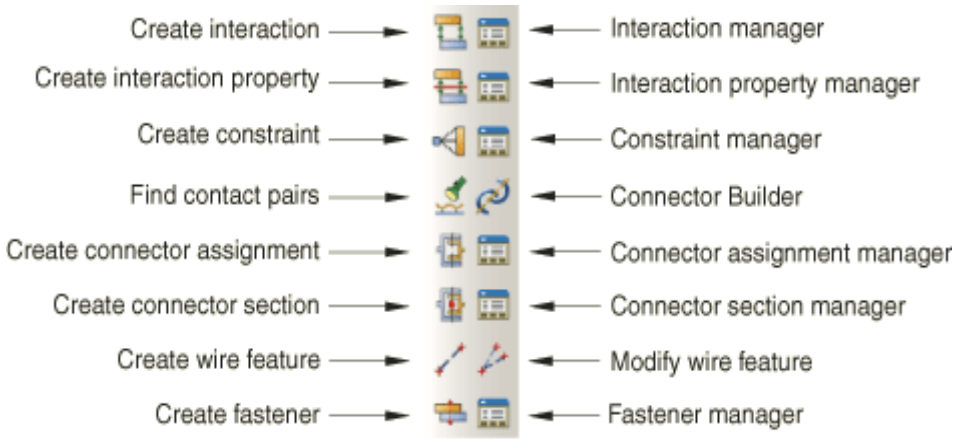


Fig 2.6: Toolbox Area nel modulo Interaction

2.7 Modulo Load

Il modulo Load viene utilizzato per definire carichi, vincoli e "*predefined fields*", ovvero campi predefiniti, quali possono essere campi di velocità o campi di pressione. Come nel

modulo Interaction le entità qui definite sono dipendenti dallo step e si può decidere se propagarle o meno tra i vari step.

In generale sia i carichi che i vincoli possono essere di varia natura [Fig 2.7]:

- meccanica, quali ad esempio forze concentrate o vincoli di simmetria
- elettrica/magnetica, quali ad esempio applicazione di una carica elettrica o di un potenziale elettrico
- di altro tipo, quali ad esempio campi di temperatura

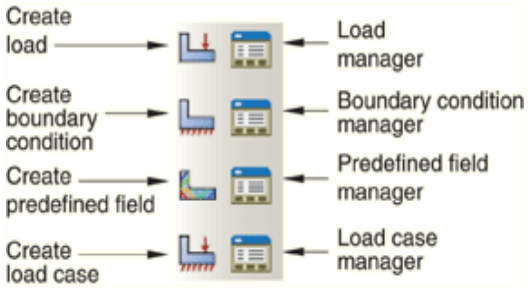


Fig 2.7:Toolbox Area nel modulo Load

2.8 Modulo Mesh

Nel modulo Mesh si può generare la mesh su parti e assiemi, tramite una serie di comandi differenti che caratterizzano la dimensione dell'elemento di mesh, il tipo di elemento di mesh, il tipo di mesh da effettuare. A seconda del tipo di analisi, e dunque della scelta dell'utente, tale processo può essere più o meno automatizzato.

Le caratteristiche della parte e dell'assieme influenzano la forma della mesh stessa, che dunque varia a seconda delle variazioni che vengono effettuate nei livelli precedenti del Module Tree.

L'insieme dei controlli di questo modulo caratterizza la bontà o meno della mesh stessa: affinché l'analisi sia ingegneristicamente valida è necessario assicurare una buona qualità di mesh.

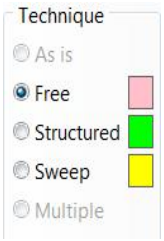


Fig 2.8: Colori legati al tipo di tecnica

Tra i comandi presenti in questo modulo vi è la possibilità di [Fig 2.10]:

- dimensionare sia localmente che globalmente gli elementi [Fig 2.9a]
- definire la tecnica di mesh, scegliendo tra "*free*", "*structured*" o "*swept*" e con supporto visivo di colori differenti associati a mesh di tipo differente [es. Fig 2.8]
- definire l'algoritmo di mesh, in base al tipo di elemento utilizzato, tra "*medial axis*" e "*advancing front*"
- verificare la qualità di mesh, con indicazione di eventuali elementi distorti
- assegnare il tipo di elemento [Fig 2.9b]
- effettuare un miglioramento della mesh
- salvare la mesh generata come una part

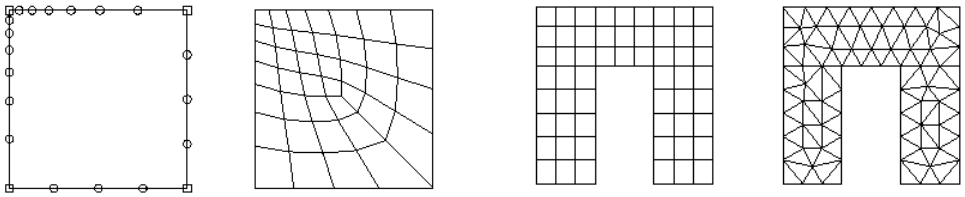


Fig 2.9: a) definizione di dimensioni differenti lungo il bordo dell'elemento
b) scelta di elementi quadrilateri o triangolari

Le tecniche e gli algoritmi di mesh saranno meglio descritti nel capitolo successivo, in relazione al caso studiato.

In generale Abaqus/CAE può generare la mesh in due differenti:

- "*Top-down meshing*", se si effettua la mesh della geometria, utilizzando gli elementi disponibili. Vi è perfetta corrispondenza tra geometria e mesh, ma vi possono essere complicazioni in geometrie particolarmente complesse
- "*Bottom-up meshing*", se la mesh viene generata a partire da entità bidimensionali per creare mesh tridimensionali. La mesh finale può variare notevolmente dalla geometria di partenza

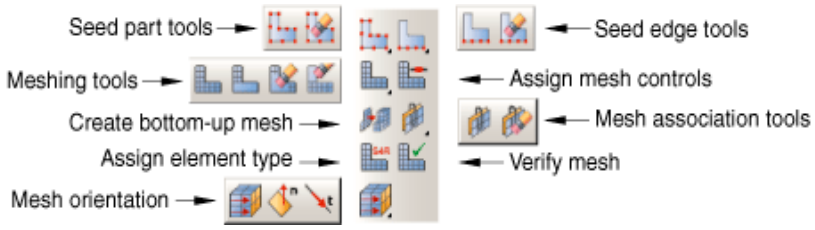


Fig 2.10:Toolbox Area nel modulo Mesh

2.9 Modulo Job

Nel modulo Job si effettua l'analisi del modello, definito in tutti i suoi aspetti nei moduli precedenti.

Si procede generando un "*job*", ovvero un processo, che sarà risolto e monitorato.

Nel "*Job Manager*" [Fig 2.11] si può:

- creare un processo di analisi, associandolo ad uno specifico modello, ed eventualmente modificarlo, rinominarlo, eliminarlo
- generare un "*input file*" senza necessità di far partire l'analisi
- effettuare l'analisi

- monitorare l'analisi e i suoi progressi, tramite l'apposita finestra di dialogo [Fig 2.12] e le indicazioni fornite nella stessa finestra del "*job manager*" ("none", "check submitted", "running", ecc)
- visualizzare i risultati
- eliminare, o meglio uccidere ("*kill*"), il processo, interrompendo qualsiasi analisi sia stata iniziata

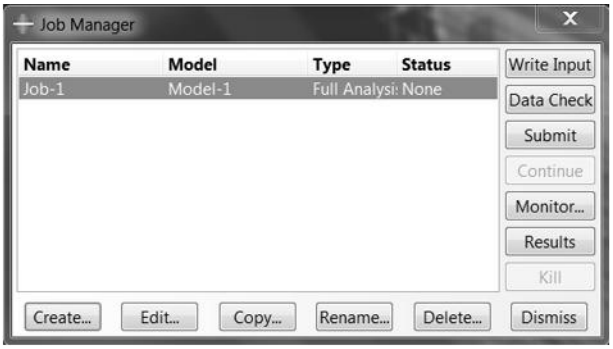


Fig 2.11 Job Manager

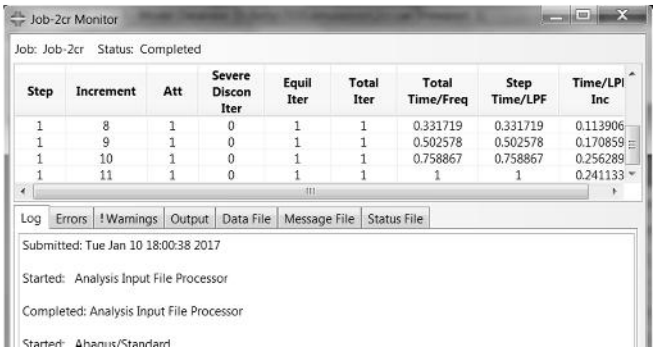


Fig 2.12 Monitoraggio analisi

2.10 Modulo Visualization

Il modulo Visualization consente, anche ad analisi non completata, di visualizzare i risultati e il modello dell'analisi stessa in modo grafico. Le informazioni raffigurate possono essere rappresentate in differenti modi a seconda del comando utilizzato.

In particolare si può scegliere di visualizzare:

- il modello in modo indeformato, deformato o in entrambi i modi
- il modello colorato in maniera opportuna, ad esempio per caratterizzare le concentrazioni di stress, i valori della deformazione o i valori dei carichi applicati
- il modello rappresentato da appositi simboli, che rappresentano l'intensità e la direzione di vettori o tensori specificati
- l'orientamento dei materiali in uno step ben preciso
- una rappresentazione grafica bidimensionale di due specifiche variabili
- l'andamento nel tempo, tramite la rappresentazione consequenziale di immagini a step ben precisi

Capitolo 3 MODELLO NUMERICO DI UN PANNELLO CON DIFETTI MULTIPLI

3.1 Introduzione al modello

In questo capitolo si affronta lo studio di un pannello in materiale composito, rinforzato da un numero variabile di stringers e da due ribs, danneggiato da una cricca passante [Fig 3.1]. La posizione della cricca è limitata alla zona interna del pannello, ovvero alla zona delimitata dai due ribs e dai due stringers esterni e taglia in due lo stringer di interesse. Il modello è stato costruito per un numero variabile di danneggiamenti [11]. In particolare si è valutata l'efficacia di una doppia riparazione, effettuata sia sullo skin, che sullo stringer danneggiato, volgendo particolare attenzione alla zona di bullonatura. Il modello è costruito utilizzando esclusivamente elementi piastra.

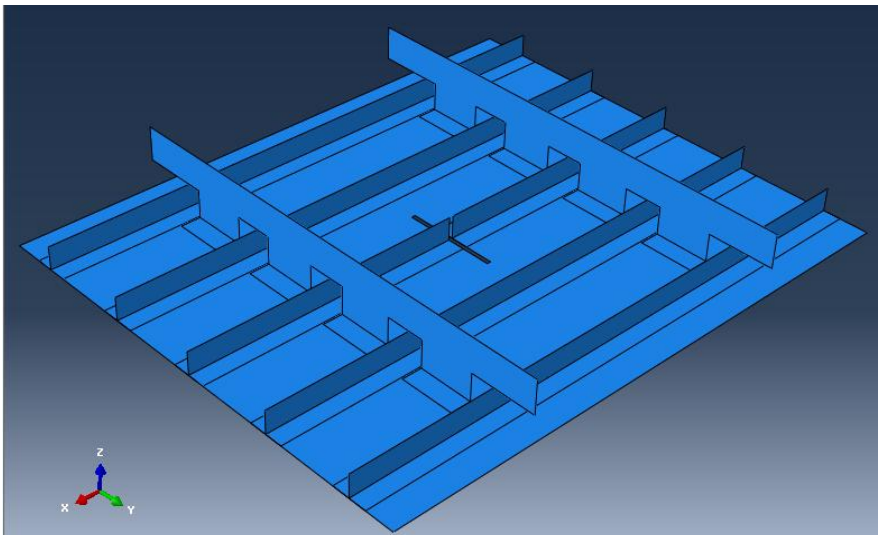


Fig 3.1 Modello geometrico

3.1.1 Materiali e lay-up

Lo skin e i suoi rinforzi sono in materiale composito IMS/977-2, le cui caratteristiche meccaniche sono riportate nel seguito [Tab 3.1]:

E_1, modulo di Young in direzione delle fibre [MPa]	152305
E_2, E_3, modulo di Young in direzione trasversale alle fibre [MPa]	8756
ν_{12}, rapporto di Poisson	0.34
G_{12}, modulo di elasticità tangenziale nel piano [MPa]	3937
G_{23}, G_{13}, modulo di elasticità tangenziale fuori dal piano [MPa]	2834

Tabella 3.1: Proprietà del materiale IMS/977-2

Per ognuno degli elementi la disposizione dei ply, nonché il numero degli stessi è differente:

- skin: 24 ply, orientamento a partire dal basso:

$$(45, 90, 0, 0, -45, -45, 45, 90, 0, -45, 45, 0)_{sym}$$

- stringer: 12 ply, orientamento a partire dall'alto:

$$(45, 90, 0, 0, -45, 0)_{sym}$$

- rib: 12 ply, orientamento a partire dall'alto:

$$(45,0,-45,90)_{|3}$$

Invece le piastre di riparazione sono in materiale metallico, una lega 2000 di Alluminio, con caratteristiche [Tab 3.2]:

Densità [g/mm3]	2,768e-3
Modulo elastico [MPa]	73800
Modulo di taglio [MPa]	27600
Modulo di Poisson	0.34

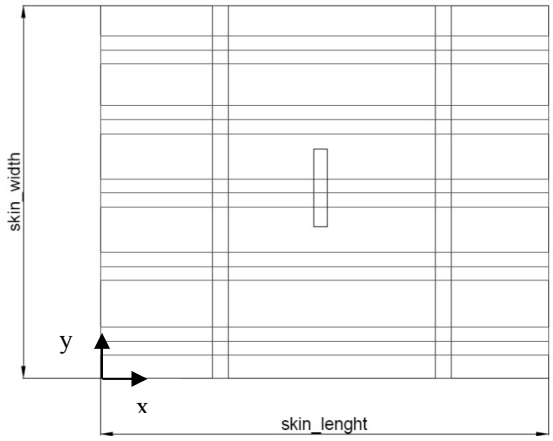
Tabella 3.2: Proprietà del materiale Al 2219-T87

3.1.2 Parametrizzazione

Nel seguito è indicata la nomenclatura utilizzata per ognuna delle parti di interesse [Tab 3.3]:

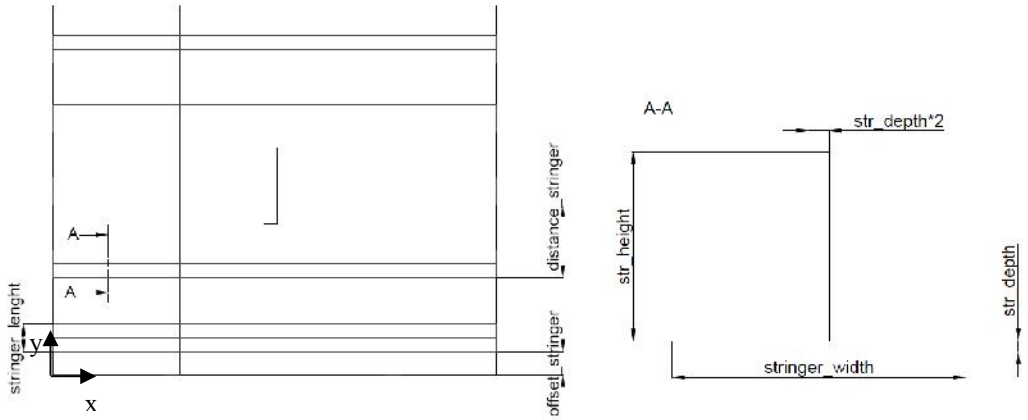
- skin

GRANDEZZA DI INTERESSE	*DENOMINAZIONE*	*VALORE ADOTTATO IN SIMULAZIONE*
Lunghezza in direzione x	skin_lenght	914.4 [mm]
Larghezza in direzione y	skin_width	762 [mm]
Spessore	skin_depth	3.81 [mm]



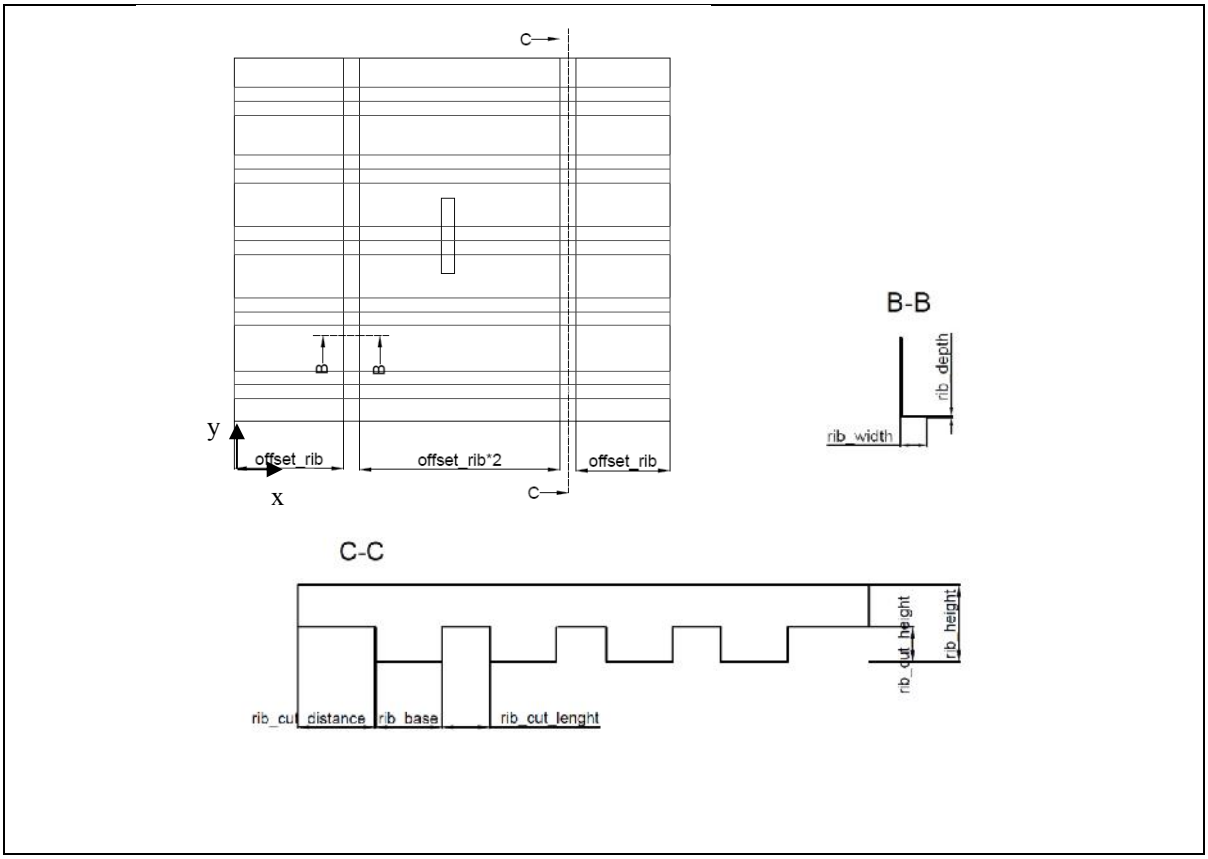
- stringer

GRANDEZZA DI INTERESSE	*DENOMINAZIONE*	*VALORE ADOTTATO IN SIMULAZIONE*
Numero di stringer	n_stringer	5
Lunghezza in direzione y	str_lenght	57.56 [mm]
Altezza in direzione z	str_height	38.1 [mm]
Spessore	str_depth	2.2352 [mm]
Distanza tra due stringer consecutivi	distance_stringer	152.4 [mm]

Distanza del primo stringer dal bordo dello skin	offset_stringer	152.4 [mm]
		

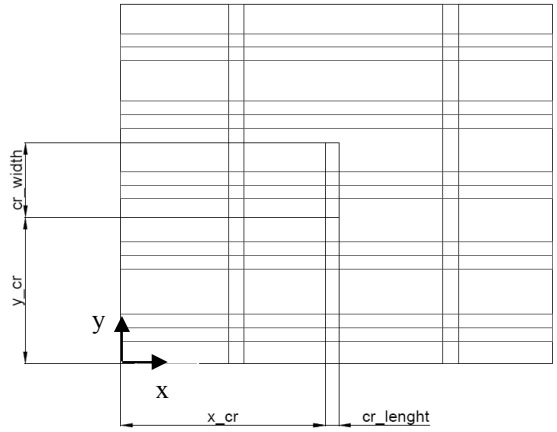
- rib

GRANDEZZA DI INTERESSE	*DENOMINAZIONE*	*VALORE ADOTTATO IN SIMULAZIONE*
Numero di rib	n_rib	2
Lunghezza in direzione x	rib_width	33 [mm]
Altezza in direzione z	rib_height	101.6 [mm]
Spessore	rib_depth	2.4892 [mm]
Altezza del taglio	rib_cut_height	45.72 [mm]
Larghezza del taglio	rib_cut_lenght	63.316 [mm]
Distanza rib dal bordo dello skin	offset_rib	228.6 [mm]
Distanza del bordo dal primo piedino	rib_cut_distance	107.86 [mm]



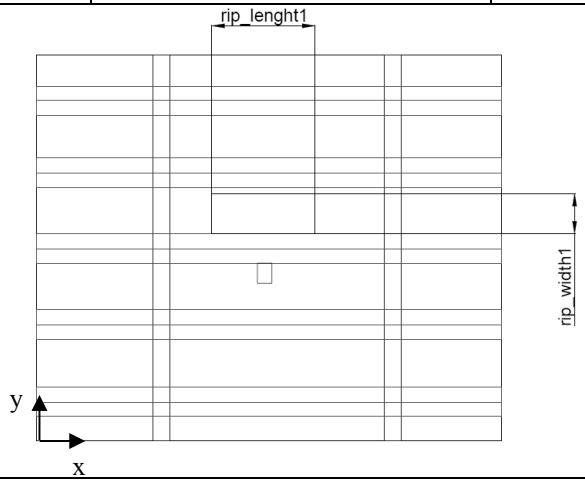
- cricche

GRANDEZZA DI INTERESSE	*DENOMINAZIONE*	*VALORE ADOTTATO IN SIMULAZIONE*
Numero di cricche	n_cr	1
Posizione x rispetto al sdr globale	x_cr	454 [mm]
Posizione y rispetto al sdr globale	y_cr	304.8 [mm]
Lunghezza della cricca in direzione x	cr_lenght	6.35 [mm]
Lunghezza della cricca in direzione y	cr_width	152.4 [mm]



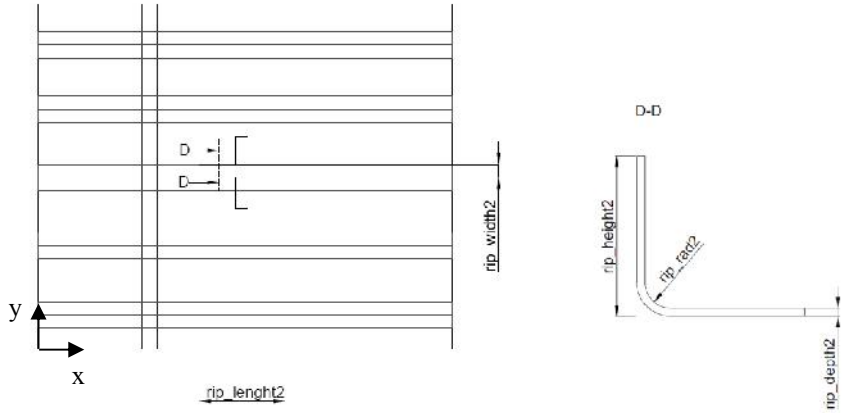
- piastra di riparazione dello skin

GRANDEZZA DI INTERESSE	*DENOMINAZIONE*	*VALORE ADOTTATO IN SIMULAZIONE*
Numero di piastre	n_rip1	2
Lunghezza in direzione x	rip_lenght1	200 [mm]
Lunghezza in direzione y	rip_width1	91.96 [mm]
Spessore	rip_depth1	1[mm]



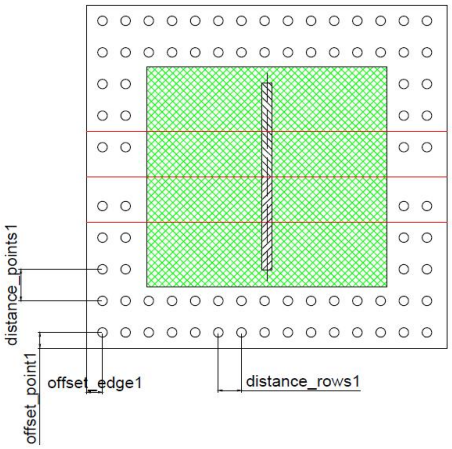
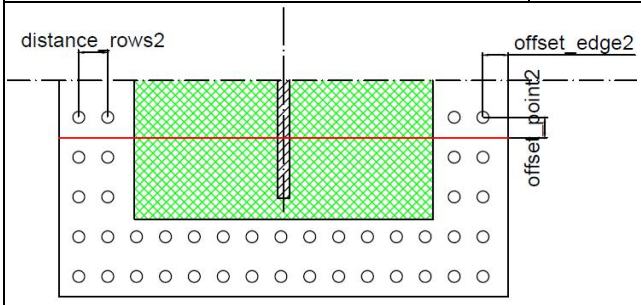
- piastra di riparazione dello stringer

GRANDEZZA DI INTERESSE	*DENOMINAZIONE*	*VALORE ADOTTATO IN SIMULAZIONE*
Numero di piastre	n_rip2	2
Lunghezza in direzione x	rip_lenght2	200 [mm]
Lunghezza in direzione y	rip_width2	26,04 [mm]
Spessore	rip_depth2	1 [mm]
Raggio di curvatura	rip_rad2	2.9972 [mm]
Altezza in direzione z	rip_height2	36.48[mm]



- zona di bullonatura

GRANDEZZA DI INTERESSE	*DENOMINAZIONE*	*VALORE ADOTTATO IN SIMULAZIONE*
Numero di punti nella singola riga di rip1	n_points1	5
Numero di punti nella singola colonna di rip1	n_rows1	10
Distanza del primo punto dal bordo della lastra in direzione x	offset_edge1	10 [mm]
Distanza del primo punto dal bordo della lastra in direzione y	offset_points1	10 [mm]

Distanza tra due punti della stessa riga	distance_rows1	20 [mm]
Distanza tra due punti della stessa colonna	distance_points1	17.9 [mm]
	In verde è indicata la zona di non bullonatura. Le linee rosse indicano l'inizio, la mezzeria e la fine dello stringer interessato dalla cricca. Le zone di riparazione dello skin sono quelle inferiori alla linea rossa inferiore e superiori alla linea rossa superiore nell'immagine.	
Numero di punti nella singola colonna di rip2	n_rows2	10
Distanza del primo punto dal bordo della lastra in direzione x	offset_edge2	10 [mm]
Distanza tra due punti della stessa riga	distance_rows2	20 [mm]
	In verde è indicata la zona di non bullonatura. La linea rossa indica l'inizio dello stringer interessato dalla cricca. Le zone di riparazione dello stringer sono quelle comprese tra la linea rossa e l'asse di simmetria.	

Si osservi che, essendo il numero di file fissato ad 1, il numero di parametri per la riparazione dello stringer è inferiore al numero di parametri per la riparazione dello skin

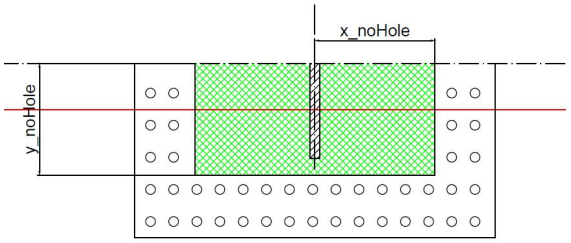
Raggio dei bulloni	fast_rad	3 [mm]
Zona di non bullonatura in direzione x	x_noHole	50 [mm]
Zona di non bullonatura in direzione y	y_noHole	75 [mm]
		In verde è indicata la zona di non bullonatura. La linea rossa indica l'inizio dello stringer interessato dalla cricca.

Tabella 3.3: Nomenclatura del modello parametrico

3.2 Creazione delle parti

3.2.1 Skin

Lo skin è stato ottenuto come modello nello spazio tridimensionale, di tipo deformabile, con caratteristiche di elemento shell planare [Fig 3.2].

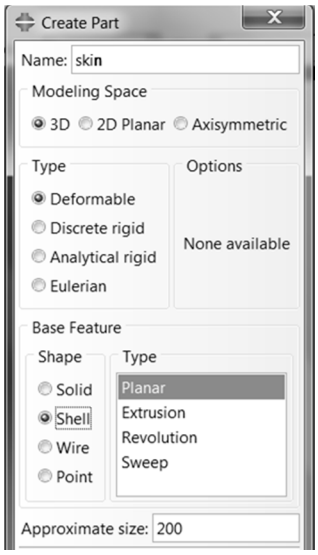


Fig 3.2 Creazione della parte skin

Lo sketch è dunque sufficiente per definirne il profilo, lo spessore sarà caratterizzato in seguito. Si tratta di un profilo rettangolare di larghezza skin_width e lunghezza skin_lenght.

Il foro della cricca è stato eseguito tramite il comando "*CutExtrude* " e inserito all'interno di un ciclo "*for*", per garantire l'applicabilità del codice ad un numero di danneggiamenti superiore ad uno. Il singolo foro ha forma rettangolare [Fig 3.3], con dimensioni: lunghezza cr_lenght e larghezza cr_width; per comodità si è fissata l'origine dello sketch coincidente con la posizione x e la posizione y della cricca.

```
#Creazione cricca
for i in range (n_cr):
    mdb.models['Model-1'].ConstrainedSketch(gridSpacing=59.51,
      name='__profile__',
    sheetSize=2380.57, transform=
    mdb.models['Model-1'].parts['skin'].MakeSketchTransform(
    sketchPlane=mdb.models['Model-1'].parts['skin'].faces.findAt(
      (skin_lenght/2,skin_width/2,0),),
    sketchPlaneSide=SIDE1,
    sketchUpEdge=mdb.models['Model-1'].parts['skin'].edges.findAt(
      (skin_lenght,skin_width/2,0),),
    sketchOrientation=RIGHT, origin=(x_cr[i],y_cr[i],0)))
    mdb.models['Model-1'].parts['skin'].
      projectReferencesOntoSketch(filter=
    COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])
    mdb.models['Model-1'].sketches['__profile__'].
      rectangle(point1=(0, 0),point2=(cr_lenght[i], cr_width[i]))
    mdb.models['Model-  1'].parts['skin'].
      CutExtrude(flipExtrudeDirection=OFF,
    sketch=mdb.models['Model-1'].sketches['__profile__'],
      sketchOrientation=
    RIGHT, sketchPlane=mdb.models['Model-1'].parts['skin'].faces.findAt(
      (skin_lenght/2,skin_width/2,0),),
    sketchPlaneSide=SIDE1, sketchUpEdge=
    mdb.models['Model-1'].parts['skin'].edges.findAt(
      (skin_lenght,skin_width/2,0),))
    del mdb.models['Model-1'].sketches['__profile__']
```

Si osservi, come verrà meglio esplicato in seguito, che tutte le grandezze relative alla cricca sono inserite all'interno di appositi vettori; ad esempio nel caso di tre cricche si scriverà:

```
#Posizione cricche rispetto allo skin
x_cr=[454,554,475]
y_cr=[304.8,325,543]
```

```
#Geometria Cricche
cr_width=[152.4,100,100]
cr_lenght=[6.35,6.35,7]
```

cosicché la posizione zero del vettore, ovvero la prima posizione, sarà caratteristica della prima cricca e così per le successive.

Nel codice riportato si visualizza per la prima volta il comando "*FindAt*", di fondamentale importanza nella costruzione del modello, in quanto sostituisce il generico "*GetSequenceFromMask*", generato dal software Abaqus. Tale sostituzione è necessaria ogni qualvolta nella costruzione del modello si vogliono selezione delle entità geometriche.

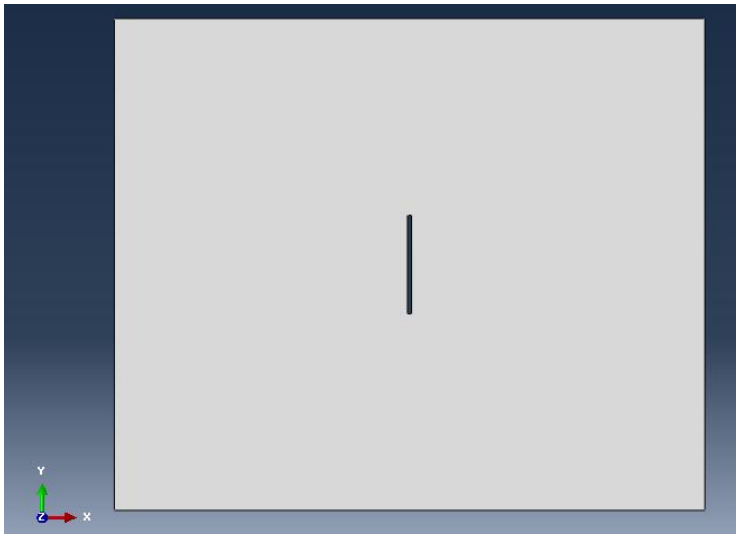


Fig 3.3 Particolare dello skin con una cricca

3.2.2 Stringer

La parte stringer è stata ottenuta dalla combinazione di due diverse superfici piane, la base, che sarà a contatto con lo skin, ed il rinforzo, in direzione perpendicolare alla base, come visibile in [Fig. 3.4]:

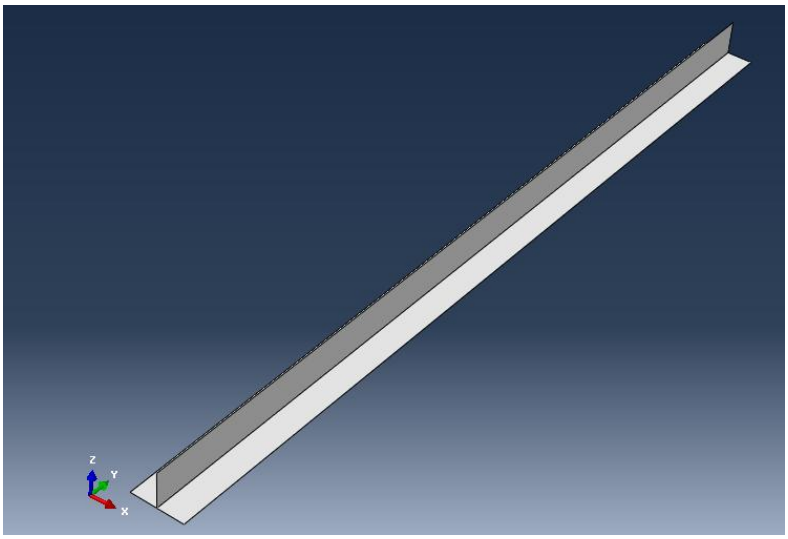


Fig 3.4 Particolare dello stringer

Entrambe le superfici sono ottenute come superfici nello spazio tridimensionale, deformabili da elementi shell piani:

- la base è ottenuta dallo sketch di un rettangolo di larghezza str_lenght e lunghezza skin_lenght, poiché lo stringer ha la stessa lunghezza dello skin su cui è montato
- il rinforzo è ottenuto tramite il comando "*ShellExtrude*", disegnando una linea nella mezzeria della base e estrudendola in direzione z, nel sistema di riferimento relativo allo stringer, per una quantità pari a str_height

3.2.3 Rib

Il rib è ottenuto come estrusione di un profilo ad L e successivo taglio dello stesso. La particolarità di tale procedura risiede nella variazione di forma a cui è soggetto il rib al variare del numero e delle dimensioni degli stringers [Fig 3.5].

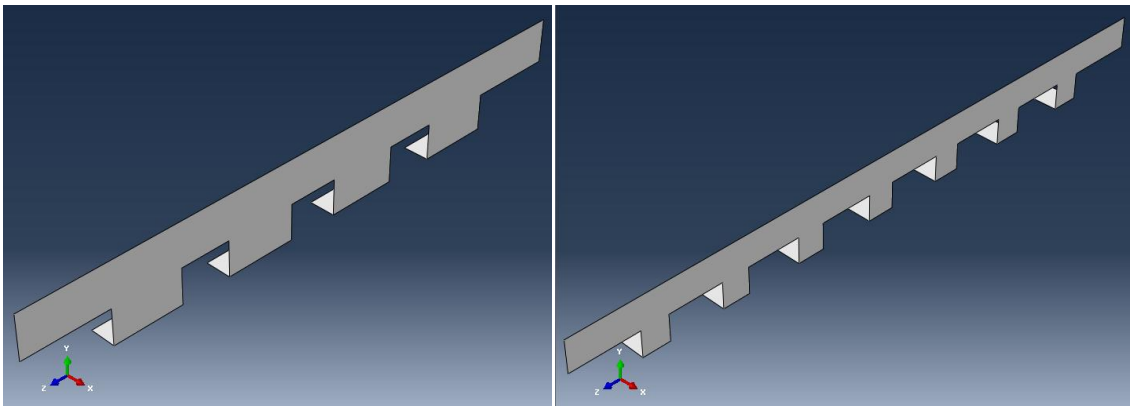


Fig 3.5 Confronto tra un rib generato per 5 stringers di altezza 38.1 [mm] a sinistra
con quello generato per 8 stringers di altezza 25 [mm] a destra

Il profilo del rib è ottenuto come sketch di una L [Fig 3.7-1], di larghezza rib_width e altezza rib_height, imponendo che la parte sia deformabile, appartenente allo spazio tridimensionale e si ottenga dalla estrusione di un elemento shell [Fig 3.6].

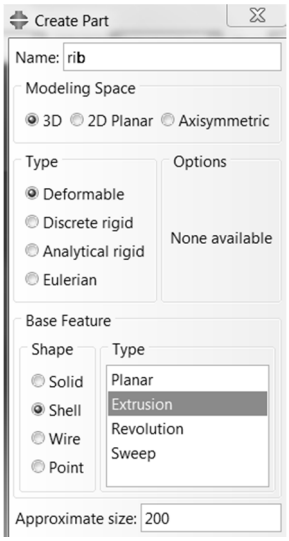


Fig 3.6 Creazione della parte rib

La fase successiva è quella di taglio, eseguita tramite il comando "*CutExtrude*". Nel codice è possibile distinguere [Fig 3.7-2]:

- punti periodici, la cui posizione e numero dipende dal numero degli stringers
- punti non periodici, che delimitano il profilo del rib esternamente

```
#punti periodici, funzione del numero di stringer
for i in range(n_stringer-1):
    mdb.models['Model-1'].sketches['__profile__'].Line(
point1=((2*i+1)*rib_base/2+(i+1)*rib_cut_lenght,-rib_cut_height),
point2=((2*i+1)*rib_base/2+(i+1)*rib_cut_lenght,rib_cut_height/2))
    mdb.models['Model-1'].sketches['__profile__'].Line(
point1=((2*i+1)*rib_base/2+(i+1)*rib_cut_lenght,rib_cut_height/2),
point2=((2*(i+1)+1)*rib_base/2+(i+1)*rib_cut_lenght,rib_cut_height/2))
    mdb.models['Model-1'].sketches['__profile__'].Line(
point1=((2*(i+1)+1)*rib_base/2+(i+1)*rib_cut_lenght,rib_cut_height/2),
point2=((2*(i+1)+1)*rib_base/2+(i+1)*rib_cut_lenght,-rib_cut_height))
    mdb.models['Model-1'].sketches['__profile__'].Line(
point1=((2*(i+1)+1)*rib_base/2+(i+1)*rib_cut_lenght,-rib_cut_height),
point2=((2*(i+1)+1)*rib_base/2+(i+2)*rib_cut_lenght,-rib_cut_height))

#punti non periodici
mdb.models['Model-1'].sketches['__profile__'].Line(
point1=((2*(n_stringer-1)+1)*rib_base/2+n_stringer*rib_cut_lenght,-
rib_cut_height),
point2=(skin_width,-rib_cut_height))
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(skin_width,-
rib_cut_height),

point2=(skin_width,rib_cut_height))
mdb.models['Model-
1'].sketches['__profile__'].Line(point1=(skin_width,rib_cut_height),

point2=(0,rib_cut_height))
mdb.models['Model-
1'].sketches['__profile__'].Line(point1=(0,rib_cut_height),
                                                    point2=(0,-
rib_cut_height))
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0,           -
rib_cut_height),
point2=(rib_base/2+rib_cut_lenght, -rib_cut_height))
```

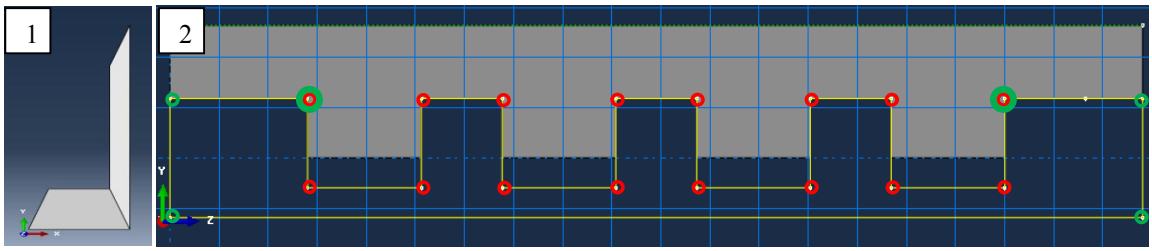


Fig. 3.7 Procedura di creazione del rib: 1) estrusione di un profilo a L

2) taglio dello stesso, con indicazione dei punti periodici in rosso, non periodici in verde

3.2.4 Stringer criccato

Per conoscere il numero di parti "stringer criccato" è necessario confrontare la posizione delle cricche con la posizione degli stringers: può infatti accadere che un solo stringer sia interessato da due diverse cricche, pertanto la parte "stringer criccato" sarà unica, o viceversa due cricche diverse tagliano due stringers differenti, per cui sarà necessaria la creazione di due diverse parti "stringer criccato".

Tale valutazione viene compiuta nel codice tramite il supporto di un ciclo "*if*", in cui si confrontano le posizioni relative allo skin, e dunque al sistema di riferimento globale, di cricche e stringers. Come accennato precedentemente, per la caratterizzazione delle cricche si utilizzano appositi vettori:

```
#Creazione cricca sullo stringer
for j in range (n_stringer):
    for i in range (n_cr):
        if           ((y_cr[i]<((j+1)*distance_stringer+(offset_stringer-
str_lenght)/2)\
        and                                                    cr_width[i]>(-
y_cr[i]+(j+1)*distance_stringer+(offset_stringer-
      str_lenght)/2))):
            str_cr.append(j+1)
            str_cr_cr.append((j+1,i))
            str_cr = list(set(str_cr))
            n_str_cr=len(str_cr)

for k in range (n_str_cr):
            mdb.models['Model-1'].Part(name='stringer-cricca'+str(k+1),
objectToCopy=mdb.models['Model-1'].parts['stringer'])
```

Nel dettaglio, qualora sia soddisfatto il sistema:

$$\begin{cases} inizio\ della\ cricca\ \ prima\ dell'inizio\ dello\ stringer\ k-esimo \\ \quad fine\ della\ cricca\ dopo\ la\ fine\ dello\ stringer\ k-esimo \end{cases}$$

si costruiscono due diversi vettori di supporto, "str_cr" e " str_cr_cr":

- str_cr contiene l'elenco non ripetuto degli stringers intaccati da una o più cricche
- str_cr_cr contiene l'elenco degli stringers intaccati da una cricca, con l'indicazione della stessa

Se, ad esempio [Fig. 3.8], vi sono 3 cricche, di cui la terza agisce sullo stringer 4 e le prime due sullo stringer 5, i vettori avranno rispettivamente forma:

- str_cr = [4,5]
- str_cr_cr = [(4,3), (5,1), (5,2)]

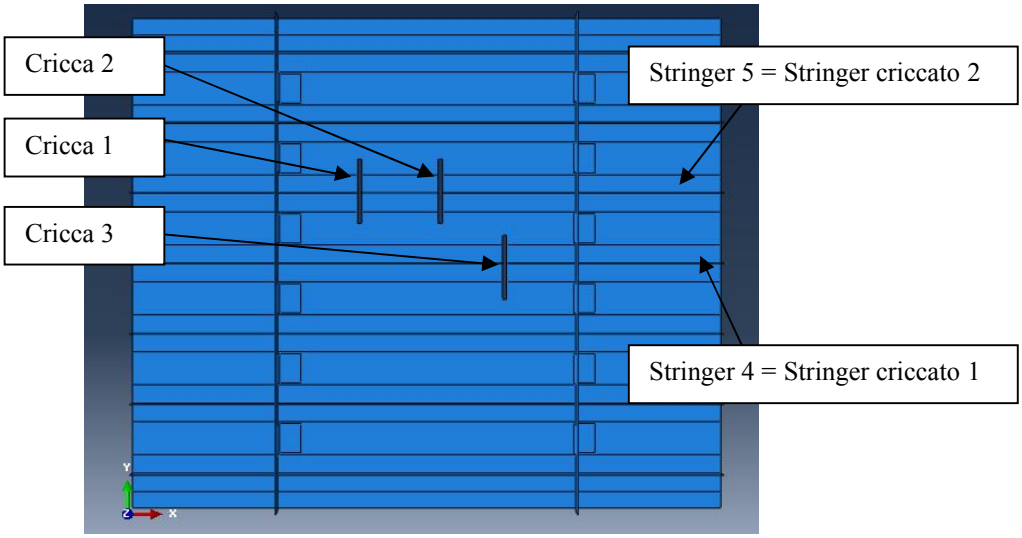


Fig. 3.8 Particolare dell'assieme con tre cricche posizionate su due stringer diversi

Per ottenere gli stringers criccati, così valutati, si effettua un numero di copie adeguato della parte "*stringer*", tramite il comando "*objectToCopy*", e successivamente si taglia tramite il comando "*CutExtrude*".

3.2.5 Piastra di riparazione dello skin

La creazione della parte "*rip1*" è simile a quella di creazione dello skin, trattandosi semplicemente di un rettangolo, ottenuto come elemento shell planare, deformabile, nello spazio tridimensionale.

Le dimensioni dello sketch rettangolare sono pari a rip_lenght1 per rip_width1, dove:

- rip_lenght1 viene inserito dall'utente

- rip_width1 viene valutato, di modo che la fine della piastra di riparazione coincida con il termine del piedino del rib, che corrisponde allo stringer intaccato dalla cricca [Fig 3.9]:

```
rip_width1=distance_stringer-rib_cut_lenght/2-str_lenght/2
```

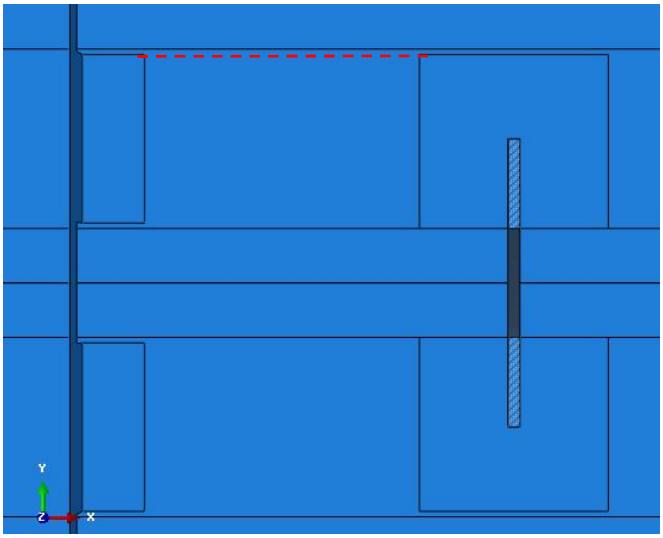


Fig 3.9 Allineamento piastra di riparazione con il piedino del rib

3.2.6 Piastra di riparazione dello stringer

La piastra di riparazione dello stringer si ottiene come estrusione, tramite il comando "*ShellExtrude*", di una forma a L, raccordata nell'angolo [Fig 3.10]. Si tratta di una parte deformabile, nello spazio tridimensionale.

Le dimensioni caratteristiche sono:

- rip_rad2, ovvero raggio di curvatura, inserito dall'utente
- rip_width2, ovvero lunghezza in direzione x, calcolata in modo che il termine della piastra coincida con il termine dello stringer

```
rip_width2=(str_lenght-2*str_depth)/2-rip_depth2/2
```

- rip_height2, ovvero altezza in direzione z, calcolata, ancora, in modo che il termine della piastra coincida con il termine dello stringer

```
rip_height2=str_height-(rip_depth2+str_depth)/2
```

- rip_lenght2, ovvero profondità di estrusione del profilo a L, inserito dall'utente, di norma uguale a rip_lenght1

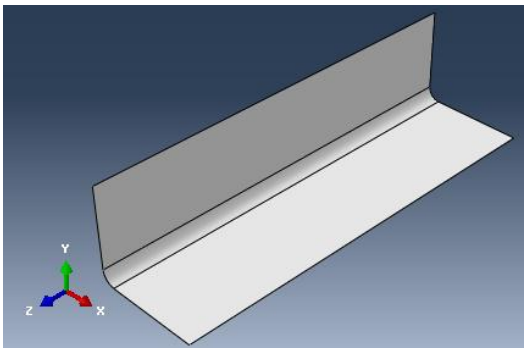


Fig 3.10 Rappresentazione piastra di riparazione dello stringer

3.3 Definizione delle proprietà

3.3.1 Assegnazione dei materiali

I materiali utilizzati per la costruzione del modello sono di natura differente:

- un materiale composito, IMS/977-2, a cui è assegnato comportamento elastico di tipo ortotropo, tramite l'utilizzo del comando "*Engineering Constants*", che consente di caratterizzare le costanti meccaniche del materiale singolarmente. Come già noto, trattandosi di materiale ortotropo, alcune grandezze risultano essere uguali [Fig 3.11]

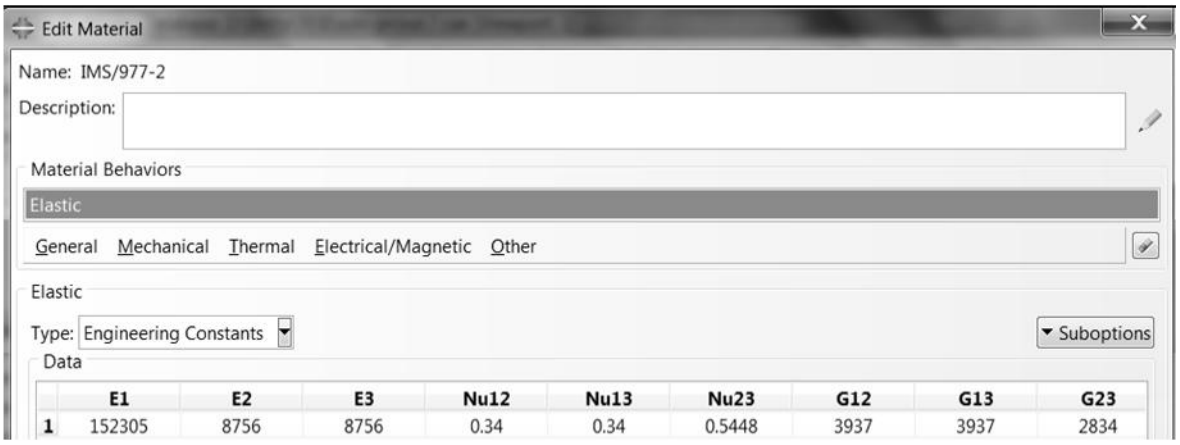


Fig 3.11 Creazione e caratterizzazione del materiale IMS/977-2

- un materiale metallico, la lega di Alluminio Al 2219-T87, a cui verrà assegnato comportamento elastico isotropo, tramite la scelta "*Elastic*" per "*Material Behaviors*" e "*Isotropic*" per "*Type*". Viene altresì definita la densità, con distribuzione uniforme [Fig 3.12]

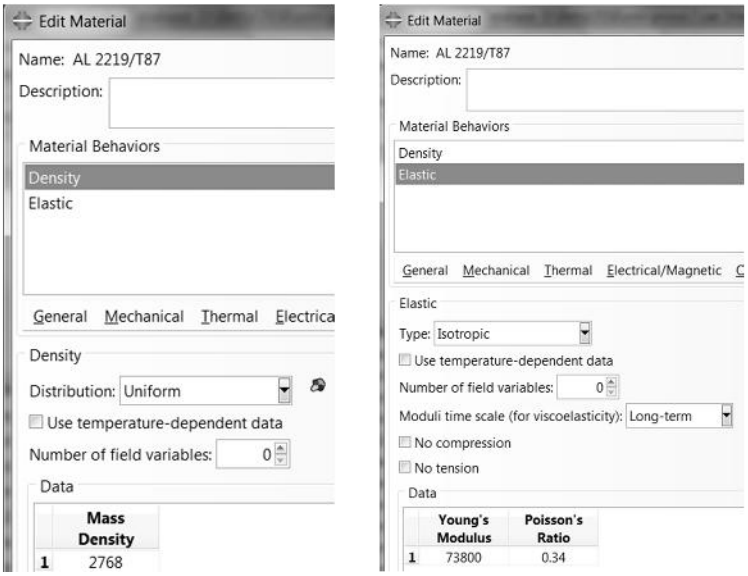


Fig 3.12 Creazione e caratterizzazione del materiale Al 2219-T87

3.3.2 Assegnazione delle Sezioni

Anche per ciò che riguarda le sezioni, la procedura seguita per caratterizzare le parti in composito, cioè skin, stringer e rib, è differente da quella utilizzata per caratterizzare le parti in materiale metallico, cioè le piastre di riparazione.

Iniziando dalle parti in composito, bisogna in primo luogo definire un sistema di riferimento relativo, il cui asse z coincida con la direzione di impilaggio delle singole lamine del laminato. In particolare, per le diverse parti, si definisce:

- skin, un solo sistema di riferimento relativo per la base, che risulta coincidere con il sistema di riferimento assoluto [Fig 3.13 a]
- stringer, due diversi sistemi di riferimento [Fig 3.13 b]:
    - un sistema di riferimento relativo per la base, che risulta coincidere con il sistema di riferimento assoluto della parte stringer
    - un sistema di riferimento relativo per il rinforzo, il cui asse z coincide con l'asse x del sdr assoluto della parte stringer

  Poichè i ply dello stringer hanno andamento simmetrico, la direzione di impilaggio non è di fondamentale importanza, purché si mantenga coerenza tra le due diverse parti del componente
- rib, due diversi sistemi di riferimento [Fig 3.13 c]:
    - un sistema di riferimento relativo per la base, il cui asse z coincide con l'asse y del sdr assoluto della parte rib, ma ha verso opposto. Infatti nelle indicazioni sull'impilamento dei ply si procede dall'alto verso il basso
    - un sistema di riferimento relativo per il rinforzo, il cui asse z coincide con l'asse x del sdr assoluto della parte rib

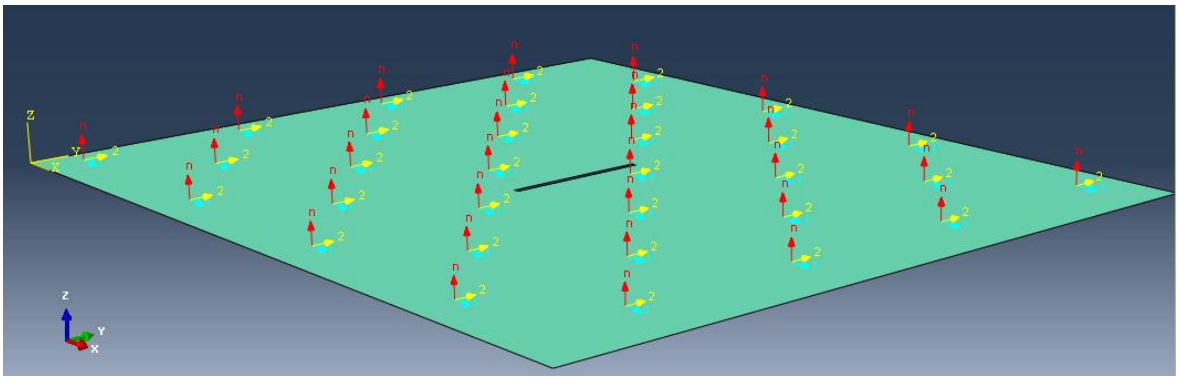


Fig 3.13 a): Sistema di riferimento relativo allo skin, con orientamento del materiale.

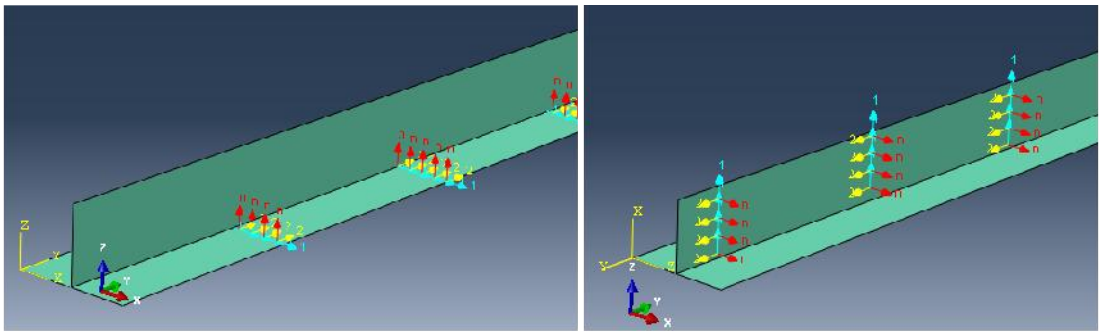


Fig 3.13 b): Sistemi di riferimento relativi allo stringer, con orientamento del materiale.

A sinistra il sdr della base, a destra il sdr del rinforzo

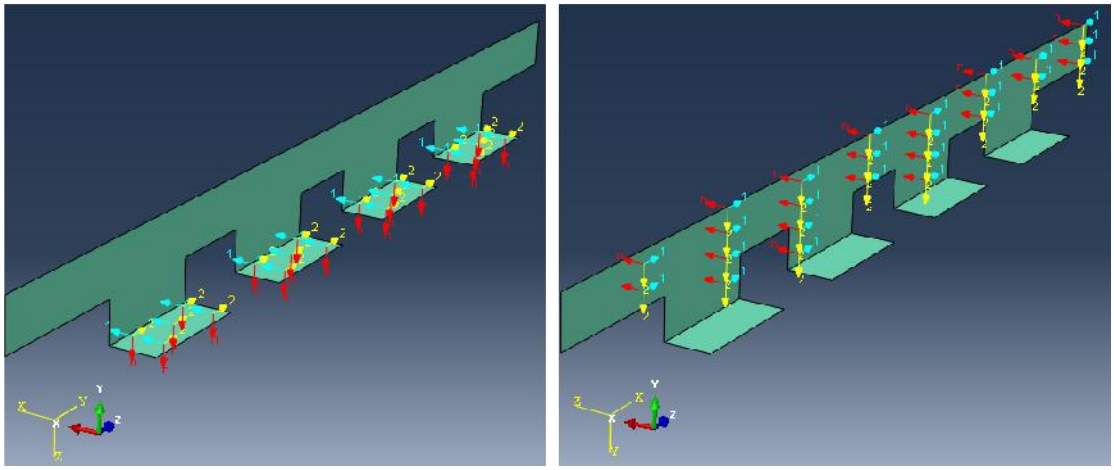


Fig 3.13 c): Sistemi di riferimento relativi al rib, con orientamento del materiale.

A sinistra il sdr della base, a destra il sdr del rinforzo

Si riporta, a titolo di esempio, il codice relativo alla creazione dei due sistemi di riferimento dello stringer:

```
#sdr layup stringer base
sdr_str_base=mdb.models['Model-1'].parts['stringer'].
DatumCsysByThreePoints(coordSysType=
    CARTESIAN,name='Datum   csys-str-base',   origin=(0.0,   0.0,   0.0),
point1=(1.0, 0.0, 0.0), point2=(1.0, 1.0, 0.0))
#sdr layup stringer rinforzo
sdr_str_rinf=mdb.models['Model-1'].parts['stringer'].
DatumCsysByThreePoints(coordSysType=
    CARTESIAN,   name='Datum   csys-str-rinf',   origin=(0.0,   0.0,   0.0),
point1=(0.0,0.0, 1.0), point2=(0.0, -1.0, 1.0))
```

Definiti i sistemi di riferimento occorre creare un "*Composite Layup*", ovvero bisogna creare il laminato di materiale composito, come sovrapposizione di un numero variabile di strati a seconda della parte considerata [Fig 3.14]:

- 24 "ply", ovvero lamine, per lo skin [Fig 3.16 a]
- 12 "ply" per stringer e rib, distinguendo tra base e rinforzo

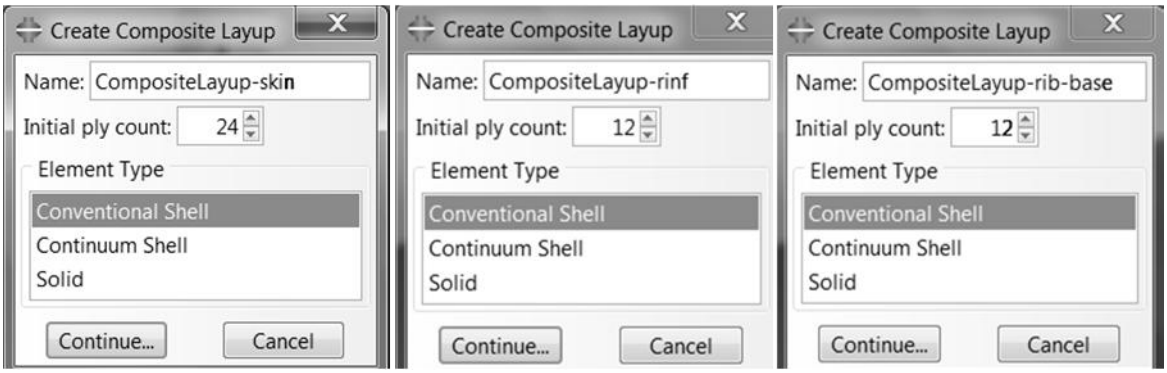


Fig 3.14 Creazione del laminato in composito per skin, rinforzo dello stringer, base del rib

Si utilizzano elementi di tipo "*Conventional Shell*", in accordo con quanto verrà fatto nella mesh. L'elemento è caratterizzato da una geometria planare, di riferimento, e uno spessore assegnato come proprietà. Si è prediletto questo al "*Continuum Shell*" per la possibilità di

utilizzare i vincoli di Fastener, di cui si parlerà in seguito, e in generale per ridurre i tempi di calcolo.

Il laminato viene identificato nel codice Python come un vettore, le cui componenti sono gli angoli di disposizione delle lamine rispetto al sistema di riferimento, adottato di volta in volta. Ad esempio per il layup del rib si scrive:

```
#Layup rib
layup_rib_or=((45,),(0,),(-45,),(90,),
              (45,),(0,),(-45,),(90,),
              (45,),(0,),(-45,),(90,),)
n_layup_rib=len(layup_str_or)
```

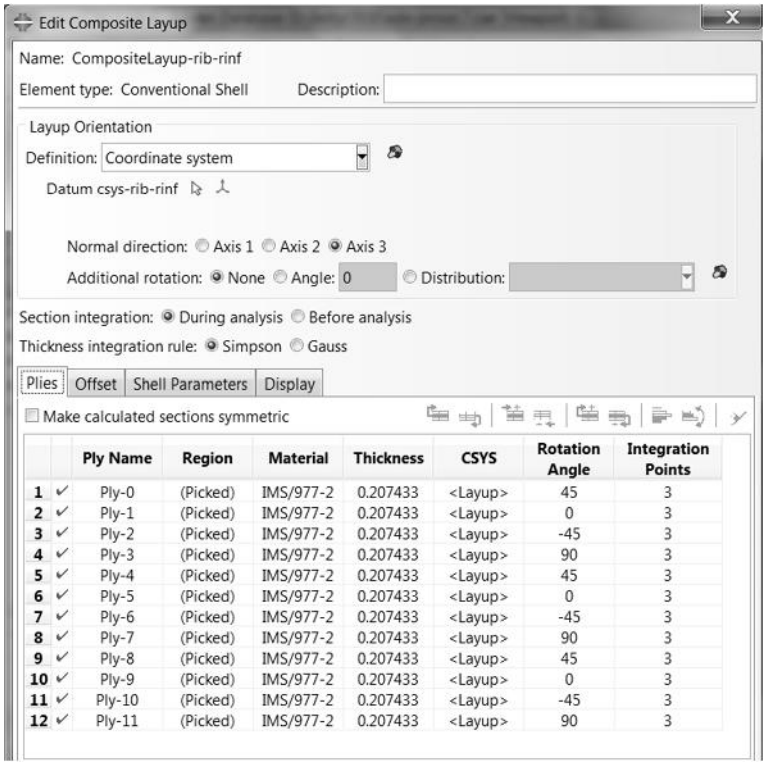
Definito tale vettore di supporto è sufficiente eseguire un ciclo "*for*" per un numero di volte pari alla lunghezza del vettore stesso, in cui si fissano le caratteristiche del laminato [Fig 3.15], ovvero:

- definizione di orientamento del layup in base al sistema di riferimento, tramite "*Coordinate system*"
- direzione normale del layup, che deve coincidere con la direzione z del sistema di riferimento relativo del laminato
- regione di interesse, selezionata tramite il comando "*FindAt*"
- materiale del laminato
- spessore del singolo strato, valutato come rapporto tra lo spessore della parte considerata e il numero di ply del laminato
- angolo di inclinazione della direzione principale della lamina, rispetto alla direzione principale del laminato

```
#Caratteristiche layup rib base
mdb.models['Model-1'].parts['rib'].CompositeLayup(description='',
elementType=SHELL,                          name='CompositeLayup-rib-base',
offsetType=MIDDLE_SURFACE, symmetric=False,
    thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['rib'].compositeLayups['CompositeLayup-rib-
base'].Section(integrationRule=SIMPSON,poissonDefinition=DEFAULT,
preIntegrate=OFF,       temperature=GRADIENT,       thicknessType=UNIFORM,
useDensity=OFF)
```

```
mdb.models['Model-1'].parts['rib'].compositeLayups['CompositeLayup-rib-
base'].ReferenceOrientation(
    additionalRotationField='',     additionalRotationType=ROTATION_NONE,
angle=0.0, axis=AXIS_3, fieldName='', localCsys=
    mdb.models['Model-1'].parts['rib'].datums[sdr_rib_base.id],
orientationType=SYSTEM, stackDirection=STACK_3)

#Orientamento layup rib base
for i in range(n_layup_rib):
    for j in range (1,n_stringer):
        mdb.models['Model-1'].parts['rib'].
compositeLayups['CompositeLayup-rib-base'].
CompositePly(additionalRotationField='',
additionalRotationType=ROTATION_NONE, angle=0.0
        ,     axis=AXIS_3,     material='IMS/977-2',     numIntPoints=3,
orientationType=SPECIFY_ORIENT,      orientationValue=layup_rib_or[i][0],
plyName='Ply'+str(i)+str(j), region=Region(
        faces=mdb.models['Model-
1'].parts['rib'].faces.findAt(((rib_width/2,0,j*(rib_cut_lenght+rib_base)
),),     )),     suppressed=False,     thickness=rib_depth/n_layup_rib,
thicknessType=SPECIFY_THICKNESS)
```



	Ply Name	Region	Material	Thickness	CSYS	Rotation Angle	Integration Points
1	Ply-0	(Picked)	IMS/977-2	0.207433	<Layup>	45	3
2	Ply-1	(Picked)	IMS/977-2	0.207433	<Layup>	0	3
3	Ply-2	(Picked)	IMS/977-2	0.207433	<Layup>	-45	3
4	Ply-3	(Picked)	IMS/977-2	0.207433	<Layup>	90	3
5	Ply-4	(Picked)	IMS/977-2	0.207433	<Layup>	45	3
6	Ply-5	(Picked)	IMS/977-2	0.207433	<Layup>	0	3
7	Ply-6	(Picked)	IMS/977-2	0.207433	<Layup>	-45	3
8	Ply-7	(Picked)	IMS/977-2	0.207433	<Layup>	90	3
9	Ply-8	(Picked)	IMS/977-2	0.207433	<Layup>	45	3
10	Ply-9	(Picked)	IMS/977-2	0.207433	<Layup>	0	3
11	Ply-10	(Picked)	IMS/977-2	0.207433	<Layup>	-45	3
12	Ply-11	(Picked)	IMS/977-2	0.207433	<Layup>	90	3

Fig 3.15 Definizione caratteristiche del laminato per il rinforzo del rib

La procedura seguita è la stessa per tutte la parti, con alcune piccole differenze per:

- rinforzo dello stringer, avendo un doppio strato di materiale composito, è caratterizzato dalla ripetizione del comando "*CompositePly*" per due volte consecutive [Fig 3.16 b]
- base del rib, in cui ogni piedino è selezionato tramite il comando "*FindAt*" singolarmente, per garantire la possibilità di variare il numero di stringers, dunque di piedini del rib, senza incorrere in errori di definizione del materiale [Fig 3.16 c]

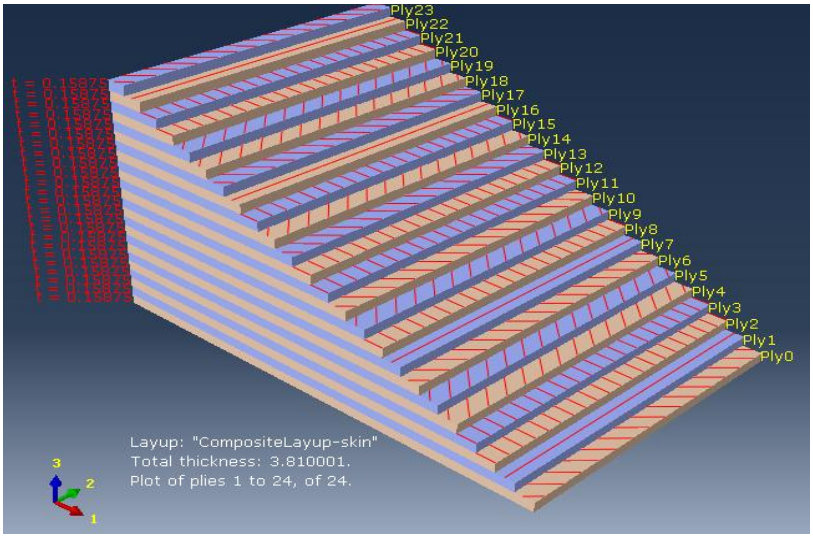


Fig 3.16 a) Visualizzazione dei ply del laminato per lo skin

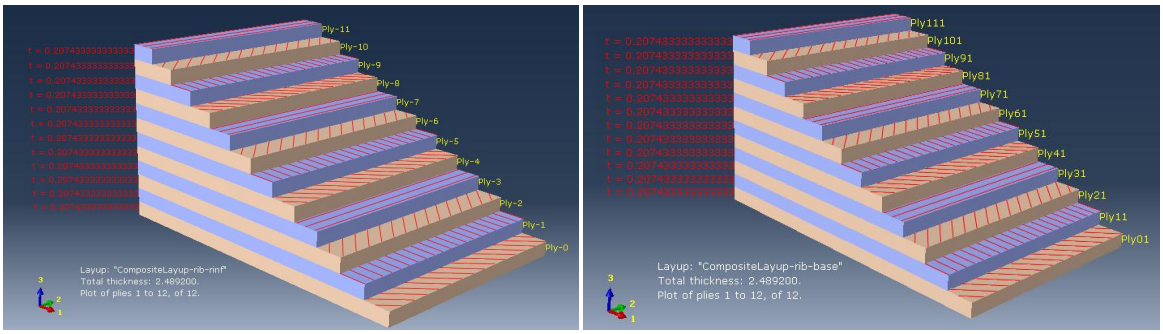


Fig 3.16 b) Visualizzazione dei ply del laminato
per il rinforzo del rib a sinistra, per la base del rib a destra

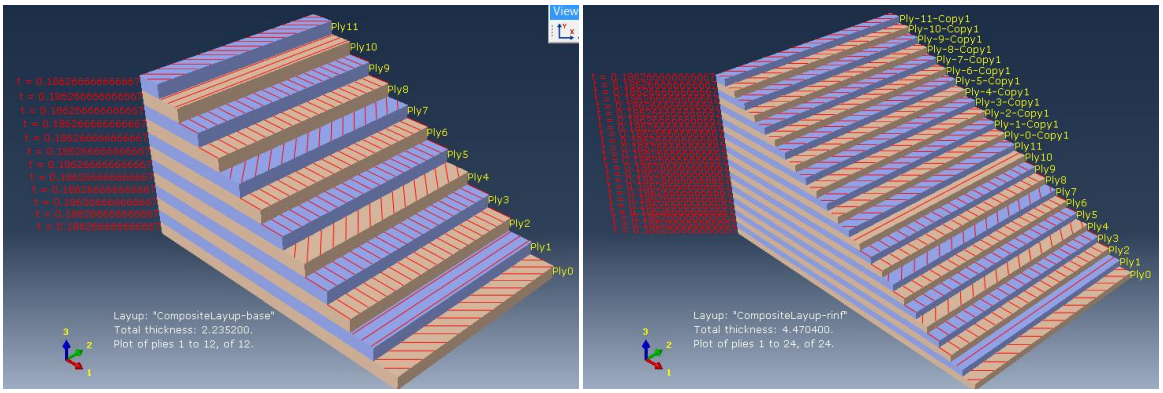


Fig 3.16 c) Visualizzazione dei ply del laminato
per la base dello stringer a sinistra, per il rinforzo dello stringer a destra

Per ciò che riguarda i materiali metallici la procedura di caratterizzazione è molto più semplice: è sufficiente generare una sezione di tipo "*Shell Homogeneous*", a cui assegnare lo spessore rip_depth1 e rip_depth2 rispettivamente e il materiale Al2219/T87. Tale sezione è poi assegnata alla parte di riferimento [Fig 3.17].

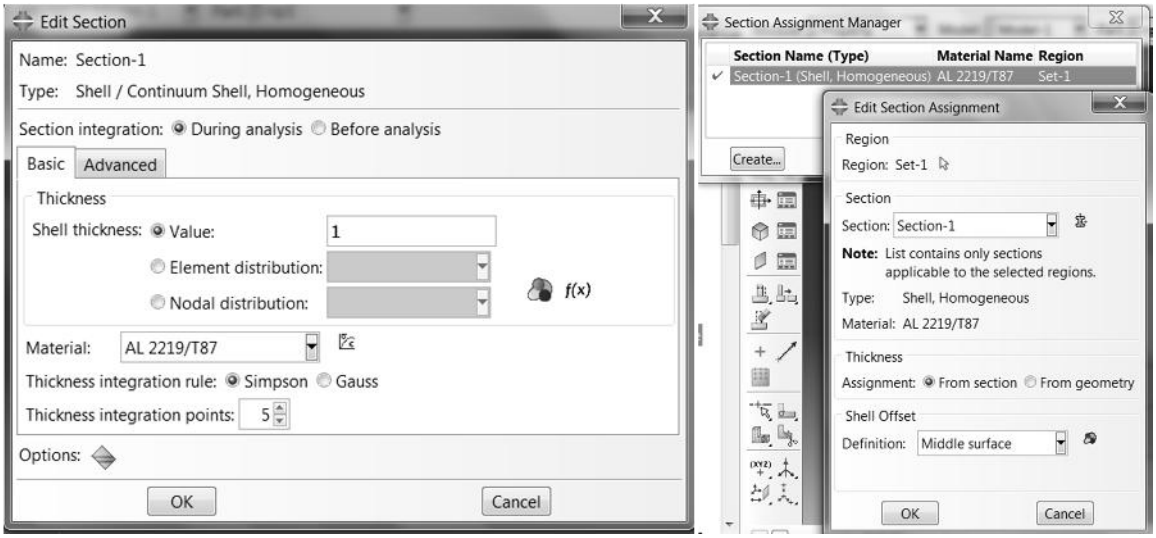


Fig 3.17 Creazione della sezione per i materiali metallici e
assegnazione della stessa alla parte rip1

3.4 Creazione della mesh

3.4.1 Skin

Per tutte le parti dell'assieme si è utilizzato l'elemento shell S4R, elemento a 4 nodi, con 6 gradi di libertà per nodo, di utilizzo per problemi generici [Fig 3.18]. Nella risoluzione delle equazioni di equilibrio e di continuità utilizza il metodo di integrazione ridotta, con controllo del fenomeno dell'hourglassing (problematica subdola dell'analisi FEM, che comporta risultati inattendibili); è adatto anche per grandi deformazioni.

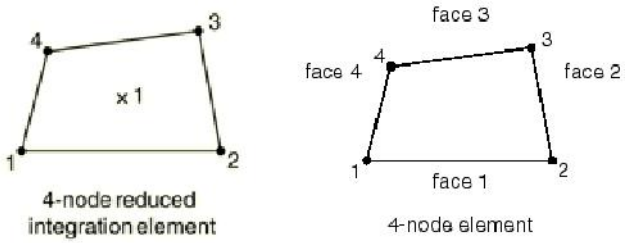


Fig 3.18 Rappresentazione di S4R, con in evidenza
il punto di integrazione e la numerazione dei lati

Definito l'elemento si procede a caratterizzare la mesh nell'intorno della cricca, essendo tale zona quella di maggior interesse dell'intero assieme. Per tale ragione tramite i "*local seeds*", ovvero il dimensionamento locale, si fissa una dimensione degli elementi pari ad un decimo di quella utilizzata complessivamente [Fig 3.19]

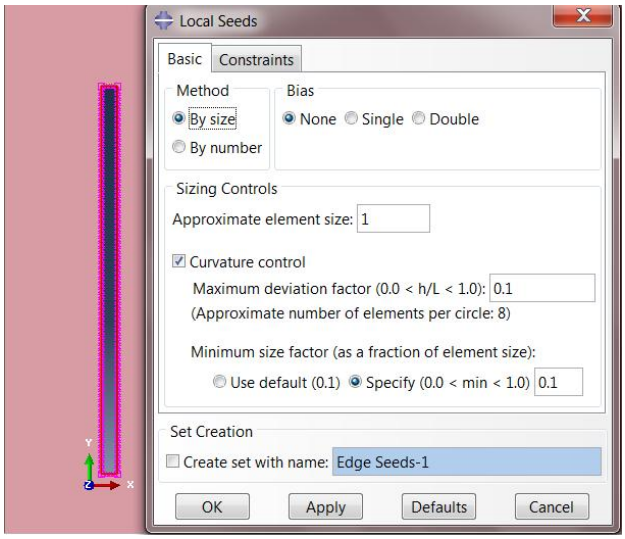


Fig 3.19 Particolare della cricca e
assegnazione di dimensioni locali di mesh

Definite tale dimensioni si procede alla creazione di una serie di partizioni di supporto per la mesh:

- una prima partizione rappresentante il profilo delle piastre di riparazione sullo skin; la partizione è a sua volta circondata da un secondo profilo, ottenuto semplicemente ingrandendo il primo
- una seconda partizione, in corrispondenza della posizione dei bulloni

La prima partizione è stata introdotta per garantire, almeno in parte, la corrispondenza tra i nodi dello skin e i nodi delle piastre di riparazione. Si ottiene tramite lo sketch di due rettangoli, il primo di dimensioni rip_lenght1 in direzione x e rip_widht1+str_lenght in direzione y e il secondo di dimensioni x_1*rip_lenght1 in direzione x e x_1*rip_widht1+str_lenght in direzione y. L'utilità di questo sketch esterno risiede nella possibilità di delimitare la zona di transizione della mesh ad una regione di spazio ben definita e garantire la possibilità di una mesh strutturata lontano dalla cricca [Fig 3.24].

Si osservi che, essendo la larghezza della piastra di riparazione rip_widht1 indipendente dalla posizione della cricca, la partizione viene costruita di modo da essere simmetrica

rispetto al centro dello stringer intaccato dalla cricca, piuttosto che dal centro della cricca stessa. Tali considerazioni non sono valide in direzione longitudinale, dove pertanto è rispettata la simmetria rispetto alla cricca [Fig 3.20].

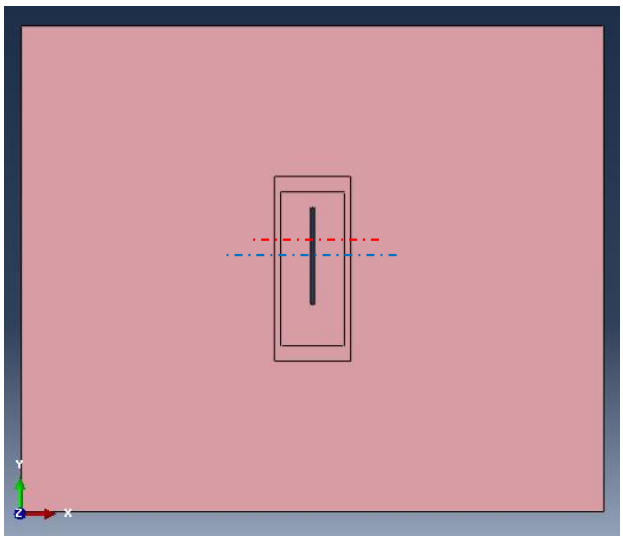


Fig 3.20 Assi di simmetria della riparazione in blu e assi di simmetria della cricca in rosso

```
#Partizione skin zona riparazione
## dimensioni zona di transizione
###(x1=fattore moltiplicativo rispetto alla zona interna)
x1=1.2
for i in range (n_cr):
    [...]
      origin=(x_cr[i]+cr_lenght[i]/2,
      str_cr_cr[i][0]*distance_stringer+offset_stringer/2, 0)))
    mdb.models['Model-  1'].parts['skin'].projectReferencesOntoSketch(
      filter=
    COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])
    mdb.models['Model-1'].sketches['__profile__'].
      rectangle(point1=(-     rip_lenght1/2,-rip_width1-str_lenght/2),
                point2=(rip_lenght1/2,rip_width1+str_lenght/2))
    mdb.models['Model-1'].sketches['__profile__'].
      rectangle(point1=(-x1*rip_lenght1/2,-x1*(rip_width1+str_lenght/2)),
                point2=(x1*rip_lenght1/2,x1*(rip_width1+str_lenght/2)))
    mdb.models['Model-1'].parts['skin'].PartitionFaceBySketch(faces=
    mdb.models['Model-1'].parts['skin'].faces.findAt((skin_lenght/100,
      skin_width/100, 0),),
    sketch=mdb.models['Model-1'].sketches['__profile__'], sketchUpEdge=
```

```
mdb.models['Model-1'].parts['skin'].edges.findAt((skin_lenght,
  skin_width/100, 0),))
del mdb.models['Model-1'].sketches['__profile__']
```

La seconda partizione invece è stata introdotta per il modo in cui è stata simulata la presenza dei bulloni nel corso dell'analisi: come verrà approfondito meglio in seguito, si sono utilizzati i "*fastener*", degli elementi di connessione indipendenti dalla mesh. Essi considerano come zona interessata dal collegamento quella delimitata dai punti della mesh disposti entro un certo raggio dal centro della connessione stessa [Fig 3.21].

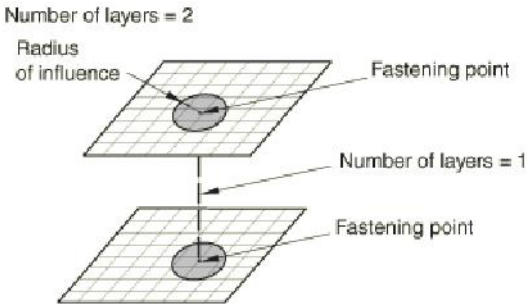


Fig 3.21 Modo di funzionamento dei fastener

Per far si che il centro del fastener coincidesse con un nodo della mesh e che il raggio di influenza fosse limitato a nodi effettivamente disposti in modo circolare rispetto al centro, si è creata una partizione, composta da una serie di cerchi [Fig 3.22].

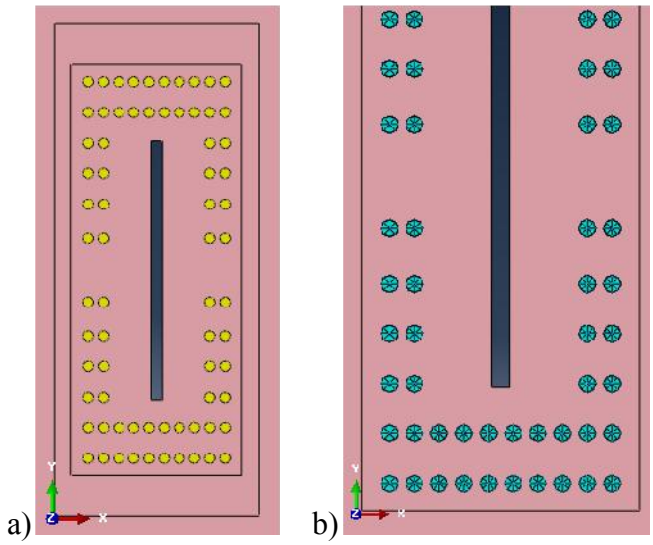


Fig 3.22 a) Particolare delle partizioni dello skin intorno alla cricca

b) Particolare della tecnica di mesh swept

Tale strategia ha però senso solo se nelle zone delimitate dalle circonferenze si genera la mesh tramite la tecnica "*sweep*" (colore giallo in [Fig 3.22 a]), che consente di far coincidere il centro della circonferenza con un nodo ben preciso [Fig 3.22 b]. Infatti la tecnica sweep genera i nodi procedendo lungo percorsi successivi: i nodi generati sul primo percorso saranno copiati anche nei percorsi successivi, sino a completare la superficie.

Nel complesso la seconda partizione garantisce un incremento di stress nell'intorno dei punti di Fastening più netto, di quanto non avvenga in assenza di partizione.

Nel resto del pannello si procede assegnando come tecnica di mesh la tecnica "*Free*", che garantisce la massima libertà possibile. Invece per ciò che riguarda le dimensioni locali degli elementi e gli algoritmi di generazione [Fig 3.23] si sono utilizzate le seguenti direttive:

1. zona lontana dalla cricca, Global Seed pari a 10 [mm], utilizzo di elementi Quadrilateri, algoritmo "*Medial Axis*", con minimizzazione della transizione di mesh
2. zona di transizione tra la zona esterna e la prima partizione, Local Seed che variano da 10 a 5 [mm], elementi "*Quad Dominated*", algoritmo "*Advancing Front*"
3. zona rettangolare interna della prima transizione, Local Seed pari a 5 [mm], elementi "*Quad Dominated*", algoritmo "*Advancing Front*"
4. seconda zona di partizione, Local Seed sulle circonferenze pari a 5 [mm]

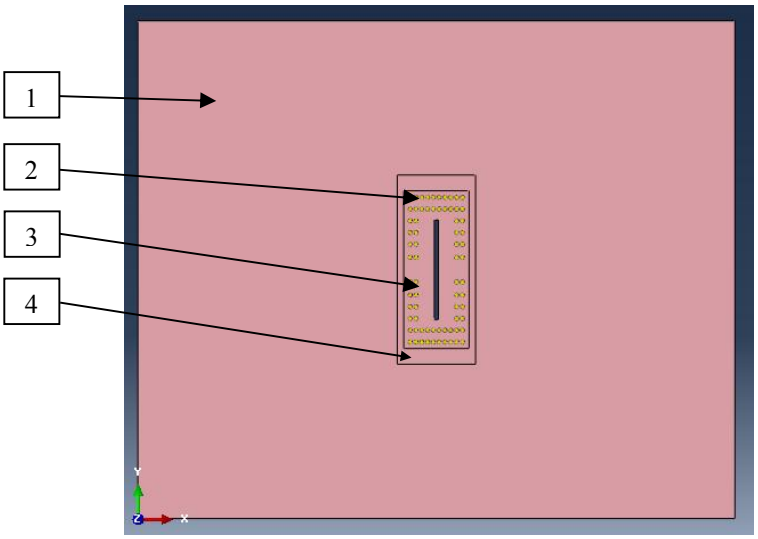


Fig 3.23 Tecniche di mesh e dimensioni utilizzate nello skin

Si osservi che l'algoritmo Medial Axis opera suddividendo la zona di interesse in microzone regolari e poi riempie ognuna di queste regioni con gli elementi di mesh. Si tratta del metodo più veloce per superfici a geometria regolare.

Invece l'algoritmo Advancing Front si presta meglio a regioni di spazio dalla geometria più complessa, poiché opera dal bordo della superficie procedendo verso l'interno. Garantisce un migliore soddisfacimento dei singoli criteri imposti sulle dimensioni.

Nel complesso, nel caso specifico, si ottiene una mesh strutturata solo nella zona esterna, lontana dalla cricca, mentre nella zona di maggiore interesse la mesh è non strutturata e infittita [Fig 3.24].

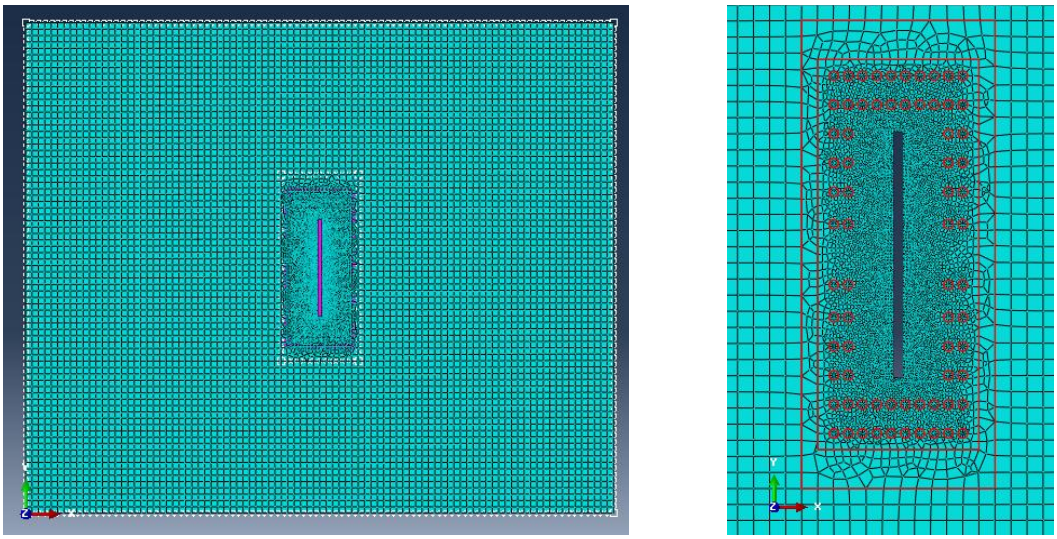


Fig 3.24 Complessivo Mesh dello skin e
particolare della zona criccata con le partizioni indicate in rosso

3.4.2 Stringer e Rib

Sia per gli stringer non danneggiati che per i rib l'esecuzione della mesh è meno complessa di quanto non sia per le altre parti: è sufficiente assegnare il canonico elemento S4R, Global Seed pari a 10 [mm] e tecnica di mesh dominata da elementi quadrilateri, con algoritmo Medial Axis [Fig 3.25].

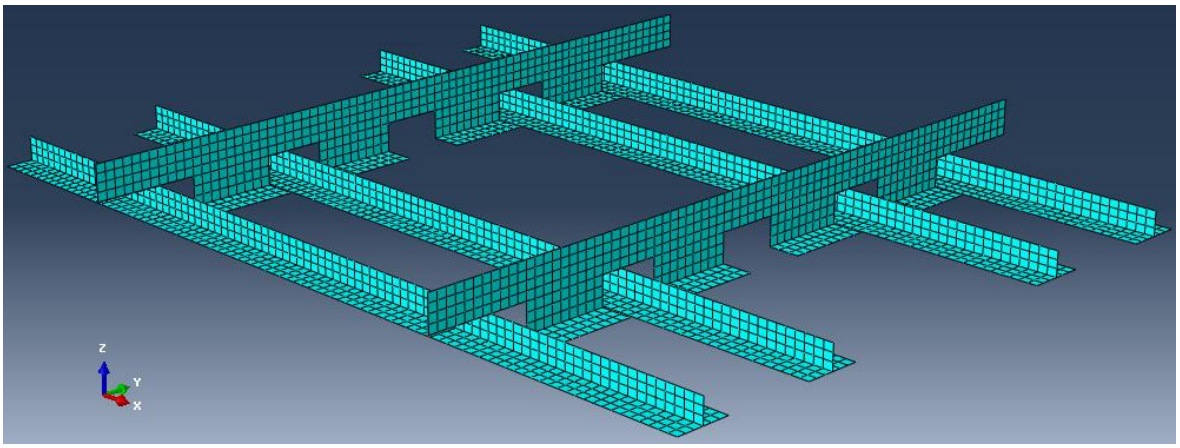


Fig 3.25 Assieme dei rib e degli stringer non danneggiati con rappresentazione della mesh

3.4.3 Stringer criccati

Si procede in analogia a quanto fatto per lo skin, tramite la creazione di due differenti partizioni, una in corrispondenza della piastra di riparazione e la seconda in corrispondenza dei fastener [Fig 3.26].

Vi sono però alcune differenze, la prima delle quali consiste nel fatto che la prima partizione si ottiene creando un piano di riferimento, o "*datum plane*", in corrispondenza del quale effettuare la suddivisione della parte:

```
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
#creazione piani offset per la partizione
        xz_strR=mdb.models['Model-1'].parts['stringer-
       cricca'+str(k)].DatumPlaneByOffset(flip=SIDE1,
        offset=x_cr[int(j+a)]+cr_lenght[int(j+a)]/2+rip_lenght1/2,
        plane=mdb.models['Model-1'].parts['stringer-
       cricca'+str(k)].datums[xz_str.id])

        xz_strL=mdb.models['Model-1'].parts['stringer-
       cricca'+str(k)].DatumPlaneByOffset(flip=SIDE1,
        offset=x_cr[int(j+a)]+cr_lenght[int(j+a)]/2-rip_lenght1/2,
        plane=mdb.models['Model-1'].parts['stringer-
       cricca'+str(k)].datums[xz_str.id])
#partizione stringer cricca in corrispondenza della riparazione
        mdb.models['Model-1'].parts['stringer-
       cricca'+str(k)].PartitionFaceByDatumPlane(
        datumPlane=mdb.models['Model-1'].parts['stringer-
       cricca'+str(k)].datums[xz_strL.id],
        faces=mdb.models['Model-1'].parts[...]
    a=nsup[k-1]
```

Nel codice riportato si può notare l'utilizzo di una serie di cicli "*for*" annidati, il cui scopo è quello di garantire il funzionamento anche in presenza di più cricche sullo stesso stringer:

```
#creazione vettore di supporto nel caso di piu cricche
a=[]
nsup=[]

for j in range (n_str_cr+1):
    nsup.append(0)
for j in range (n_str_cr):
    for i in range (len(str_cr_cr)):
        a.append(str_cr_cr[i].count(str_cr[j]))

          nsup[j]=sum(a)-nsup[j-1]
```

Il vettore "*nsup*" rappresenta il numero di cricche presenti in ogni stringer danneggiato; se ad esempio lo stringer 5 ha due cricche e lo stringer 4 ne ha una, il vettore nsup assume la forma:

```
nsup=[1,2,0]
```

ovvero la procedura di partizione è eseguita una volta sullo stringer 4 e due volta sullo stringer 5. Tale notazione sarà utilizzata spesso anche nel seguito.

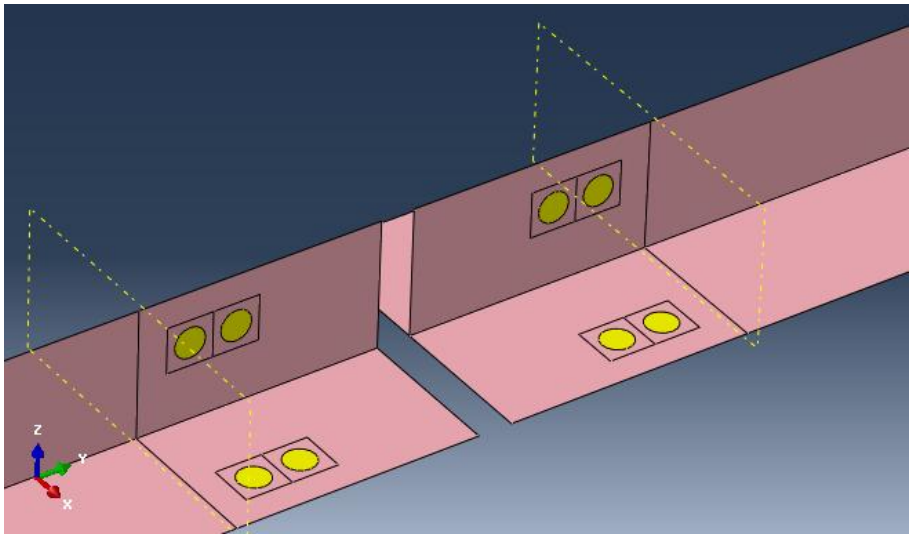


Fig 3.26 Assieme delle partizioni sullo stringer criccato
in corrispondenza della cricca

La seconda differenza risiede nell'utilizzo di una zona di transizione nel passare dalla zona circolare, rappresentante il fastener, a quella esterna: tale scelta è dettata dalla selezione di un algoritmo "*Medial Axis*", accoppiato ad un infittimento degli elementi, mano a mano che ci si avvicina alla cricca. In pratica si è utilizzato il comando "*Local Seed*", assegnando una direzione di Bias, che indica la direzione di infittimento [Fig 3.27].

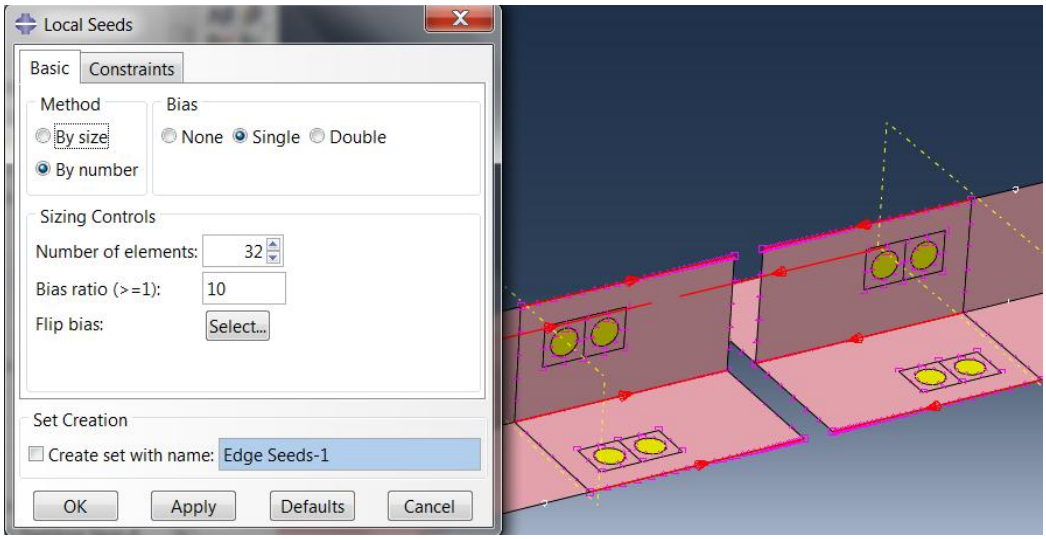
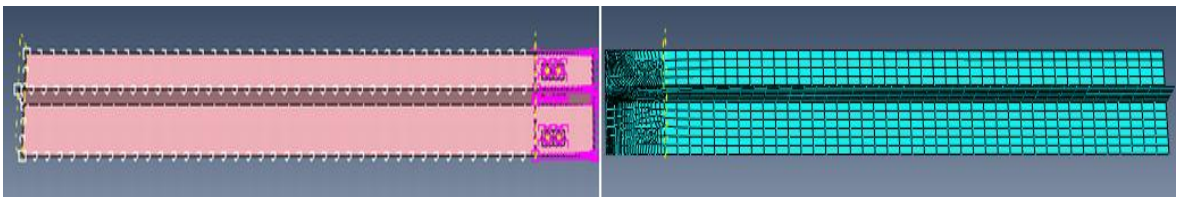


Fig 3.27 Particolare del comando "SeedEdgeByBias",
con indicazione della direzione di infittimento tramite frecce rosse

Nel complesso le tecniche e le dimensioni adottate per lo stinger criccato sono le seguenti [Fig. 3.28]:

- Global Seed pari a 10 [mm]
- elementi quadrilateri e algoritmo Medial Axis per la zona lontana dalla cricca
- Local Seed in corrispondenza della cricca 5 [mm]
- tecnica Sweep per i cerchi dei fastener
- algoritmo Advancing Front per la zona di transizione tra cerchi e partizione in corrispondenza della cricca
- Local Seed con direzionalità per i lati in direzione y della zona di partizione
- elementi quadrilateri dominanti e algoritmo Medial Axis per la prima partizione



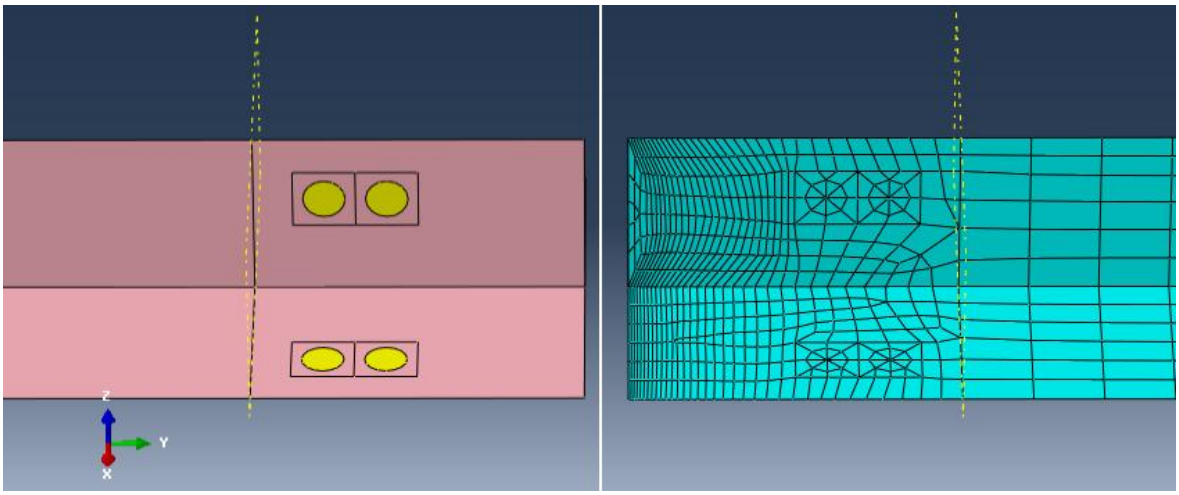


Fig 3.28 Complessivo e particolare della mesh dello stringer danneggiato

3.4.4 Piastre di riparazione

La procedura è simile a quella delle parti che vanno a riparare:

- rip1 [Fig 3.29]:
  - creazione delle circonferenze dei fastener e tecnica di mesh Sweep per le stesse
  - elementi quadrilateri dominanti, Global Seed pari a 5 [mm], algoritmo Advanced Front nel complesso

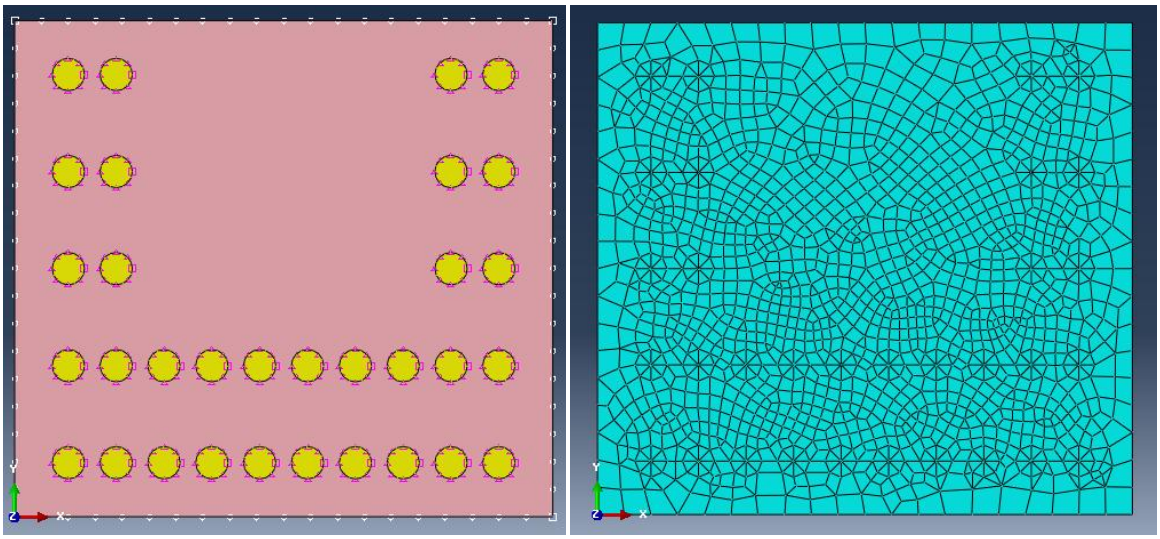


Fig 3.29 Mesh sulla piastra di riparazione dello skin

- rip2 [Fig 3.30]:
  - creazione delle circonferenze dei fastener e tecnica di mesh Sweep per le stesse
  - creazione dei rettangoli di transizione attorno alle circonferenze e utilizzo dell'algoritmo Advanced Front
  - elementi quadrilateri dominanti, Global Seed pari a 5 [mm], algoritmo Advanced Front nel complesso

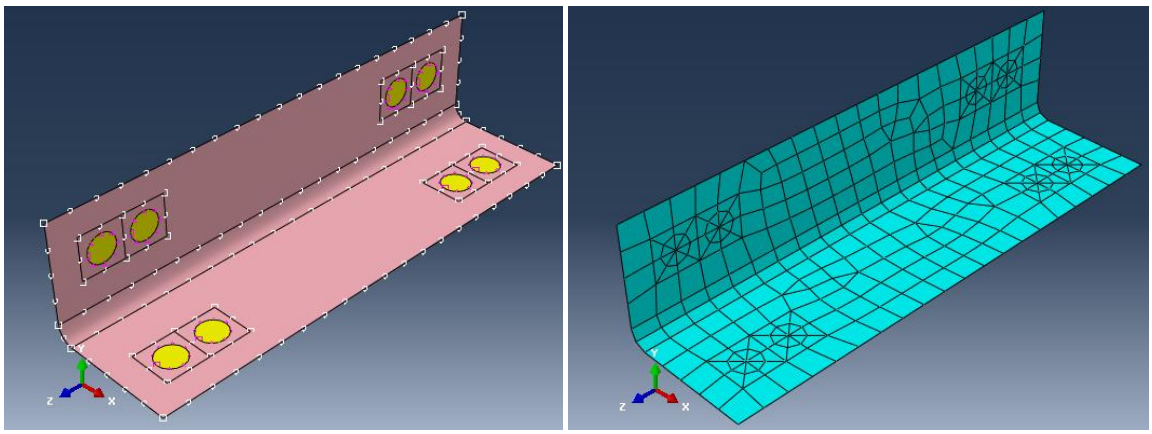


Fig 3.30 Mesh sulla piastra di riparazione dello stringer

3.5 Creazione dell'Assembly

Nella creazione dell'assieme [Fig 3.33] si è mantenuta l'ipotesi di sistema di riferimento globale coincidente con il sistema di riferimento dello skin, che pertanto è stata la prima istanza ad essere inserita nell'assembly [Fig 3.31]. Le altre sono state introdotte distanziandole dallo skin stesso di una quantità pari a lunghezza dello skin per "n", dove n rappresenta il numero di istanze introdotte precedentemente.

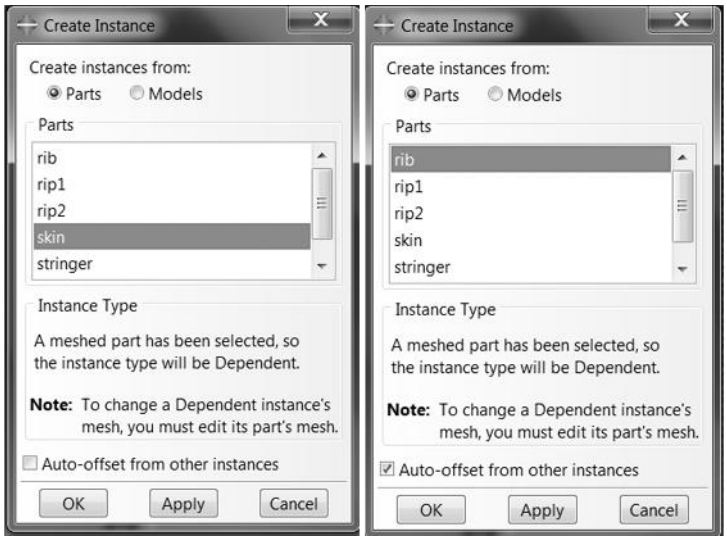


Fig 3.31 Creazione degli elementi che compongono l'assieme

Successivamente si sono fissati una serie di vincoli geometrici, che posizionassero le varie istanze nelle posizioni adeguate:

- contatto tra la superficie media di base dello skin e la superficie media di base delle altre istanze, tramite il comando "*FacetoFace*" con "*clearance*" variabile caso per caso: per coerenza si è fissato ogni volta un offset che tenesse conto degli spessori di ogni elemento [Fig 3.32]

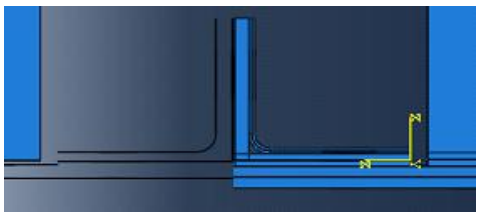


Fig 3.32 Visualizzazione dello spessore negli elementi shell

- offset in direzione x e in direzione y di ogni istanza rispetto al sistema di riferimento globale. Per facilitare tale procedura, nella definizione della parti, si sono introdotti una serie di "*datum plane*", corrispondenti ai piani del sistema di riferimento relativo alla parte:

```
#Creazione piani di supporto per i vincoli
xz_str=mdb.models['Model-
1'].parts['stringer'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=XZPLANE)
yz_str=mdb.models['Model-
1'].parts['stringer'].DatumPlaneByPrincipalPlane(offset=str_lenght/2,
   principalPlane=YZPLANE)
```

  Il vincolo di contatto tra le superfici è stato dunque imposto tra tali piani di supporto.

Un'attenzione particolare è stata posta al posizionamento delle piastre di riparazione, essendo disposte in modo antisimmetrico rispetto al centro dello stringer di riferimento.

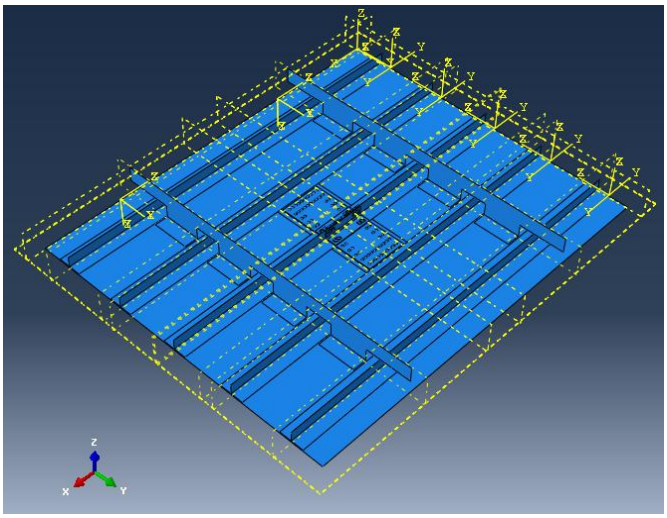


Fig 3.33 Assembly con le posizioni corrette per ogni istanza

3.6 Interazioni tra istanze

3.6.1 Vincolo Tie

Tra le singole componenti dell'assieme, escluse le riparazioni, si sono imposti dei vincoli di tipo "*Tie*", ovvero incastro. Tale tipo di vincolo lega due superfici durante la simulazione, imponendo che i nodi della superficie "*slave*", letteralmente "servo", si muovano come i nodi della superficie "*master*", letteralmente "padrone". La scelta della superficie master è vincolata volta per volta da una serie di valutazioni, in questo ordine, su:

1. grandezza di una superficie rispetto all'altra
2. rigidezza di una superficie rispetto all'altra
3. dimensioni di mesh

In genere la superficie più grande, più rigida o con mesh più grossolana è scelta come superficie master. Nel caso specifico la scelta della superficie dello skin come superficie master risulta immediata.

Vi sono due fattori di cui tenere conto per applicare correttamente il vincolo nel caso specifico:

- le varie partizioni che sono state eseguite nel corso della mesh fanno si che si renda necessaria l'unione di tutte le singole superfici della parte in considerazione in un'unica superficie di riferimento. Per farlo si esegue l'operazione Booleana di unione tra superfici:

```
mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf-1',
side1Faces=
    mdb.models['Model-1'].rootAssembly.instances['skin-
1'].faces.findAt(
    ((skin_lenght/100,skin_width/100,0),), ))
for i in range (n_cr):
    mdb.models['Model-
1'].rootAssembly.Surface(name='m_Surf'+str(i+1), side1Faces=
    mdb.models['Model-1'].rootAssembly.instances['skin-
1'].faces.findAt(
    ((x_cr[i]-cr_lenght[i]/100,y_cr[i]+0.99*cr_width[i]/2,0),), ))
    mdb.models['Model-
1'].rootAssembly.SurfaceByBoolean(name='m_Surf-1', surfaces=(
    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'],
 mdb.models['Model-1'].rootAssembly.surfaces['m_Surf'+str(i+1)]))
```

- l'utilizzo degli elementi shell non consente di visualizzare a priori quale sia la superficie superiore e quale quella inferiore, ma mostra, e riconosce, solo la superficie media. L'interfaccia grafica di Abaqus consente tale distinzione tramite colorazioni differenti (viola o marrone, [Fig 3.34]), mentre nella stesura del codice bisogna distinguere tra "*side1Faces*" e "*side2Faces*" a seconda dei casi
- per lo stringer criccato, essendo esso diviso in due parti differenti, che in simulazione devono mantenere la loro indipendenza, sono necessari più tie. Per una cricca vi saranno due parti separate sullo stringer e dunque due tie, per 2 cricche vi saranno tre parti separate e dunque tre tie

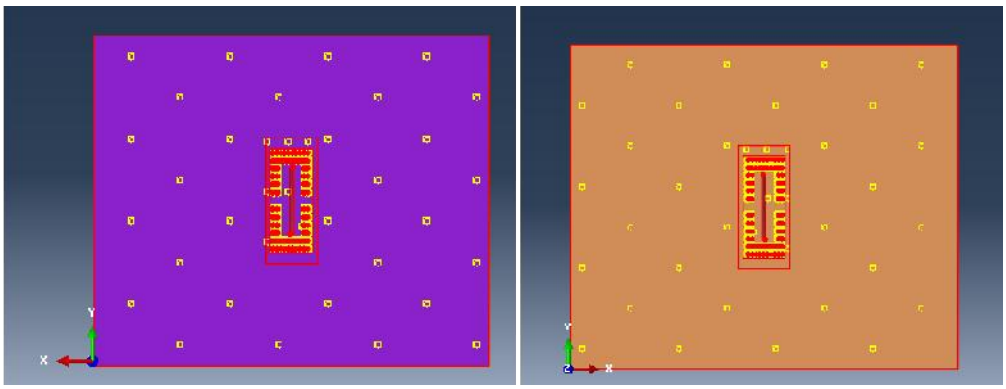


Fig 3.34 Selezione della superficie interna

3.6.2 Vincolo di contatto Surface-to-Surface

Tra le piastre di riparazione e il pannello da riparare si utilizzano due differenti tipi di interazioni: il vincolo di contatto tra le superfici e i fastener in specifici punti.

Per ciò che riguarda il vincolo di contatto è necessario caratterizzare lo stesso tramite apposite proprietà [Fig 3.35].

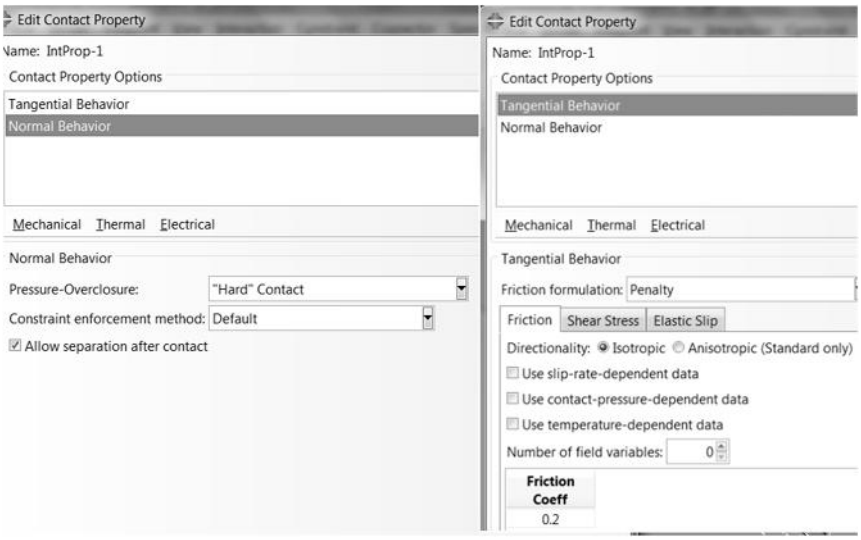


Fig 3.35 Caratterizzazione della relazione di contatto

Si sceglie un contatto di tipo superficie/superficie perché più accurato rispetto al contatto nodo/superficie in condizioni standard, cioè di punti sufficientemente lontani dai bordi o da altre zone di singolarità.

L'algoritmo di scorrimento è di tipo "*finite-sliding*", computazionalmente più pesante dell'algoritmo "*small-sliding*", in quanto tiene conto anche di grandi spostamenti reciproci tra le superfici.

Dopodiché si definisce il comportamento in direzione normale, caratterizzando la pressione di chiusura come "*Hard Contact*", il cui comportamento è visibile in figura [Fig 3.36]: la pressione viene scambiata tra le due superfici solo se sono in contatto, ovvero con distanza inferiore a zero.

Il comportamento in direzione tangenziale è invece caratterizzato dall'uso di un metodo alle penalità, che consente il moto relativo delle superfici quando le stesse sono in condizioni di aderenza, ma lo limita tramite il coefficiente di aderenza. L'algoritmo di risoluzione varia continuativamente il valore del vincolo di penalità per rappresentare al meglio questa condizione.

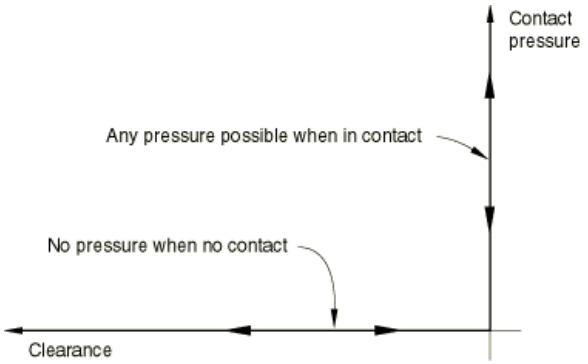


Fig 3.36 Andamento della pressione di chiusura in direzione normale

Definite le condizioni di contatto si selezionano le superfici interessate [Tab 3.4]:

Superficie Master	*Superficie Slave*
Skin	Rip1
Stringer criccato-base	Rip2-base
Stringer criccato-rinforzo	Rip2-rinforzo

Tab 3.4 Superfici master e slave del vincolo di contatto

3.6.3 Fastener

L'utilizzo dei fastener è dettato dalla volontà di riprodurre il comportamento dei bulloni, senza doverli modellare ex novo. Tale approccio consente di definire un nodo e un'area di interesse e in automatico il software simula il comportamento voluto.

Il fastener è infatti un conveniente metodo per definire la connessione punto a punto tra due o più superfici, indipendentemente dalla mesh. Ogni punto di fastening è connesso alla superficie utilizzando un vincolo di accoppiamento, che lega spostamenti e rotazioni del punto stesso agli spostamenti e alle rotazioni medie dei nodi nelle vicinanze.

I nodi delle superfici di interesse sono selezionati in modo automatico, definendo un raggio fisico: tutti i nodi la cui distanza dal fastening point è inferiore a tale raggio, sono considerati soggetti al vincolo di accoppiamento.

Come già accennato nell'analisi delle partizioni utilizzate per la mesh, la presenza di una mesh poco fitta attorno al punto centrale implica una distorsione del vincolo di fastener, nel senso che agisce su punti anche molto lontani dal centro. Viceversa una mesh troppo fitta comporta una intensificazione degli stress eccessiva.

Per caratterizzare i fastener si definiscono una serie di "*reference points*" sullo skin e sul rinforzo dello stringer, in numero variabile [Fig 3.37]. Il profilo della cricca viene distanziato di una certa quantità, sia in direzione x che in direzione y, per evitare il propagarsi della stessa.

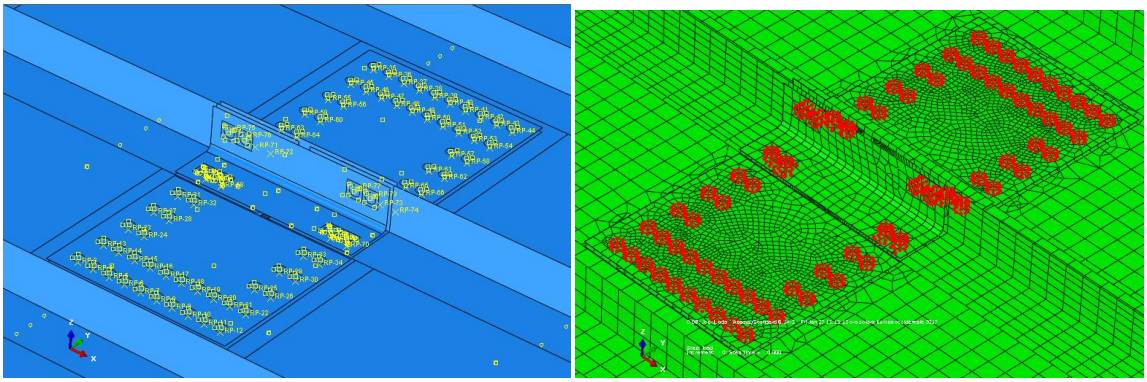


Fig 3.37 Reference Points e Cloud Nodes del vincolo di Fastener

Nel codice questa accortezza viene simulata tramite un controllo sulla posizione del singolo reference point:

```
#FASTENER
#Attachment Points
rp=mdb.models['Model-1'].rootAssembly.ReferencePoint(point=
                    (offset_edge1+x_cr[0]+cr_lenght[0]/2-rip_lenght1/2,
                      offset_point1+y_cr[0]+cr_width[0]/2-(rip_width1+str_lenght/2),
                      0))
mdb.models['Model-1'].rootAssembly.Set(name='Set-RP', referencePoints=(
    mdb.models['Model-1'].rootAssembly.referencePoints[rp.id],))

for i in range (n_cr):
    for k in range (n_points1):
        for q in range (n_rows1):
            if rip_lenght1/2-x_noHole/2 < offset_edge2+distance_rows2*q <
       rip_lenght1/2+x_noHole/2 and\
               rip_width1/2+str_lenght/2-
       y_noHole/2<distance_points1*k+offset_point1
                    <rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                rp=mdb.models['Model-1'].rootAssembly.ReferencePoint
                      (point=
```

```
            (offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
          distance_points1*k+offset_point1+
str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                    0))
         mdb.models['Model-1'].rootAssembly.Set(name='SetRP',
referencePoints=(
         mdb.models['Model-
        1'].rootAssembly.referencePoints[rp.id],))
         mdb.models['Model-
        1'].rootAssembly.SetByBoolean(name='Set-RP', sets=(
         mdb.models['Model-1'].rootAssembly.sets['Set-RP'],
         mdb.models['Model-1'].rootAssembly.sets['SetRP']))
```

Si osservi che si è utilizzata ancora la funzione booleana di unione per raggruppare tutti i punti di fastening in un unico "*Set*", e che il primo Reference Point è di solo supporto, non farà parte del set finale.

Si procede infine alla caratterizzazione vera e propria del vincolo, tramite:

- la generazione di una sezione Beam [Fig 3.38], che vincola tutti i gradi di libertà
- l'assegnazione di un raggio fisico di fastening
- la selezione del set di punti precedentemente definito

```
mdb.models['Model-1'].rootAssembly.engineeringFeatures.PointFastener(name=
    'Fasteners-1', physicalRadius=2.0, region=
    mdb.models['Model-1'].rootAssembly.sets['Set-RP'], sectionName='ConnSect')
```

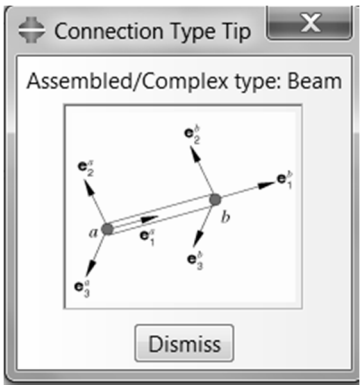


Fig 3.38 Gradi di libertà vincolati nella sezione Beam

3.7 Condizioni al contorno e di risoluzione

3.7.1 Creazione Step

Si è già discussa nel secondo capitolo l'importanza della definizione di uno step per garantire la corretta risoluzione del problema. Bisogna avere l'accortezza di fissare un incremento iniziale molto più piccolo del tempo di analisi globale, di modo da avere l'effettiva convergenza della soluzione [Fig. 3.39].

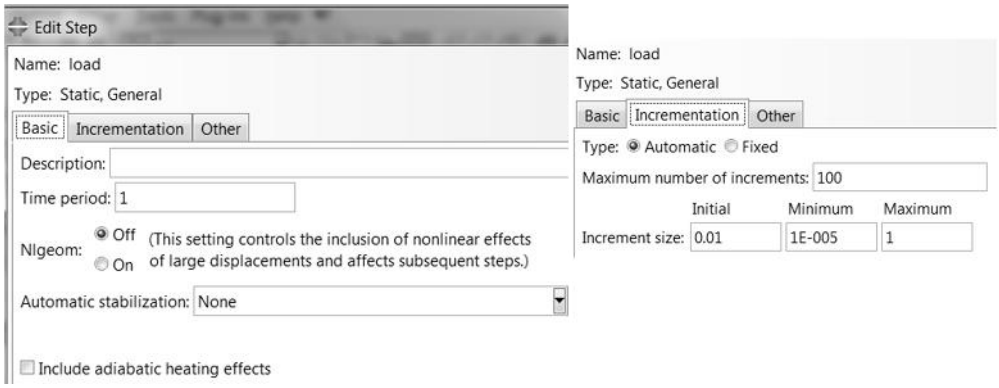


Fig 3.39 Parametri dello step di carico

Gli effetti non lineari non sono considerati nell'analisi, per garantire la convergenza e un costo computazionale non eccessivo.

3.7.2 Applicazione dei vincoli

Il pannello è vincolato nella superficie laterale dello skin [Fig 3.40], quella con coordinata x nulla, tramite un vincolo di tipo di incastro, creato nello step iniziale e propagato nello step di carico.

Il vincolo incastro elimina tutti i gradi di libertà alla superficie su cui è applicato.

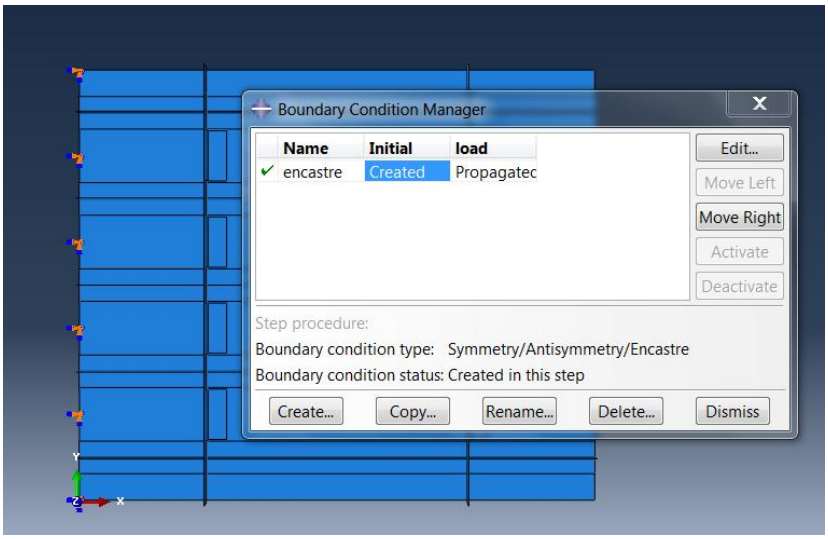


Fig 3.40 Definizione del vincolo di incastro

3.7.3 Applicazione dei carichi

Il carico è un carico di trazione applicato alla superficie laterale dello skin, quella con coordinata x pari a skin_lenght. Per rappresentare al meglio la condizione di carico reale si utilizza un vincolo di accoppiamento tra questa superficie ed un punto esterno (*reference point*) e un carico di trazione applicato solo sul punto.

Il carico di trazione è diretto nel verso positivo dell'asse x ed é possibile modificare il suo valore [Fig 3.41].

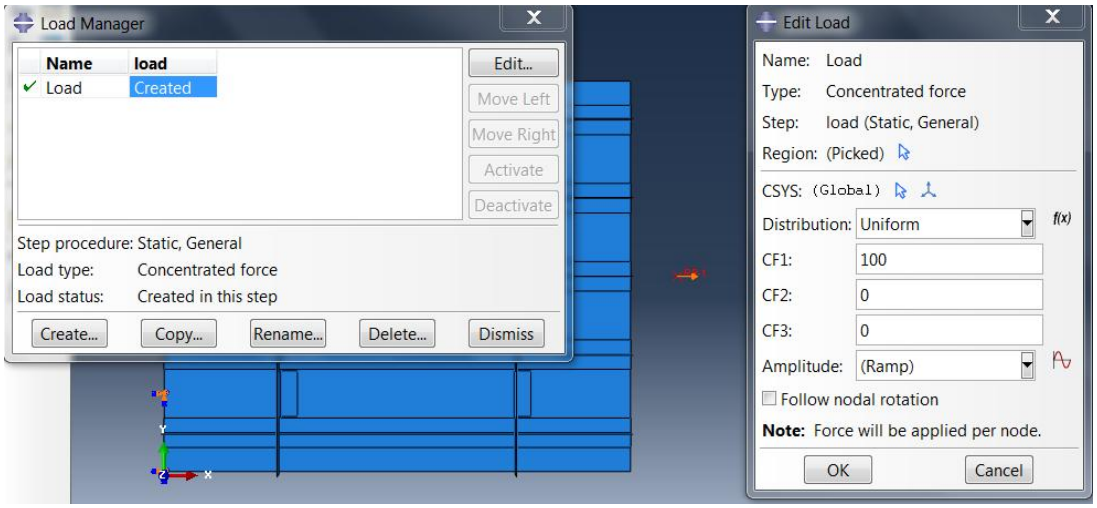


Fig 3.41 Definizione del carico di trazione

Il vincolo di accoppiamento lega il moto di una serie di nodi al moto del reference point. [Fig 3.42]. Si è utilizzato il tipo cinematico, che consente la scelta dei gradi di libertà eliminati. Nel caso specifico sono tutti vincolati.

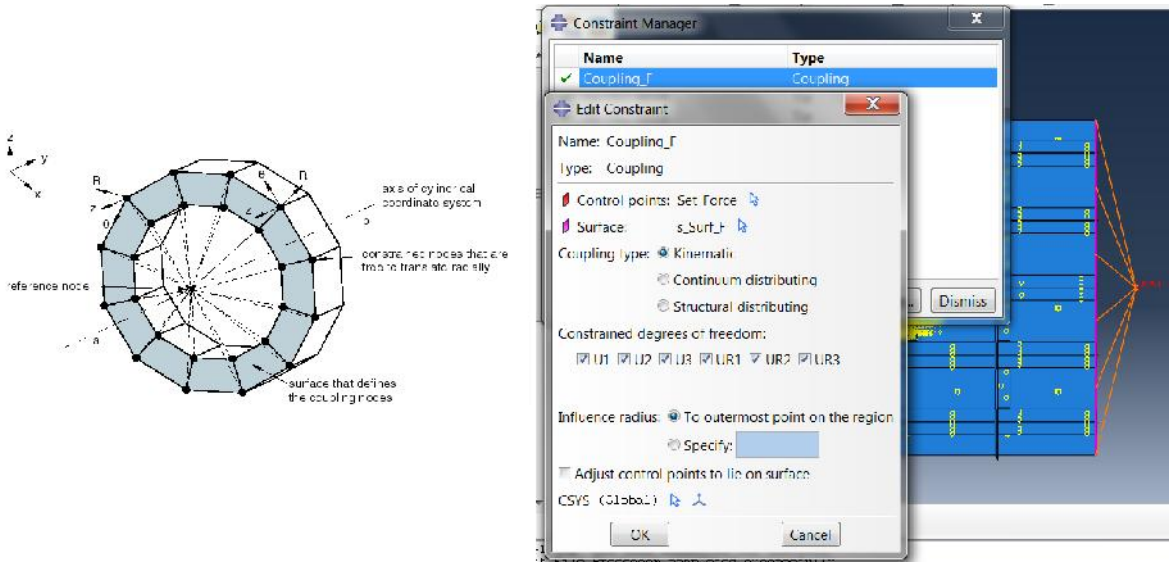


Fig 3.42 Definizione del vincolo di accoppiamento cinematico

Capitolo 4 SIMULAZIONI E CASI DI STUDIO

4.1 Configurazioni

Data la flessibilità del codice implementato, è stato possibile realizzare una pseudo-ottimizzazione, considerando più cricche di posizione variabile.

In particolare si sono studiate due macrocasistiche:

- configurazione A, in cui due cricche intaccano lo stesso stringer, che nel caso specifico è lo stringer centrale
- configurazione B, in cui due cricche intaccano due stringer diversi, il secondo e il quarto.

4.2 Configurazione A: due cricche sul medesimo stringer

Nella configurazione A una cricca è in posizione fissa, a ridosso del rib di sinistra, la seconda cricca è in posizione variabile.

Per garantire sufficiente differenza tra una configurazione e l'altra si è aumentata la distanza tra i rib di una quantità pari al 10% della lunghezza dello skin.

Si studiano tre diverse configurazioni [Tab 4.1]:

1. seconda riparazione in prossimità della prima, denominata *configurazione A-lx* (lx=left) [Fig.4.1-1]
2. seconda riparazione al centro dello spazio utile tra i due rib, denominata *configurazione A-middle* [Fig.4.1-2]
3. seconda riparazione in prossimità del secondo rib, denominata *configurazione A-rx* (rx=right) [[Fig.4.1-3]

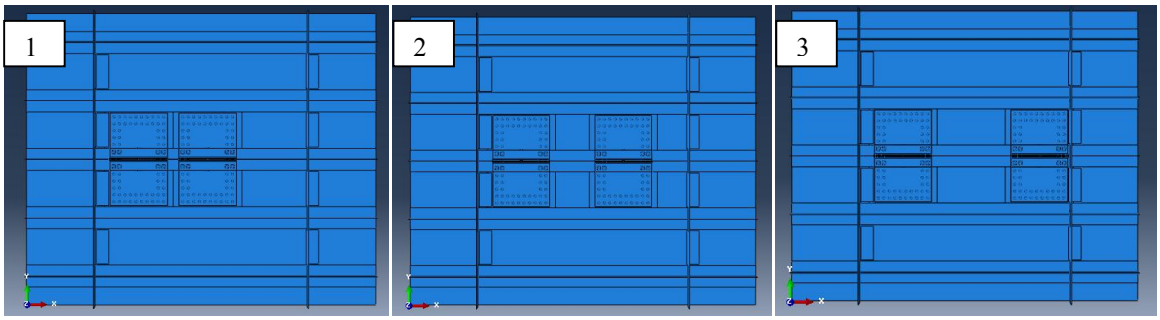


Fig 4.1 Configurazione A: seconda riparazione a 1) sinistra, 2) centro, 3) destra

	Origine della cricca in direzione x nel sdr assoluto	**VALORE [mm]**	**Distanza tra la cricca fissa e la cricca mobile**	**VALORE [mm]**
CRICCA FISSA	offset_rib+rib_width+ rip_lenght/2	290.88	/	/
CRICCA LX	offset_rib+rib_width+ rip_lenght/2+ +1*x1*rip_lenght	470.88	1*x1*rip_lenght	180
CRICCA MIDDLE	offset_rib+rib_width+ rip_lenght/2+ +1.5*x1*rip_lenght	560.88	1.5*x1*rip_lenght	270
CRICCA RX	offset_rib+rib_width+ rip_lenght/2+ +2*x1*rip_lenght	650.88	2*x1*rip_lenght	360

Tab 4.1 Configurazione A, posizione x delle cricche

Analizzando nel dettaglio le singole configurazioni:

- configurazione A-lx [Fig 4.2]: il valore di stress maggiore si ha in corrispondenza della cricca mobile. Gli apici della cricca sono punti singolari ed in quanto tali

presentano concentrazioni di stress maggiori, tra una riparazione e l'altra il valore di tensione è molto basso

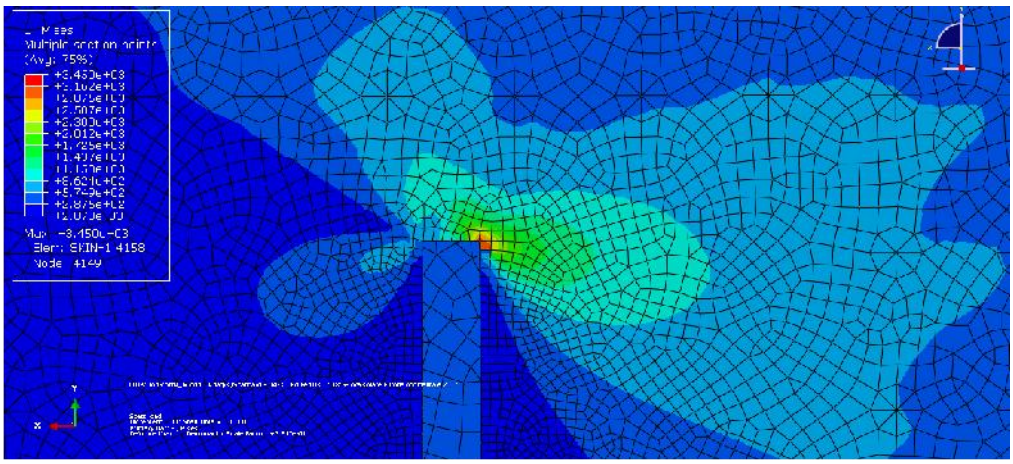


Fig 4.2 Configurazione a-lx: a) Particolare apice della cricca fissa

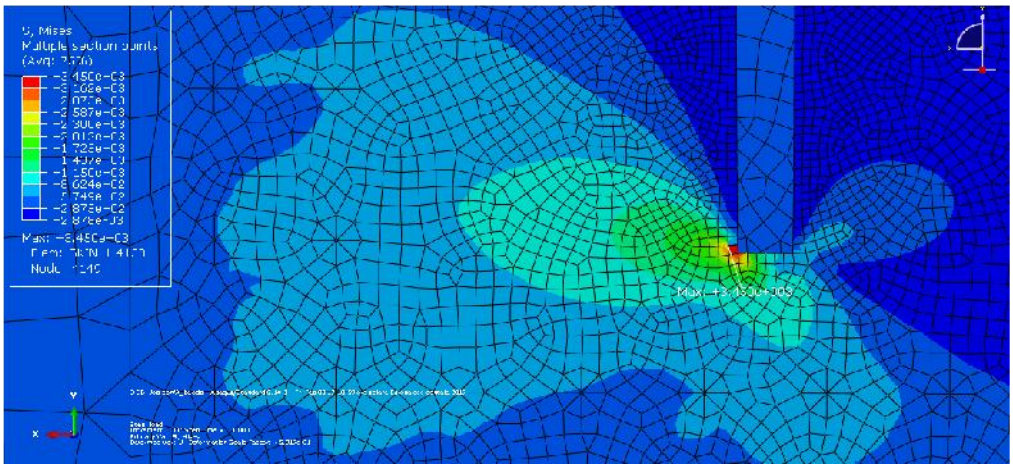


Fig 4.2 Configurazione a-lx: b) Particolare apice della cricca mobile

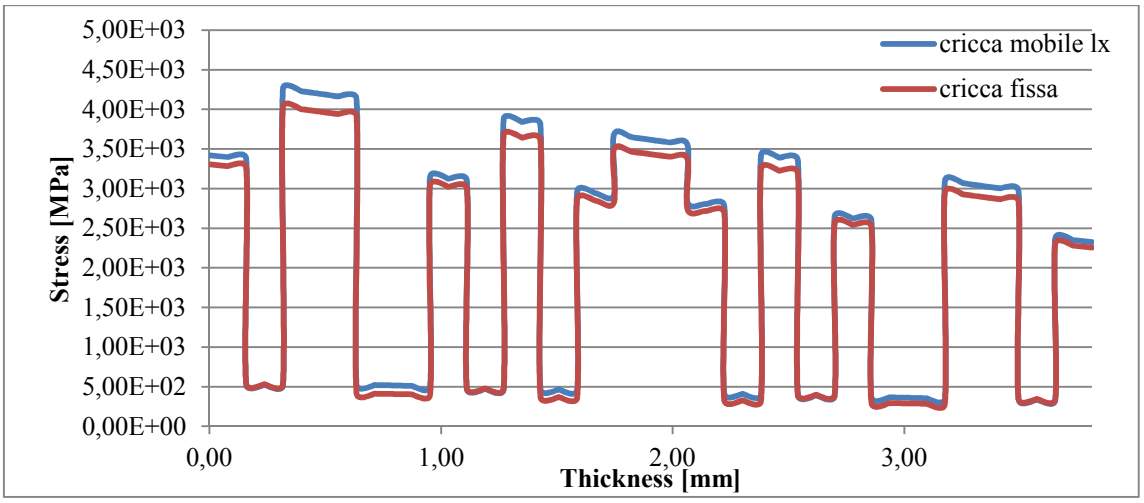


Fig 4.2 Configurazione a-lx: c) andamento degli stress lungo lo spessore

- configurazione A-middle [Fig. 4.3]: il massimo valore di stress si ha in corrispondenza della cricca mobile

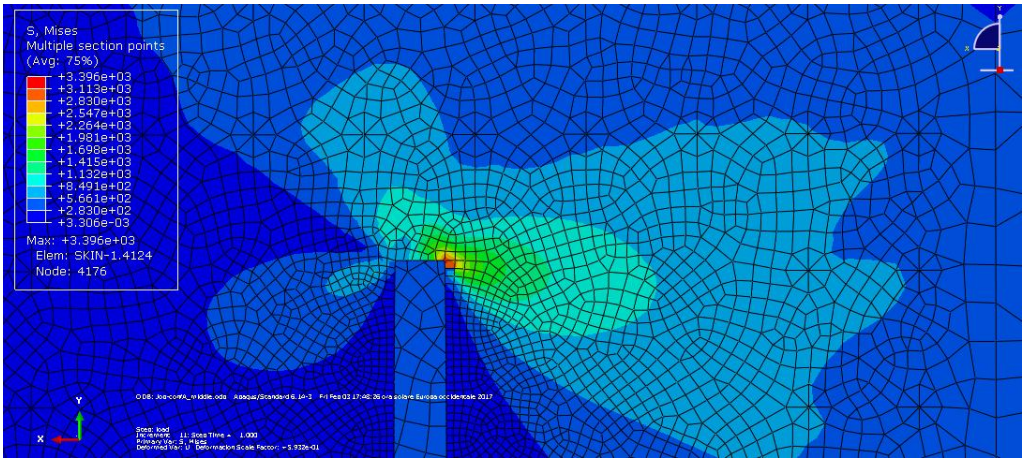


Fig 4.3 Configurazione a-middle: a) Particolare apice della cricca fissa

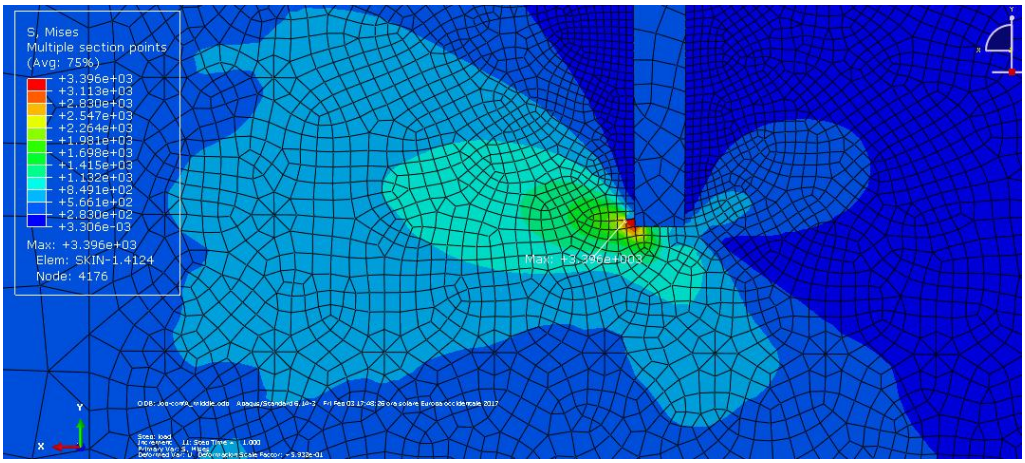


Fig 4.3 Configurazione a-middle: b) Particolare apice della cricca mobile

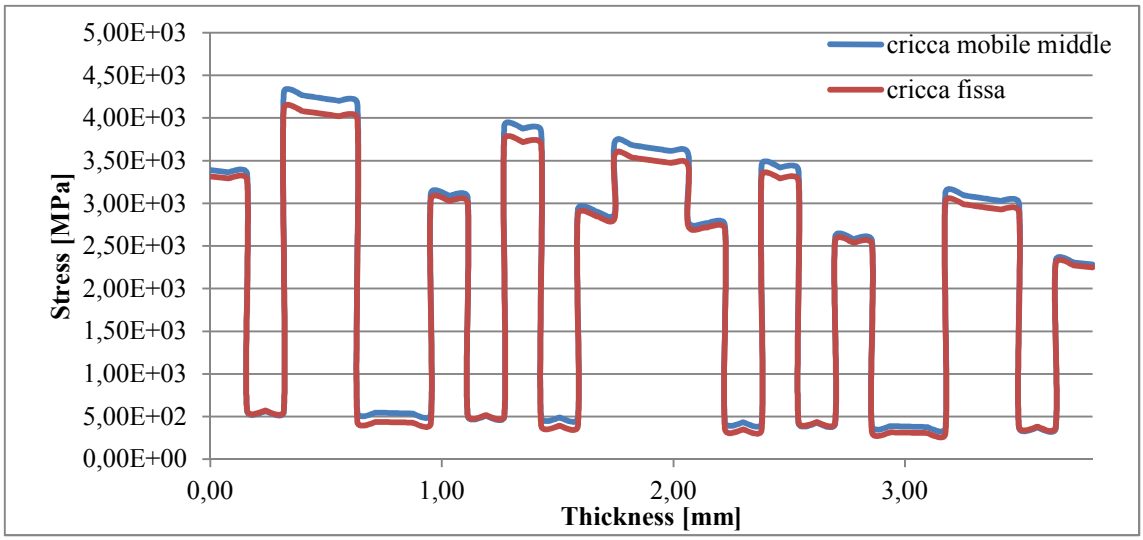


Fig 4.3 Configurazione a-middle: c) andamento degli stress lungo lo spessore

- configurazione A-rx [Fig. 4.4]: il massimo valore di stress si ha in corrispondenza della cricca fissa, al contrario che nelle precedenti configurazioni

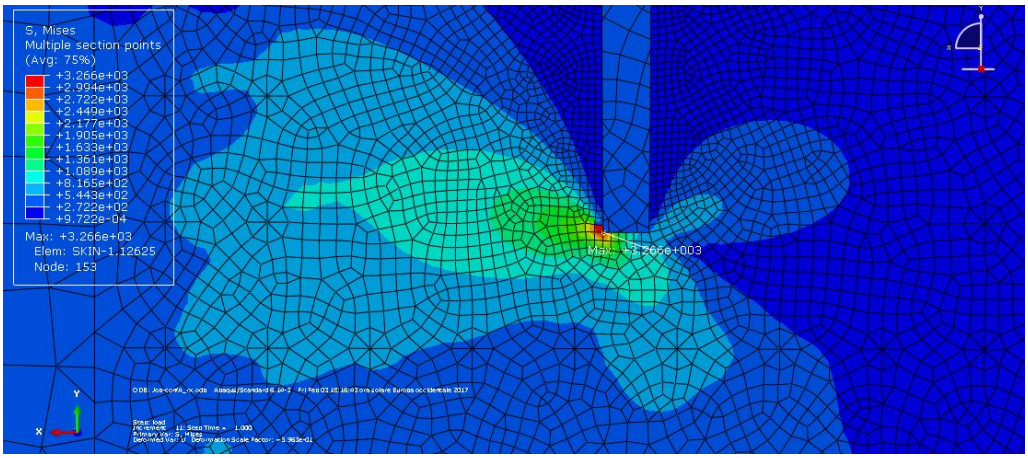


Fig 4.4 Configurazione a-rx: a) Particolare apice della cricca fissa

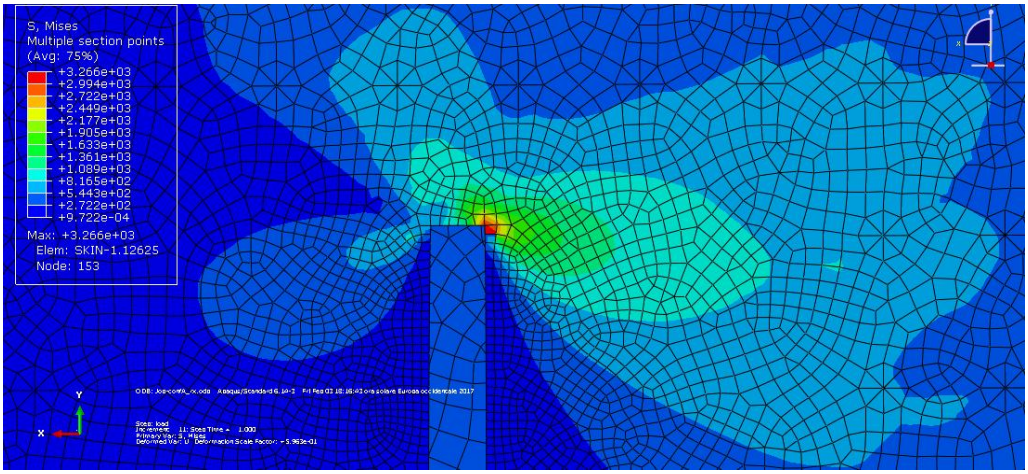


Fig 4.4 Configurazione a-rx: b) Particolare apice della cricca mobile

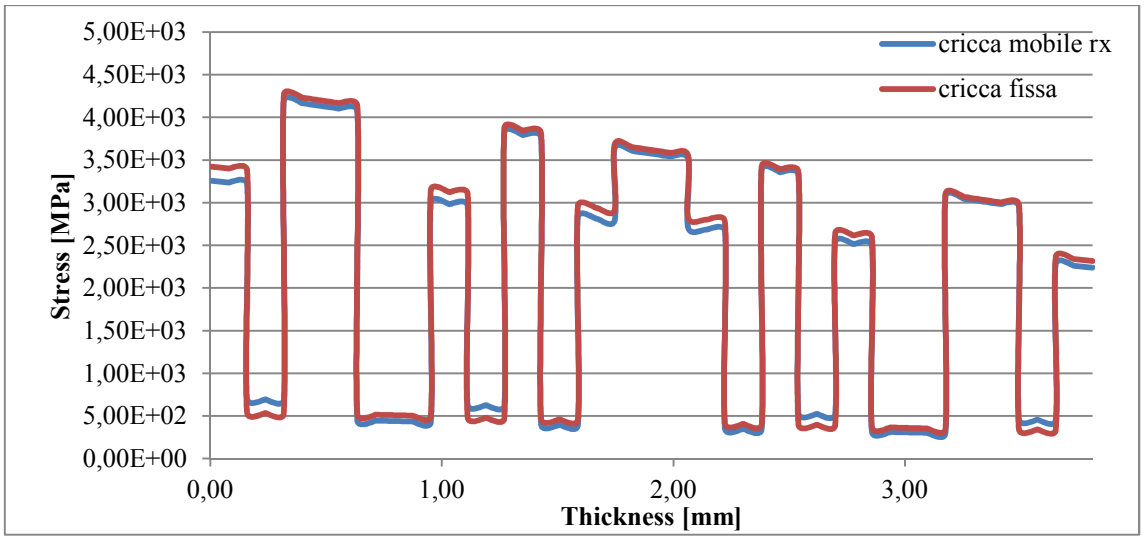


Fig 4.4 Configurazione a-rx: c) andamento degli stress lungo lo spessore

Nel complesso la configurazione più pericolosa risulta essere quella con le due cricche vicine, ovvero la configurazione A-lx. Mano a mano che la seconda cricca è posizionata più lontana dalla cricca in posizione fissa le tensioni diminuiscono [Fig. 4.5].

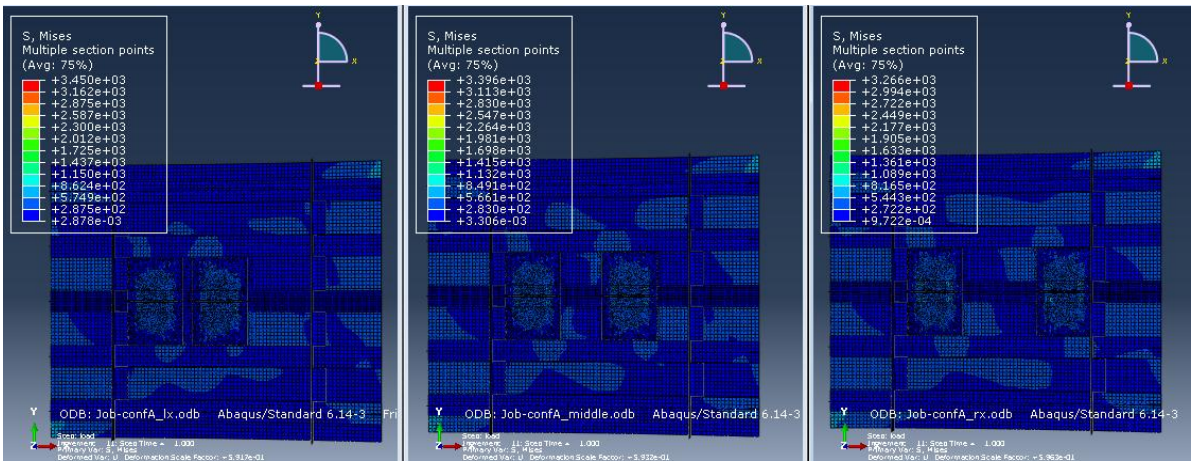


Fig 4.5 Rappresentazione degli stress per le tre configurazioni

Gli stress maggiori, e più grandi in assoluto, sono registrati in corrispondenza della cricca mobile, per la configurazione A-middle [Fig 4.6a]. Viceversa gli stress maggiori per la cricca fissa sono registrati per la configurazione A-rx [Fig 4.6b].

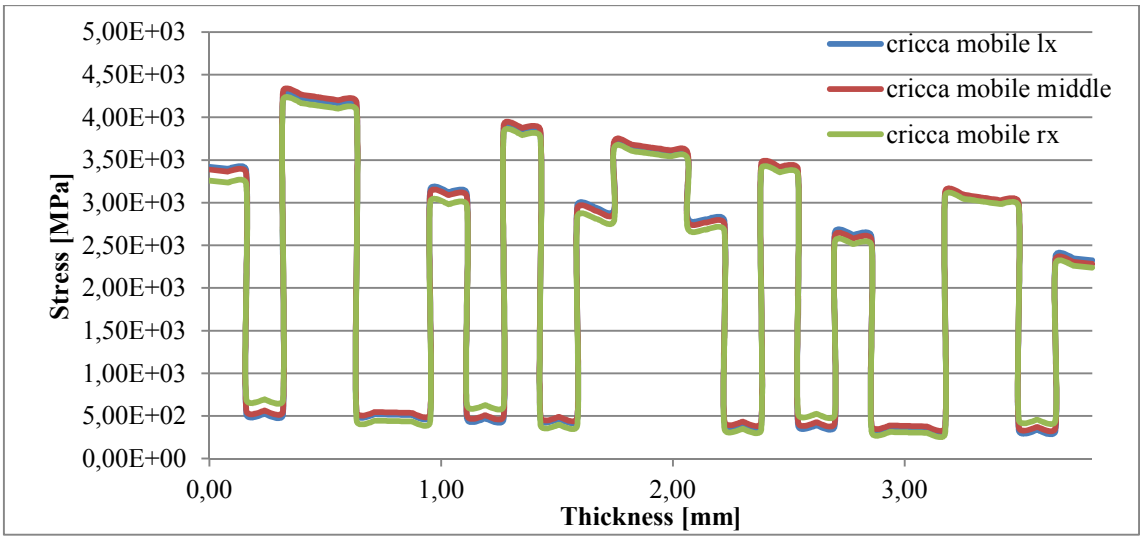


Fig 4.6 a) Confronto degli stress lungo lo spessore per la cricca di posizione variabile

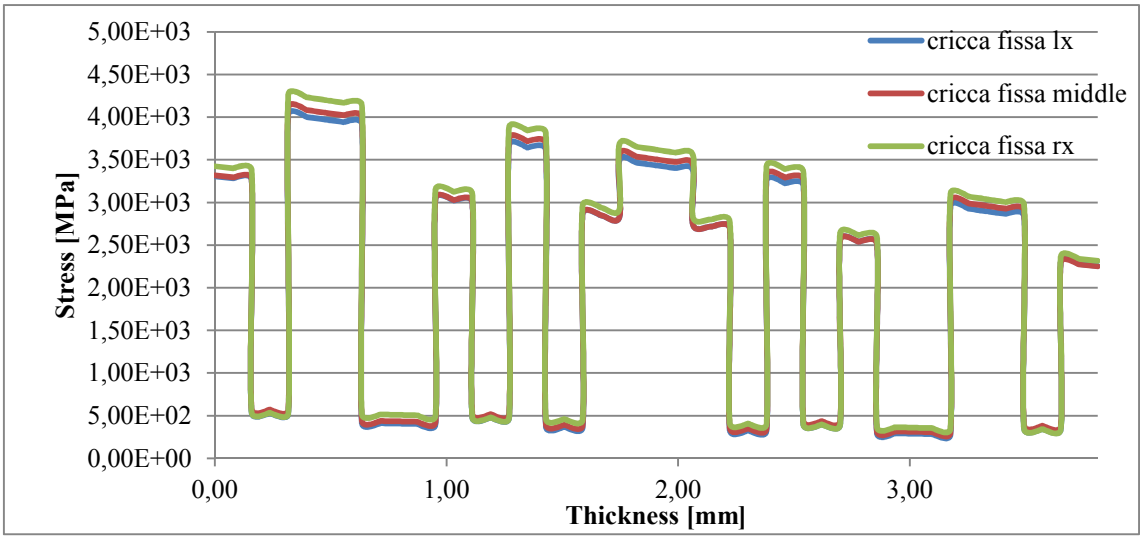


Fig 4.6 b) Confronto degli stress lungo lo spessore per la cricca fissa

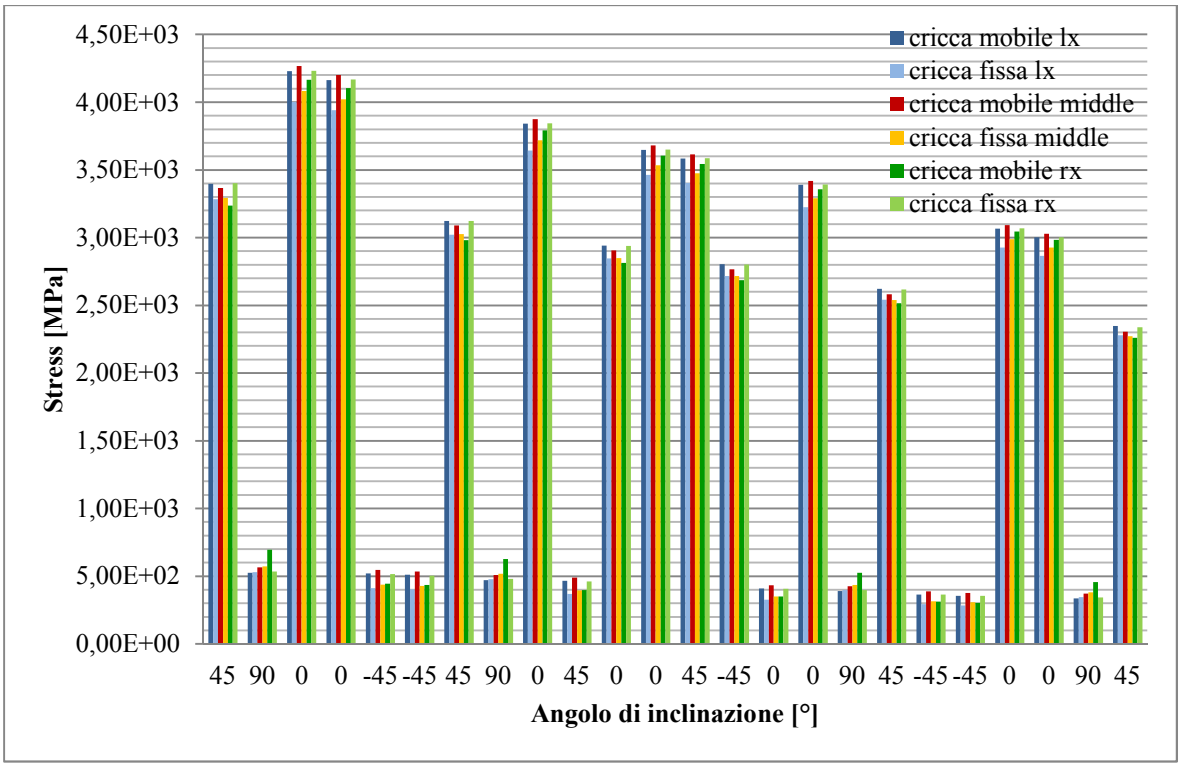


Fig 4.6 c) Confronto degli stress in relazione all'angolo di orientamento dei ply

Si possono fare una serie di osservazioni riguardo l'andamento delle tensioni lungo lo spessore [Fig 4.6c]:

- l'orientamento dei ply influenza la risposta del pannello, in quanto si ottiene massimo stress per i ply orientati a 0° rispetto al carico esterno, minimo stress per i ply orientati a 90° rispetto al carico esterno
- i ply inclinati a 45° e -45° hanno comportamento variabile a seconda dei ply che li circondano
- lo stress tende ad aumentare per i punti a distanza maggiore dalla riparazione, ovvero per i punti che giacciono sul piano xy, risentendo però dell'influenza dell'angolo di inclinazione dei ply. Lo stress massimo si ha per quei punti a massima distanza dalla riparazione, con angolo di inclinazione dei ply nullo
- nella mezzeria del pannello vi è una concentrazione di stress, a prescindere dall'angolo di inclinazione

Per ciò che riguarda le deformazioni si è considerato lo spostamento in direzione x del punto di applicazione del carico [Fig 4.7]. La configurazione A-rx subisce il massimo spostamento, pari a 4.38 [mm].

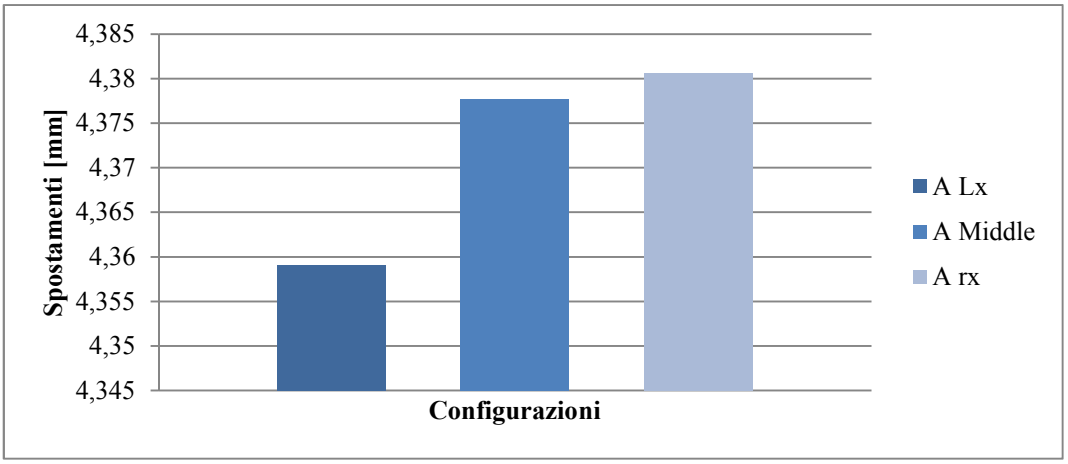


Fig 4.7 Spostamenti nella configurazione A

Si osservi che i punti considerati per estrarre i valori di stress sono stati scelti agli apici delle cricche, in corrispondenza dei valori maggiori, secondo indicazione della legenda.

La scelta di un elemento anziché di un altro porterebbe a sottostimare i valori di stress, motivo per cui si è proceduto tramite confronto grafico tra gli elementi. Si mostra la procedura per la sola cricca mobile del caso A-lx, essendo i passaggi seguiti gli stessi per tutte le configurazioni.

Innanzitutto si selezionano gli elementi che, da legenda, hanno il maggior valore di stress e se ne fa il "*plot*", ovvero la rappresentazione grafica [Fig. 4.8]:

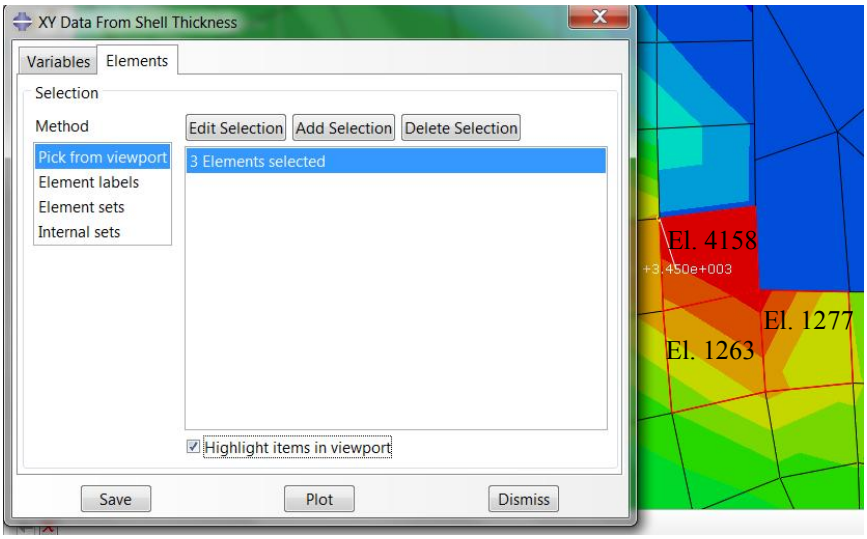


Fig 4.8 a) Selezione degli elementi di maggiore stress

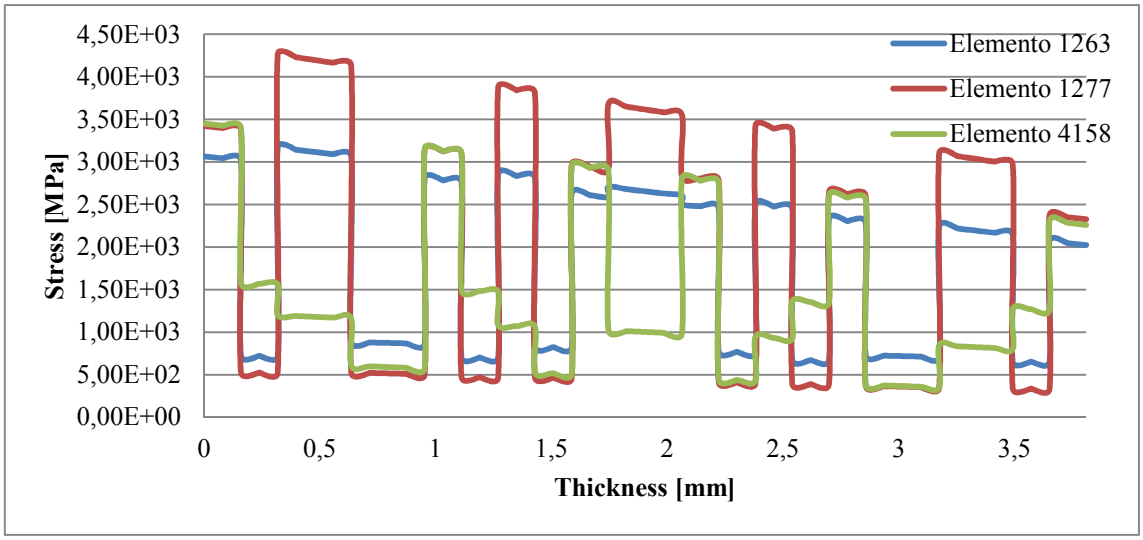


Fig 4.8 b) Confronto grafico degli elementi di maggiore stress

Tramite confronto grafico è immediato valutare gli elementi soggetti al maggior stato di tensione, in tal caso si tratta dell'elemento 1277.

Si osservi che una selezione poco attenta comporterebbe, oltre ad una sottostima delle tensioni, anche a una differente correlazione tra stress e spessore: poiché i punti in questione si trovano in corrispondenza dei bordi della cricca gli effetti di bordo assumono particolare importanza ai fini della valutazione finale.

4.3 Configurazione B: due cricche su due stringer

Nella configurazione B la cricca sul secondo stringer si mantiene in posizione centrale rispetto al pannello, quella sul quarto stringer trasla in direzione x, ricoprendo tre diverse posizioni nello spazio delimitato dai due rib [Fig 4.9].

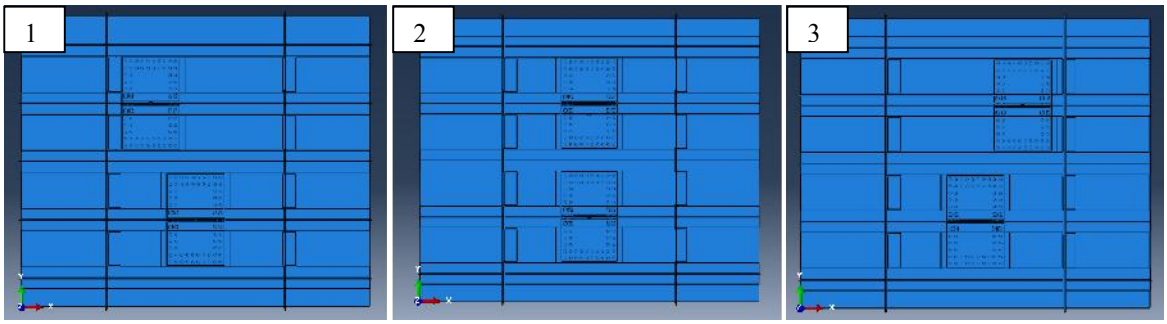


Fig 4.9 Configurazione B: seconda riparazione a 1) sinistra, 2) centro, 3) destra

Così come la configurazione A, anche la configurazione B presenta tre sottocasi [Tab 4.2]:

- cricca mobile sullo stringer 4, in prossimità del rib di sinistra, denominata *configurazione B-lx* [Fig 4.9-1]
- cricca mobile sullo stringer 4, in corrispondenza della cricca sullo stringer 2, denominata *configurazione B-middle* [Fig 4.9-2]
- cricca mobile sullo stringer 4, in prossimità del rib di destra, denominata *configurazione B-rx* [Fig 4.9-3]

	Origine della cricca in direzione x nel sdr assoluto	**VALORE [mm]**	**Origine della cricca in direzione y nel sdr assoluto**	**VALORE [mm]**
CRICCA FISSA	metà dello skin-cr_lenght/2	454	posizione stringer2 - metà cricca	152.4
CRICCA LX	rib_width+offset_rib+ rip_lenght1*0.5-cr_lenght/2	336.6	posizione stringer4 - metà cricca	457.2

CRICCA MIDDLE	metà dello skin-cr_lenght/2	454	posizione stringer4 - metà cricca	457.2
CRICCA RX	skin_lenght-offset_rib-rip_lenght1/2-cr_lenght/2	577.8	posizione stringer4 - metà cricca	457.2

Tab 4.2 Configurazione B, posizione delle cricche

Analizzando nel dettaglio le tre configurazioni:

- configurazione B-lx [Fig 4.10]: stress maggiori sulla cricca fissa, ovvero sulla cricca più vicina al punto di applicazione del carico

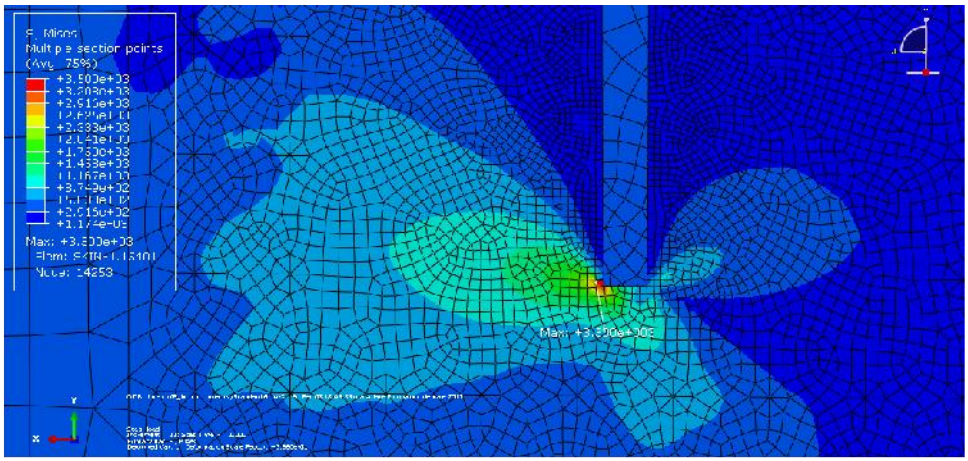


Fig 4.10 Configurazione b-lx: a) Particolare apice della cricca fissa

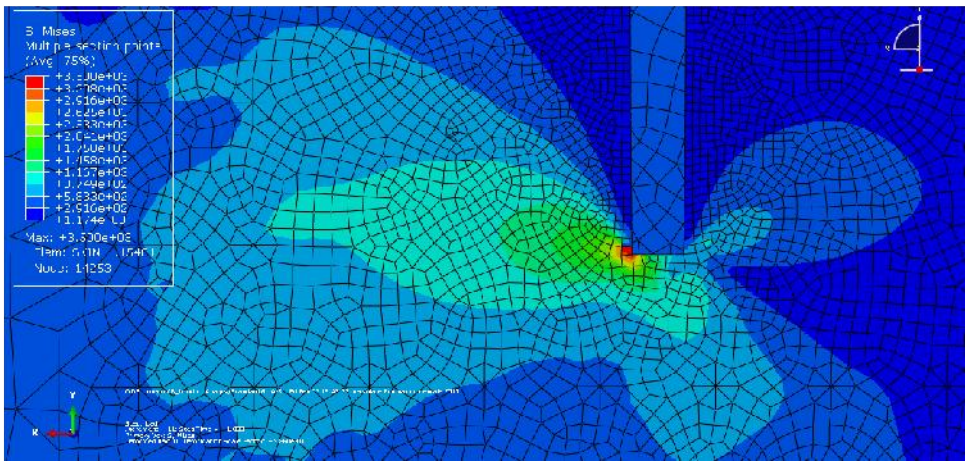


Fig 4.10 Configurazione b-lx: b) Particolare apice della cricca mobile

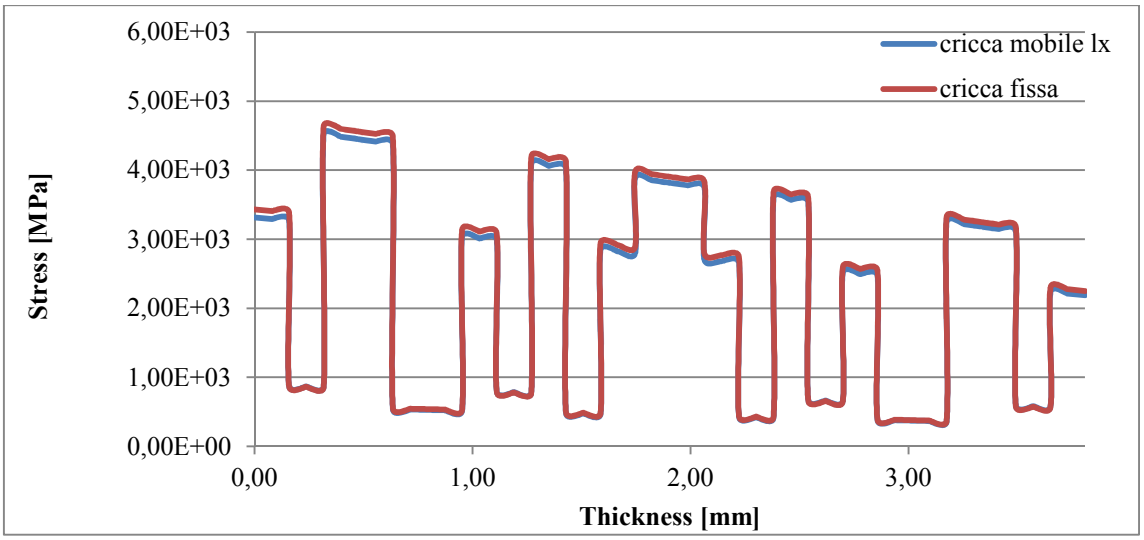


Fig 4.10 Configurazione b-lx: c) andamento degli stress lungo lo spessore

- configurazione B-middle [Fig 4.11]: stress molto simili su entrambe le cricche, con minima prevalenza della cricca fissa

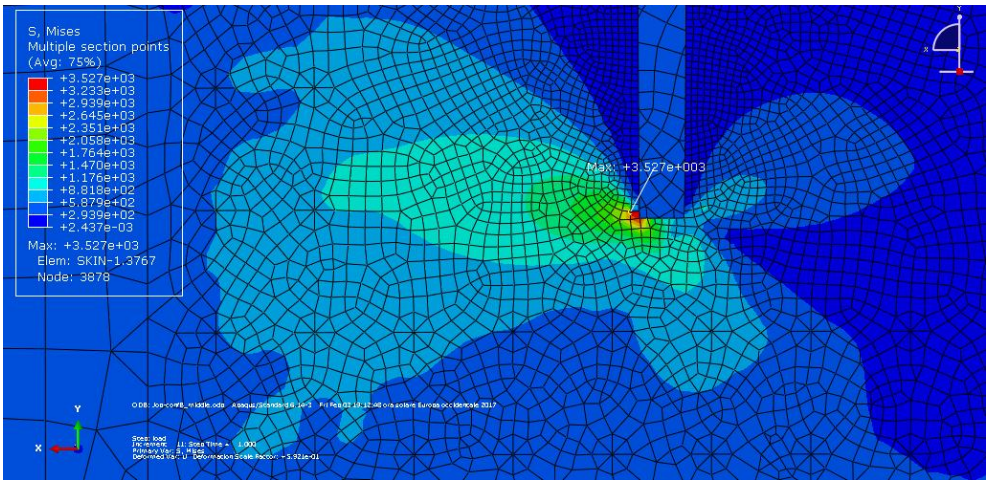


Fig 4.11 Configurazione b-middle: a) Particolare apice della cricca fissa

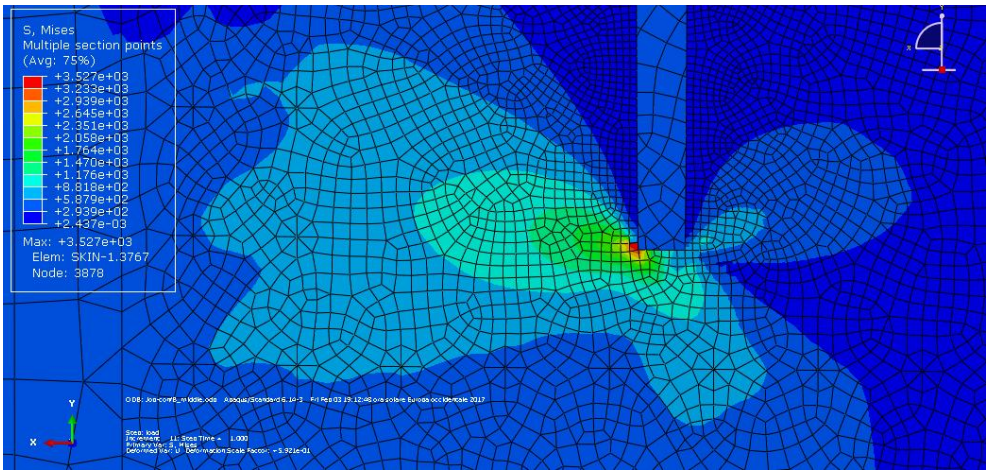


Fig 4.11 Configurazione b-middle: b) Particolare apice della cricca mobile

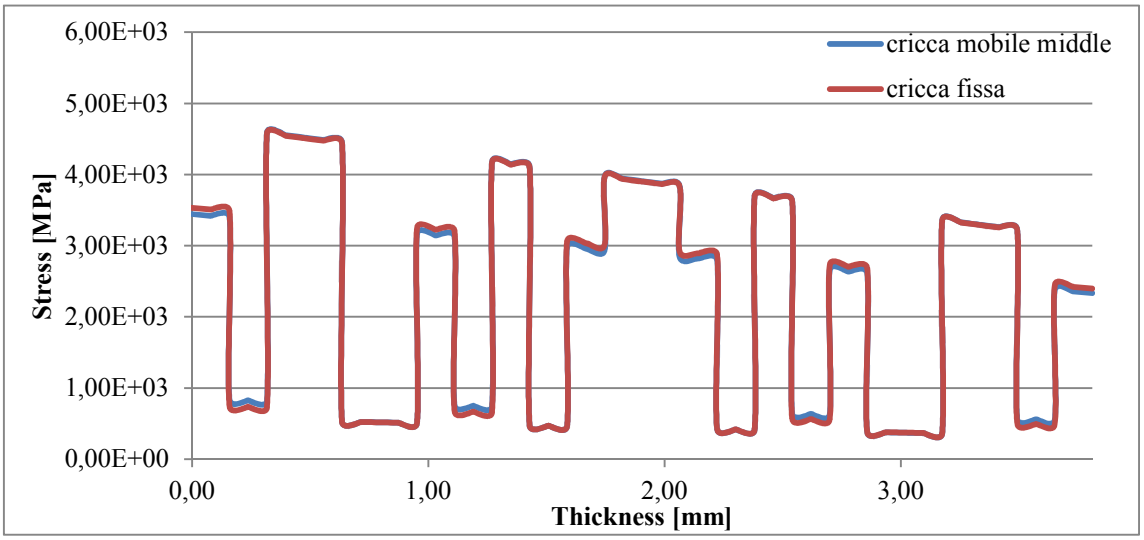


Fig 4.11 Configurazione b-middle: c) andamento degli stress lungo lo spessore

- configurazione B-rx [Fig 4.12]: stress maggiori sulla cricca mobile, ovvero, in analogia al caso B-lx, sulla cricca più vicina al punto di applicazione del carico

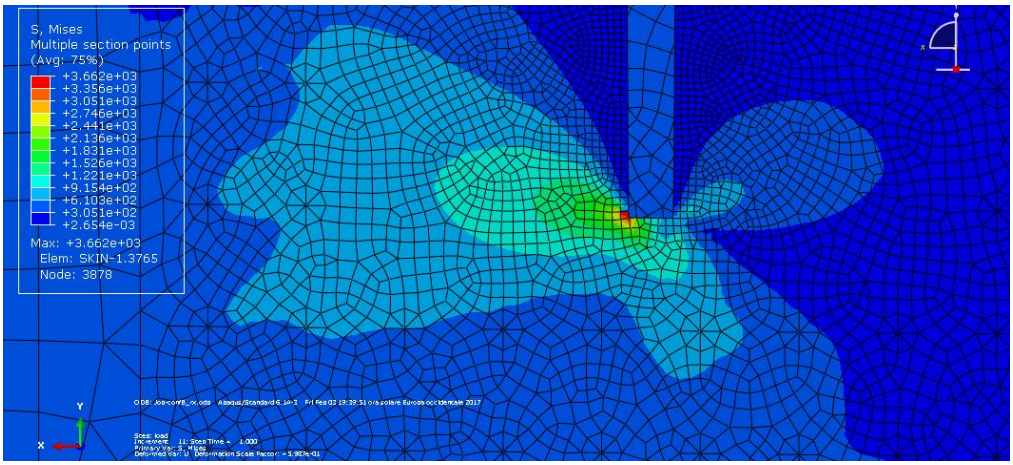


Fig 4.12 Configurazione b-rx: a) Particolare apice della cricca fissa

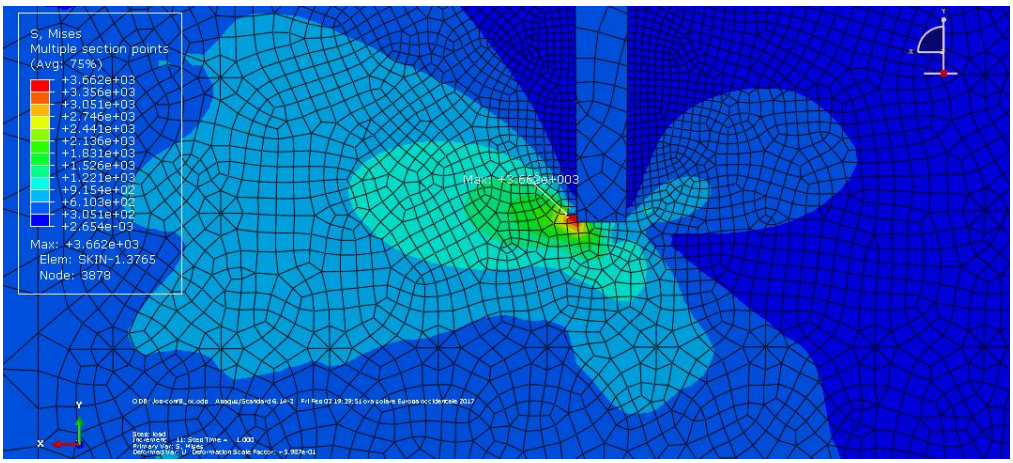


Fig 4.12 Configurazione b-rx: b) Particolare apice della cricca mobile

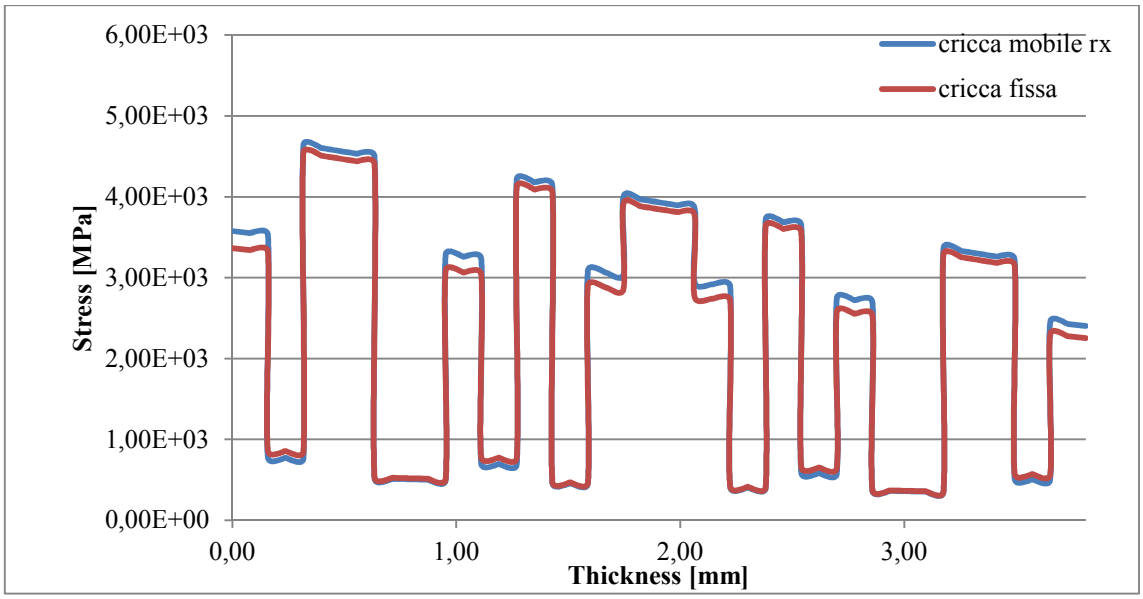


Fig 4.12 Configurazione b-rx: c) andamento degli stress lungo lo spessore

Si osservi che le distribuzioni di tensione tra le due diverse cricche, all'interno della stessa configurazione, sono molto simili.

Nel complesso si osserva che gli stress aumentano mano a mano che la cricca di posizione variabile si avvicina al punto di applicazione del carico [Fig 4.13].

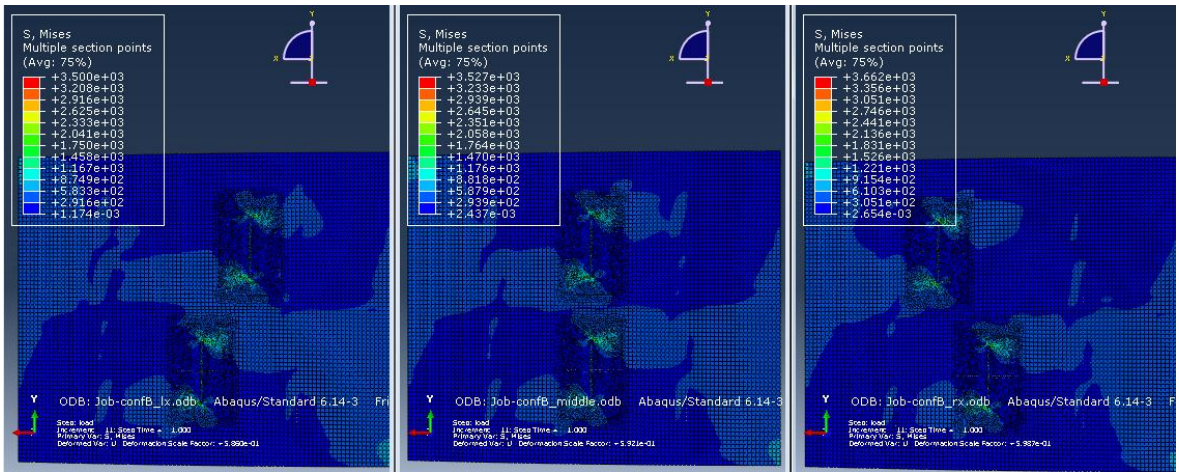


Fig 4.13 Rappresentazione degli stress per le tre configurazioni

I valori di stress maggiori in assoluto si hanno in corrispondenza della cricca mobile per la configurazione B-rx [Fig 4.14a]. I valori di stress maggiori sulla cricca fissa si hanno invece in corrispondenza della configurazione B-middle [Fig 4.14b].

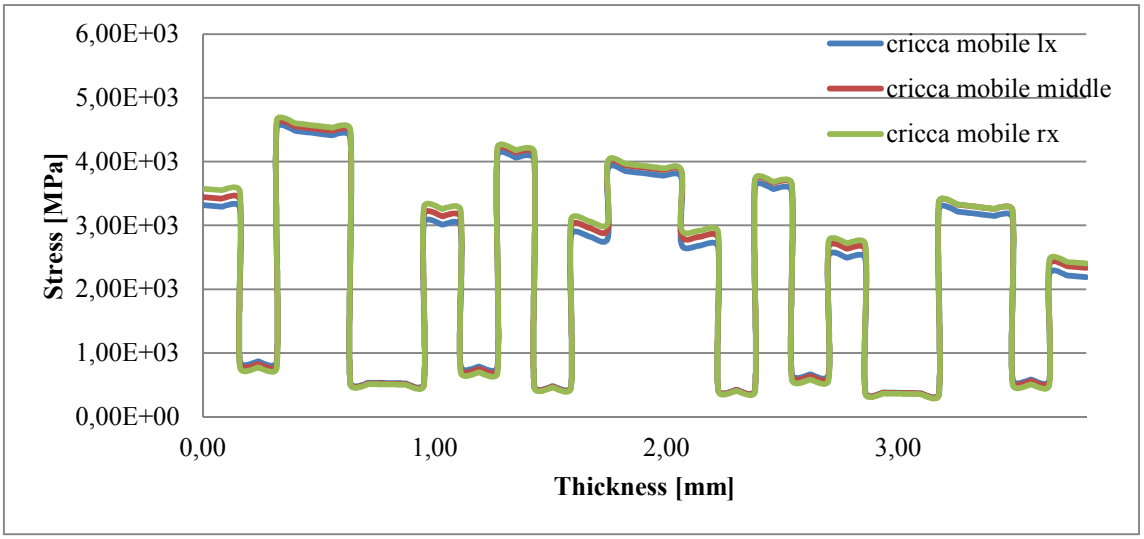


Fig 4.14 a) Confronto degli stress lungo lo spessore per la cricca di posizione variabile

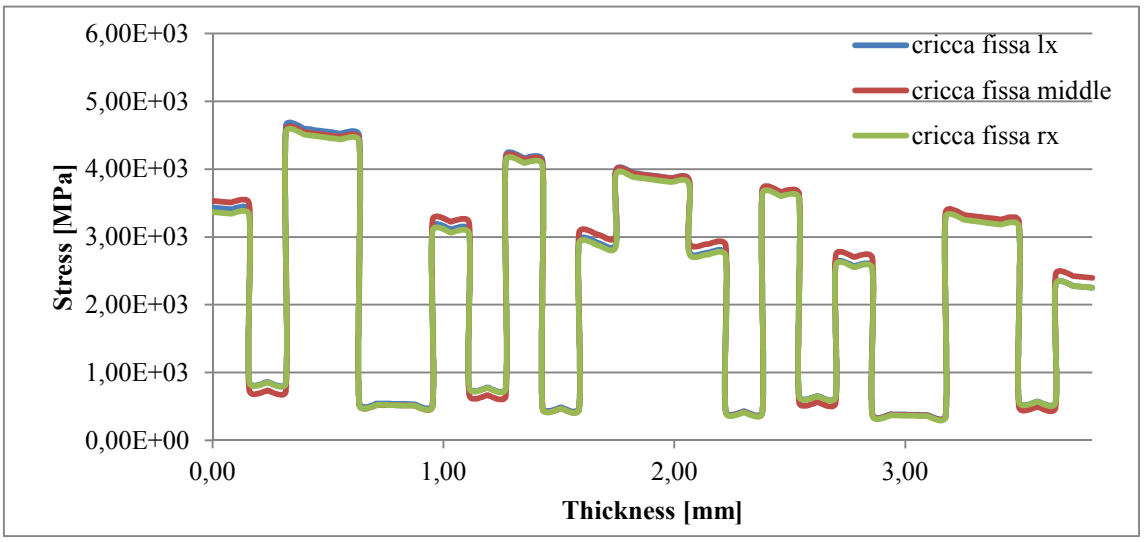


Fig 4.14 b) Confronto degli stress lungo lo spessore per la cricca fissa

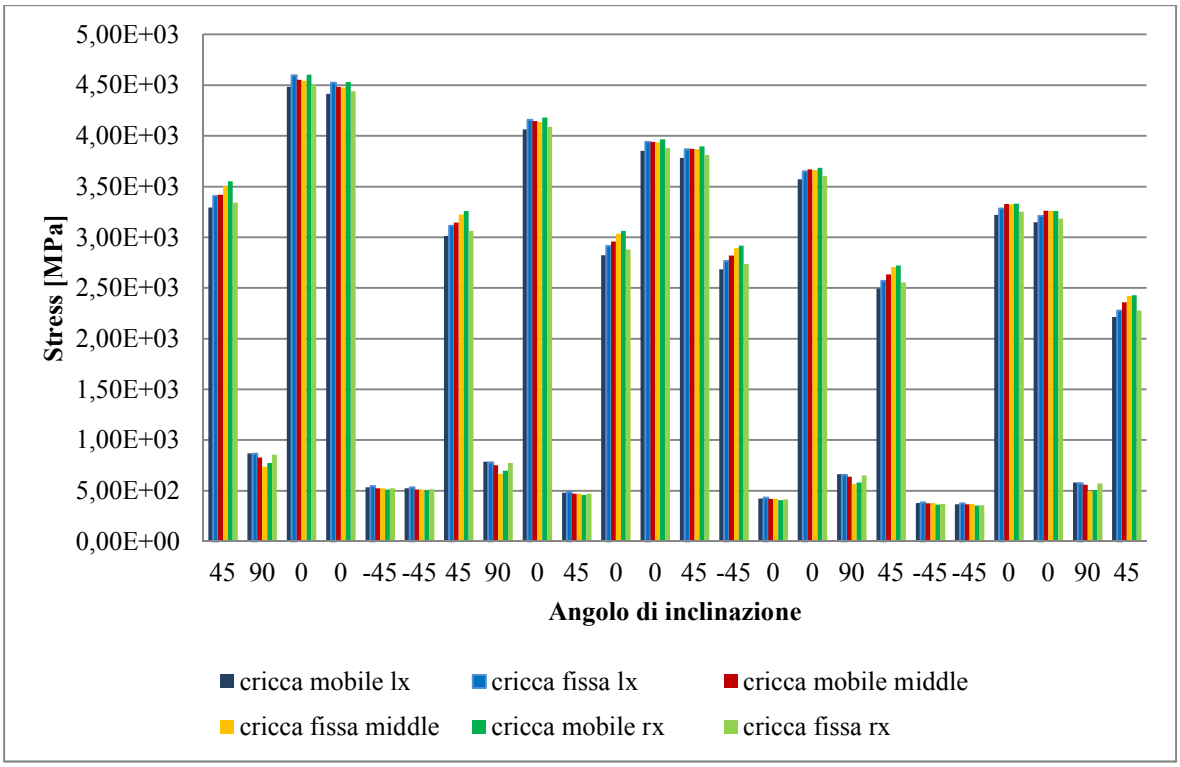


Fig 4.14 c) Confronto degli stress in relazione all'angolo di orientamento dei ply

Così come osservato per la macroconfigurazione A, l'orientamento dei ply influenza la risposta complessiva dello skin [Fig 4.14c]:

- lo stress è massimo per quei ply inclinati a 0° rispetto all'asse x e, dunque, rispetto alla direzione di applicazione del carico esterno
- lo stress è minimo per quei ply inclinati a 90° rispetto all'asse x e, dunque, perpendicolari rispetto alla direzione di applicazione del carico esterno

Il valore di stress maggiore in assoluto si ricava per quei ply inclinati a 0°, la cui distanza dalla riparazione è massima. Vi è ancora una concentrazione di stress nella mezzeria dello spessore.

Gli spostamenti [Fig 4.15] sono valutati, come nel caso A, per il punto di applicazione del carico. Il valore massimo si ha in corrispondenza della configurazione B-middle ed è pari a 4.41 [mm]

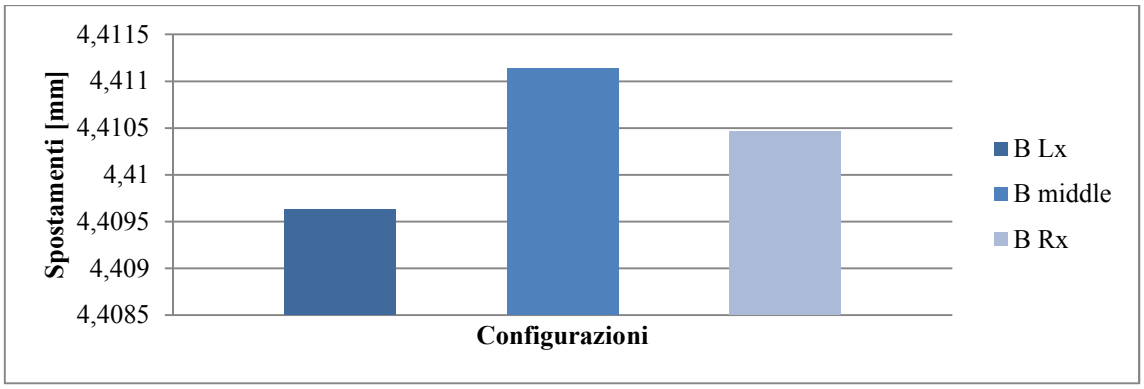


Fig 4.15 Spostamenti nella configurazione b

4.4 Confronto tra le configurazioni A e B

Nel seguito si confrontano gli stati di tensione in entrambe le configurazioni [Fig. 4.16]:

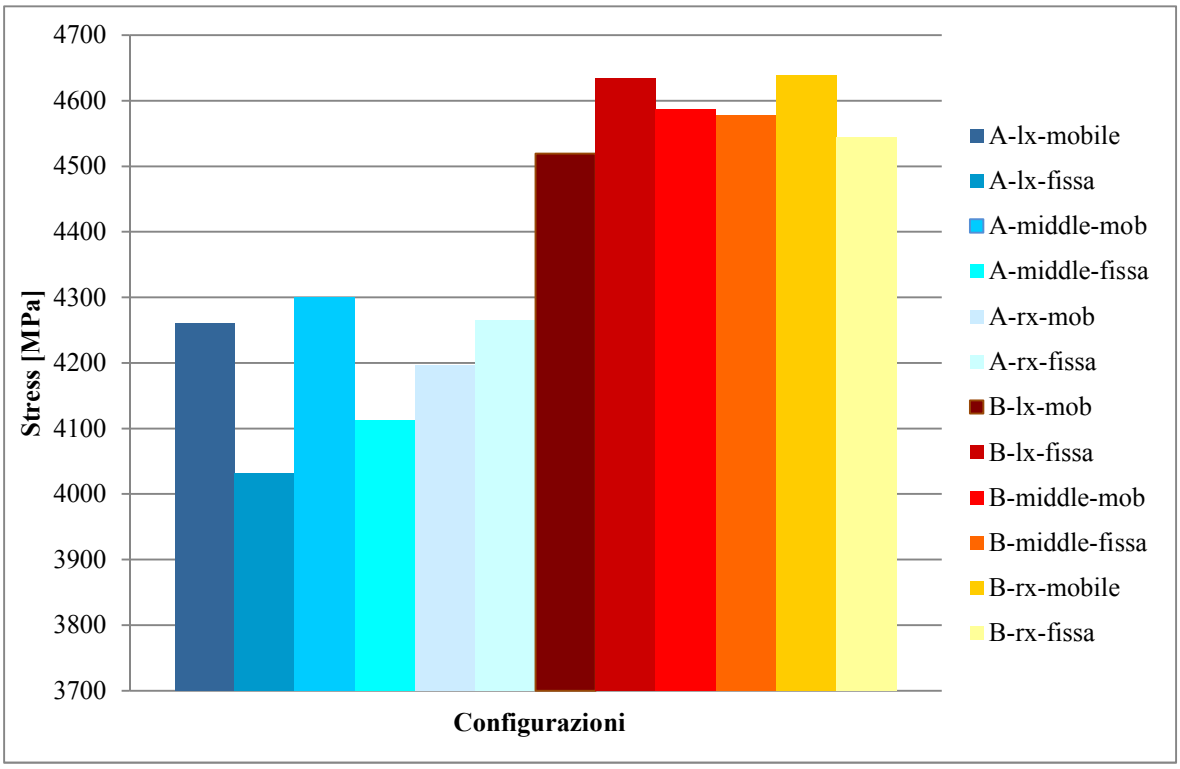


Fig 4.16 Confronto delle tensioni massime tra tutte le configurazioni possibili

In generale si verifica la maggiore pericolosità della configurazione B rispetto a quella A, ovvero danneggiamenti singoli su stringer differenti sono più gravi che danneggiamenti multipli sullo stesso stringer.

I valori di stress del caso B-rx sono, circa, del 9% maggiori rispetto a quelli massimi ottenuti nella configurazione A, nelle tre sottocasistiche.

Per ciò che riguarda l'angolo di inclinazione [Fig 4.17] si effettua il confronto delle sole configurazioni più pericolose a livello di cricca fissa e di cricca mobile: nello specifico si considerano i casi A-middle e B-rx per la cricca mobile, A-rx e B-middle per la cricca fissa.

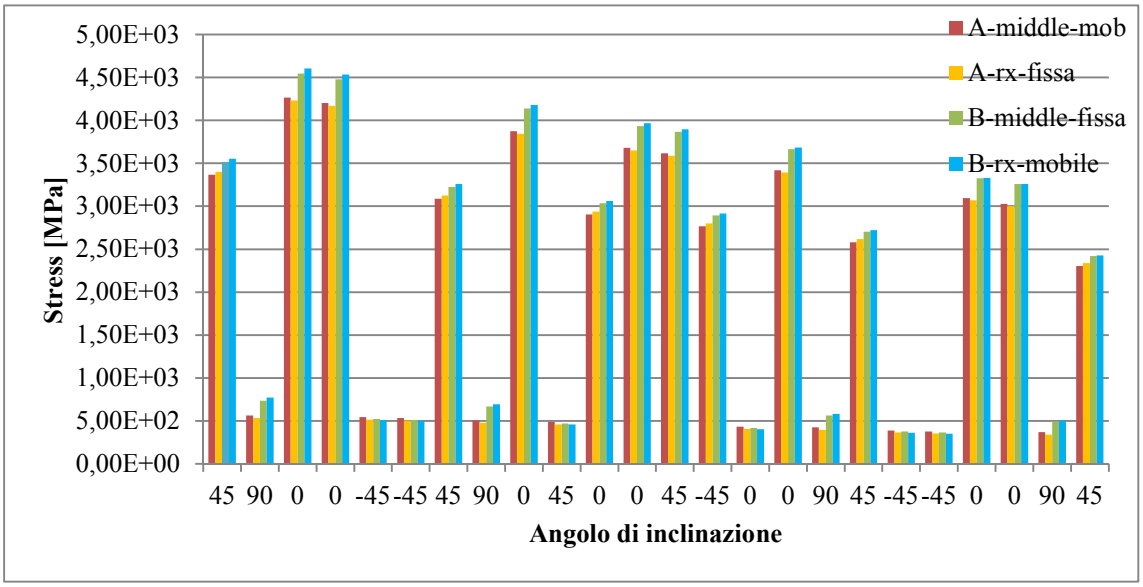


Fig 4.17 Confronto delle tensioni rispetto all'angolo di inclinazione

Anche per il confronto degli spostamenti in direzione longitudinale [Fig. 4.18] si sono considerati i soli valori massimi delle due casistiche, ovvero si sono esaminate la configurazione A-rx e B-middle.

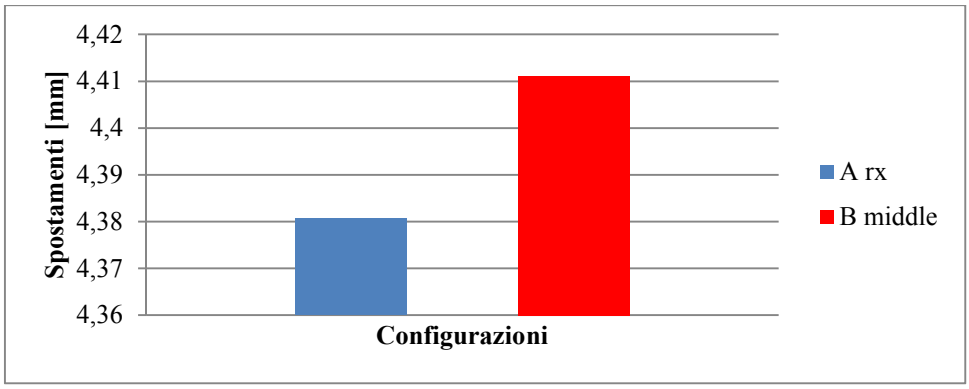


Fig 4.18 Spostamento massimo nei casi più critici

Il valore massimo è quello della configurazione B-middle, che supera il valore della configurazione A-rx di circa 0.3 [mm].

In conclusione quindi è possibile osservare che:

- danneggiamenti su stringer diversi sono più pericolosi che danneggiamenti sullo stesso stringer. Ciò può essere imputabile al fatto che nel pannello sono intaccati più rinforzi contemporaneamente, pertanto la funzione di rafforzamento dello skin diminuisce
- danneggiamenti sullo stesso stringer sono tanto più pericolosi quanto più sono vicini tra di loro: mantenendo una cricca fissa e facendo traslare la seconda lo stress aumenta mano a mano che i danneggiamenti sono in posizione reciproca più vicina
- danneggiamenti su stringer diversi sono tanto più pericolosi quanto più sono vicini al carico di applicazione: mantenendo un danneggiamento in posizione fissa e facendo traslare il secondo, gli stress sono massimi quando la seconda cricca è più vicina al punto di applicazione del carico

Capitolo 5 ANALISI DI UN PANNELLO IN PRESENZA DI UN DIFETTO

5.1 Ottimizzazione strutturale

In generale un problema di ottimizzazione si traduce nel voler minimizzare o massimizzare una certa funzione, chiamata funzione obiettivo, di N variabili x1, …, xN indipendenti, chiamate variabili di design [12][13]. I vincoli fisici del problema vengono espressi con funzioni di vincolo delle variabili di design, e possono essere di uguaglianza o di disuguaglianza. Infine, vi sono i limiti direttamente sulle variabili di design. Matematicamente si può esprimere tutto come segue:

Minimizza / Massimizza $W(X)$ *funzione obiettivo*

Soggetta a:

$g_j(X) < 0$ $j = l, m$ *vincoli di disuguaglianza*

$h_k(X) = 0$ $k = 1, l$ *vincoli di uguaglianza*

$X^l_i < X_i < X^u_i$ $i = l, n$ *limiti*

Dove $X = x_1, \dots, x_N$ *variabili di design*

La funzione obiettivo $W(X)$, come anche le funzioni di vincolo $g(X)$ e $h(X)$ sono funzioni lineari o non lineari delle variabili di design X, e possono essere esplicite o implicite.

Tipicamente data la complessità del problema di ottimizzazione, si ricorre a metodi di risoluzione iterativi; per raggiungere la convergenza, ad ogni iterazione i si valuta il fattore di convergenza eps calc conv definito come segue:

$$\varepsilon_{conv}^{calc} = max\left\{\left|\frac{(W_i - W_{i-1})}{W_{i-1}}\right|, \left|\frac{(X_i - X_{i-1})}{X_{i-1}}\right|\right\}$$

dove X_i e X_{i-1} sono i vettori soluzione ottenuti in due successive iterazioni; W_i e W_{i-1} sono i corrispondenti valori della funzione obiettivo. Questo fattore si confronta quindi con il limite di convergenza ε_{conv}^{lim} che in genere è minore di 10^{-15}:

$$\varepsilon_{conv}^{calc} < \varepsilon_{conv}^{lim}$$

5.1.1 Ottimizzazione Multi-objective

Spesso accade che ci sia la necessità di minimizzare o massimizzare contemporaneamente più funzioni obiettivo, e tipicamente sono funzioni che competono tra di loro, come per esempio il peso di una struttura aeronautica e la sua resistenza meccanica. Un modo molto pratico per affrontare questo tipo di problemi è convertirli in un problema con un'unica funzione obiettivo, così da ricondursi al caso precedente; questa unica funzione obiettivo può semplicemente essere una somma pesata delle funzioni obiettivo del problema originale, dove i pesi rappresentano l'importanza di ognuna. Quindi la funzione obiettivo sarà del tipo:

Minimizza $W(X) = \sum_{k=1}^{K} \alpha_k F_k(X)$

I fattori α_k sono da scegliere con accortezza in modo da non far prevalere inaspettatamente una funzione sulle altre.

5.1.2 Algoritmo genetico

Gli algoritmi genetici (GA) si basano sul principio darwiniano di evoluzione attraverso la selezione genetica. Il GA opera su una popolazione di cromosomi artificiali; ogni

cromosoma rappresenta una soluzione al problema in esame, ed è identificato da stringhe, tipicamente rappresentate con codice binario. Per ogni cromosoma si valuta la sua qualità ad essere soluzione del problema, quantificata tramite un numero reale chiamato fitness.

Partendo da una popolazione di cromosomi randomicament generata, un GA porta avanti un processo di selezione e ricombinazione per produrre la generazione successiva. La selezione avviene utilizzando il numero di fitness, quindi i cromosomi con un fitness maggiore hanno una probabilità maggiore di essere selezionati, anche più di una volta e anche ricombinati con sé stessi. Successivamente quindi i cromosomi selezionati sono ricombinati per formare i nuovi membri della popolazione successiva. La ricombinazione può avvenire come incrocio (crossover) e mutazione (mutation). Nell'operazione di incrocio si parte da due cromosomi genitori precedentemente selezionati; viene quindi selezionato un punto di incrocio in maniera random. I cromosomi figli vengono quindi costruiti utilizzando i caratteri del primo genitore, dall'inizio della stringa fino al punto di incrocio, e i caratteri del secondo genitore, dal punto di incrocio fino alla fine. La probabilità con cui si applica l'operazione di incrocio può variare tra 0.6 e 0.8; se non c'è incrocio, i cromosomi genitori passano inalterati nella generazione successiva. L'operazione di mutazione invece consiste nel cambiare uno o più elementi della stringa di un singolo cromosoma. La probabilità con cui avviene questa operazione è molto bassa, per esempio 0.01 o 0.001. Il GA viene iterato fin quando non si raggiungono certi criteri, come per esempio un numero massimo di generazioni o una variazione non più significativa della funzione obiettivo. In letteratura è possibile trovare esempi del GA applicato a ottimizzazione di laminati compositi [14][15][16].

5.2 Modello numerico

5.2.1 Pannello danneggiato non riparato

Nel seguente capitolo si è voluto studiare l'influenza della presenza di un difetto sui parametri di progettazione di un pannello in materiale composito rinforzato con stringer e rib. Il modello presentato nei capitoli precendenti è stato quindi modificato: è stata rimossa la riparazione ed è stata implementata la parametrizzazione delle caratteristiche geometriche che si è scelto di variare in fase di ottimizzazione.

Il modello geometrico consiste in un pannello in materiale composito IMS/977-2 di dimensioni 550mmX400mm con 2 rib e 2 stringer, e un difetto rappresentato da una cricca passante di dimensioni 2mmX50mm posizionata al centro della baia. I parametri che si è scelto di variare in fase di ottimizzazione sono:

- Layup dello skin: numero di layer `n_layup_skin` e percentuali delle fibre a 90° `n90_skin` e a ±45° `n45_skin`;
- Spessore e altezza degli stringer `str_width` e `str_height`

La percentuale di fibre a 0° `n0_skin` viene calcolata a partire dalle altre due percentuali:

```
n0_skin = 1 - n90_skin - n45_skin
```

Il codice Python è stato modificato in modo da poter includere l'aggiornamento delle variabili di design in fase di ottimizzazione. Questo viene fatto tramite il file `var_def.py` così fatto:

```
cliCommand("""n90_skin = 0.79065""")
cliCommand("""n45_skin = 0.075982""")
cliCommand("""n_layup_skin = 14""")
cliCommand("""str_width = 63.8265""")
cliCommand("""str_height = 18.9986""")
```

Il comando `cliCommand(x)` prende la stringa `x` e la esegue in Abaqus, aggiornando così le variabili di design. Nel codice Python viene richiamata l'esecuzione di `var_def.py` :

```
execfile('C:/.../var_def.py', __main__.__dict__)
```

Il modello prevede anche delle modifiche sulla mesh: non sono più presenti infatti le partizioni per i bulloni, ma rimane la partizione per la cricca.

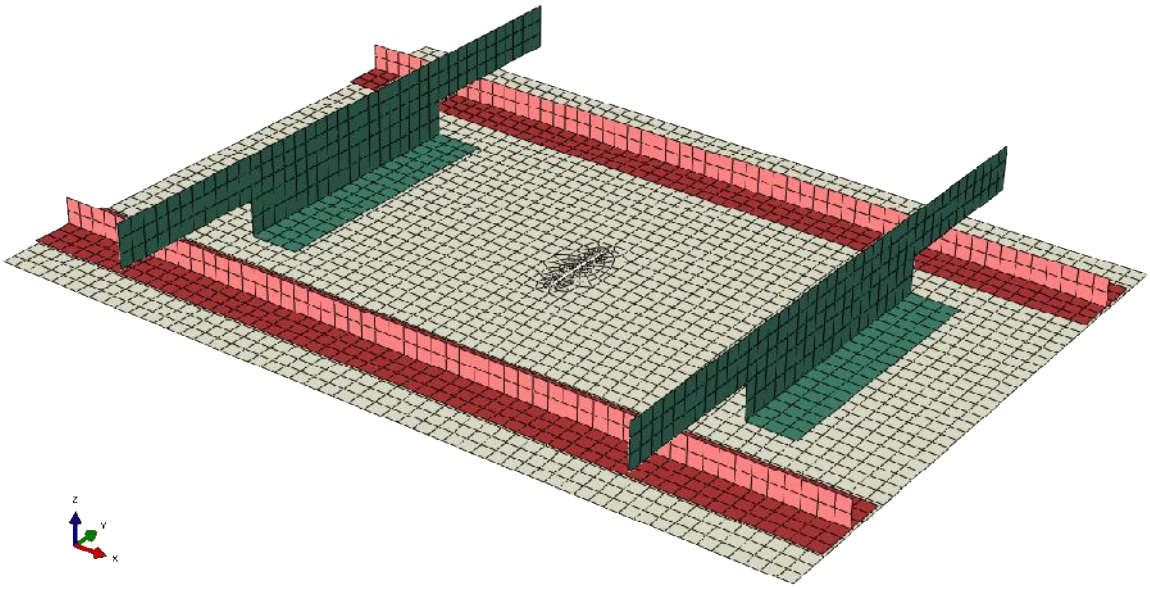


Figura 5-1 Assemblato del pannello con presenza di difetto (verde – rib; rosso – stringer; bianco – skin)

Il danneggiamento inoltre viene modellato con una geometria rettangolare, come nel precedente modello, ma con i bordi arrotondati:

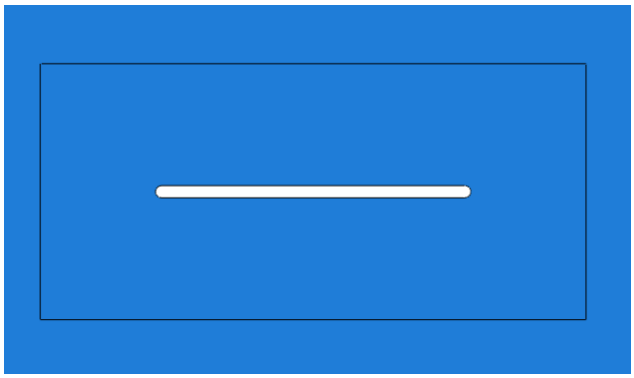


Figura 5-2 Il danneggiamento viene modellato con forma rettangolare con i bordi arrotondati

I materiali, i vincoli restano invariati rispetto al modello esposto precedentemente (Capitolo 4). Il carico, identificato come uno spostamento imposto su uno dei lati corti del pannello, viene applicato direttamente e non tramite un *reference point*.

Infine, nel modello viene creato anche il *job* e viene sottomesso:

```
job_name = 'opt_job'

mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
   explicitPrecision=SINGLE, getMemoryFromAnalysis=False,
   historyPrint=OFF, memory=80, memoryUnits=PERCENTAGE, model='Model-1',
   modelPrint=OFF, multiprocessingMode=DEFAULT, name=job_name,
   nodalOutputPrecision=SINGLE, numCpus=2, numDomains=2, numGPUs=0,
   queue=None, resultsFormat=ODB, scratch='', type=ANALYSIS,
   userSubroutine='', waitHours=0, waitMinutes=0)

mdb.jobs[job_name].submit(consistencyChecking=OFF)

mdb.jobs[job_name].waitForCompletion()
```

Si aspetta il completamento della simulazione e quindi si calcolano i risultati che saranno usati poi in fase di ottimizzazione: si apre il file dei risultati .odb; si leggono i valori di massimo e minimo stress di Von Mises e si calcola il peso totale della struttura; si produce un file di testo che contiene questi risultati.

```
odb = openOdb(job_name+'.odb')
```

```
# Max , Min Mises Stress
result = ('S','Mises')
result_field, result_invariant = result
_max = -1.0e20
_min = 1.0e20
step = odb.steps.values()[2]
for frame in step.frames:
   if frame.frameValue == 1.0:
       allFields = frame.fieldOutputs
       if (allFields.has_key(result_field)):
           stressSet = allFields[result_field]
           for stressValue in stressSet.values:
               if result_invariant:
                   if hasattr(stressValue, result_invariant.lower()):
                       val =
getattr(stressValue,result_invariant.lower())
                   else:
                       raise ValueError('Field value does not have
invariant %s' % (result_invariant,))
               else:
                   val = stressValue.data
               if ( val > _max):
                   _max = val
               if ( val < _min):
                   _min = val
       else:
           raise ValueError('Field output does not have field %s' %
(result_field,))

S_max = _max
S_min = _min


# Weight
skin_weight = skin_depth*skin_width*skin_lenght
str_weight = str_depth*str_width*skin_lenght +
str_height*(2*str_width)*skin_lenght

tot_weight = skin_weight + n_stringer*str_weight

file_temp = open('C:/Users/Utente/simulazioni/output_file.txt', 'w')
file_temp.write(str(S_max)+" "+str(tot_weight))
file_temp.close()
```

5.2.2 Pannello danneggiato con riparazione

Partendo dal modello descritto nel paragrafo precedente (5.2.1), si è aggiunta la riparazione come modellata nel Capitolo 4. Nel dettaglio, rispetto al modello precedente vi sono le seguenti modifiche.

La mesh non viene partizionata in corrispondenza dei bulloni né della lastra; infatti ai fini computazionali queste partizioni non hanno una influenza sui risultati, ma aumentano il numero degli elementi di mesh e quindi il tempo computazionale, laddove invece in fase di ottimizzazione si tende a contenerlo.

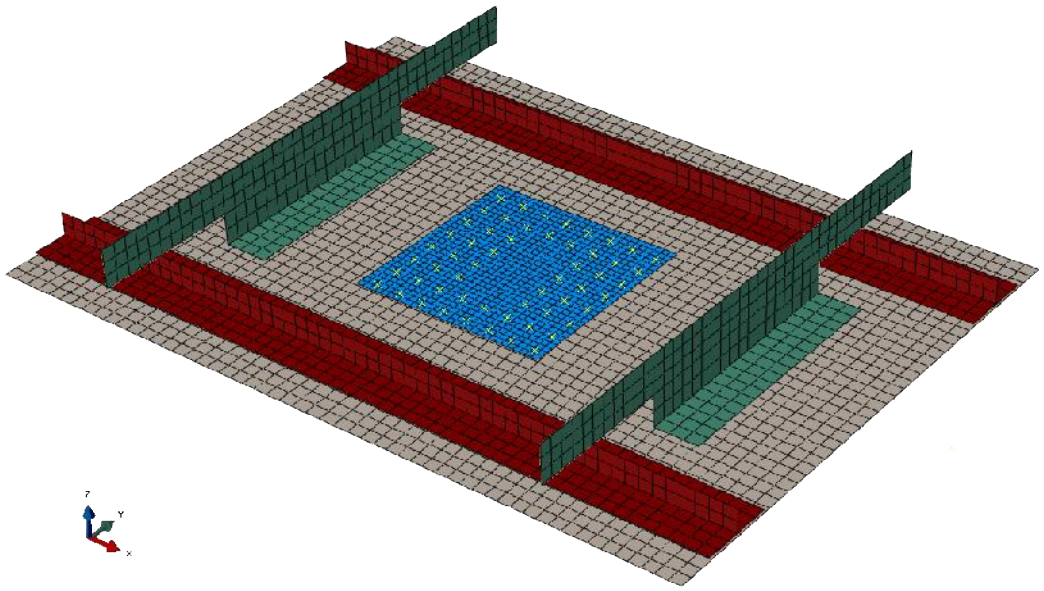


Figura 5-3 Assemblato del pannello con presenza di difetto, riparato con la piastra (verde – rib; rosso – stringer; bianco – skin; azzuro – piastra di riparazione). In giallo sono evidenziati i reference point che individuano i fastener.

Vengono attivati gli effetti della non linearità geometrica [17], concordemente con la presenza dell'interazione di contatto tra la piastra di riparazione e la skin:

```
mdb.models['Model-1'].steps['load'].setValues(nlgeom=ON)
```

5.3 Interfaccia Abaqus – MATLAB

L'ottimizzazione viene eseguita interfacciando Abaqus e Matlab. Come si è detto nel paragrafo precedente, le variabili di design sono: numero di layer dello skin e relativi numeri di fibre a 90° e a ±45°; altezza e spessore degli stringer. Le funzioni obbiettivo sono due: si vuole minimizzare contemporaneamente il peso totale della struttura P_{tot} e il valore del massimo stress di Von Mises S_{max}. Si è scelto quindi di minimizzare un'unica funzione obbiettivo F(X):

$$F(X) = \alpha S_{max} + \beta P_{tot}$$

Dove α e β sono stati inizialmente scelti pari a 0.5.

Per l'ottimizzazione si utilizza l'algoritmo genetico già implementato su Matlab [18]:

```
x = ga(fun,nvars,A,b,Aeq,beq,lb,ub,nonlcon,IntCon,options)
```

che trova un minimo locale `x` per la funzione obbiettivo `fun`; `nvars` è il numero di variabili di design. È possibile poi impostare:

- vincoli di disuguaglianza lineari `A*x ≤ b`
- vincoli di uguaglianza lineari `Aeq*x = beq`
- limiti inferiori e superiori delle variabili di design `lb ≤ x ≤ ub`
- vincoli non lineari tramite la funzione `nonlcon`
- quali variabili siano intere indicandole nel vettore `IntCon`
- opzioni di ottimizzazione personalizzate con `options`

L'opzione `IntCon` richiede che non vi siano vincoli di uguaglianza lineari e non lineari.

In questo caso si ha che:

```
nvars = 5;
lb = [0 0 12 55 18];
ub = [1 1 30 65 23];
Aeq = []
beq = []
A = [1 1 0 0 0];
b = 1;
IntCon = [3];
options = optimoptions('ga', 'FunctionTolerance', 2000,
'MaxGenerations',200,'PlotFcn',@gaplotbestf);
```

Le 5 variabili di design sono rispettivamente: numero di fibre a 0°; numero di fibre a 90°; numero di layer; spessore stringer; altezza stringer. Il vincolo lineare che si impone è che la percentuale di fibre a 0° sommata a quella delle fibre a 90° non superi l'unità. La terza variabile viene identificata come variabile intera. Le opzioni fissano a 2000 la variazione della funzione obbiettivo minima per interrompere l'algoritmo, e a 200 il numero massimo di iterazioni; si sceglie inoltre di plottare il valore della funzione obiettivo ad ogni iterazione.

La funzione obbiettivo è così strutturata: si genera un file di python `var_def.py` che contiene i valori aggiornati delle variabili di design:

```
var_file = 'var_def.py';

fileID = fopen(var_file,'wt');

fprintf(fileID,'cliCommand("""n90_skin = %s""")',num2str(x(1)));
fprintf(fileID,'\n');
fprintf(fileID,'cliCommand("""n45_skin = %s""")',num2str(x(2)));
fprintf(fileID,'\n');
fprintf(fileID,'cliCommand("""n_layup_skin = %s""")',num2str(x(3)));
fprintf(fileID,'\n');

fprintf(fileID,'cliCommand("""str_width = %s""")',num2str(x(4)));
fprintf(fileID,'\n');
fprintf(fileID,'cliCommand("""str_height = %s""")',num2str(x(5)));
fprintf(fileID,'\n');

fclose(fileID);
```

Quindi si richiama l'esecuzione in Abaqus del file di Python che genera il modello numerico del pannello:

```
dos('abaqus cae noGUI=C:\...\Shell_opt.py');
```

In questo file di Python, come è stato mostrato nel paragrafo precedente, è chiamata l'esecuzione del file var_def.py che aggiorna le variabili di design; crea il modello numerico; viene sottomessa la simulazione e viene prodotto un file di testo che contiene i risultati utili all'ottimizzazione, in questo caso massimo stress di V. Mises e peso totale.

In ultimo, nella funzione obbiettivo viene letto il suddetto file di testo e calcolata la funzione obbiettivo vera e propria:

```
[freq_1e,buck_1e,S_max,S_min,tot_weight] = textread('output_file.txt','%f
%f %f %f %f');

a = 0.5;
b = 0.5;
S_max
tot_weight
out = a*S_max + b*tot_weight
```

Capitolo 6 CONCLUSIONI E SVILUPPI FUTURI

Nel lavoro di tesi si è realizzato un modello di un pannello in materiale composito, rinforzato con rib e stringer, su cui è presente uno o più danneggiamenti riparati da una piastra bullonata in materiale metallico. Il modello è parametrico in modo da rendere possibile l'implementazione di un processo di ottimizzazione, volto a studiare l'interazione tra più difetti e l'effetto dei parametri di progettazione sul pannello sprovvisto di riparazione.

Gli sviluppi che si possono portare avanti sono molteplici e si possono così riassumere:

- Validazione sperimentale del modello FEM: riproducendo il pannello e le condizioni di carico si possono avere risultati sperimentali in termini di curva carico-spostamento e risultati estensimetrici con i quali validare il modello numerico;
- Aumentare la complessità del modello numerico. Si potrebbe per esempio sostituire il vincolo Tie tra skin e stringer, skin e rib, con un'interazione che tenga conto delle proprietà del collante; si potrebbero introdurre dei criteri di danneggiamento come per esempio il criterio di Hashin [19]; si potrebbe studiare l'effetto della curvatura del pannello.
- Modellazione dei bulloni. I bulloni sono modellati come collegamenti rigidi; sarebbe interessante introdurre le proprietà meccaniche elastiche di rigidezza a trazione e a taglio del giunto. In letteratura è possibile trovare diversi modi per calcolare queste rigidezze (formula di Huth [20]). Inoltre, servendosi di risultati sperimentali sul giunto stesso, si può pensare di introdurre un comportamento elasto-plastico del giunto, che possa in questo modo tener conto della ridistribuzione del carico tra i bulloni [21][22]. Questo approccio è più realistico e permette un'ottimizzazione meno conservativa.

- Completare e proseguire le ottimizzazioni. Si potrebbe condurre un'ottimizzazione della riparazione, partendo dal layup originale e cambiando i parametri caratteristici della riparazione, come per esempio numero e distanza dei bulloni, e confrontare il pannello ottimizzato con difetto non riparato, e il pannello danneggiato con riparazione ottimizzata. Si potrebbe inoltre introdurre nell'ottimizzazione altre funzioni obbiettivo come per esempio la massimizzazione del carico di buckling e la risposta in frequenza, oppure confrontare algoritmi diversi di ottimizzazione.

BIBLIOGRAFIA

[1] REINFORCEDplastic (2012). "The challange of composite fuselage repairs". Maggio/Giugno. 2012, pp. 30-35.

[2] S.C. Her, D.L. Shie. "The failure analysis of bolted repair on composite laminate". International Journal of Solids and Structures. Vol 35, Issue 15, 1998

[3] Z. Kapidzic, L. Nilsson, H. Ansell. "Finite element modeling of mechanically fastened composite-aluminum joints in aircraft structures". Composite Structures, Vol. 109, pp. 198-210, 2014.

[4] Abaqus Documentation, http://abaqus.software.polimi.it

[5] Python, https://www.python.org/

[6] N.E. Dowling. "Mechanical behavior of materials", second edition, Prentice Hall, 1998.

[7] S. P., Tesi di dottorato "I materiali compositi fibrorinforzati per il recupero degli edifici storici", Napoli: Università degli Studi di Napoli, 2005.

[8] R. Borelli (2010/2011). Tesi di dottorato "Sviluppo di procedure numeriche per la simulazione del danno in strutture composite", Università degli studi di Napoli.

[9] G.R. Liu and S.S. Quek. "The Finite Element Method: A practical course", Ed. Butterworth-Heinemann

[10] B. Atzori. "Moderni Metodi e Procedimenti di Calcolo nella Progettazione Meccanica", Laterza, Bari, 1999

[11] E. Toma (2016). Tesi di laurea. "Analisi e ottimizzazione di strutture in composito con riparazioni multiple", Politecnico di Bari

[12] F. Granieri (2007-2009). Tesi di dottorato "Ottimizzazione di strutture con metodi metaeuristici", Politecnico di Bari

[13] G.N. Vanderplaats, "Numerical Optimization Techinques for Engineering Desing", McGraw-Hill Book Company, New York. 1984

[14] G. Soremekun, Z. Gurdal, R.T. Haftka, L.T. Watson. “Composite laminate design optimization by genetic algorithms with generalized elitist selection”. Computers and Structures, Vol. 79, pp. 131-143, 2001.

[15] C.C. Lin, Y.J. Lee. “Stacking sequence optimization of laminated composites using genetic algorithms with local improvement”. Composite Structures, Vol. 63, pp. 339-345, 2004.

[16] T.U. Kim, J.W. Shin, I.H. Hwang. “Stacking sequence design of a composite wing under a random gust using a genetic algorithm”. Computers and Structures, Vol. 85, pp. 579-585, 2007

[17] L. Lamberti, S. Venkataraman, R.T. Haftka, T.F. Johnson. “Preliminary design optimization of stiffened panels using approximate analysis models", International Journal for Numerical Methods in Engineering, 2003.

[18] MATLAB Documentation, https://it.mathworks.com/help/matlab/index.html

[19] Z. Hashin, “Failure criteria for unidirectional fiber composites”, Journal of Applied Mechanics, 47, 329-334, (1980)

[20] H. Huth. “Influence on Fastener Flexibility on the Prediction of Load Transfer and Fatigue Life for Multiple-Row Joints”. ASTM STP 927 (1986).

[21] F. Liu, X. Lu, L. Zhao, J. Zhang, J. Xu, N. Hu. “Investigation of bolt load redistribution and its effect on failure prediction in double-lap, multi-bolt composite joints”. Composite Structures, Vol. 202, pp. 397-405, 2018.

[22] P.J. Gray, C.T. McCarthy. “A global bolted joint model for finite element analysis of load distributions in multi-bolt composite joints”. Composite: Part B, Vol. 41, pp. 317-325, 2010.

CODICI IN PYTHON

```
from __future__ import division # per le divisioni

from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *



#------#
# DATI #
#------#


##Le lunghezze sono tutte espresse in [mm]
##Di conseguenza si ricavano le altre unita' di misura


#DATI MATERIALE IMS/977-2
E1=152305.0
E2=8756.0
E3=8756.0
ni12=0.34
ni13=0.34
ni23=0.5448
G12=3937
G13=3937
G23=2834


#DATI MATERIALE AL 2219/T87
E=73800
ni=0.34
G=27600
ro=2768


#DATI SKIN
#Geometria Skin
skin_depth=3.81
skin_width=762
skin_lenght=914.4

#Layup Skin
layup_skin_or=((45,),(90,),(0,),(0,),(-45,),(-45,),
               (45,),(90,),(0,),(-45,),(45,),(0,),
               (0,),(45,),(-45,),(0,),(90,),(45,),
               (-45,),(-45,),(0,),(0,),(90,),(45,),)
n_layup_skin=len(layup_skin_or)
```

```
#CRICCA
#Numero cricche
n_cr=1

#Posizione cricche rispetto allo skin
x_cr=[454,554,475]
y_cr=[304.8,304.8,543]


#Geometria Cricche
cr_width=[152.4,152.4,100]
cr_lenght=[6.35,6.35,7]


#DATI STRINGER
#Numero stringers
n_stringer=5
n_rib=2


#Geometria stringer
str_lenght=57.56
str_depth=2.2352
str_height=25

#Layup stringer
layup_str_or=((45,),(90,),(0,),(0,),(-45,),(-0,),
              (0,),(-45,),(0,),(0,),(90,),(45,),)
n_layup_str=len(layup_str_or)



#DATI RIB
#Geometria rib
rib_width=33
rib_depth=2.4892
rib_height=str_height/0.375
rib_cut_height=1.2*str_height
rib_cut_lenght=1.1*str_lenght
rib_base=skin_width/n_stringer-rib_cut_lenght
rib_cut_distance=rib_cut_lenght+rib_base/2


#Layup rib
layup_rib_or=((45,),(0,),(-45,),(90,),
              (45,),(0,),(-45,),(90,),
              (45,),(0,),(-45,),(90,),)
n_layup_rib=len(layup_str_or)


#DISPOSIZIONE STRINGER E RIB
offset_stringer=(skin_width)/(n_stringer)
distance_stringer=offset_stringer

offset_rib=(skin_lenght)/4
distance_rib=skin_lenght-offset_rib*2



#PIASTRA DI RIPARAZIONE SKIN
rip_lenght1=150
rip_depth1=1
rip_width1=distance_stringer-rib_cut_lenght/2-str_lenght/2
n_rip1=2
```

```
###(x1=fattore moltiplicativo rispetto alla zona interna)
x1=1.2

#PIASTRA DI RIPARAZIONE stringer
rip_lenght2=150
rip_depth2=1
rip_rad2=2.9972
rip_width2=(str_lenght-2*str_depth)/2-rip_depth2/2
rip_height2=str_height-(rip_depth2+str_depth)/2
n_rip2=2



#Reference Points e Fastener
n_points1=5
n_points2=1
n_rows1=10
n_rows2=10
offset_edge1=10
offset_edge2=10
offset_point1=10
offset_point2=10
distance_rows1=(rip_lenght1-2*offset_edge1)/(n_rows1-1)
distance_rows2=(rip_lenght2-2*offset_edge2)/(n_rows2-1)
distance_points1=(rip_width1-2*offset_point1)/(n_points1-1)
fast_rad=3
lato_quad=min(fast_rad*1.5,distance_rows2/2)

##zona di non bullonatura
x_noHole=75
y_noHole=75



#Posizionamento variabile cricche
#CRICCA
#Numero cricche
#n_cr=2

#Posizione cricche rispetto allo skin
### B
#x_cr=[454,rib_width+offset_rib+rip_lenght1/2,454,skin_lenght-
(rib_width+offset_rib+rip_lenght1/2)]
#y_cr=[152.4,457.2]

### A
#x_cr=[offset_rib+rib_width+rip_lenght1/2,offset_rib+rib_width+rip_lenght1/2+1.0*x1*rip_len
ght1]
#y_cr=[304.8,304.8]

#Geometria Cricche
#cr_width=[152.4,152.4]
#cr_lenght=[6.35,6.35]



#---------#
#GEOMETRIA#
#---------#

#Creazione materiale
mdb.models['Model-1'].Material(name='IMS/977-2')
```

```
mdb.models['Model-1'].materials['IMS/977-
2'].Elastic(table=((E1,E2,E3,ni12,ni13,ni23,G12,G13,G23),                                    ),
type=ENGINEERING_CONSTANTS)

#Creazione materiale AL 2219/T87
mdb.models['Model-1'].Material(name='AL 2219/T87')
mdb.models['Model-1'].materials['AL 2219/T87'].Density(table=((ro, ), ))
mdb.models['Model-1'].materials['AL 2219/T87'].Elastic(table=((E,ni), ))


#SKIN
#Creazione skin
mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=1000.0)
mdb.models['Model-1'].sketches['__profile__'].rectangle(point1=(0.0, 0.0),
    point2=(skin_lenght,skin_width))
mdb.models['Model-1'].Part(dimensionality=THREE_D, name='skin', type=
    DEFORMABLE_BODY)
mdb.models['Model-1'].parts['skin'].BaseShell(sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']


#Creazione cricca
for i in range (n_cr):
    mdb.models['Model-1'].ConstrainedSketch(gridSpacing=59.51, name='__profile__',
    sheetSize=2380.57, transform=
    mdb.models['Model-1'].parts['skin'].MakeSketchTransform(
    sketchPlane=mdb.models['Model-
1'].parts['skin'].faces.findAt((skin_lenght/2,skin_width/2,0),),
    sketchPlaneSide=SIDE1,
    sketchUpEdge=mdb.models['Model-
1'].parts['skin'].edges.findAt((skin_lenght,skin_width/2,0),),
    sketchOrientation=RIGHT, origin=(x_cr[i],y_cr[i],0)))
    mdb.models['Model-1'].parts['skin'].projectReferencesOntoSketch(filter=
    COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])
    mdb.models['Model-1'].sketches['__profile__'].rectangle(point1=(0, 0),
    point2=(cr_lenght[i], cr_width[i]))
    mdb.models['Model-1'].parts['skin'].CutExtrude(flipExtrudeDirection=OFF,
    sketch=mdb.models['Model-1'].sketches['__profile__'], sketchOrientation=
    RIGHT,                                                  sketchPlane=mdb.models['Model-
1'].parts['skin'].faces.findAt((skin_lenght/2,skin_width/2,0),),
    sketchPlaneSide=SIDE1, sketchUpEdge=
    mdb.models['Model-1'].parts['skin'].edges.findAt((skin_lenght,skin_width/2,0),))
    del mdb.models['Model-1'].sketches['__profile__']



#Layup skin
##sdr layup skin
sdr_layup_skin=mdb.models['Model-1'].parts['skin'].DatumCsysByThreePoints(coordSysType=
    CARTESIAN, name='Datum csys-skin', origin=(0.0, 0.0, 0.0), point1=(1.0,
    0.0, 0.0), point2=(1.0, 1.0, 0.0))


##Caratteristiche layup skin
mdb.models['Model-1'].parts['skin'].CompositeLayup(description='', elementType=
    SHELL, name='CompositeLayup-skin',  offsetType=MIDDLE_SURFACE, symmetric=False,
    thicknessAssignment=FROM_SECTION)

mdb.models['Model-1'].parts['skin'].compositeLayups['CompositeLayup-skin'].Section(
    integrationRule=SIMPSON, poissonDefinition=DEFAULT, preIntegrate=OFF,
    temperature=GRADIENT, thicknessType=UNIFORM, useDensity=OFF)
```

```
mdb.models['Model-1'].parts['skin'].compositeLayups['CompositeLayup-
skin'].ReferenceOrientation(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0,
    axis=AXIS_3, fieldName='',
    localCsys=mdb.models['Model-1'].parts['skin'].datums[sdr_layup_skin.id],
    orientationType=SYSTEM,
    stackDirection=STACK_3)


#Orientamento layup skin
for i in range(n_layup_skin):
    mdb.models['Model-1'].parts['skin'].compositeLayups['CompositeLayup-
skin'].CompositePly(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
    , axis=AXIS_3, material='IMS/977-2', numIntPoints=3, orientationType=
    SPECIFY_ORIENT,       orientationValue=layup_skin_or[i][0],       plyName='Ply'+str(i),
region=Region(
    faces=mdb.models['Model-
1'].parts['skin'].faces.findAt(((skin_lenght/100,skin_width/100,0),)),),
    suppressed=False, thickness=skin_depth/n_layup_skin, thicknessType=SPECIFY_THICKNESS)



#Creazione piani di supporto per i vincoli
xz_skin=mdb.models['Model-1'].parts['skin'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=XZPLANE)
yz_skin=mdb.models['Model-1'].parts['skin'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=YZPLANE)



#STRINGER
#Creazione base stringer
mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=100.0)
mdb.models['Model-1'].sketches['__profile__'].rectangle(point1=(0.0, 0.0),
    point2=(str_lenght, skin_lenght))
mdb.models['Model-1'].Part(dimensionality=THREE_D, name='stringer', type=
    DEFORMABLE_BODY)
mdb.models['Model-1'].parts['stringer'].BaseShell(sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']



#Creazione rinforzo stringer
mdb.models['Model-1'].ConstrainedSketch(gridSpacing=45.81, name='__profile__',
    sheetSize=1832.42, transform=
    mdb.models['Model-1'].parts['stringer'].MakeSketchTransform(
    sketchPlane=mdb.models['Model-
1'].parts['stringer'].faces.findAt((str_lenght/2,skin_lenght/2,0), ),
    sketchPlaneSide=SIDE1,
    sketchUpEdge=mdb.models['Model-
1'].parts['stringer'].edges.findAt((str_lenght,skin_lenght/2,0), ),
    sketchOrientation=RIGHT, origin=(str_lenght/2,skin_lenght/2,0)))
mdb.models['Model-1'].parts['stringer'].projectReferencesOntoSketch(filter=
    COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0, -skin_lenght/2),
    point2=(0, skin_lenght/2))
mdb.models['Model-1'].parts['stringer'].ShellExtrude(depth=str_height,
    flipExtrudeDirection=OFF, sketch=
    mdb.models['Model-1'].sketches['__profile__'], sketchOrientation=RIGHT,
    sketchPlane=mdb.models['Model-
1'].parts['stringer'].faces.findAt((str_lenght/2,skin_lenght/2,0), ),
    sketchPlaneSide=SIDE1, sketchUpEdge=
```

```
    mdb.models['Model-1'].parts['stringer'].edges.findAt((str_lenght,skin_lenght/2,0), ))
del mdb.models['Model-1'].sketches['__profile__']



#sdr layup stringer base
sdr_str_base=mdb.models['Model-1'].parts['stringer'].DatumCsysByThreePoints(coordSysType=
    CARTESIAN, name='Datum csys-str-base', origin=(0.0, 0.0, 0.0), point1=(1.0,
    0.0, 0.0), point2=(1.0, 1.0, 0.0))


#Caratteristiche layup stringer base
mdb.models['Model-1'].parts['stringer'].CompositeLayup(description='', elementType=
    SHELL, name='CompositeLayup-base',  offsetType=MIDDLE_SURFACE, symmetric=False,
    thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['stringer'].compositeLayups['CompositeLayup-base'].Section(
    integrationRule=SIMPSON, poissonDefinition=DEFAULT, preIntegrate=OFF,
    temperature=GRADIENT, thicknessType=UNIFORM, useDensity=OFF)
mdb.models['Model-1'].parts['stringer'].compositeLayups['CompositeLayup-
base'].ReferenceOrientation(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
    , axis=AXIS_3, fieldName='', localCsys=
    mdb.models['Model-1'].parts['stringer'].datums[sdr_str_base.id],
orientationType=SYSTEM,
    stackDirection=STACK_3)


#Orientamento layup base stringer
for i in range(n_layup_str):
    mdb.models['Model-1'].parts['stringer'].compositeLayups['CompositeLayup-
base'].CompositePly(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
    , axis=AXIS_3, material='IMS/977-2', numIntPoints=3, orientationType=
    SPECIFY_ORIENT,        orientationValue=layup_str_or[i][0],        plyName='Ply'+str(i),
region=Region(
    faces=mdb.models['Model-
1'].parts['stringer'].faces.findAt(((str_lenght/100,skin_lenght/100,0),),

((str_lenght*90/100,skin_lenght*90/100,0),)
                                                                    ),),
    suppressed=False, thickness=str_depth/n_layup_str, thicknessType=SPECIFY_THICKNESS)


#sdr layup stringer rinforzo
sdr_str_rinf=mdb.models['Model-1'].parts['stringer'].DatumCsysByThreePoints(coordSysType=
    CARTESIAN, name='Datum csys-str-rinf', origin=(0.0, 0.0, 0.0), point1=(0.0,
    0.0, 1.0), point2=(0.0, -1.0, 1.0))


#Caratteristiche layup rinforzo stringer
mdb.models['Model-1'].parts['stringer'].CompositeLayup(description='', elementType=
    SHELL, name='CompositeLayup-rinf',  offsetType=MIDDLE_SURFACE, symmetric=False,
    thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['stringer'].compositeLayups['CompositeLayup-rinf'].Section(
    integrationRule=SIMPSON, poissonDefinition=DEFAULT, preIntegrate=OFF,
    temperature=GRADIENT, thicknessType=UNIFORM, useDensity=OFF)
mdb.models['Model-1'].parts['stringer'].compositeLayups['CompositeLayup-
rinf'].ReferenceOrientation(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
    , axis=AXIS_3, fieldName='', localCsys=
    mdb.models['Model-1'].parts['stringer'].datums[sdr_str_rinf.id],
orientationType=SYSTEM,
    stackDirection=STACK_3)
```

```
#Orientamento layup rinforzo stringer
for i in range(n_layup_str):
    mdb.models['Model-1'].parts['stringer'].compositeLayups['CompositeLayup-
rinf'].CompositePly(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
    , axis=AXIS_3, material='IMS/977-2', numIntPoints=3, orientationType=
    SPECIFY_ORIENT,        orientationValue=layup_str_or[i][0],        plyName='Ply'+str(i),
region=Region(
    faces=mdb.models['Model-
1'].parts['stringer'].faces.findAt(((str_lenght/2,skin_lenght/100,str_height/2),)),),
    suppressed=False, thickness=str_depth/n_layup_str, thicknessType=SPECIFY_THICKNESS)

##Orientamento layup rinforzo stringer 2
#Il ply e' nel complesso formato da 24 ply, simmetrici tra di loro
for i in range(n_layup_str):
    mdb.models['Model-1'].parts['stringer'].compositeLayups['CompositeLayup-
rinf'].CompositePly(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
    , axis=AXIS_3, material='IMS/977-2', numIntPoints=3, orientationType=
    SPECIFY_ORIENT, orientationValue=45.0, plyName='Ply-'+str(i)+'-Copy1', region=
    Region(
    faces=mdb.models['Model-
1'].parts['stringer'].faces.findAt(((str_lenght/2,skin_lenght/100,str_height/2),)),),
    suppressed=False, thickness=str_depth/n_layup_str, thicknessType=SPECIFY_THICKNESS)


#Creazione piani di supporto per i vincoli
xz_str=mdb.models['Model-1'].parts['stringer'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=XZPLANE)
yz_str=mdb.models['Model-
1'].parts['stringer'].DatumPlaneByPrincipalPlane(offset=str_lenght/2,
   principalPlane=YZPLANE)
xy_str=mdb.models['Model-1'].parts['stringer'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=XYPLANE)



#RIB
#Creazione rib profilo a L
mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=100.0)
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0.0,   0.0),   point2=(rib_width,
0.0))
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(rib_width,                  0.0),
point2=(rib_width, rib_height))
mdb.models['Model-1'].Part(dimensionality=THREE_D, name='rib', type=DEFORMABLE_BODY)
mdb.models['Model-1'].parts['rib'].BaseShellExtrude(depth=skin_width,
                                                    sketch=mdb.models['Model-
1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']

#Taglio profilo a L
mdb.models['Model-1'].ConstrainedSketch(gridSpacing=38.47, name='__profile__',
    sheetSize=1538.9, transform=
    mdb.models['Model-1'].parts['rib'].MakeSketchTransform(
    sketchPlane=mdb.models['Model-
1'].parts['rib'].faces.findAt((rib_width,rib_height/2,skin_width/2),),
    sketchPlaneSide=SIDE1,
    sketchUpEdge=mdb.models['Model-
1'].parts['rib'].edges.findAt((rib_width,rib_height/2,skin_width),),
    sketchOrientation=RIGHT, origin=(rib_width, 0.0, 0.0)))
mdb.models['Model-1'].parts['rib'].projectReferencesOntoSketch(filter=
    COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0, -rib_cut_height),
```

```
                                                    point2=(rib_base/2+rib_cut_lenght,      -
rib_cut_height))
#punti periodici, funzione del numero di stringer
for i in range(n_stringer-1):
    mdb.models['Model-
1'].sketches['__profile__'].Line(point1=((2*i+1)*rib_base/2+(i+1)*rib_cut_lenght,-
rib_cut_height),

point2=((2*i+1)*rib_base/2+(i+1)*rib_cut_lenght,rib_cut_height/2))
    mdb.models['Model-
1'].sketches['__profile__'].Line(point1=((2*i+1)*rib_base/2+(i+1)*rib_cut_lenght,rib_cut_he
ight/2),

point2=((2*(i+1)+1)*rib_base/2+(i+1)*rib_cut_lenght,rib_cut_height/2))
    mdb.models['Model-
1'].sketches['__profile__'].Line(point1=((2*(i+1)+1)*rib_base/2+(i+1)*rib_cut_lenght,rib_cu
t_height/2),

point2=((2*(i+1)+1)*rib_base/2+(i+1)*rib_cut_lenght,-rib_cut_height))
    mdb.models['Model-
1'].sketches['__profile__'].Line(point1=((2*(i+1)+1)*rib_base/2+(i+1)*rib_cut_lenght,-
rib_cut_height),

point2=((2*(i+1)+1)*rib_base/2+(i+2)*rib_cut_lenght,-rib_cut_height))
#punti non periodici
mdb.models['Model-1'].sketches['__profile__'].Line(point1=((2*(n_stringer-
1)+1)*rib_base/2+n_stringer*rib_cut_lenght,-rib_cut_height),
                                                    point2=(skin_width,-rib_cut_height))
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(skin_width,-rib_cut_height),
                                                    point2=(skin_width,rib_cut_height))
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(skin_width,rib_cut_height),
                                                    point2=(0,rib_cut_height))
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0,rib_cut_height),
                                                    point2=(0,-rib_cut_height))

mdb.models['Model-1'].parts['rib'].CutExtrude(flipExtrudeDirection=OFF, sketch=
    mdb.models['Model-1'].sketches['__profile__'], sketchOrientation=RIGHT,
    sketchPlane=mdb.models['Model-
1'].parts['rib'].faces.findAt((rib_width,rib_height/2,skin_width/2),),
    sketchPlaneSide=SIDE1,
    sketchUpEdge=mdb.models['Model-
1'].parts['rib'].edges.findAt((rib_width,rib_height/2,skin_width),),)
del mdb.models['Model-1'].sketches['__profile__']



#sdr rib base
sdr_rib_base=mdb.models['Model-1'].parts['rib'].DatumCsysByThreePoints(coordSysType=
    CARTESIAN, name='Datum csys-rib-base', origin=(0.0, 0.0, 0.0), point1=(1.0,
    0.0, 0.0), point2=(1.0, 0.0, 1.0))



#Caratteristiche layup rib base
mdb.models['Model-1'].parts['rib'].CompositeLayup(description='', elementType=
    SHELL, name='CompositeLayup-rib-base',  offsetType=MIDDLE_SURFACE, symmetric=False,
    thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['rib'].compositeLayups['CompositeLayup-rib-base'].Section(
    integrationRule=SIMPSON, poissonDefinition=DEFAULT, preIntegrate=OFF,
    temperature=GRADIENT, thicknessType=UNIFORM, useDensity=OFF)
mdb.models['Model-1'].parts['rib'].compositeLayups['CompositeLayup-rib-
base'].ReferenceOrientation(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
    , axis=AXIS_3, fieldName='', localCsys=
```

```
    mdb.models['Model-1'].parts['rib'].datums[sdr_rib_base.id], orientationType=SYSTEM,
    stackDirection=STACK_3)


#Orientamento layup rib base
for i in range(n_layup_rib):
    for j in range (1,n_stringer):
        mdb.models['Model-1'].parts['rib'].compositeLayups['CompositeLayup-rib-
base'].CompositePly(
        additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
        , axis=AXIS_3, material='IMS/977-2', numIntPoints=3, orientationType=
        SPECIFY_ORIENT,  orientationValue=layup_rib_or[i][0],  plyName='Ply'+str(i)+str(j),
region=Region(
        faces=mdb.models['Model-
1'].parts['rib'].faces.findAt(((rib_width/2,0,j*(rib_cut_lenght+rib_base)),), ))
         ,                suppressed=False,                thickness=rib_depth/n_layup_rib,
thicknessType=SPECIFY_THICKNESS)


#sdr rib rinforzo
sdr_rib_rinf=mdb.models['Model-1'].parts['rib'].DatumCsysByThreePoints(coordSysType=
    CARTESIAN, name='Datum csys-rib-rinf', origin=(0.0, 0.0, 0.0), point1=(0.0,
    0.0, 1.0), point2=(0.0, -1.0, 1.0))


#Caratteristiche layup rib rinforzo
mdb.models['Model-1'].parts['rib'].CompositeLayup(description='', elementType=
    SHELL, name='CompositeLayup-rib-rinf',   offsetType=MIDDLE_SURFACE, symmetric=False,
    thicknessAssignment=FROM_SECTION)
mdb.models['Model-1'].parts['rib'].compositeLayups['CompositeLayup-rib-rinf'].Section(
    integrationRule=SIMPSON, poissonDefinition=DEFAULT, preIntegrate=OFF,
    temperature=GRADIENT, thicknessModulus=None, useDensity=OFF)
mdb.models['Model-1'].parts['rib'].compositeLayups['CompositeLayup-rib-
rinf'].ReferenceOrientation(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
    , axis=AXIS_3, fieldName='', localCsys=
    mdb.models['Model-1'].parts['rib'].datums[sdr_rib_rinf.id], orientationType=SYSTEM,
    stackDirection=STACK_3)


#Orientamento layup rib rinforzo
for i in range(n_layup_rib):
    mdb.models['Model-1'].parts['rib'].compositeLayups['CompositeLayup-rib-
rinf'].CompositePly(
    additionalRotationField='', additionalRotationType=ROTATION_NONE, angle=0.0
    , axis=AXIS_3, material='IMS/977-2', numIntPoints=3, orientationType=
    SPECIFY_ORIENT,       orientationValue=layup_rib_or[i][0],       plyName='Ply-'+str(i),
region=Region(
    faces=mdb.models['Model-
1'].parts['rib'].faces.findAt(((rib_width,rib_height/2,skin_width/2),),))
    , suppressed=False, thickness=rib_depth/n_layup_rib, thicknessType=SPECIFY_THICKNESS)


#Creazione piani di supporto per i vincoli
xy_rib=mdb.models['Model-1'].parts['rib'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=XYPLANE)
yz_rib=mdb.models['Model-1'].parts['rib'].DatumPlaneByPrincipalPlane(offset=rib_width,
    principalPlane=YZPLANE)



#STRINGER CRICCATI
#Il vettore str_cr contiene l'elenco non ripetuto degli stringers intaccati da una/piu
cricche
```

```
#Il vettore str_cr_cr contiene l'elenco degli stringers intaccati da una cricca, con
l'indicazione della cricca
    #Es. cricca 1 su stringer 2:str_cr-cr=((2,1))
str_cr=[]
str_cr_cr=[]


#Creazione cricca sullo stringer
for j in range (n_stringer):
    for i in range (n_cr):
        if ((y_cr[i] < ((j+1)*distance_stringer+(offset_stringer-str_lenght)/2) \
        and      cr_width[i]      >      (-y_cr[i]+(j+1)*distance_stringer+(offset_stringer-
str_lenght)/2))) \
        or ((y_cr[i] > ((j+1)*distance_stringer+(offset_stringer-str_lenght)/2) \
        and y_cr[i] < ((j+1)*distance_stringer+(offset_stringer+str_lenght)/2))):
            str_cr.append(j+1)
            str_cr_cr.append((j+1,i))
            str_cr = list(set(str_cr))
            n_str_cr=len(str_cr)

for k in range (n_str_cr):
            mdb.models['Model-1'].Part(name='stringer-cricca'+str(k+1), objectToCopy=
            mdb.models['Model-1'].parts['stringer'])


#posizionamento cricca nel sdr relativo allo stringer
cr_width_str=[]
for i in range (len(str_cr_cr)):
    cr_width_str.append(0)

#posizionamento cricca nel sdr relativo allo stringer
x_cr_str=[]
for i in range (len(str_cr_cr)):
    x_cr_str.append(0)


for k in range (len(str_cr_cr)):
                cr_width_str[k]=min(str_lenght,
                        cr_width[str_cr_cr[k][1]]+
                        y_cr[str_cr_cr[k][1]]-((str_cr_cr[k][0])*distance_stringer+
                                (offset_stringer-str_lenght)/2))
                x_cr_str[k]=max(0,y_cr[str_cr_cr[k][1]]-
                    ((str_cr_cr[k][0])*distance_stringer+
                     (offset_stringer-str_lenght)/2))

                mdb.models['Model-1'].ConstrainedSketch(gridSpacing=45.82,
name='__profile__',
                sheetSize=1832.81, transform=
                mdb.models['Model-1'].parts['stringer-
cricca'+str(str_cr.index(str_cr_cr[k][0])+1)].MakeSketchTransform(
                sketchPlane=mdb.models['Model-1'].parts['stringer-
cricca'+str(str_cr.index(str_cr_cr[k][0])+1)].
                    faces.findAt((str_lenght*1.01/2,skin_lenght/2,0),),
                sketchPlaneSide=SIDE1,
                sketchUpEdge=mdb.models['Model-1'].parts['stringer-
cricca'+str(str_cr.index(str_cr_cr[k][0])+1)].
                    edges.findAt((0,skin_lenght/2,0),),
                sketchOrientation=RIGHT,                                origin=(str_lenght-
x_cr_str[k],x_cr[str_cr_cr[k][1]],0)))
                mdb.models['Model-1'].parts['stringer-
cricca'+str(str_cr.index(str_cr_cr[k][0])+1)].projectReferencesOntoSketch(
                filter=COPLANAR_EDGES,                                  sketch=mdb.models['Model-
1'].sketches['__profile__'])
                mdb.models['Model-1'].sketches['__profile__'].rectangle(point1=(0,0)
```

```
                , point2=(cr_width_str[k],-cr_lenght[str_cr_cr[k][1]]))
               mdb.models['Model-1'].parts['stringer-
cricca'+str(str_cr.index(str_cr_cr[k][0])+1)].CutExtrude(flipExtrudeDirection=
               ON, sketch=mdb.models['Model-1'].sketches['__profile__'],
               sketchOrientation=RIGHT, sketchPlane=
               mdb.models['Model-1'].parts['stringer-
cricca'+str(str_cr.index(str_cr_cr[k][0])+1)]
                     .faces.findAt((str_lenght*1.01/2,skin_lenght/2,0),),
               sketchPlaneSide=SIDE1,
               sketchUpEdge=mdb.models['Model-1'].parts['stringer-
cricca'+str(str_cr.index(str_cr_cr[k][0])+1)]
                      .edges.findAt((0,skin_lenght/2,0),))
               del mdb.models['Model-1'].sketches['__profile__']



#PIASTRA DI RIPARAZIONE SKIN
#Creazione piastra_skin
mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=1000.0)
mdb.models['Model-1'].sketches['__profile__'].rectangle(point1=(0.0, 0.0),
    point2=(rip_lenght1,rip_width1))
mdb.models['Model-1'].Part(dimensionality=THREE_D, name='rip1', type=
    DEFORMABLE_BODY)
mdb.models['Model-1'].parts['rip1'].BaseShell(sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']



#Assegnazione sezione shell AL rip1
mdb.models['Model-1'].parts['rip1'].Set(faces=
    mdb.models['Model-1'].parts['rip1'].faces.findAt(((rip_lenght1,rip_width1/2,0), ),
    ), name='Set-1')
mdb.models['Model-1'].HomogeneousShellSection(idealization=NO_IDEALIZATION,
    integrationRule=SIMPSON, material='AL 2219/T87', name='Section-1',
    numIntPts=5, poissonDefinition=DEFAULT, preIntegrate=OFF, temperature=
    GRADIENT, thickness=rip_depth1, thicknessField='', thicknessModulus=None,
    thicknessType=UNIFORM, useDensity=OFF)
mdb.models['Model-1'].parts['rip1'].SectionAssignment(offset=0.0, offsetField=
    '', offsetType=MIDDLE_SURFACE, region=
    mdb.models['Model-1'].parts['rip1'].sets['Set-1'], sectionName='Section-1',
    thicknessAssignment=FROM_SECTION)



#PIASTRA DI RIPARAZIONE STRINGER
#Creazione piastra_stringer
mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=1000.0)
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(rip_rad2, 0),
                                                   point2=(rip_width2,0))
mdb.models['Model-1'].sketches['__profile__'].Line(point1=(0,rip_height2),
                                                   point2=(0,rip_rad2))
mdb.models['Model-1'].sketches['__profile__'].ArcByCenterEnds(center=(rip_rad2,rip_rad2),
                                                              direction=CLOCKWISE,
                                                              point1=(rip_rad2, 0),
                                                              point2=(0, rip_rad2))
mdb.models['Model-1'].Part(dimensionality=THREE_D, name='rip2', type=
    DEFORMABLE_BODY)

mdb.models['Model-1'].parts['rip2'].BaseShellExtrude(depth=rip_lenght2,
                                                     sketch=mdb.models['Model-
1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']
```

```
#Assegnazione sezione shell AL rip2
mdb.models['Model-1'].parts['rip2'].Set(faces=
    mdb.models['Model-1'].parts['rip2'].faces.findAt(((rip_width2/2,0,rip_lenght2/2), ),
                                                     ((0,rip_height2,rip_lenght2/2), ),
                                                     ((rip_rad2*(1-cos(pi/4)),rip_rad2*(1-
sin(pi/4)),rip_lenght2/2), ),
    ), name='Set-1')
mdb.models['Model-1'].HomogeneousShellSection(idealization=NO_IDEALIZATION,
    integrationRule=SIMPSON, material='AL 2219/T87', name='Section-1',
    numIntPts=5, poissonDefinition=DEFAULT, preIntegrate=OFF, temperature=
    GRADIENT, thickness=rip_depth2, thicknessField='', thicknessModulus=None,
    thicknessType=UNIFORM, useDensity=OFF)
mdb.models['Model-1'].parts['rip2'].SectionAssignment(offset=0.0, offsetField=
    '', offsetType=MIDDLE_SURFACE, region=
    mdb.models['Model-1'].parts['rip2'].sets['Set-1'], sectionName='Section-1',
    thicknessAssignment=FROM_SECTION)



#---------#
#---------#
#   MESH  #
#---------#
#---------#



#Mesh skin
#Element type
mdb.models['Model-1'].parts['skin'].setElementType(elemTypes=(ElemType(
    elemCode=S4R, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT), ElemType(elemCode=S3, elemLibrary=STANDARD),), regions=(
    mdb.models['Model-1'].parts['skin'].faces.findAt((skin_lenght/100,skin_width/100,0),
),))


#Tecnica di mesh locale
for i in range (n_cr):
    mdb.models['Model-1'].parts['skin'].setMeshControls(algorithm=ADVANCING_FRONT,
    regions=mdb.models['Model-1'].parts['skin'].faces.findAt(((
        x_cr[i]-cr_lenght[i]/100,y_cr[i]-cr_width[i]/100,0), ),))


#Element size locale cricca
for i in range (n_cr):
    mdb.models['Model-1'].parts['skin'].seedEdgeBySize(constraint=FINER,
    deviationFactor=0.1, edges=
    mdb.models['Model-1'].parts['skin'].edges.findAt(
        ((x_cr[i]+cr_lenght[i]/2,y_cr[i],0),),
        ((x_cr[i]+cr_lenght[i]/2,y_cr[i]+cr_width[i],0),),
        ((x_cr[i],y_cr[i]+cr_width[i]/2,0),),
        ((x_cr[i]+cr_lenght[i],y_cr[i]+cr_width[i]/2,0),),
         ),
        minSizeFactor=0.1,size=1)



#Partizione skin zona riparazione
## dimensioni zona di transizione
```

```
###(x1=fattore moltiplicativo rispetto alla zona interna)
x1=1.2
for i in range (n_cr):
    mdb.models['Model-
1'].parts['skin'].faces.findAt(((skin_lenght/100,skin_width/100,0),),)
    mdb.models['Model-1'].ConstrainedSketch(gridSpacing=59.51, name='__profile__',
    sheetSize=2380.57, transform=
    mdb.models['Model-1'].parts['skin'].MakeSketchTransform(
    sketchPlane=mdb.models['Model-1'].parts['skin'].faces.findAt((skin_lenght/100,
skin_width/100, 0),),
    sketchPlaneSide=SIDE1,
    sketchUpEdge=mdb.models['Model-1'].parts['skin'].edges.findAt((skin_lenght,
skin_width/100, 0),),
    sketchOrientation=RIGHT,                                origin=(x_cr[i]+cr_lenght[i]/2,
str_cr_cr[i][0]*distance_stringer+offset_stringer/2, 0)))
    mdb.models['Model-1'].parts['skin'].projectReferencesOntoSketch(filter=
    COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])
    mdb.models['Model-1'].sketches['__profile__'].rectangle(point1=(-rip_lenght1/2,-
rip_width1-str_lenght/2),

point2=(rip_lenght1/2,rip_width1+str_lenght/2))
    mdb.models['Model-1'].sketches['__profile__'].rectangle(point1=(-x1*rip_lenght1/2,-
x1*(rip_width1+str_lenght/2)),

point2=(x1*rip_lenght1/2,x1*(rip_width1+str_lenght/2)))
    mdb.models['Model-1'].parts['skin'].PartitionFaceBySketch(faces=
    mdb.models['Model-1'].parts['skin'].faces.findAt((skin_lenght/100,      skin_width/100,
0),),
    sketch=mdb.models['Model-1'].sketches['__profile__'], sketchUpEdge=
    mdb.models['Model-1'].parts['skin'].edges.findAt((skin_lenght, skin_width/100, 0),))
    del mdb.models['Model-1'].sketches['__profile__']


#Tecnica di mesh locale
for i in range (n_cr):
    mdb.models['Model-1'].parts['skin'].setMeshControls(algorithm=ADVANCING_FRONT,
    regions=mdb.models['Model-1'].parts['skin'].faces.findAt(((
        x_cr[i],y_cr[i]+cr_width[i]/2-0.5*(1+x1)*(rip_width1+str_lenght/2),0), ),))


#Element size locale zona riparazione
for i in range (n_cr):
    mdb.models['Model-1'].parts['skin'].seedEdgeBySize(constraint=FINER,
    deviationFactor=0.1, edges=
    mdb.models['Model-1'].parts['skin'].edges.findAt(
        ((x_cr[i]+cr_lenght[i]/2,str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),0),),

((x_cr[i]+cr_lenght[i]/2,str_cr_cr[i][0]*distance_stringer+offset_stringer/2+(rip_width1+st
r_lenght/2),0),),
        ((x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,str_cr_cr[i][0]*distance_stringer+offset_stringer/2,0),),

((x_cr[i]+cr_lenght[i]/2+rip_lenght1/2,str_cr_cr[i][0]*distance_stringer+offset_stringer/2,
0),
         )),
        minSizeFactor=0.1,size=5)


#Partizione Fastener
##Creazione cerchio di simulazione fastener
for i in range (n_cr):
    mdb.models['Model-1'].parts['skin'].faces.findAt((
```

```
        (x_cr[i],str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
0.5*(rip_width1+str_lenght/2),0),),)
    mdb.models['Model-1'].ConstrainedSketch(gridSpacing=59.51, name='__profile__',
        sheetSize=2380.57, transform=
        mdb.models['Model-1'].parts['skin'].MakeSketchTransform(
        sketchPlane=mdb.models['Model-1'].parts['skin'].faces.findAt(
        (x_cr[i],str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
0.5*(rip_width1+str_lenght/2),0),),
        sketchPlaneSide=SIDE1,
        sketchUpEdge=mdb.models['Model-1'].parts['skin'].edges.findAt((
        x_cr[i],str_cr_cr[i][0]*distance_stringer+offset_stringer/2,0),),
        sketchOrientation=RIGHT,                              origin=(x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),0)))
    mdb.models['Model-1'].parts['skin'].projectReferencesOntoSketch(filter=
        COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])

#rip1
    for k in range (n_points1):
        for q in range (n_rows1):
            if      rip_lenght1/2-x_noHole/2      <      offset_edge1+distance_rows1*q      <
rip_lenght1/2+x_noHole/2 and\
               rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
                    center=(offset_edge1+distance_rows1*q,
distance_points1*k+offset_point1),
                    point1=(offset_edge1+fast_rad+distance_rows1*q,
distance_points1*k+offset_point1))

                mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
                 center=(offset_edge1+distance_rows1*q,
                        2*(rip_width1+str_lenght/2)-(distance_points1*k+offset_point1)),
                 point1=(offset_edge1+fast_rad+distance_rows1*q,
                        2*(rip_width1+str_lenght/2)-(distance_points1*k+offset_point1)))

#rip2
    for q in range (n_rows2):
        if       rip_lenght1/2-x_noHole/2       <       offset_edge2+distance_rows2*q       <
rip_lenght1/2+x_noHole/2:
            bandiera=1
        else:
            mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
            center=(offset_edge2+distance_rows2*q,
                        (rip_width1)+(offset_point2)),
            point1=(offset_edge2+fast_rad+distance_rows2*q,
                        (rip_width1)+(offset_point2)))

            mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
            center=(offset_edge2+distance_rows2*q,
                        rip_width1+str_lenght-offset_point2),
            point1=(offset_edge2+fast_rad+distance_rows2*q,
                        (rip_width1+str_lenght)-(offset_point2)))

    mdb.models['Model-1'].parts['skin'].PartitionFaceBySketch(faces=
        mdb.models['Model-1'].parts['skin'].faces.findAt(((
        x_cr[i],y_cr[i]+cr_width[i]/2-0.5*(rip_width1+rip_width2),0), ),
        ), sketch=mdb.models['Model-1'].sketches['__profile__'], sketchUpEdge=
        mdb.models['Model-1'].parts['skin'].edges.findAt((
        x_cr[i],y_cr[i]+cr_width[i]/2,0),))
    del mdb.models['Model-1'].sketches['__profile__']
```

```
#Mesh intorno dei fastener cerchio
for i in range (n_cr):
    for k in range (n_points1):
        for q in range (n_rows1):
            if      rip_lenght1/2-x_noHole/2      <      offset_edge2+distance_rows2*q      <
rip_lenght1/2+x_noHole/2 and\
               rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].parts['skin'].setMeshControls(regions=
                mdb.models['Model-1'].parts['skin'].faces.findAt(
                ((offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,

distance_points1*k+offset_point1+str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                    0),
                ), ), technique=SWEEP)


for i in range (n_cr):
    for k in range (n_points1):
        for q in range (n_rows1):
            if      rip_lenght1/2-x_noHole/2      <      offset_edge2+distance_rows2*q      <
rip_lenght1/2+x_noHole/2 and\
               rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].parts['skin'].setMeshControls(regions=
                mdb.models['Model-1'].parts['skin'].faces.findAt(
                ((offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                    rip_width1+str_lenght/2-
(distance_points1*k+offset_point1)+str_cr_cr[i][0]*distance_stringer+offset_stringer/2,
                    0),
                ), ), technique=SWEEP)


for i in range (n_cr):
    for q in range (n_rows2):
        if       rip_lenght1/2-x_noHole/2       <       offset_edge2+distance_rows2*q       <
rip_lenght1/2+x_noHole/2:
            bandiera=1
        else:
            mdb.models['Model-1'].parts['skin'].setMeshControls(regions=
                    mdb.models['Model-1'].parts['skin'].faces.findAt(
                    ((offset_edge1+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      offset_point2+str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
str_lenght/2,
                      0),
                    ), ), technique=SWEEP)

for i in range (n_cr):
    for q in range (n_rows2):
        if       rip_lenght1/2-x_noHole/2       <       offset_edge2+distance_rows2*q       <
rip_lenght1/2+x_noHole/2:
            bandiera=1
        else:
            mdb.models['Model-1'].parts['skin'].setMeshControls(regions=
                    mdb.models['Model-1'].parts['skin'].faces.findAt(
                    ((offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      str_lenght/2-
offset_point2+str_cr_cr[i][0]*distance_stringer+offset_stringer/2,
```

```
                    0),
                ), ), technique=SWEEP)


#Element size locale zona fastener rip1
for i in range (n_cr):
    for k in range (n_points1):
        for q in range (n_rows1):
            if     rip_lenght1/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght1/2+x_noHole/2 and\
               rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                if       fast_rad*1.1+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2-x_cr[i]>0:
                    mdb.models['Model-1'].parts['skin'].seedEdgeBySize(constraint=FINER,
                        deviationFactor=0.1, edges=
                        mdb.models['Model-1'].parts['skin'].edges.findAt(

((sin(pi/4)*fast_rad+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                            sin(pi/4)*fast_rad+distance_points1*k+offset_point1+
                            str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                            0),),

((sin(pi/4)*fast_rad+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                            sin(pi/4)*fast_rad+2*(rip_width1+str_lenght/2)-
(distance_points1*k+offset_point1)+
                            str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                            0),),

((sin(pi/4)*fast_rad+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                            -sin(pi/4)*fast_rad+distance_points1*k+offset_point1+
str_cr_cr[i][0]*distance_stringer+offset_stringer/2-(rip_width1+str_lenght/2),
                            0),),

((sin(pi/4)*fast_rad+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                            -sin(pi/4)*fast_rad+2*(rip_width1+str_lenght/2)-
(distance_points1*k+offset_point1)+
str_cr_cr[i][0]*distance_stringer+offset_stringer/2-(rip_width1+str_lenght/2),
                            0),),
                           ),
                        minSizeFactor=0.1,size=5)
                else:
                    mdb.models['Model-1'].parts['skin'].seedEdgeBySize(constraint=FINER,
                        deviationFactor=0.1, edges=
                        mdb.models['Model-1'].parts['skin'].edges.findAt(
                        ((-fast_rad+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
                            distance_points1*k+offset_point1+
str_cr_cr[i][0]*distance_stringer+offset_stringer/2-(rip_width1+str_lenght/2),
                            0),),
                        ((-fast_rad+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
                            2*(rip_width1+str_lenght/2)-(distance_points1*k+offset_point1)+
str_cr_cr[i][0]*distance_stringer+offset_stringer/2-(rip_width1+str_lenght/2),
                            0),),
                         ),
                        minSizeFactor=0.1,size=5)


#Element size locale zona fastener rip2
```

```
for i in range (n_cr):
    for q in range (n_rows2):
        if      rip_lenght1/2-x_noHole/2      <      offset_edge2+distance_rows2*q      <
rip_lenght1/2+x_noHole/2:
            bandiera=1
        else:
            if  fast_rad+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2-
x_cr[i]>0:
                mdb.models['Model-1'].parts['skin'].seedEdgeBySize(constraint=FINER,
                    deviationFactor=0.1, edges=
                    mdb.models['Model-1'].parts['skin'].edges.findAt(

((sin(pi/4)*fast_rad+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      sin(pi/4)*fast_rad+rip_width1+offset_point1+
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),

((sin(pi/4)*fast_rad+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      sin(pi/4)*fast_rad+rip_width1+str_lenght-offset_point2+
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),

((sin(pi/4)*fast_rad+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      -sin(pi/4)*fast_rad+rip_width1+offset_point1+
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),

((sin(pi/4)*fast_rad+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      -sin(pi/4)*fast_rad+rip_width1+str_lenght-offset_point2+
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
                     ),
                    minSizeFactor=0.1,size=5)
            else:
                mdb.models['Model-1'].parts['skin'].seedEdgeBySize(constraint=FINER,
                    deviationFactor=0.1, edges=
                    mdb.models['Model-1'].parts['skin'].edges.findAt(
                    ((-
sin(pi/4)*fast_rad+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      sin(pi/4)*fast_rad+rip_width1+offset_point1+
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
                    ((-
sin(pi/4)*fast_rad+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      sin(pi/4)*fast_rad+rip_width1+str_lenght-offset_point2+
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),

                    ((-
sin(pi/4)*fast_rad+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      -sin(pi/4)*fast_rad+rip_width1+offset_point1+
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
                    ((-
sin(pi/4)*fast_rad+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      -sin(pi/4)*fast_rad+rip_width1+str_lenght-offset_point2+
```

```
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
                     ),
                minSizeFactor=0.1,size=5)

#Tecnica di mesh globale
mdb.models['Model-1'].parts['skin'].setMeshControls(elemShape=QUAD,
                                                    algorithm=MEDIAL_AXIS,
    regions=mdb.models['Model-1'].parts['skin'].faces.findAt((
        (skin_lenght/100,skin_width/100,0),),) )

#Element size globale
mdb.models['Model-1'].parts['skin'].seedPart(deviationFactor=0.1,
    minSizeFactor=0.1, size=10.0)

#Creazione mesh
mdb.models['Model-1'].parts['skin'].generateMesh()


#Mesh stringer
#Element type
mdb.models['Model-1'].parts['stringer'].setElementType(elemTypes=(ElemType(
    elemCode=S4R, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT), ElemType(elemCode=S3, elemLibrary=STANDARD),), regions=(
    mdb.models['Model-1'].parts['stringer'].faces.findAt(
        ((str_lenght/2,skin_lenght/100,0), ),
        ((str_lenght/2,skin_lenght/100,str_height/2), ) ),))


#Element size
mdb.models['Model-1'].parts['stringer'].seedPart(deviationFactor=0.1,     minSizeFactor=0.1,
size=10.0)

#Creazione mesh
mdb.models['Model-1'].parts['stringer'].generateMesh()


#Mesh rib
#Element type
mdb.models['Model-1'].parts['rib'].setElementType(elemTypes=(ElemType(
    elemCode=S4R, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT), ElemType(elemCode=S3, elemLibrary=STANDARD),), regions=(
    mdb.models['Model-1'].parts['rib'].faces.findAt(
        ((rib_width,rib_height/2,skin_width/2),),
        ),))

for i in range (1,n_stringer):
    mdb.models['Model-1'].parts['rib'].setElementType(elemTypes=(ElemType(
    elemCode=S4R, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT), ElemType(elemCode=S4R, elemLibrary=STANDARD),), regions=(
    mdb.models['Model-1'].parts['rib'].faces.findAt(
        ((rib_width/2,0,i*(rib_base+rib_cut_lenght)),),
        ),))


#Tecnica di mesh
mdb.models['Model-1'].parts['rib'].setMeshControls(algorithm=MEDIAL_AXIS,
    regions=mdb.models['Model-1'].parts['rib'].faces.findAt(
        ((rib_width,rib_height/2,skin_width/2),),))

for i in range (1,n_stringer):
    mdb.models['Model-1'].parts['rib'].setMeshControls(algorithm=MEDIAL_AXIS,
    regions=mdb.models['Model-1'].parts['rib'].faces.findAt(
```

```
        ((rib_width/2,0,i*(rib_base+rib_cut_lenght)),),))


#Element size
mdb.models['Model-1'].parts['rib'].seedPart(deviationFactor=0.1, minSizeFactor=
    0.1, size=10.0)

#Creazione mesh
mdb.models['Model-1'].parts['rib'].generateMesh()


#Mesh stringer cricca
#Element type
for i in range (n_str_cr):
    mdb.models['Model-1'].parts['stringer-cricca'+str(i+1)].setElementType(elemTypes=(
    ElemType(elemCode=S4R, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT), ElemType(elemCode=S3, elemLibrary=STANDARD),), regions=(
    mdb.models['Model-1'].parts['stringer-cricca'+str(i+1)].faces.findAt(
    ((str_lenght/10,skin_lenght/100,0), ),
    ((str_lenght*9/10,skin_lenght/100,0), ),
    ((str_lenght/2,skin_lenght/100,str_height/2),),
    ((str_lenght/10,skin_lenght*90/100,0), ),
    ((str_lenght*9/10,skin_lenght*90/100,0), ),
    ((str_lenght/2,skin_lenght*90/100,str_height/2),),),))


#creazione vettore di supporto nel caso di piu cricche
#creazione vettore di supporto nel caso di piu cricche
a=[]
nsup=[]

for j in range (n_str_cr+1):
    nsup.append(0)
for j in range (n_str_cr):
    for i in range (len(str_cr_cr)):
        if str_cr_cr[i][0]==str_cr[j]:
            a.append(str_cr_cr[i].count(str_cr[j]))
            nsup[j]=sum(a)-nsup[j-1]
    if nsup[j]==0:
        nsup[j]=1

a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
#creazione piani offset per la partizione
        xz_strR=mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].DatumPlaneByOffset(flip=SIDE1,
        offset=x_cr[int(j+a)]+cr_lenght[int(j+a)]/2+rip_lenght1/2,
        plane=mdb.models['Model-1'].parts['stringer-cricca'+str(k)].datums[xz_str.id])

        xz_strL=mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].DatumPlaneByOffset(flip=SIDE1,
        offset=x_cr[int(j+a)]+cr_lenght[int(j+a)]/2-rip_lenght1/2,
        plane=mdb.models['Model-1'].parts['stringer-cricca'+str(k)].datums[xz_str.id])

#partizione stringer cricca in corrispondenza della riparazione
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].PartitionFaceByDatumPlane(
        datumPlane=mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].datums[xz_strL.id],
        faces=mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
            ((str_lenght/10,x_cr[int(j+a)]+cr_lenght[int(j+a)]/2-rip_lenght1/10,0),),
            ((str_lenght/2,x_cr[int(j+a)]+cr_lenght[int(j+a)]/2-
rip_lenght1/10,str_height/2),),
            ((str_lenght*9/10,x_cr[int(j+a)]+cr_lenght[int(j+a)]/2-rip_lenght1/10,0),)))
```

```
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].PartitionFaceByDatumPlane(
        datumPlane=mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].datums[xz_strR.id],
        faces=mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
            ((str_lenght/10,x_cr[int(j+a)]+cr_lenght[int(j+a)]/2+rip_lenght1/10,0),),

((str_lenght/2,x_cr[int(j+a)]+cr_lenght[int(j+a)]/2+rip_lenght1/10,str_height/2),),
            ((str_lenght*9/10,x_cr[int(j+a)]+cr_lenght[int(j+a)]/2+rip_lenght1/10,0),)))
    a=nsup[k-1]


#Element size locale zona riparazione
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].seedEdgeBySize(constraint=FINER
        , deviationFactor=0.1, edges=
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].edges.findAt(
        ((str_lenght/10,x_cr[int(j+a)],0), ),
        ((str_lenght*9/10,x_cr[int(j+a)],0), ),
        ((str_lenght/2,x_cr[int(j+a)],str_height/2),),
        ((str_lenght,x_cr[int(j+a)]*0.99,0),),
        ((str_lenght/10,x_cr[int(j+a)]+cr_lenght[int(j+a)],0), ),
        ((str_lenght*9/10,x_cr[int(j+a)]+cr_lenght[int(j+a)],0), ),
        ((str_lenght/2,x_cr[int(j+a)]+cr_lenght[int(j+a)],str_height/2),),
        ((str_lenght,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],0), ),
        ((0,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],0), ),
        ((0,x_cr[int(j+a)]*0.99,0), ),
        ((str_lenght/2,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],str_height),),
        ((str_lenght/2,x_cr[int(j+a)]*0.99,str_height),),

        ((str_lenght/2,x_cr[int(j+a)]*0.99,0),),
        ((str_lenght/2,x_cr[int(j+a)]-
rip_lenght2/2+0.5*cr_lenght[int(j+a)],str_height/2),),
        ((str_lenght/10,x_cr[int(j+a)]-rip_lenght2/2+0.5*cr_lenght[int(j+a)],0),),
        ((str_lenght*9/10,x_cr[int(j+a)]-rip_lenght2/2+0.5*cr_lenght[int(j+a)],0),),
        ((str_lenght/2,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],0),),

((str_lenght/2,x_cr[int(j+a)]+rip_lenght2/2+0.5*cr_lenght[int(j+a)],str_height/2),),
        ((str_lenght/10,x_cr[int(j+a)]+rip_lenght2/2+0.5*cr_lenght[int(j+a)],0),),
        ((str_lenght*9/10,x_cr[int(j+a)]+rip_lenght2/2+0.5*cr_lenght[int(j+a)],0),),

       ), minSizeFactor=0.1,size=5.0)
    a=nsup[k-1]


#Tecnica di mesh locale zona riparazione
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].setMeshControls(algorithm=
        MEDIAL_AXIS, elemShape=QUAD_DOMINATED, regions=
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
        ((str_lenght/10,x_cr[int(j+a)]-rip_lenght1/10,0), ),
        ((str_lenght*9/10,x_cr[int(j+a)]-rip_lenght1/10,0), ),
        ((str_lenght/2,x_cr[int(j+a)]-rip_lenght1/10,str_height/2),),
        ((str_lenght/10,x_cr[int(j+a)]+rip_lenght1/10,0), ),
        ((str_lenght*9/10,x_cr[int(j+a)]+rip_lenght1/10,0), ),
        ((str_lenght/2,x_cr[int(j+a)]+rip_lenght1/10,str_height/2),),))
    a=nsup[k-1]
```

```
#Tecnica di mesh
for j in range (n_str_cr):
    mdb.models['Model-1'].parts['stringer-
cricca'+str(j+1)].setMeshControls(algorithm=MEDIAL_AXIS,
    elemShape=QUAD,
    regions=mdb.models['Model-1'].parts['stringer-cricca'+str(j+1)].faces.findAt(
        ((str_lenght/10,skin_lenght/100,0), ),
        ((str_lenght*9/10,skin_lenght/100,0), ),
        ((str_lenght/2,skin_lenght/100,str_height/2),),
        ((str_lenght/10,skin_lenght*90/100,0), ),
        ((str_lenght*9/10,skin_lenght*90/100,0), ),
        ((str_lenght/2,skin_lenght*90/100,str_height/2),)))

#Element size
    mdb.models['Model-1'].parts['stringer-cricca'+str(j+1)].seedPart(deviationFactor=0.1,
minSizeFactor=0.1, size=10.0)


#Partizione Fastener sul rinforzo
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
                (str_lenght/2,x_cr[int(j+a)]*0.99,str_height/2),
                (str_lenght/2,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],str_height/2),)
        mdb.models['Model-1'].ConstrainedSketch(gridSpacing=22.88, name='__profile__',
            sheetSize=915.26, transform=
            mdb.models['Model-1'].parts['stringer-cricca'+str(k)].MakeSketchTransform(
            sketchPlane=mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
                (str_lenght/2,x_cr[int(j+a)]*0.99,str_height/2),),
            sketchPlaneSide=SIDE1,
            sketchUpEdge=mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].edges.findAt(
                (str_lenght/2,x_cr[int(j+a)],str_height/2),),
            sketchOrientation=RIGHT, origin=(0.0,0,0)))
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].projectReferencesOntoSketch(
            filter=COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])
        for q in range (n_rows2):
            if     rip_lenght2/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght2/2+x_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
                center=(x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2, str_height-offset_point2),
                point1=(x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2, str_height-offset_point2+fast_rad))
                mdb.models['Model-1'].sketches['__profile__'].rectangle(
                point1=(x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2-lato_quad,  str_height-offset_point2-
lato_quad)
                , point2=(x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2+lato_quad,               str_height-
offset_point2+lato_quad))

        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].PartitionFaceBySketch(faces=
            mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
                ((str_lenght/2,x_cr[int(j+a)]*0.99,str_height/2),),
                ((str_lenght/2,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],str_height/2),),),
             sketch=mdb.models['Model-1'].sketches['__profile__'], sketchUpEdge=
            mdb.models['Model-1'].parts['stringer-cricca'+str(k)].edges.findAt((
                str_lenght/2,x_cr[int(j+a)],str_height/2),))
        del mdb.models['Model-1'].sketches['__profile__']
    a=nsup[k-1]
```

```
#Partizione Fastener sulla base
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
                ((str_lenght/10,x_cr[int(j+a)]*0.99,0),),
                ((str_lenght/10,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],0),),
                ((str_lenght*9/10,x_cr[int(j+a)]*0.99,0),),
                ((str_lenght*9/10,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],0),),
                )
        mdb.models['Model-1'].ConstrainedSketch(gridSpacing=22.88, name='__profile__',
            sheetSize=915.26, transform=
            mdb.models['Model-1'].parts['stringer-cricca'+str(k)].MakeSketchTransform(
            sketchPlane=mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
                (str_lenght/10,x_cr[int(j+a)]*0.99,0),),
            sketchPlaneSide=SIDE1,
            sketchUpEdge=mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].edges.findAt(
                (str_lenght/10,x_cr[int(j+a)],0),),
            sketchOrientation=RIGHT, origin=(0.0,0,0)))
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].projectReferencesOntoSketch(
            filter=COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])

        for q in range (n_rows2):
            if     rip_lenght2/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght2/2+x_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
                center=(x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                     -offset_point2),
                point1=(x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                     -offset_point2+fast_rad))

                mdb.models['Model-1'].sketches['__profile__'].rectangle(
                point1=(x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2-
lato_quad+distance_rows2*q,
                     -offset_point2-lato_quad),
                   point2=(x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+lato_quad+distance_rows2*q,
                     -offset_point2+lato_quad))


                mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
                center=(x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                     -str_lenght+offset_point2),
                point1=(x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                     -str_lenght+offset_point2+fast_rad))

                mdb.models['Model-1'].sketches['__profile__'].rectangle(
                point1=(x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2-
lato_quad+distance_rows2*q,
                     -str_lenght+offset_point2-lato_quad),
                point2=(x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+lato_quad+distance_rows2*q,
                     -str_lenght+offset_point2+lato_quad))

        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].PartitionFaceBySketch(faces=
```

```
            mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
                ((str_lenght/10,x_cr[int(j+a)]*0.99,0),),
                ((str_lenght/10,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],0),),
                ((str_lenght*9/10,x_cr[int(j+a)]*0.99,0),),
                ((str_lenght*9/10,x_cr[int(j+a)]+1.01*cr_lenght[int(j+a)],0),)),
             sketch=mdb.models['Model-1'].sketches['__profile__'], sketchUpEdge=
            mdb.models['Model-1'].parts['stringer-cricca'+str(k)].edges.findAt((
                str_lenght/10,x_cr[int(j+a)],0),))
        del mdb.models['Model-1'].sketches['__profile__']
    a=nsup[k-1]


#Mesh zona cerchio fastener
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for q in range (n_rows2):
            if     rip_lenght2/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght2/2+x_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].setMeshControls(regions=
                mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
                ((str_lenght-offset_point2,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                  0),),
                ((offset_point2,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                  0),),
                ((str_lenght/2,
                  x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2,
                  str_height-offset_point2),)
                ), technique=SWEEP)
    a=nsup[k-1]


#Mesh zona quadrato fastener
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for q in range (n_rows2):
            if     rip_lenght2/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght2/2+x_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].setMeshControls(regions=
                mdb.models['Model-1'].parts['stringer-cricca'+str(k)].faces.findAt(
                ((str_lenght-offset_point2+1.01*fast_rad,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                  0),),
                ((offset_point2+1.01*fast_rad,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                  0),),
                ((str_lenght/2,
                  x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2,
                  str_height-offset_point2+1.01*fast_rad),)
```

```
                ), algorithm=ADVANCING_FRONT)
    a=nsup[k-1]


#Element size locale zona fastener rip2
#cerchio
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for q in range (n_rows2):
            if      rip_lenght2/2-x_noHole/2      <      offset_edge2+distance_rows2*q      <
rip_lenght2/2+x_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].seedEdgeBySize(constraint=FINER,
                    deviationFactor=0.1, edges=
                mdb.models['Model-1'].parts['stringer-cricca'+str(k)].edges.findAt(
#base dx
                    ((str_lenght-offset_point2+fast_rad,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                      0),),
#base sx
                    ((offset_point2+fast_rad,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                      0),),
#rinforzo
                    ((str_lenght/2,
                      x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2,
                      str_height-offset_point2+fast_rad),),
                                        ),
                    minSizeFactor=0.1,size=5)
    a=nsup[k-1]


##quadrato
a=0
lati_quad=4
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for q in range (n_rows2):
            if      rip_lenght2/2-x_noHole/2      <      offset_edge2+distance_rows2*q      <
rip_lenght2/2+x_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].parts['stringer-
cricca'+str(k)].seedEdgeBySize(constraint=FINER,
                        deviationFactor=0.1, edges=
                mdb.models['Model-1'].parts['stringer-cricca'+str(k)].edges.findAt(
#base dx
                ((str_lenght-offset_point2-lato_quad,
                x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                     0),),
                ((str_lenght-offset_point2,
                x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q-lato_quad,
                    0),),
                ((str_lenght-offset_point2+lato_quad,
                x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
```

```
                    0),),
                ((str_lenght-offset_point2,
                x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q+lato_quad,
                    0),),
#base sx
                ((offset_point2+lato_quad,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                    0),),
                ((offset_point2,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q+lato_quad,
                    0),),
                ((offset_point2-lato_quad,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q,
                    0),),
                ((offset_point2,
                    x_cr[int(j+a)]-
rip_lenght2/2+cr_lenght[int(j+a)]/2+offset_edge2+distance_rows2*q-lato_quad,
                    0),),
#base rinforzo
                    ((str_lenght/2,
                      x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2,
                      str_height-offset_point2+lato_quad),),
                    ((str_lenght/2,
                      x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2+lato_quad,
                      str_height-offset_point2),),
                    ((str_lenght/2,
                      x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2,
                      str_height-offset_point2-lato_quad),),
                    ((str_lenght/2,
                      x_cr[int(j+a)]-rip_lenght2/2+cr_lenght[int(j+a)]/2+
                        distance_rows2*q+offset_edge2-lato_quad,
                      str_height-offset_point2),),
                     ),
                    minSizeFactor=0.1,size=5)
    a=nsup[k-1]



#Infittimento mesh in direzione della cricca
##Lx rispetto alla cricca
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].seedEdgeByBias(biasMethod=
        SINGLE, constraint=FINER, end1Edges=
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].edges.findAt(
        ((str_lenght/2,             x_cr[int(j+a)]-0.9*rip_lenght2/2+cr_lenght[int(j+a)]/2,
str_height),),
        ((str_lenght/2, x_cr[int(j+a)]-0.9*rip_lenght2/2+cr_lenght[int(j+a)]/2,0),),
        ((str_lenght,  x_cr[int(j+a)]-0.9*rip_lenght2/2+cr_lenght[int(j+a)]/2, 0),),),
        end2Edges=
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].edges.findAt(
        ((0,x_cr[int(j+a)]-0.1*rip_lenght2/2+cr_lenght[int(j+a)]/2, 0),),
        ), number=32, ratio=10.0)
    a=nsup[k-1]
```

```
##Rx rispetto alla cricca
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].seedEdgeByBias(biasMethod=
        SINGLE, constraint=FINER, end1Edges=
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].edges.findAt(
        ((0,  x_cr[int(j+a)]+0.9*rip_lenght2/2+cr_lenght[int(j+a)]/2, 0),),
                ),
        end2Edges=
        mdb.models['Model-1'].parts['stringer-cricca'+str(k)].edges.findAt(
        ((str_lenght/2,             x_cr[int(j+a)]+0.9*rip_lenght2/2+cr_lenght[int(j+a)]/2,
str_height),),
        ((str_lenght/2, x_cr[int(j+a)]+0.9*rip_lenght2/2+cr_lenght[int(j+a)]/2,0),),
        ((str_lenght,  x_cr[int(j+a)]+0.9*rip_lenght2/2+cr_lenght[int(j+a)]/2, 0),),
                ),
number=32, ratio=10.0)
    a=nsup[k-1]


#Tecnica di mesh
for j in range (n_str_cr):
    mdb.models['Model-1'].parts['stringer-
cricca'+str(j+1)].setMeshControls(algorithm=MEDIAL_AXIS,
    elemShape=QUAD,
    regions=mdb.models['Model-1'].parts['stringer-cricca'+str(j+1)].faces.findAt(
        ((str_lenght/10,skin_lenght/100,0), ),
        ((str_lenght*9/10,skin_lenght/100,0), ),
        ((str_lenght/2,skin_lenght/100,str_height/2),),
        ((str_lenght/10,skin_lenght*90/100,0), ),
        ((str_lenght*9/10,skin_lenght*90/100,0), ),
        ((str_lenght/2,skin_lenght*90/100,str_height/2),)))


#Element size
    mdb.models['Model-1'].parts['stringer-cricca'+str(j+1)].seedPart(deviationFactor=0.1,
minSizeFactor=0.1, size=10.0)

#creazione mesh
    mdb.models['Model-1'].parts['stringer-cricca'+str(j+1)].generateMesh()



#Mesh riparazione skin
#Element type
mdb.models['Model-1'].parts['rip1'].setElementType(elemTypes=(ElemType(
    elemCode=S4R, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT), ElemType(elemCode=S3, elemLibrary=STANDARD),), regions=(
    mdb.models['Model-1'].parts['rip1'].faces.findAt((rip_lenght1,rip_width1/2,0), ),))


#Partizione Fastener
mdb.models['Model-1'].parts['rip1'].faces.findAt(((rip_lenght1/2,rip_width1/2,0),),)
mdb.models['Model-1'].ConstrainedSketch(gridSpacing=59.51, name='__profile__',
sheetSize=2380.57, transform=
mdb.models['Model-1'].parts['rip1'].MakeSketchTransform(
sketchPlane=mdb.models['Model-
1'].parts['rip1'].faces.findAt((rip_lenght1/2,rip_width1/2,0),),
sketchPlaneSide=SIDE1,
sketchUpEdge=mdb.models['Model-
1'].parts['rip1'].edges.findAt((rip_lenght1,rip_width1/2,0),),
sketchOrientation=RIGHT, origin=(0,0, 0)))
mdb.models['Model-1'].parts['rip1'].projectReferencesOntoSketch(filter=
COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])
```

```
for k in range (n_points1):
        for q in range (n_rows1):
            if      rip_lenght1/2-x_noHole/2      <      offset_edge1+distance_rows1*q      <
rip_lenght1/2+x_noHole/2 and\
                rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
                center=(offset_edge1+distance_rows1*q, distance_points1*k+offset_point1),
                point1=(offset_edge1+fast_rad+distance_rows1*q,
distance_points1*k+offset_point1))

mdb.models['Model-1'].parts['rip1'].PartitionFaceBySketch(faces=
    mdb.models['Model-1'].parts['rip1'].faces.findAt(((rip_lenght1/2,rip_width1/2,0), ),
    ), sketch=mdb.models['Model-1'].sketches['__profile__'], sketchUpEdge=
    mdb.models['Model-1'].parts['rip1'].edges.findAt((rip_lenght1,rip_width1/2,0),))
del mdb.models['Model-1'].sketches['__profile__']

for k in range (n_points1):
    for q in range (n_rows1):
            if      rip_lenght1/2-x_noHole/2      <      offset_edge1+distance_rows1*q      <
rip_lenght1/2+x_noHole/2 and\
                rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].parts['rip1'].setMeshControls(regions=
                mdb.models['Model-1'].parts['rip1'].faces.findAt(
                ((offset_edge1+distance_rows1*q, distance_points1*k+offset_point1,0),
                ), ), technique=SWEEP)

#Tecnica di mesh
mdb.models['Model-1'].parts['rip1'].setMeshControls(algorithm=ADVANCING_FRONT,
    regions=mdb.models['Model-1'].parts['rip1'].faces.findAt((
        (rip_lenght1,rip_width1/2,0),),) )

#Element size
mdb.models['Model-1'].parts['rip1'].seedPart(deviationFactor=0.1,
    minSizeFactor=0.1,size=5)


#Element size locale zona fastener
for k in range (n_points1):
    for q in range (n_rows1):
        if       rip_lenght1/2-x_noHole/2       <       offset_edge1+distance_rows1*q       <
rip_lenght1/2+x_noHole/2 and\
                rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
        else:
            mdb.models['Model-1'].parts['rip1'].seedEdgeBySize(constraint=FINER,
            deviationFactor=0.1, edges=
            mdb.models['Model-1'].parts['rip1'].edges.findAt(
            ((offset_edge1+distance_rows1*q+fast_rad,
distance_points1*k+offset_point1,0),),
                    ),
            minSizeFactor=0.1,size=5)

#Creazione mesh
mdb.models['Model-1'].parts['rip1'].generateMesh()
```

```
#Mesh riparazione stringer
#Element type
mdb.models['Model-1'].parts['rip2'].setElementType(elemTypes=(ElemType(
    elemCode=S4R, elemLibrary=STANDARD, secondOrderAccuracy=OFF,
    hourglassControl=DEFAULT), ElemType(elemCode=S3, elemLibrary=STANDARD),), regions=(
    mdb.models['Model-1'].parts['rip2'].faces.findAt(((rip_width2/2,0,rip_lenght2/2), ),
                                                     ((0,rip_height2/2,rip_lenght2/2), ),
                                                     ((rip_rad2*(1-cos(pi/4)),rip_rad2*(1-
sin(pi/4)),rip_lenght2/2), ),),))

#Partizione Fastener sulla base
mdb.models['Model-1'].parts['rip2'].faces.findAt(((rip_width2/2,0,rip_lenght2/2),),)
mdb.models['Model-1'].ConstrainedSketch(gridSpacing=59.51, name='__profile__',
sheetSize=2380.57, transform=
mdb.models['Model-1'].parts['rip2'].MakeSketchTransform(
sketchPlane=mdb.models['Model-
1'].parts['rip2'].faces.findAt((rip_width2/2,0,rip_lenght2/2),),
sketchPlaneSide=SIDE1,
sketchUpEdge=mdb.models['Model-
1'].parts['rip2'].edges.findAt((rip_width2/2,0,rip_lenght2),),
sketchOrientation=RIGHT, origin=(0,0,0)))
mdb.models['Model-1'].parts['rip2'].projectReferencesOntoSketch(filter=
COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])

for k in range (n_rows2):
    if rip_lenght2/2-x_noHole/2 < offset_edge2+distance_rows2*k < rip_lenght2/2+x_noHole/2:
        bandiera=1
    else:
        mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
        center=(distance_rows2*k+offset_edge2, rip_width2-offset_point2),
        point1=(distance_rows2*k+offset_edge2+fast_rad, rip_width2-offset_point2))
        mdb.models['Model-1'].sketches['__profile__'].rectangle(
        point1=(distance_rows2*k+offset_edge2-lato_quad,          rip_width2-offset_point2-
lato_quad)
        ,           point2=(distance_rows2*k+offset_edge2+lato_quad,           rip_width2-
offset_point2+lato_quad))

mdb.models['Model-1'].parts['rip2'].PartitionFaceBySketch(faces=
    mdb.models['Model-1'].parts['rip2'].faces.findAt(((rip_width2/2,0,rip_lenght2/2), ),
    ), sketch=mdb.models['Model-1'].sketches['__profile__'], sketchUpEdge=
    mdb.models['Model-1'].parts['rip2'].edges.findAt((rip_width2/2,0,rip_lenght2),))
del mdb.models['Model-1'].sketches['__profile__']

#Partizione Fastener sull'altezza
mdb.models['Model-1'].parts['rip2'].faces.findAt(((0,rip_height2/2,rip_lenght2/2),),)
mdb.models['Model-1'].ConstrainedSketch(gridSpacing=59.51, name='__profile__',
sheetSize=2380.57, transform=
mdb.models['Model-1'].parts['rip2'].MakeSketchTransform(
sketchPlane=mdb.models['Model-
1'].parts['rip2'].faces.findAt((0,rip_height2/2,rip_lenght2/2),),
sketchPlaneSide=SIDE1,
sketchUpEdge=mdb.models['Model-
1'].parts['rip2'].edges.findAt((0,rip_height2,rip_lenght2/2),),
sketchOrientation=RIGHT, origin=(0,0,0)))
mdb.models['Model-1'].parts['rip2'].projectReferencesOntoSketch(filter=
COPLANAR_EDGES, sketch=mdb.models['Model-1'].sketches['__profile__'])

for k in range (n_rows2):
    if rip_lenght2/2-x_noHole/2 < offset_edge2+distance_rows2*k < rip_lenght2/2+x_noHole/2:
        bandiera=1
    else:
        mdb.models['Model-1'].sketches['__profile__'].CircleByCenterPerimeter(
        center=(rip_height2-offset_point2,distance_rows2*k+offset_edge2),
```

```
        point1=(rip_height2-offset_point2,distance_rows2*k+offset_edge2+fast_rad))
        mdb.models['Model-1'].sketches['__profile__'].rectangle(
        point1=(rip_height2-offset_point2-lato_quad,distance_rows2*k+offset_edge2-
lato_quad)
        ,                                                                point2=(rip_height2-
offset_point2+lato_quad,distance_rows2*k+offset_edge2+lato_quad))

mdb.models['Model-1'].parts['rip2'].PartitionFaceBySketch(faces=
    mdb.models['Model-1'].parts['rip2'].faces.findAt(((0,rip_height2/2,rip_lenght2/2), ),
    ), sketch=mdb.models['Model-1'].sketches['__profile__'], sketchUpEdge=
    mdb.models['Model-1'].parts['rip2'].edges.findAt((0,rip_height2,rip_lenght2/2),))
del mdb.models['Model-1'].sketches['__profile__']


#Tecnica di mesh fastener
for k in range (n_rows2):
    if rip_lenght2/2-x_noHole/2 < offset_edge2+distance_rows2*k < rip_lenght2/2+x_noHole/2:
        bandiera=1
    else:
        mdb.models['Model-1'].parts['rip2'].setMeshControls(regions=
        mdb.models['Model-1'].parts['rip2'].faces.findAt(
        ((rip_width2-offset_point2,0,distance_rows2*k+offset_edge2),),
        ((0,rip_height2-offset_point2,distance_rows2*k+offset_edge2),),
        ), technique=SWEEP)

#Tecnica di mesh
mdb.models['Model-1'].parts['rip2'].setMeshControls(algorithm=ADVANCING_FRONT,
    regions=mdb.models['Model-
1'].parts['rip2'].faces.findAt(((rip_width2*0.99,0,rip_lenght2/2), ),
                                                        ((0,rip_height2/2,rip_lenght2*0.99),
),
                                                        ((rip_rad2*(1-cos(pi/4)),rip_rad2*(1-
sin(pi/4)),rip_lenght2/2), )))

#Element size
mdb.models['Model-1'].parts['rip2'].seedPart(deviationFactor=0.1,
    minSizeFactor=0.1,size=5)

#Element size locale zona fastener
for k in range (n_rows2):
    if rip_lenght2/2-x_noHole/2 < offset_edge2+distance_rows2*k < rip_lenght2/2+x_noHole/2:
        bandiera=1
    else:
        mdb.models['Model-1'].parts['rip2'].seedEdgeBySize(constraint=FINER,
        deviationFactor=0.1, edges=
        mdb.models['Model-1'].parts['rip2'].edges.findAt(
        ((rip_width2-offset_point2,0,distance_rows2*k+offset_edge2+fast_rad),),
        ((0,rip_height2-offset_point2,distance_rows2*k+offset_edge2+fast_rad),),
            ),
        minSizeFactor=0.1,size=5)

#Creazione mesh
mdb.models['Model-1'].parts['rip2'].generateMesh()


#----------#
#----------#
# ASSEMBLY #
#----------#
#----------#

#CREAZIONE assembly
mdb.models['Model-1'].rootAssembly.DatumCsysByDefault(CARTESIAN)
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name='skin-1', part=
```

```
    mdb.models['Model-1'].parts['skin'])


for i in range(1,n_rib+1):
    mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name='rib-'+str(i), part=
    mdb.models['Model-1'].parts['rib'])
    mdb.models['Model-1'].rootAssembly.instances['rib-'+str(i)].translate(vector=(
    (1+i)*skin_lenght, 0.0, 0.0))


for i in range(1,n_stringer-n_str_cr+1):
    mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name='stringer-'+str(i),
    part=mdb.models['Model-1'].parts['stringer'])
    mdb.models['Model-1'].rootAssembly.instances['stringer-'+str(i)].translate(vector=(
    (n_rib+1+i)*skin_lenght, 0.0, 0.0))

for i in range(1,n_str_cr+1):
    mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name=
    'stringer-cricca-'+str(i), part=mdb.models['Model-1'].parts['stringer-cricca'+str(i)])
    mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-'+str(i)].translate(
    vector=((n_rib+n_stringer-n_str_cr+2+i)*skin_lenght, 0.0, 0.0))

for j in range (n_cr):
    for i in range(1,n_rip1+1):
        mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name=
        'rip1-'+str(i)+'-cr'+str(j+1), part=mdb.models['Model-1'].parts['rip1'])
        mdb.models['Model-1'].rootAssembly.instances['rip1-'+str(i)+'-
cr'+str(j+1)].translate(
        vector=((n_rib+n_stringer+3+i)*skin_lenght, 0.0, 0.0))


for j in range (n_cr):
    for i in range(1,n_rip2+1):
        mdb.models['Model-1'].rootAssembly.Instance(dependent=ON, name=
        'rip2-'+str(i)+'-cr'+str(j+1), part=mdb.models['Model-1'].parts['rip2'])
        mdb.models['Model-1'].rootAssembly.instances['rip2-'+str(i)+'-
cr'+str(j+1)].translate(
        vector=((n_rib+n_stringer+n_rip1+4+i)*skin_lenght, 0.0, 0.0))


#vincoli rib-skin
for i in range (1,n_rib+1):
    #contatto superfici base
    mdb.models['Model-1'].rootAssembly.FaceToFace(clearance=(skin_depth+rib_depth)/2,
fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['skin-
1'].faces.findAt((skin_lenght/100,skin_width/100,0),),
                                            flip=ON,
    movablePlane=mdb.models['Model-1'].rootAssembly.instances['rib-'+str(i)].faces.

findAt((skin_lenght*(1+i)+rib_width/2,0,rib_base+rib_cut_lenght),))

    #contatto superfici laterali
    mdb.models['Model-1'].rootAssembly.FaceToFace(clearance=0.0, fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[xz_skin.id], flip=OFF,
    movablePlane=
    mdb.models['Model-1'].rootAssembly.instances['rib-'+str(i)].datums[xy_rib.id])

    #offset rib skin
    mdb.models['Model-1'].rootAssembly.FaceToFace(clearance=offset_rib+distance_rib*(i-1),
fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[yz_skin.id],
                                            flip=ON,
```

```
    movablePlane=mdb.models['Model-1'].rootAssembly.instances['rib-
'+str(i)].datums[yz_rib.id])


#vincoli stringers-skin
#creazione del vettore contenente gli stringer non criccati
stringer=[]
for j in range (n_stringer):
    stringer.append(j)
str_ncr= list(set(stringer)-set(str_cr))

for i in range(1,n_stringer+1-n_str_cr):
    #contatto superfici base
    mdb.models['Model-1'].rootAssembly.FaceToFace(clearance=(skin_depth+str_depth)/2,
fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['skin-
1'].faces.findAt((skin_lenght/100,skin_width/100,0),),
                                                flip=OFF,
    movablePlane=mdb.models['Model-1'].rootAssembly.instances['stringer-'+str(i)].faces.
    findAt(((n_rib+1+i)*skin_lenght+str_lenght/2,skin_lenght/2,0),))

    #contatto superfici laterali
    mdb.models['Model-1'].rootAssembly.FaceToFace(clearance=0.0, fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[yz_skin.id],
    flip=OFF,
    movablePlane=mdb.models['Model-1'].rootAssembly.instances['stringer-
'+str(i)].datums[xz_str.id])

    #offset stringer skin
    mdb.models['Model-1'].rootAssembly.FaceToFace(clearance=(str_ncr[i-
1]*distance_stringer+offset_stringer/2), fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[xz_skin.id],
                                            flip=ON,
    movablePlane=mdb.models['Model-1'].rootAssembly.instances['stringer-
'+str(i)].datums[yz_str.id])


#vincoli stringer criccati-skin
for i in range (1,n_str_cr+1):
    #contatto superfici base
    mdb.models['Model-1'].rootAssembly.FaceToFace(clearance=(skin_depth+str_depth)/2,
fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['skin-
1'].faces.findAt((skin_lenght/100,skin_width/100,0),),
                                                flip=OFF,
    movablePlane=mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(i)].datums[xy_str.id])

    #contatto superfici laterali
    mdb.models['Model-1'].rootAssembly.FaceToFace(clearance=0.0, fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[yz_skin.id],
    flip=OFF,
    movablePlane=mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(i)].datums[xz_str.id])

    #offset stringer skin
    mdb.models['Model-1'].rootAssembly.FaceToFace(clearance=(str_cr[i-
1]*distance_stringer+offset_stringer/2), fixedPlane=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[xz_skin.id],
                                            flip=ON,
    movablePlane=mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(i)].datums[yz_str.id])
```

```
#vincoli rip1-skin
xz_rip1=mdb.models['Model-1'].parts['rip1'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=XZPLANE)
yz_rip1=mdb.models['Model-1'].parts['rip1'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=YZPLANE)
xy_rip1=mdb.models['Model-1'].parts['rip1'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=XYPLANE)

for j in range (n_cr):
    for i in range (1,n_rip1+1):
        #contatto superfici base
        mdb.models['Model-1'].rootAssembly.FaceToFace(
            clearance=(skin_depth+rip_depth1)/2, fixedPlane=
        mdb.models['Model-1'].rootAssembly.instances['skin-
1'].faces.findAt((skin_lenght/100,skin_width/100,0),),
                                                flip=ON,
        movablePlane=mdb.models['Model-1'].rootAssembly.instances['rip1-'+str(i)+'-
cr'+str(j+1)].datums[xy_rip1.id])

        if (i<2):
        #contatto superfici laterali
            mdb.models['Model-1'].rootAssembly.FaceToFace(
                clearance=(str_cr_cr[j][0]*distance_stringer+offset_stringer/2)-rip_width1-
str_lenght/2,
            fixedPlane=mdb.models['Model-1'].rootAssembly.instances['skin-
1'].datums[xz_skin.id],
            flip=OFF,
            movablePlane=mdb.models['Model-1'].rootAssembly.instances['rip1-'+str(i)+'-
cr'+str(j+1)].datums[xz_rip1.id])
        #offset rip1 skin
            mdb.models['Model-1'].rootAssembly.FaceToFace(
            clearance=(rip_lenght1+cr_lenght[str_cr_cr[j][1]])/2+x_cr[str_cr_cr[j][1]],
            fixedPlane=mdb.models['Model-1'].rootAssembly.instances['skin-
1'].datums[yz_skin.id],
                                                flip=ON,
            movablePlane=mdb.models['Model-1'].rootAssembly.instances['rip1-'+str(i)+'-
cr'+str(j+1)].datums[yz_rip1.id])


        else:
        #contatto superfici laterali
            mdb.models['Model-1'].rootAssembly.FaceToFace(

clearance=(str_cr_cr[j][0]*distance_stringer+offset_stringer/2)+rip_width1+str_lenght/2,
            fixedPlane=mdb.models['Model-1'].rootAssembly.instances['skin-
1'].datums[xz_skin.id],
            flip=ON,
            movablePlane=mdb.models['Model-1'].rootAssembly.instances['rip1-'+str(i)+'-
cr'+str(j+1)].datums[xz_rip1.id])
        #offset rip1 skin
            mdb.models['Model-1'].rootAssembly.FaceToFace(
            clearance=((cr_lenght[str_cr_cr[j][1]])/2+x_cr[str_cr_cr[j][1]])-rip_lenght1/2,
            fixedPlane=mdb.models['Model-1'].rootAssembly.instances['skin-
1'].datums[yz_skin.id],
                                                flip=OFF,
            movablePlane=mdb.models['Model-1'].rootAssembly.instances['rip1-'+str(i)+'-
cr'+str(j+1)].datums[yz_rip1.id])


#vincoli rip2-skin
xz_rip2=mdb.models['Model-1'].parts['rip2'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=XZPLANE)
yz_rip2=mdb.models['Model-1'].parts['rip2'].DatumPlaneByPrincipalPlane(offset=0.0,
```

```
    principalPlane=YZPLANE)
xy_rip2=mdb.models['Model-1'].parts['rip2'].DatumPlaneByPrincipalPlane(offset=0.0,
    principalPlane=XYPLANE)

for j in range (n_cr):
    for i in range (1,n_rip2+1):
        #contatto superfici base
        mdb.models['Model-
1'].rootAssembly.FaceToFace(clearance=(skin_depth+2*str_depth+rip_depth2)/2, fixedPlane=
        mdb.models['Model-1'].rootAssembly.instances['skin-
1'].faces.findAt((skin_lenght/100,skin_width/100,0),),
                                                flip=OFF,
        movablePlane=mdb.models['Model-1'].rootAssembly.instances['rip2-'+str(i)+'-
cr'+str(j+1)].datums[xz_rip2.id])

        if (i<2):
            #contatto superfici laterali
            mdb.models['Model-1'].rootAssembly.FaceToFace(

clearance=(str_cr_cr[j][0]*distance_stringer+offset_stringer/2)+str_depth+rip_depth2/2
                                                    , fixedPlane=
            mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[xz_skin.id],
flip=OFF,
            movablePlane=
            mdb.models['Model-1'].rootAssembly.instances['rip2-'+str(i)+'-
cr'+str(j+1)].datums[yz_rip2.id])

            #offset rip skin
            mdb.models['Model-1'].rootAssembly.FaceToFace(
                clearance=x_cr[str_cr_cr[j][1]]-(rip_lenght2-cr_lenght[str_cr_cr[j][1]])/2,
fixedPlane=
            mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[yz_skin.id],
                                                  flip=OFF,
            movablePlane=mdb.models['Model-1'].rootAssembly.instances['rip2-'+str(i)+'-
cr'+str(j+1)].datums[xy_rip2.id])


        else:
            #contatto superfici laterali
            mdb.models['Model-1'].rootAssembly.FaceToFace(
                clearance=(str_cr_cr[j][0]*distance_stringer+offset_stringer/2)-str_depth-
rip_depth2/2
            , fixedPlane=
            mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[xz_skin.id],
flip=ON,
            movablePlane=
            mdb.models['Model-1'].rootAssembly.instances['rip2-'+str(i)+'-
cr'+str(j+1)].datums[yz_rip2.id])

            #offset rip skin
            mdb.models['Model-1'].rootAssembly.FaceToFace(
                clearance=rip_lenght2+x_cr[str_cr_cr[j][1]]-(rip_lenght2-
cr_lenght[str_cr_cr[j][1]])/2, fixedPlane=
            mdb.models['Model-1'].rootAssembly.instances['skin-1'].datums[yz_skin.id],
                                                flip=ON,
            movablePlane=mdb.models['Model-1'].rootAssembly.instances['rip2-'+str(i)+'-
cr'+str(j+1)].datums[xy_rip2.id])


# ----------- #
# ----------- #
# INTERACTION #
# ----------- #
# ----------- #
```

```
#tie rib-skin
#selezione superficie skin (in tutte le sue partizioni)
mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf-1', side1Faces=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].faces.findAt(
    ((skin_lenght/100,skin_width/100,0),), ))


for i in range (n_cr):
    mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf'+str(i+1), side1Faces=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].faces.findAt(
    ((x_cr[i]-cr_lenght[i]/100,y_cr[i]+0.99*cr_width[i]/2,0),),
        ))
    mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='m_Surf-1', surfaces=(
    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'],
    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf'+str(i+1)]))


    mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf'+str(i+1), side1Faces=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].faces.findAt(
    ((x_cr[i],str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
0.5*(1+0.9*x1)*(rip_width1+rip_width2),0),),
        ))
    mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='m_Surf-1', surfaces=(
    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'],
    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf'+str(i+1)]))


    for k in range (n_points1):
        for q in range (n_rows1):
            if     rip_lenght1/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght1/2+x_noHole/2 and\
               rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf'+str(i+1),
side1Faces=
                mdb.models['Model-1'].rootAssembly.instances['skin-1'].faces.findAt(
                    ((fast_rad*0.9+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,

distance_points1*k+offset_point1+str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
                    ((-fast_rad*0.9+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,

distance_points1*k+offset_point1+str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
                        ))
                mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='m_Surf-1',
surfaces=(
                mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'],
                mdb.models['Model-1'].rootAssembly.surfaces['m_Surf'+str(i+1)]))


    for k in range (n_points1):
        for q in range (n_rows1):
            if     rip_lenght1/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght1/2+x_noHole/2 and\
```

```
              rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf'+str(i+1),
side1Faces=
                mdb.models['Model-1'].rootAssembly.instances['skin-1'].faces.findAt(
                    ((fast_rad*0.9+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
                      2*(rip_width1+str_lenght/2)-(distance_points1*k+offset_point1)+
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
                    ((-fast_rad*0.9+offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
                      2*(rip_width1+str_lenght/2)-(distance_points1*k+offset_point1)+
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
                        ))
                mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='m_Surf-1',
surfaces=(
                mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'],
                mdb.models['Model-1'].rootAssembly.surfaces['m_Surf'+str(i+1)]))


    for q in range (n_rows2):
        if      rip_lenght1/2-x_noHole/2      <       offset_edge2+distance_rows2*q      <
rip_lenght1/2+x_noHole/2:
            bandiera=1
        else:
            mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf'+str(i+1), side1Faces=
            mdb.models['Model-1'].rootAssembly.instances['skin-1'].faces.findAt(
            ((-fast_rad*0.5+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
                      rip_width1+offset_point1+
              str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
            ((fast_rad*0.5+offset_edge2+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
                      rip_width1+offset_point1+
              str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0),),
            ))
            mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='m_Surf-1', surfaces=(
            mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'],
            mdb.models['Model-1'].rootAssembly.surfaces['m_Surf'+str(i+1)]))



    for q in range (n_rows2):
        if      rip_lenght1/2-x_noHole/2      <       offset_edge2+distance_rows2*q      <
rip_lenght1/2+x_noHole/2:
            bandiera=1
        else:
            mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf'+str(i+1), side1Faces=
            mdb.models['Model-1'].rootAssembly.instances['skin-1'].faces.findAt(
            ((-fast_rad*0.5+offset_edge1+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
                      rip_width1+str_lenght-offset_point1+
              str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
```

```
                    0),),
            ((fast_rad*0.5+offset_edge1+distance_rows2*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
                    rip_width1+str_lenght-offset_point1+
             str_cr_cr[i][0]*distance_stringer+offset_stringer/2-(rip_width1+str_lenght/2),
                    0),),        ))
            mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='m_Surf-1', surfaces=(
            mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'],
            mdb.models['Model-1'].rootAssembly.surfaces['m_Surf'+str(i+1)]))

#selezione base rib (in tutti i suoi piedini)
for i in range(1,n_rib+1):
    mdb.models['Model-1'].rootAssembly.Surface(name='s_Surf'+str(i), side1Faces=
    mdb.models['Model-1'].rootAssembly.instances['rib-'+str(i)].faces.findAt(
    ((offset_rib+distance_rib*(i-
1)+rib_width/2,1*(rib_base+rib_cut_lenght),(skin_depth+rib_depth)/2),),))

for i in range(1,n_rib+1):
    for j in range(1,n_stringer):
        mdb.models['Model-1'].rootAssembly.Surface(name='s_Surf'+str(i)+str(j), side1Faces=
        mdb.models['Model-1'].rootAssembly.instances['rib-'+str(i)].faces.findAt(
        ((offset_rib+distance_rib*(i-
1)+rib_width/2,j*(rib_base+rib_cut_lenght),(skin_depth+rib_depth)/2),),))
        mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='s_Surf'+str(i),
surfaces=(
        mdb.models['Model-1'].rootAssembly.surfaces['s_Surf'+str(i)],
        mdb.models['Model-1'].rootAssembly.surfaces['s_Surf'+str(i)+str(j)]))

#creazione tie
    mdb.models['Model-1'].Tie(adjust=ON, master=
    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'], name=
    'tie-rib'+str(i)+'-skin'+str(j), positionToleranceMethod=COMPUTED, slave=
    mdb.models['Model-1'].rootAssembly.surfaces['s_Surf'+str(i)], thickness=ON,
    tieRotations=ON)



#tie str-skin
for i in range (1,n_stringer-n_str_cr+1):
    mdb.models['Model-1'].rootAssembly.Surface(name='s_Surf-'+str(i+n_rib), side1Faces=
    mdb.models['Model-1'].rootAssembly.instances['stringer-'+str(i)].faces.findAt(
    ((skin_lenght/2,(str_ncr[i-
1]*distance_stringer+offset_stringer/2)+str_lenght/10,(skin_depth+str_depth)/2),),
    ((skin_lenght/2,(str_ncr[i-1]*distance_stringer+offset_stringer/2)-
str_lenght/10,(skin_depth+str_depth)/2),),))

    mdb.models['Model-1'].Tie(adjust=ON, master=
    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'], name=
    'tie-str'+str(i)+'-skin', positionToleranceMethod=COMPUTED, slave=
    mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-'+str(i+n_rib)], thickness=ON,
    tieRotations=ON)


#tie str cr-skin
##stringer con cricca passante

a=0
for i in range (1,n_str_cr+1):
    for k in range (int(nsup[i-1])):
        #LEFT
        mdb.models['Model-1'].rootAssembly.Surface(name=
            's_Surf-str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a)), side1Faces=
        mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(i)].faces.findAt(
```

```
        ((x_cr[str_cr_cr[int(k+a)][1]]*0.99,
              (str_cr[i-
1]*distance_stringer+offset_stringer/2)+str_lenght/10,(skin_depth+str_depth)/2),),
        ((x_cr[str_cr_cr[int(k+a)][1]]*0.99,
              (str_cr[i-1]*distance_stringer+offset_stringer/2)-
str_lenght/10,(skin_depth+str_depth)/2),),
        ((x_cr[str_cr_cr[int(k+a)][1]]-
1.01*rip_lenght2/2+cr_lenght[str_cr_cr[int(k+a)][1]]/2,
          str_lenght*0.1+str_cr[i-
1]*distance_stringer+offset_stringer/2,(skin_depth+str_depth)/2),),
        ((x_cr[str_cr_cr[int(k+a)][1]]-
1.01*rip_lenght2/2+cr_lenght[str_cr_cr[int(k+a)][1]]/2,
          -str_lenght*0.1+str_cr[i-
1]*distance_stringer+offset_stringer/2,(skin_depth+str_depth)/2),),))

        for q in range (int(n_rows2/2)):
            if     rip_lenght1/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght1/2+x_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].rootAssembly.Surface(name=
                    's_Surf-str'+str(int(str_cr_cr[int(k+a)][0]))+'supp-'+str(int(k+a)),
side1Faces=
                mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(i)].faces.findAt(
                ((x_cr[str_cr_cr[int(k+a)][1]]-
rip_lenght2/2+cr_lenght[str_cr_cr[int(k+a)][1]]/2+offset_edge2+distance_rows2*q,
                  (str_cr[i-1]*distance_stringer+offset_stringer/2)+str_lenght/2-
offset_point2,(skin_depth+str_depth)/2),),
                ((x_cr[str_cr_cr[int(k+a)][1]]-
rip_lenght2/2+cr_lenght[str_cr_cr[int(k+a)][1]]/2+offset_edge2+distance_rows2*q,
                  (str_cr[i-1]*distance_stringer+offset_stringer/2)-
str_lenght/2+offset_point2,(skin_depth+str_depth)/2),),
                ((x_cr[str_cr_cr[int(k+a)][1]]-
rip_lenght2/2+1.1*fast_rad+cr_lenght[str_cr_cr[int(k+a)][1]]/2+offset_edge2+distance_rows2*
q,
                  (str_cr[i-1]*distance_stringer+offset_stringer/2)+str_lenght/2-
offset_point2,(skin_depth+str_depth)/2),),
                ((x_cr[str_cr_cr[int(k+a)][1]]-
rip_lenght2/2+1.1*fast_rad+cr_lenght[str_cr_cr[int(k+a)][1]]/2+offset_edge2+distance_rows2*
q,
                  (str_cr[i-1]*distance_stringer+offset_stringer/2)-
str_lenght/2+offset_point2,(skin_depth+str_depth)/2),),
                ))

                mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name=
                    's_Surf-str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a)),
surfaces=(
                mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
str'+str(int(str_cr_cr[int(k+a)][0]))+'supp-'+str(int(k+a))],
                mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a))]))

                mdb.models['Model-1'].Tie(adjust=ON, master=
                mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'], name=
                'tie-str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a)),
                positionToleranceMethod=COMPUTED, slave=
                mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a))],
                thickness=ON,
                tieRotations=ON)

        #RIGHT
        mdb.models['Model-1'].rootAssembly.Surface(name=
```

```
            's_Surf-str'+str(int(str_cr_cr[int(k+a)][0]))+'sup-'+str(int(k+a+1)),
side1Faces=
        mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(i)].faces.findAt(
        ((x_cr[str_cr_cr[int(k+a)][1]]+cr_lenght[str_cr_cr[int(k+a)][1]]*1.01,
            (str_cr[i-
1]*distance_stringer+offset_stringer/2)+str_lenght/10,(skin_depth+str_depth)/2),),
        ((x_cr[str_cr_cr[int(k+a)][1]]+cr_lenght[str_cr_cr[int(k+a)][1]]*1.01,
            (str_cr[i-1]*distance_stringer+offset_stringer/2)-
str_lenght/10,(skin_depth+str_depth)/2),),

((x_cr[str_cr_cr[int(k+a)][1]]+1.01*rip_lenght2/2+cr_lenght[str_cr_cr[int(k+a)][1]]/2,
          str_lenght*0.1+str_cr[i-
1]*distance_stringer+offset_stringer/2,(skin_depth+str_depth)/2),),

((x_cr[str_cr_cr[int(k+a)][1]]+1.01*rip_lenght2/2+cr_lenght[str_cr_cr[int(k+a)][1]]/2,
          -str_lenght*0.1+str_cr[i-
1]*distance_stringer+offset_stringer/2,(skin_depth+str_depth)/2),),))

        for q in range (int(n_rows2/2),n_rows2):
            if     rip_lenght1/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght1/2+x_noHole/2:
                bandiera=1
            else:
                mdb.models['Model-1'].rootAssembly.Surface(name=
                    's_Surf-str'+str(int(str_cr_cr[int(k+a)][0]))+'supp'+str(int(k+a+1)),
side1Faces=
                mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(i)].faces.findAt(
                ((x_cr[str_cr_cr[int(k+a)][1]]-
rip_lenght2/2+cr_lenght[str_cr_cr[int(k+a)][1]]/2+offset_edge2+distance_rows2*q,
                  (str_cr[i-1]*distance_stringer+offset_stringer/2)+str_lenght/2-
offset_point2,(skin_depth+str_depth)/2),),
                ((x_cr[str_cr_cr[int(k+a)][1]]-
rip_lenght2/2+cr_lenght[str_cr_cr[int(k+a)][1]]/2+offset_edge2+distance_rows2*q,
                  (str_cr[i-1]*distance_stringer+offset_stringer/2)-
str_lenght/2+offset_point2,(skin_depth+str_depth)/2),),
                ((x_cr[str_cr_cr[int(k+a)][1]]-
rip_lenght2/2+1.1*fast_rad+cr_lenght[str_cr_cr[int(k+a)][1]]/2+offset_edge2+distance_rows2*
q,
                  (str_cr[i-1]*distance_stringer+offset_stringer/2)+str_lenght/2-
offset_point2,(skin_depth+str_depth)/2),),
                ((x_cr[str_cr_cr[int(k+a)][1]]-
rip_lenght2/2+1.1*fast_rad+cr_lenght[str_cr_cr[int(k+a)][1]]/2+offset_edge2+distance_rows2*
q,
                  (str_cr[i-1]*distance_stringer+offset_stringer/2)-
str_lenght/2+offset_point2,(skin_depth+str_depth)/2),),
                ))

                mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name=
                    's_Surf-str'+str(int(str_cr_cr[int(k+a)][0]))+'sup-'+str(int(k+a+1)),
surfaces=(
                mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
str'+str(int(str_cr_cr[int(k+a)][0]))+'sup-'+str(int(k+a+1))],
                mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
str'+str(int(str_cr_cr[int(k+a)][0]))+'supp'+str(int(k+a+1))]))

                mdb.models['Model-1'].Tie(adjust=ON, master=
                mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'], name=
                'tie-str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a+1)),
                positionToleranceMethod=COMPUTED, slave=
                mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
str'+str(int(str_cr_cr[int(k+a)][0]))+'sup-'+str(int(k+a+1))],
                thickness=ON,
```

```
                tieRotations=ON)
    a=nsup[i-1]

a=0
for i in range (1,n_str_cr+1):
    for k in range (1,int(nsup[i-1])):
        mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name=
                    's_Surf-str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a)),
surfaces=(
        mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a))],
        mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
str'+str(int(str_cr_cr[int(k+a)][0]))+'sup-'+str(int(k+a))]))

        mdb.models['Model-1'].Tie(adjust=ON, master=
        mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'], name=
        'tie-str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a)),
        positionToleranceMethod=COMPUTED, slave=
        mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
str'+str(int(str_cr_cr[int(k+a)][0]))+'sup'+str(int(k+a))],
        thickness=ON,
        tieRotations=ON)
    a=nsup[i-1]


#Contatto lamine di riparazione_skin
##proprieta contatto
mdb.models['Model-1'].ContactProperty('IntProp-1')
mdb.models['Model-1'].interactionProperties['IntProp-1'].TangentialBehavior(
    dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
    formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
    pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
    table=((0.2, ), ), temperatureDependency=OFF)
mdb.models['Model-1'].interactionProperties['IntProp-1'].NormalBehavior(
    allowSeparation=ON, constraintEnforcementMethod=DEFAULT,
    pressureOverclosure=HARD)


#rip1-skin contatto
for i in range (n_cr):
    for j in range (1,n_rip1+1):
        mdb.models['Model-1'].rootAssembly.Surface(name='s_Surf-'+str(j)+'cr'+str(i+1),
side1Faces=
        mdb.models['Model-1'].rootAssembly.instances['rip1-'+str(j)+'-
cr'+str(i+1)].faces.findAt(
        ((-0.99*(rip_lenght1-cr_lenght[str_cr_cr[i][1]])/2+x_cr[str_cr_cr[i][1]],
          y_cr[str_cr_cr[i][1]]+(j-1)*cr_width[str_cr_cr[i][1]],
          (skin_depth+rip_depth1)/2), ), ))

        for k in range (n_points1):
            for q in range (n_rows1):
                if    rip_lenght1/2-x_noHole/2    <    offset_edge1+distance_rows1*q    <
rip_lenght1/2+x_noHole/2 and\
                   rip_width1/2+str_lenght/2-y_noHole/2<(j-
1)*cr_width[str_cr_cr[i][1]]+distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_n
oHole/2:
                    bandiera=1
                else:
                    mdb.models['Model-
1'].rootAssembly.Surface(name='s_Surf'+str(j)+'cr'+str(i+1), side1Faces=
                    mdb.models['Model-1'].rootAssembly.instances['rip1-'+str(j)+'-
cr'+str(i+1)].faces.findAt(
                    ((offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      (j-1)*offset_stringer+distance_points1*k+offset_point1+
```

```
                      str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      (skin_depth+rip_depth1)/2),),
                        ))
                    mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='s_Surf-
'+str(j)+'cr'+str(i+1), surfaces=(
                    mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-
'+str(j)+'cr'+str(i+1)],
                    mdb.models['Model-
1'].rootAssembly.surfaces['s_Surf'+str(j)+'cr'+str(i+1)]))



        mdb.models['Model-1'].SurfaceToSurfaceContactStd(adjustMethod=NONE,
        clearanceRegion=None, createStepName='Initial', datumAxis=None,
        initialClearance=OMIT, interactionProperty='IntProp-1', master=
        mdb.models['Model-1'].rootAssembly.surfaces['m_Surf-1'],                 name='Int-
'+str(j)+'cr'+str(i+1),
        slave=mdb.models['Model-1'].rootAssembly.surfaces['s_Surf-'+str(j)+'cr'+str(i+1)],
                    sliding=FINITE, thickness=ON)


#rip2-stringer contatto
#superficie master su base stringer
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2+1):
            mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf2'+str(k+int(j+a))+'
1'+str(i), side1Faces=
            mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(k)].faces.findAt(
             (((-0.1*(rip_lenght2-
cr_lenght[str_cr_cr[int(j+a)][1]])/2+x_cr[str_cr_cr[int(j+a)][1]],
              (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2-(2*i-
3)*str_lenght*0.2),
              (skin_depth+str_depth)/2) ),)))
    a=nsup[k-1]


a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2+1):
            mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf2-'+str(k+int(j+a))+'
1'+str(i), side1Faces=
            mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(k)].faces.findAt(
             (((0.1*(rip_lenght2-
cr_lenght[str_cr_cr[int(j+a)][1]])/2+x_cr[str_cr_cr[int(j+a)][1]],
              (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2-(2*i-
3)*str_lenght*0.2),
              (skin_depth+str_depth)/2) ),)))
            mdb.models['Model-
1'].rootAssembly.SurfaceByBoolean(name='m_Surf2'+str(k+int(j+a))+' 1'+str(i), surfaces=(
            mdb.models['Model-1'].rootAssembly.surfaces['m_Surf2-'+str(k+int(j+a))+'
1'+str(i)],
            mdb.models['Model-1'].rootAssembly.surfaces['m_Surf2'+str(k+int(j+a))+'
1'+str(i)]))
    a=nsup[k-1]


a=0
for k in range (1,n_str_cr+1):
```

```
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2+1):
            for q in range (n_rows2):
                if    rip_lenght1/2-x_noHole/2    <    offset_edge2+distance_rows2*q    <
rip_lenght1/2+x_noHole/2:
                    bandiera=1
                else:
                    mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf2-
'+str(k+int(j+a))+' 1'+str(i), side1Faces=
                    mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(k)].faces.findAt(
                    ((fast_rad*1.1+offset_edge2+distance_rows2*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      rip_width1+str_lenght-offset_point2+
                      str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2
                      -(rip_width1+str_lenght/2)-(i-1)*(str_lenght-2*offset_point1),
                      (skin_depth+str_depth)/2),),
                    ((offset_edge2+distance_rows2*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      rip_width1+str_lenght-offset_point2+
                      str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2)-(i-1)*(str_lenght-2*offset_point1),
                      (skin_depth+str_depth)/2),),
                    ))

                    mdb.models['Model-
1'].rootAssembly.SurfaceByBoolean(name='m_Surf2'+str(k+int(j+a))+' 1'+str(i), surfaces=(
                    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf2-
'+str(k+int(j+a))+' 1'+str(i)],
                    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf2'+str(k+int(j+a))+'
1'+str(i)]))
    a=nsup[k-1]


#superficie slave su rip2
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2+1):
            mdb.models['Model-1'].rootAssembly.Surface(name='s_Surf2-'+str(k+int(j+a))+'
1'+str(i), side1Faces=
            mdb.models['Model-1'].rootAssembly.instances['rip2-'+str(i)+'-
cr'+str(int(j+a)+1)].faces.findAt(
             (((-0.1*(rip_lenght2-
cr_lenght[str_cr_cr[int(j+a)][1]])/2+x_cr[str_cr_cr[int(j+a)][1]],
              (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2-(2*i-
3)*str_lenght*0.2),
              (skin_depth+2*str_depth+rip_depth2)/2) ),)))
    a=nsup[k-1]


a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2+1):
            for q in range (n_rows2):
                if    rip_lenght1/2-x_noHole/2    <    offset_edge2+distance_rows2*q    <
rip_lenght1/2+x_noHole/2:
                    bandiera=1
                else:
                    mdb.models['Model-
1'].rootAssembly.Surface(name='s_Surf2'+str(k+int(j+a))+' 1'+str(i), side1Faces=
```

```
                    mdb.models['Model-1'].rootAssembly.instances['rip2-'+str(i)+'-
cr'+str(int(j+a)+1)].faces.findAt(
                    ((fast_rad*1.1+offset_edge2+distance_rows2*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      rip_width1+str_lenght-offset_point2+
                      str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2
                      -(rip_width1+str_lenght/2)-(i-1)*(str_lenght-2*offset_point1),
                      (skin_depth+2*str_depth+rip_depth2)/2),),
                    ((offset_edge2+distance_rows2*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      rip_width1+str_lenght-offset_point2+
                      str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2
                      -(rip_width1+str_lenght/2)-(i-1)*(str_lenght-2*offset_point1),
                      (skin_depth+2*str_depth+rip_depth2)/2),),
                    ))

                    mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='s_Surf2-
'+str(k+int(j+a))+' 1'+str(i), surfaces=(
                    mdb.models['Model-1'].rootAssembly.surfaces['s_Surf2-
'+str(k+int(j+a))+' 1'+str(i)],
                    mdb.models['Model-1'].rootAssembly.surfaces['s_Surf2'+str(k+int(j+a))+'
1'+str(i)]))
    a=nsup[k-1]


#definizione contatto rip2 base-stringer
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2+1):
            mdb.models['Model-1'].SurfaceToSurfaceContactStd(adjustMethod=NONE,
            clearanceRegion=None, createStepName='Initial', datumAxis=None,
            initialClearance=OMIT, interactionProperty='IntProp-1', master=
            mdb.models['Model-1'].rootAssembly.surfaces['m_Surf2'+str(k+int(j+a))+'
1'+str(i)],
                                        name='Int-'+str(i+n_rip1)+'cr'+str(k+int(j+a)+1),
            slave=  mdb.models['Model-1'].rootAssembly.surfaces['s_Surf2-'+str(k+int(j+a))+'
1'+str(i)],
                                                    sliding=FINITE, thickness=ON)
    a=nsup[k-1]



#superficie master su rinforzo stringer lato interno
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
            mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf3'+str(int(j+a))+'    1',
side2Faces=
            mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(k)].faces.findAt(
             (((-0.1*(rip_lenght2-
cr_lenght[str_cr_cr[int(j+a)][1]])/2+x_cr[str_cr_cr[int(j+a)][1]],
               (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2),
               (str_height)/2) ),)))
    a=nsup[k-1]

a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf3-'+str(int(j+a))+'      1',
side2Faces=
```

```
        mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(k)].faces.findAt(
            (((0.1*(rip_lenght2-
cr_lenght[str_cr_cr[int(j+a)][1]])/2+x_cr[str_cr_cr[int(j+a)][1]],
              (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2),
              (str_height)/2) ),)))
        mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='m_Surf3'+str(int(j+a))+'
1', surfaces=(
        mdb.models['Model-1'].rootAssembly.surfaces['m_Surf3'+str(int(j+a))+' 1'],
        mdb.models['Model-1'].rootAssembly.surfaces['m_Surf3-'+str(int(j+a))+' 1']))
    a=nsup[k-1]

a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2):
            for q in range (n_rows2):
                if    rip_lenght1/2-x_noHole/2    <    offset_edge2+distance_rows2*q    <
rip_lenght1/2+x_noHole/2:
                    bandiera=1
                else:
                    mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf3-
'+str(int(j+a))+' 1', side2Faces=
                    mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(k)].faces.findAt(
                    ((offset_edge1+distance_rows1*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2),
                      fast_rad*1.1+str_height-offset_point2),),
                    ((offset_edge1+distance_rows1*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2),
                      str_height-offset_point2),),
                    ))

                    mdb.models['Model-
1'].rootAssembly.SurfaceByBoolean(name='m_Surf3'+str(int(j+a))+' 1', surfaces=(
                    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf3'+str(int(j+a))+'
1'],
                    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf3-'+str(int(j+a))+'
1']))
    a=nsup[k-1]


#superficie slave su rinforzo rip2
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2+1):
            mdb.models['Model-1'].rootAssembly.Surface(name='s_Surf3-'+str(k+int(j+a))+'
1'+str(i), side2Faces=
            mdb.models['Model-1'].rootAssembly.instances['rip2-1-
cr'+str(int(j+a)+1)].faces.findAt(
            (((-0.1*(rip_lenght2-
cr_lenght[str_cr_cr[int(j+a)][1]])/2+x_cr[str_cr_cr[int(j+a)][1]],

(str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2+(str_depth+rip_depth2/2)),
            (str_height)/2) ),)))
    a=nsup[k-1]


a=0
```

```
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2):
            for q in range (n_rows2):
                if     rip_lenght1/2-x_noHole/2     <     offset_edge2+distance_rows2*q     <
rip_lenght1/2+x_noHole/2:
                    bandiera=1
                else:
                    mdb.models['Model-
1'].rootAssembly.Surface(name='s_Surf3'+str(k+int(j+a))+' 1'+str(i), side2Faces=
                    mdb.models['Model-1'].rootAssembly.instances['rip2-1-
cr'+str(int(j+a)+1)].faces.findAt(
                    ((offset_edge1+distance_rows1*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      rip_width1+str_lenght-rip_width2+
                      str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      fast_rad*1.1+str_height-offset_point2),),
                    ((offset_edge1+distance_rows1*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      rip_width1+str_lenght-rip_width2+
                      str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      str_height-offset_point2),),
                    ))

                    mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='s_Surf3-
'+str(k+int(j+a))+' 1'+str(i), surfaces=(
                    mdb.models['Model-1'].rootAssembly.surfaces['s_Surf3-
'+str(k+int(j+a))+' 1'+str(i)],
                    mdb.models['Model-1'].rootAssembly.surfaces['s_Surf3'+str(k+int(j+a))+'
1'+str(i)]))
    a=nsup[k-1]


a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].SurfaceToSurfaceContactStd(adjustMethod=NONE,
        clearanceRegion=None, createStepName='Initial', datumAxis=None,
        initialClearance=OMIT, interactionProperty='IntProp-1', master=
        mdb.models['Model-1'].rootAssembly.surfaces['m_Surf3'+str(int(j+a))+' 1'],
        name='Int-'+str(1+n_rip1+n_rip2)+'cr'+str(int(j+a)+1),
        slave=mdb.models['Model-1'].rootAssembly.surfaces['s_Surf3-'+str(k+int(j+a))+'
1'+str(i)],
                                                    sliding=FINITE, thickness=ON)
    a=nsup[k-1]



#superficie master su rinforzo stringer lato esterno
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
            mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf4'+str(int(j+a))+'   1',
side1Faces=
            mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(k)].faces.findAt(
             (((-0.1*(rip_lenght2-
cr_lenght[str_cr_cr[int(j+a)][1]])/2+x_cr[str_cr_cr[int(j+a)][1]],
              (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2),
              (str_height)/2) ),)))
```

```
    a=nsup[k-1]

a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf4-'+str(int(j+a))+'      1',
side1Faces=
        mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(k)].faces.findAt(
            (((0.1*(rip_lenght2-
cr_lenght[str_cr_cr[int(j+a)][1]])/2+x_cr[str_cr_cr[int(j+a)][1]],
              (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2),
              (str_height)/2) ),)))
        mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='m_Surf4'+str(int(j+a))+'
1', surfaces=(
        mdb.models['Model-1'].rootAssembly.surfaces['m_Surf4'+str(int(j+a))+' 1'],
        mdb.models['Model-1'].rootAssembly.surfaces['m_Surf4-'+str(int(j+a))+' 1']))
    a=nsup[k-1]

a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2):
            for q in range (n_rows2):
                if    rip_lenght1/2-x_noHole/2    <    offset_edge2+distance_rows2*q    <
rip_lenght1/2+x_noHole/2:
                    bandiera=1
                else:
                    mdb.models['Model-1'].rootAssembly.Surface(name='m_Surf4-
'+str(int(j+a))+' 1', side1Faces=
                    mdb.models['Model-1'].rootAssembly.instances['stringer-cricca-
'+str(k)].faces.findAt(
                    ((offset_edge1+distance_rows1*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                       (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2),
                       fast_rad*1.1+str_height-offset_point2),),
                    ((offset_edge1+distance_rows1*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                       (str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2),
                      str_height-offset_point2),),
                    ))

                    mdb.models['Model-
1'].rootAssembly.SurfaceByBoolean(name='m_Surf4'+str(int(j+a))+' 1', surfaces=(
                    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf4'+str(int(j+a))+'
1'],
                    mdb.models['Model-1'].rootAssembly.surfaces['m_Surf4-'+str(int(j+a))+'
1']))
    a=nsup[k-1]


#superficie slave su rinforzo rip2
a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2+1):
            mdb.models['Model-1'].rootAssembly.Surface(name='s_Surf4-'+str(k+int(j+a))+'
1'+str(i), side2Faces=
            mdb.models['Model-1'].rootAssembly.instances['rip2-2-
cr'+str(int(j+a)+1)].faces.findAt(
            (((-0.1*(rip_lenght2-
cr_lenght[str_cr_cr[int(j+a)][1]])/2+x_cr[str_cr_cr[int(j+a)][1]],
```

```
(str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2+(str_depth+rip_depth2/2))-
2*(str_lenght/2-rip_width2),
            (str_height)/2) ),)))
    a=nsup[k-1]


a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        for i in range (1,n_rip2):
            for q in range (n_rows2):
                if    rip_lenght1/2-x_noHole/2    <    offset_edge2+distance_rows2*q    <
rip_lenght1/2+x_noHole/2:
                    bandiera=1
                else:
                    mdb.models['Model-
1'].rootAssembly.Surface(name='s_Surf4'+str(k+int(j+a))+' 1'+str(i), side2Faces=
                    mdb.models['Model-1'].rootAssembly.instances['rip2-2-
cr'+str(int(j+a)+1)].faces.findAt(
                    ((offset_edge1+distance_rows1*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      rip_width1+str_lenght-rip_width2+
                      str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2
                      -(rip_width1+str_lenght/2)-2*(str_lenght/2-rip_width2),
                      fast_rad*1.1+str_height-offset_point2),),
                    ((offset_edge1+distance_rows1*q+
                      x_cr[str_cr_cr[int(j+a)][1]]+cr_lenght[str_cr_cr[int(j+a)][1]]/2-
rip_lenght1/2,
                      rip_width1+str_lenght-rip_width2+
                      str_cr_cr[int(j+a)][0]*distance_stringer+offset_stringer/2
                      -(rip_width1+str_lenght/2)-2*(str_lenght/2-rip_width2),
                      str_height-offset_point2),),
                    ))

                    mdb.models['Model-1'].rootAssembly.SurfaceByBoolean(name='s_Surf4-
'+str(k+int(j+a))+' 1'+str(i), surfaces=(
                    mdb.models['Model-1'].rootAssembly.surfaces['s_Surf4-
'+str(k+int(j+a))+' 1'+str(i)],
                    mdb.models['Model-1'].rootAssembly.surfaces['s_Surf4'+str(k+int(j+a))+'
1'+str(i)]))
    a=nsup[k-1]


a=0
for k in range (1,n_str_cr+1):
    for j in range (int(nsup[k-1])):
        mdb.models['Model-1'].SurfaceToSurfaceContactStd(adjustMethod=NONE,
        clearanceRegion=None, createStepName='Initial', datumAxis=None,
        initialClearance=OMIT, interactionProperty='IntProp-1', master=
        mdb.models['Model-1'].rootAssembly.surfaces['m_Surf4'+str(int(j+a))+' 1'],
        name='Int-'+str(2+n_rip1+n_rip2)+'cr'+str(int(j+a)+1),
        slave=mdb.models['Model-1'].rootAssembly.surfaces['s_Surf4-'+str(k+int(j+a))+'
1'+str(i)],
                                                    sliding=FINITE, thickness=ON)
    a=nsup[k-1]

#creazione step
mdb.models['Model-1'].StaticStep(initialInc=0.01, name='load', previous=
    'Initial')


#applicazione vincoli
```

```
mdb.models['Model-1'].rootAssembly.Set(edges=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].edges.findAt(
    ((0,skin_width/2,0), ), ), name='Set costraint')
mdb.models['Model-1'].EncastreBC(createStepName='Initial', localCsys=None,
    name='encastre', region=mdb.models['Model-1'].rootAssembly.sets['Set costraint'])


#applicazione carichi
rp_F=mdb.models['Model-1'].rootAssembly.ReferencePoint(point=(skin_lenght*1.1,
skin_width/2,0))
mdb.models['Model-1'].rootAssembly.Set(name='Set_Force', referencePoints=(
    mdb.models['Model-1'].rootAssembly.referencePoints[rp_F.id], ))
mdb.models['Model-1'].rootAssembly.Surface(name='s_Surf_F', side1Edges=
    mdb.models['Model-1'].rootAssembly.instances['skin-1'].edges.findAt(
    ((skin_lenght, skin_width/2, 0),), ))

mdb.models['Model-1'].Coupling(controlPoint=
    mdb.models['Model-1'].rootAssembly.sets['Set_Force'], couplingType=KINEMATIC,
    influenceRadius=WHOLE_SURFACE, localCsys=None, name='Coupling_F',
    surface=mdb.models['Model-1'].rootAssembly.surfaces['s_Surf_F'], u1=ON,
    u2=ON, u3=ON, ur1=ON, ur2=ON, ur3=ON)

mdb.models['Model-1'].ConcentratedForce(cf1=10e5, createStepName='load',
    distributionType=UNIFORM, field='', localCsys=None, name='Load', region=
    Region(referencePoints=(mdb.models['Model-1'].rootAssembly.referencePoints[rp_F.id],
)))


#FASTENER
#Attachment Points
rp=mdb.models['Model-1'].rootAssembly.ReferencePoint(point=
                    (offset_edge1+x_cr[0]+cr_lenght[0]/2-rip_lenght1/2,
                      offset_point1+y_cr[0]+cr_width[0]/2-(rip_width1+str_lenght/2),
                      0))
mdb.models['Model-1'].rootAssembly.Set(name='Set-RP', referencePoints=(
    mdb.models['Model-1'].rootAssembly.referencePoints[rp.id],))


for i in range (n_cr):
    for k in range (n_points1):
        for q in range (n_rows1):
            if      rip_lenght1/2-x_noHole/2      <      offset_edge2+distance_rows2*q      <
rip_lenght1/2+x_noHole/2 and\
               rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                bandiera=1
            else:
                rp=mdb.models['Model-1'].rootAssembly.ReferencePoint(point=
                        (offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-
rip_lenght1/2,
                          distance_points1*k+offset_point1+
                         str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                          0))
                mdb.models['Model-1'].rootAssembly.Set(name='SetRP', referencePoints=(
                mdb.models['Model-1'].rootAssembly.referencePoints[rp.id],))
                mdb.models['Model-1'].rootAssembly.SetByBoolean(name='Set-RP', sets=(
                mdb.models['Model-1'].rootAssembly.sets['Set-RP'],
                mdb.models['Model-1'].rootAssembly.sets['SetRP']))


for i in range (n_cr):
    for k in range (n_points1):
        for q in range (n_rows1):
```

```
            if      rip_lenght1/2-x_noHole/2      <      offset_edge2+distance_rows2*q      <
rip_lenght1/2+x_noHole/2 and\
                rip_width1/2+str_lenght/2-
y_noHole/2<distance_points1*k+offset_point1<rip_width1/2+str_lenght/2+y_noHole/2:
                 bandiera=1
            else:
                rp=mdb.models['Model-1'].rootAssembly.ReferencePoint(point=
                ((offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                          2*(rip_width1+str_lenght/2)-(distance_points1*k+offset_point1)+
                  str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                          0)))
                mdb.models['Model-1'].rootAssembly.Set(name='SetRP', referencePoints=(
                mdb.models['Model-1'].rootAssembly.referencePoints[rp.id],))
                mdb.models['Model-1'].rootAssembly.SetByBoolean(name='Set-RP', sets=(
                mdb.models['Model-1'].rootAssembly.sets['Set-RP'],
                mdb.models['Model-1'].rootAssembly.sets['SetRP']))


for i in range (n_cr):
    for q in range (n_rows2):
        if       rip_lenght1/2-x_noHole/2       <       offset_edge2+distance_rows2*q       <
rip_lenght1/2+x_noHole/2:
            bandiera=1
        else:
            rp=mdb.models['Model-1'].rootAssembly.ReferencePoint(point=
            ((offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      rip_width1+offset_point1+
              str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0)))
            mdb.models['Model-1'].rootAssembly.Set(name='SetRP', referencePoints=(
            mdb.models['Model-1'].rootAssembly.referencePoints[rp.id],))
            mdb.models['Model-1'].rootAssembly.SetByBoolean(name='Set-RP', sets=(
            mdb.models['Model-1'].rootAssembly.sets['Set-RP'],
            mdb.models['Model-1'].rootAssembly.sets['SetRP']))

for i in range (n_cr):
    for q in range (n_rows2):
        if       rip_lenght1/2-x_noHole/2       <       offset_edge2+distance_rows2*q       <
rip_lenght1/2+x_noHole/2:
            bandiera=1
        else:
            rp=mdb.models['Model-1'].rootAssembly.ReferencePoint(point=
            ((offset_edge1+distance_rows1*q+x_cr[i]+cr_lenght[i]/2-rip_lenght1/2,
                      rip_width1+str_lenght-offset_point1+
              str_cr_cr[i][0]*distance_stringer+offset_stringer/2-
(rip_width1+str_lenght/2),
                      0)))
            mdb.models['Model-1'].rootAssembly.Set(name='SetRP', referencePoints=(
            mdb.models['Model-1'].rootAssembly.referencePoints[rp.id],))
            mdb.models['Model-1'].rootAssembly.SetByBoolean(name='Set-RP', sets=(
            mdb.models['Model-1'].rootAssembly.sets['Set-RP'],
            mdb.models['Model-1'].rootAssembly.sets['SetRP']))


for i in range (n_cr):
    for q in range (n_rows2):
        if       rip_lenght1/2-x_noHole/2       <       offset_edge2+distance_rows2*q       <
rip_lenght1/2+x_noHole/2:
            bandiera=1
        else:
            rp=mdb.models['Model-1'].rootAssembly.ReferencePoint(point=
            ((offset_edge1+distance_rows1*q+
```

```
                    x_cr[str_cr_cr[i][1]]+cr_lenght[str_cr_cr[i][1]]/2-rip_lenght1/2,
                    (str_cr_cr[i][0]*distance_stringer+offset_stringer/2),
                    fast_rad+str_height-offset_point2)))
            mdb.models['Model-1'].rootAssembly.Set(name='SetRP', referencePoints=(
            mdb.models['Model-1'].rootAssembly.referencePoints[rp.id],))
            mdb.models['Model-1'].rootAssembly.SetByBoolean(name='Set-RP', sets=(
            mdb.models['Model-1'].rootAssembly.sets['Set-RP'],
            mdb.models['Model-1'].rootAssembly.sets['SetRP']))


#Eliminazione punto di supporto
del mdb.models['Model-1'].rootAssembly.features['RP-2']

#Connector Section
mdb.models['Model-1'].ConnectorSection(assembledType=BEAM, name='ConnSect')

mdb.models['Model-1'].rootAssembly.engineeringFeatures.PointFastener(name=
    'Fasteners-1', physicalRadius=2.0, region=
    mdb.models['Model-1'].rootAssembly.sets['Set-RP'], sectionName='ConnSect')
```