Programmazione e calcolo scientifico

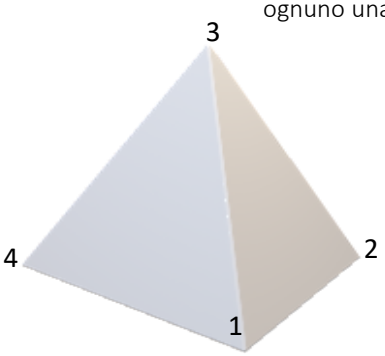
Tetraedrizzazione

Gabriele Fioravanti

Gianluigi Lopardo

Introduzione e struttura

Il programma è strutturato in un file main (*index.m*), che costituisce il cuore del progetto e richiama le diverse funzioni definite in file separati. Vengono lette dai file tutte le informazioni necessarie con la funzione *fscanf*, generando per ognuno una matrice nome con il rispettivo contenuto e una variabile contenente il primo carattere utile del file, relativo al numero degli elementi del file corrispondente (ad esempio la matrice corrispondente al file *barra.1.ele* (contenente le informazioni sui tetraedri) si chiamerà *ele*, la variabile corrispondente al numero di triangoli *n_ele*). Le variabili così generate vengono dichiarate come globali, in quanto dovranno tutte essere accessibili da più di una funzione esterna.



Una volta ricavate le informazioni necessarie dai file, utilizziamo la seguente struttura per eseguire il programma.

I sei lati sono ordinati come segue:

1	*1 – 2*
2	*1 – 3*
3	*1 – 4*
4	*2 – 3*
5	*2 – 4*
6	*3 – 4*

Dove *i – j* sono i rispettivi nodi di interesse.

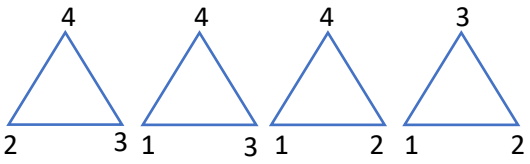
Le quattro facce sono ordinati come segue:

1	*4 – 5 – 6*
2	*2 – 3 – 6*
3	*1 – 3 – 5*
4	*1 – 2 – 4*

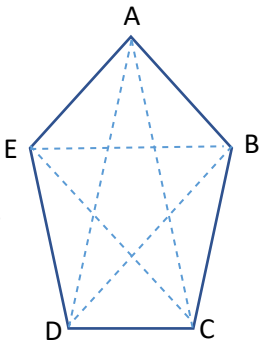
Dove *i – j – k* sono i lati identificati come descritto a fianco.

Le facce sono ordinate avendo i tre lati indicizzati come segue:

Faccia / Lati	**1**	**2**	**3**	**4**
1	*2 – 3*	*1 – 3*	*1 – 2*	*1 – 2*
2	*3 – 4*	*3 – 4*	*2 – 4*	*2 – 3*
3	*4 – 2*	*4 – 1*	*4 – 1*	*3 – 1*



Trovapiano e posnodi



Con la funzione *trovapiano* ricaviamo l'equazione del piano su cui giace la frattura.

Partiamo dalla generica equazione del piano π: $ax + by + cz + d = 0$

Ottenendo il vettore normale n ricaviamo i coefficienti a, b e c.

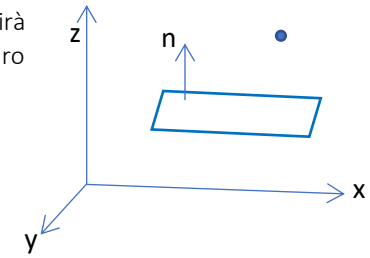
Viene eseguito il prodotto vettoriale di tutte le possibili coppie di punti dei vertici della frattura (che sono $\binom{n\ punti}{2}$) e trovato il vettore normale medio n. Infine, sostituendo i valori dei punti della frattura ai coefficienti a, b e c per ricavare il valore medio di d, individuando così univocamente il piano.

$$d = -(v_1 x_p + v_2 y_p + v_3 z_p)$$

Tramite la funzione *posnodi* stabiliamo la posizione dei nodi rispetto al piano.

Il piano dividerà lo spazio in due aree, conoscere la posizione relativa dei nodi ci servirà per valutare se può sussistere o meno una intersezione. Per ogni nodo del tetraedro eseguiamo il prodotto scalare $\mathrm{r} = \mathrm{node}(\mathrm{x},\mathrm{y},\mathrm{z}) \cdot \mathrm{n}(\mathrm{x},\mathrm{y},\mathrm{z}) + \mathrm{d}$.

Assegniamo quindi valore 1 se $\mathrm{r} > 0$, 0 se $\mathrm{r} = 0$, -1 se $\mathrm{r} < 0$.



Intersezione e suddivisione della frattura

Dal main viene chiamata la funzione *tetra*, che gestisce le funzioni per le intersezioni, la creazione dei sottopoliedri e la suddivisione della frattura in poligoni. Viene inizializzato *tetra_inters* come vettore nullo di dimensione *n_ele* (numero dei tetraedri). Inizia il controllo in ordine crescente, dal primo tetraedro. Quando si arriva ad una intersezione il programma entra nella funzione *Coda.m*. Appena trovo una intersezione in un tetraedro, verifico tra i suoi vicini e metto in coda a loro volta i vicini dei vicini nel caso in cui ci sia intersezione. Dopo aver verificato che il vicino j-esimo esista (*if neigh(i,j) ~= -1*), ci assicuriamo di non aver già controllato il tetraedro in esame (*if tetra_inters(neigh(i,j)) == 0*) e che non sia già stato inserito nella coda (*if neigh(i,j)==coda(z).* In caso affermativo inserisco il tetraedro nella coda dei tetraedri da controllare. Così facendo, invece di eseguire il ciclo su ogni tetraedro, eseguo i controlli solo sui tetraedri inseriti nella coda, in cui potrebbe esserci intersezione fino ad esaurire la coda.

Inizialmente tramite la funzione *Intersezione_punti* calcoliamo i punti di intersezione tra il piano π su cui giace la frattura (individuata dai parametri n e d calcolati precedentemente) ed i lati del tetraedro (individuato di volta in volta dall'indice i) che sono caratterizzati dai nodi N_1, N_2, N_3 e N_4, del tetraedro in questione.

La funzione riceverà inoltre in output la variabile *posizionenodi* che dice dove stanno i nodi rispetto al piano π.

Poiché in *posnodi* vi è la posizione di tutti i nodi definiti nella variabile *node* devo prima ricavare i nodi di interesse del *tetraedro* individuato da i.

Dati due nodi vi sarà intersezione tra il piano π ed il lato di interesse solo se un nodo sta sopra il piano mentre un altro sta sotto il piano. Dunque ciclo sui lati, che sono sei, e quando la condizione è verificata andiamo a calcolare i punti di intersezione come segue:

L'equazione cartesiana del piano sarà: $\pi: (\mathrm{x},\mathrm{y},\mathrm{z}) \cdot n^t + d = 0$

Equazione implicita del lato: $\gamma: P_1 + t(P_2 - P_1)$

Dove P_2 e P_1 sono due nodi del tetraedro.

Allora sostituendo l'equazione di γ nell'equazione del piano ricavo l'ascissa curvilinea t che mi individua il punto di intersezione:

$$\mathrm{t} = -\frac{P_1 n^T + d}{(P_2 - P_1)n^T}$$

Allora il punto di intersezione sarà: $\mathrm{P} = P_1 + t(P_2 - P_1)$

Ottenuto sostituendo l'ascissa curvilinea t nell'equazione della curva γ.

1 – 2	Lato 1
1 – 3	Lato 2
1 – 4	Lato 3
2 – 3	Lato 4
2 – 4	Lato 5
3 – 4	Lato 6

Inoltre salvo la posizione del lato a cui appartiene il punto come segue:

Avrò come output:

- *punti_intersezione*: matrice con i punti calcolati
- *pos_int*: vettore che rispettivamente dice a quale lato appartiene il punto di intersezione
- *pos_nodi*: vettore con la posizione dei nodi del tetraedro di interesse rispetto al piano π.

La funzione *Poliedri_intersezione* serve per trovare i poliedri che vengono intersecati dalla frattura e per andare a suddividere la frattura in poligoni ottenuti dall'intersezione. Gli elementi in ingresso di tale funzione sono:

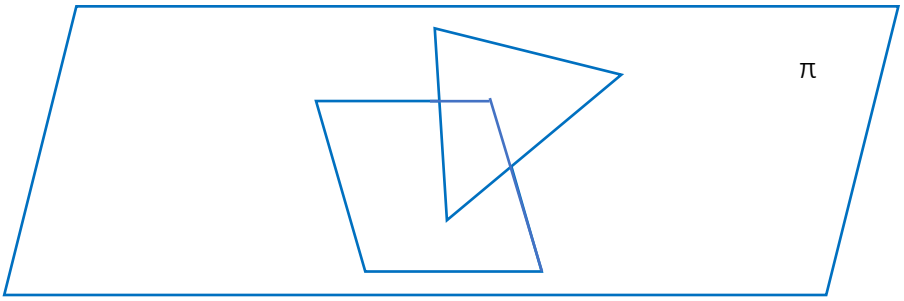
- *facc_punti*: i punti di intersezione relativi al tetreaedro considerato e la frattura
- *fra_punti*: i punti della frattura
- *posnodi*: posizione dei nodi rispetto al piano su cui giace la frattura
- *i_tetra*: indice del tetraedro considerato
- *punti_fratt*: punti che suddividono la frattura; verrà aggiornato ogni volta che si trova un'intersezione.
- *n*: versore normale al piano della frattura.

Innanzitutto, escludo i casi in cui tutti i nodi del tetraedro stanno sopra la frattura (sommo la posizione dei nodi e se essa è $\mathrm{p} == 4$ o $\mathrm{p} == -4$ so che non potrà esservi intersezione.

Dopodiché se la somma $\mathrm{p} == 3$ o $\mathrm{p} == -3$ vuol dire che tutti i nodi stanno rispettivamente sopra o sotto la frattura tranne un vertice. Anche in questo caso non può esservi intersezione.

Nel caso in cui la frattura è parallela ad una faccia del tetraedro salvo i nodi di interesse della faccia considerata e con un flag che pongo uguale a zero tengo conto di questa situazione.

Esauriti questi casi, andiamo a capire se vi è intersezione tra il tetraedro e la frattura.

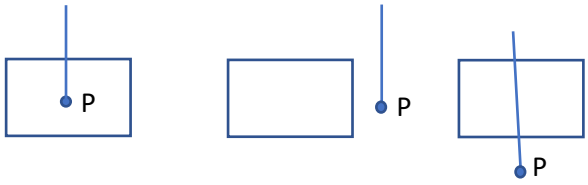


π è il piano dove giace la frattura, che nell'immagine è il quadrilatero, mentre nel triangolo è il taglio che si è andato a formare tra il tetraedro ed il piano π.

Allora, per vedere se vi è effettivamente intersezione bisogna che v sia intersezione tra i due poligoni. Quindi il nostro problema si riduce a capire se sussiste tale intersezione o meno.

Riusciamo a trovare l'intersezione se almeno un punto del taglio è interno alla frattura o se almeno un punto della frattura è interno al taglio.

Per verificarlo, fisso arbitrariamente un versore *t* che appartiene al piano π e calcolo l'intersezione della semiretta con vertice P (dove P è il punto di interesse) con ogni lato del poligono ed ho i seguenti casi:



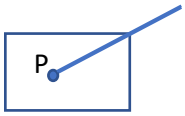
Se la semiretta interseca il poliedro una sola volta allora sicuramente il punto è interno al poliedro, se interseca il poliedro 0 o due volte il punto è esterno.

Viene chiamata la funzione *inters_interno* che serve a calcolare appunto l'intersezione tra un lato e la semiretta.

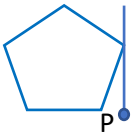
Nella funzione, a cui passo come dati $\mathrm{A, B, P}$ e $\mathrm{tan(P)}$, inizialmente trovo la forma cartesiana delle rette $\mathrm{A-B}$ e della retta che passa per P con direzione $\mathrm{tan(P)}$.

Avrò così un sistema di quattro equazioni in tre incognite. Escludendo i casi in cui le rette sono parallele so che il sistema ammette soluzione poiché le rette considerate sono complanari giacendo sul piano π.

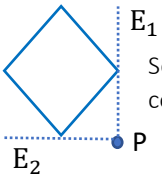
Trovo dopo il punto di intersezione e vado a calcolare le coordinate curvilinee di tale punto rispetto alla prima retta *t* e rispetto alla seconda (*tAB*). Affinchè il punto appartenga effettivamente al lato bisogna avere $\mathrm{tAB}\epsilon[0,1]$ e affinché appartenga alle semirette bisogna avere $\mathrm{t}>0$. In tal caso avrò l'intersezione. Vado solo a discernere il caso in cui il punto d'intersezione è un vertice del lato e darò il valore *int=0.5* rispetto ad 1 nel caso generale, poiché riceverò sicuramente un'altra intersezione per lo stesso vertice.



Si potrebbe però presentare il seguente problema:



In questo caso nonostante il punto sia esterno avrò che il risultato del precedente codice mi darà una sola intersezione. Allora per ovviare a tale problema eseguo lo stesso codice con tre versori E_1, E_2, E_3 (due non sono sufficienti) e per ognuno di essi calcolo il numero di intersezioni.



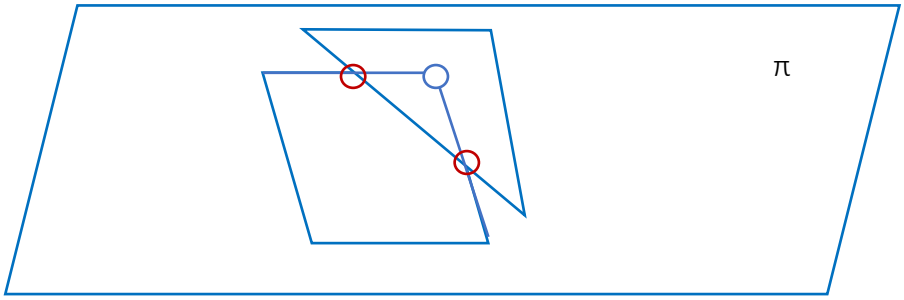
Se i tre valori coincidono e sono uguali ad uno allora il punto è interno, se non coincidono oppure coincidono ma sono diversi da 1 allora il punto è esterno.

Nel codice prima di calcolare l'intersezione si controlla se il punto *P* appartiene ad un lato tramite la funzione *punti_appartiene_segmento* che agisce come segue: per ogni coordinata *x, y, z* trovo l'ascissa curvilinea t_1, t_2, t_3 che mi dice il valore per cui la coordinata del punto *P* appartiene al segmento, ovvero considerando il lato $\mathrm{A-B}$ non parallelo gli assi ho che

$$t_1 = \frac{x_P - x_A}{x_B - x_A};\ t_2 = \frac{y_P - y_A}{y_B - y_A};\ t_3 = \frac{z_P - z_A}{z_B - z_A}.$$

Se tali valori coincidono e sono compresi tra 0 e 1 ho che il punto *P* appartiene al segmento con estremi A e B. Allora nel caso in cui il punto appartiene ad un lato del poligono o è interno ad esso salvo tale punto in una matrice *Punti_int_fra* (per la frattura) o *Punti_int_facc* (per la faccia di taglio). Appena trovo uno di tali punti so che vi è intersezione tra tetraedro e frattura.

A questo punto creo le sottofratture:



Nelle matrici sopra definite avremo il punto cerchiato in blu. Vogliamo trovare i punti cerchiati in rosso che sono dati dall'intersezione tra i lati della frattura e dalla faccia di taglio.

Supposto che tutti i punti della frattura non appartengono al taglio o viceversa vado a ciclare su tutti i lati per trovare i punti considerati tramite la funzione *intersezione_segmenti*.

Tale funzione è analoga a *inters_interno* con la differenza che invece di avere $\mathrm{A}, \mathrm{B}, \mathrm{P}, \tan(\mathrm{P})$ come dati ho quattro punti $\mathrm{A}-\mathrm{B}, \mathrm{C}-\mathrm{D}$ che formano i lati per cui voglio calcolare l'intersezione. Salvo tali punti in *Punti_fratt_taglio*.

Vado a ordinare tramite *ordina_frattura* tali punti in *Punti_frattura* che mi restituirà degli indici che andrò ad inserire nella matrice *sotto_frattura* che sarà così definita:

sottofrattura:

$\boldsymbol{i_1}$	-1	N_1	N_2	N_3	0
$\boldsymbol{i_2}$	-1	N_4	N_5	N_6	N_7
$\boldsymbol{i_3}$	-1	"	"	"	"

dove i_1, i_2, i_3 sono i poliedri con cui avviene l'intersezione, -1 mi serve come separatore e N_1, N_2,… sono i punti che andranno a formare la sottofrattura.

Crea sottopoliedri

Per creare i sottopoliedri ottenuti dall'intersezione tra i tetraedri e la frattura ed un suo possibile prolungamento andiamo a distinguere quattro casistiche corrispondenti ai punti di intersezione che non siano dei vertici del tetraedro. Appena si trova una intersezione si forniscono alcuni dati alla funzione *sottopoliedri,* quali: *punti_intersezione*, *pos_int* (contenente la posizione dei punti di intersezione ordinati come sopra), *i_tetra* (indice del tetraedro di interesse); *pos_nodi* (indice della posizione dei nodi del tetraedro rispetto la frattura: 1 se sopra, -1 se sotto, 0 se sovrapposto); *Poliedri; Facce_pol, Lati_pol; Punti_pol*. Questi ultimi quattro dati vengono aggiornati ogni volta che si trova una nuova intersezione.

Inizialmente viene eseguita la funzione *Ordina_punti* che aggiorna i punti dei futuri sottopoliedri e in uscita ritorna degli indici che individuano tali punti ordinati come segue:

$$\texttt{indice\_punti} = [N_1, N_2, N_3, N_4, P_1, P_2, P_3, P_4, \dots]$$

dove N_i sono i nodi del tetraedro e P_i sono i punti di intersezione tra frattura e tetraedro.

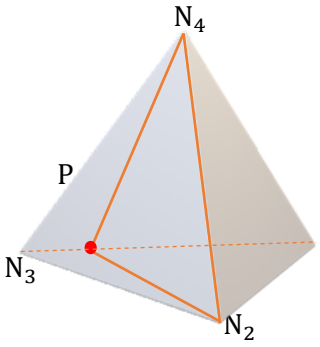
In seguito viene eseguita la funzione *posizioneinterna_facceintersecate,* dove si avrà in uscita la posizione dei punti di intersezione relativamente ad ogni faccia, dove i lati vengono ordinati facendo riferimento alla struttura sopra definita.

Questa funzione restituirà anche un vettore che indica quali facce sono intersecate ed il numero di intersezioni *n_inters.*

In base al numero di intersezioni vi sarà una differente funzione che andrà a creare i sottopoliedri, ad esempio:

`If n_inters==1 uso Crea_poliedri_1`

`Crea_poliedri_1`



In questa situazione abbiamo un altro punto di intersezione P e due vertici che appartengono alla frattura.

Inizialmente, grazie a *pos_nodi*, trovo quali sono tali vertici e ne salvo la posizione interna relativa alla struttura sopra. Creo la faccia del taglio (in figura N_2, N_4, P) e salvo l'indice *i_taglio* che la caratterizza.

Nel tetraedro avrò due facce che non hanno intersezione e due facce che ce l'hanno. Allora, grazie a *facce_inters*, inizialmente salvo le facce senza intersezione che verranno caratterizzate da *i_base:* vettore di due elementi. Poi vado a trovare le facce con l'intersezione e vado a controllare quale vertice (tra N_2 ed N_4 in figura) che appartiene alla traccia deve esser passato come dato alla funzione *Crea_facce_2.* Allora vado a creare le facce relative ai triangoli intersecati.

Infine avrò:

- *i_taglio* (un elemento);
- *i_base* (due elementi, facce non intersecate);
- *i_facce* (quattro elementi, facce intersecate).

Adesso bisogna capire quali sono le facce non intersecate (*i_base(1)).* Ad esempio avrò che tale faccia condivide un lato formato da due nodi con due triangoli di lato formato da due nodi con due triangoli di *i_facce* (nota che se ad esempio condivide un lato con *i_facce(1)* non può condividerle con *i_facce(2)*. Allora controllo per *i_facce(1)* ed *i_facce(3)* se tali facce condividono il primo lato con *i_base(1).*

Se *i_facce(1)* condivide il lato, allora andrà a formare il poliedro, se non lo condivide allora sarà *i_facce(2)* che andrà a formare il poliedro relativo ad *i_base(1).*

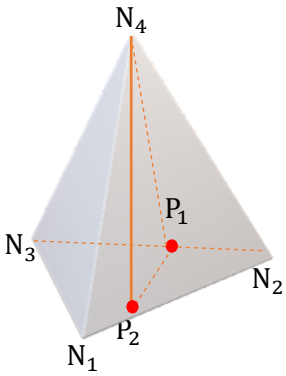
Poiché i poliedri che si creano hanno la seguente struttura:

`Poliedri = [i_base(1), i_taglio, i_facce(i), i_facce(j)]`

`Poliedri = [i_base(2), i_taglio, i_facce(i'), i_facce(j')]`

Dove i può essere 1 o 2 e rispettivamente j può essere 3 o 4. Mentre per i' e j' dovranno essere l'altro indice relativo di i e j. Ovvero se $i = 1$ e $j = 3$ allora $i' = 2$ e $j' = 4$.

`Crea_poliedri_2`

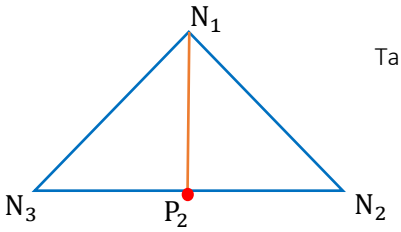


Vi sono due punti di intersezione, dunque un vertice giace sulla frattura. Grazie a *posnodi* trovo il vertice in questione e secondo la struttura sopra ne scrivo la posizione interna relativa alle singole facce.

Vado a creare la faccia di taglio che nel caso in questione sarà formata da P_1, P_2 ed N_4.

Successivamente, sempre secondo la struttura sopra, creo le facce per il triangolo opposto al vertice che giace sulla frattura, attraverso la funzione *Crea_facce_1.*

Grazie a *facce_inters* conosco le facce intersecate dove andrò a creare le sottofacce che in questo caso saranno due triangoli grazie alla funzione *Crea_facce_2*.



Tale funzione, nel caso in figura, creerà i seguenti:

$$\texttt{Lati} = [N_1\,N_2\,;\,N_2\,P_1;\,N_3\,P_2\,]$$

$$\texttt{Facce} = [i(1)\;\;i(2)\;\;i(3)\;\;-1;\;i(4)\;\;i(5)\;\;i(3)\;\;-1]$$

Inoltre vi sarà una faccia del tetraedro che non sarà intersecata, che andrò a salvare e il suo indice sarà *i_base*.

Avrò quindi *i_base* per la faccia non intersecata, *i_taglio* per il taglio e il vettore *i_facce* di sei elementi dove avrò salvato le facce ottenute dall'intersezione che avranno il seguente ordine: le prime due sono le facce ottenute da *Crea_facce_1* (quindi per il triangolo opposto al vertice) poi vi saranno le altre quattro create da *Crea_facce_2*. È importante notare che le facce ottenute da questa funzione avranno una struttura tale che il primo elemento (quindi il primo lato) sarà quello formato da due nodi. Ad esempio nella figura si avrà che che il primo elemento della prima faccia sarà $N_1 - N_2$ ed il primo elemento della seconda faccia sarà $N_1 - N_3$.

Ovviamente tali elementi saranno caratterizzati da degli indici. Per andare a creare i poliedri devo andare a trovare le due sottofacce triangolari (indicizzate da 3 e 6) che hanno un lato formato da due nodi in comune.

Nell'immagine sopra tali facce saranno $P_1 - N_2 - N_4$ e $N_4 - N_2 - P_1$ che condividono il lato $N_2 - N_4$.

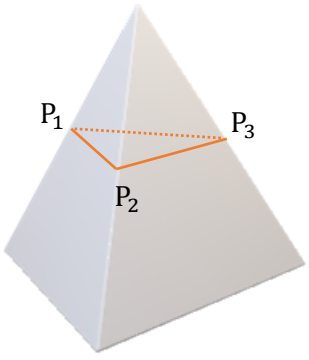
Eseguo dunque un controllo sulla prima posizione di *Facce_pol* per individuare quali delle quattro facce condividono tale lato e andrò a creare i poliedri come si vede nell'immagine.

Poliedro 1 formato da: $P_1 - N_2 - N_2$; $P_1 - N_2 - N_1$; $P_1 - N_2 - N_4$; $N_4 - P_1 - P_2$.

Poliedro 2 formato da: $N_1 - N_3 - N_4$; $N_4 - P_1 - P_2$; $N_1 - P_2 - P_1 - N_3$; $N_1 - P_2 - N_4$; $N_3 - P_1 - N_1$.

Quindi tutta tale funzione verte sul trovare il lato, formato da due nodi, in comune tra le facce create da *Crea_facce_2.*

`Crea_poliedri_3`

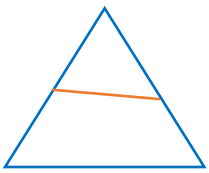


Siamo nel caso in cui vi sono tre punti di intersezione, la frattura andrà a formare la faccia $P_1 - P_2 - P_3$, allora grazie agli indici `i(5)`, `i(6)`, `i(7)` formo i lati 5 – 6, 6 – 7, 7 – 5, che vado ad ordinare tramite *ordina_lati* (analogo a ordina_punti), che restituirà degli indici per tali lati. Analogamente costituisco la faccia grazie ai lati.

Avrò tre facce intersecate a cui appartengono due dei punti di intersezione. Grazie a *facce_inters,* a cui verranno passati come argomenti i nodi di una faccia e due punti di intersezione. I nodi saranno ordinati come mostrato nella tabella sopra.

A tale funzione viene passato anche l'indice del punto di intersezione da cui si ricava a quale lato esso appartiene. Allora tramite alcuni controlli vado a creare i lati di interesse e le facce di interesse che verranno salvati rispettivamente in *Lati_pol* e *Facce_pol* e avrò indietro degli indici che individuano univocamente tali lati e tali facce.

Ad esempio, supponiamo di essere nel caso seguente:



Allora avrò i seguenti:

$$\texttt{Lati} = [N_2\,P_1,\;P_1P_2,\;P_2\,N_2\,,\;N_1\,P_1,\;N_3P_2]$$

$$\texttt{Facce} = [i\_lati(2)\;i\_lati(2)\;\;i\_lati(3)\;\;-1;\;i\_lati(4)\;i\_lati(5)\;i\_lati(6)\;i\_lati(2)]$$

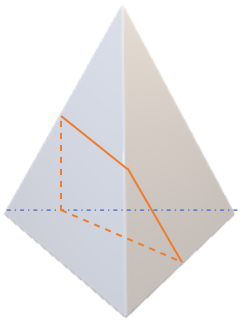
Inoltre andrò a trovare la faccia dove non vi è intersezione e analogamente a quanto sopra, verrà creata.

Infine saranno creati i poliedri tramite le facce create: avrò *indice_facce* di sei elementi che sono punti ottenuti tramite *Crea_facce_1*, *indice_taglio* e *indice_base*.

Le facce triangolari saranno in posizione 1, 3, 5 e quelle di quattro lati in posizione 2, 4, 6. Allora i sottopoliedri saranno così formati:

```
Poliedri(x+1,: ) = [i_f(1), i_f(3), i_f(5), i_taglio, -1]
Poliedri(x+2,: ) = [i_f(2), i_f(4), i_f(6), i_taglio, i_base]
```

`Crea_poliedri_4`



Trattiamo il caso in cui vi sono quattro punti di intersezione. Inizialmente viene creata la faccia del taglio attraverso la funzione *taglio_4*, a cui passo la posizione e gli indici dei punti. Tale funzione costruirà i lati della faccia, poiché alcuni di tali lati non sono ammissibili (ad esempio $P_2 - P_4$).

Dalla posizione del punto di intersezione posso stabilire quale sia il punto che non andrà a forare il lato della faccia del taglio.

In seguito creo quattro altre facce. In questo caso tutte le facce sono intersecate e tutte e quattro sono divise in un triangolo ad un quadrilatero della funzione *Crea_facce_1*. Devo solo effettuare dei controlli per capire quali sono i punti di intersezione che appartengono alla faccia che si andrà a dividere.

Allora si andrà a creare un vettore *i_facce* di otto elementi. Per capire quali sono le facce che vanno a costruire il sottopoliedro vedo che ognuno di questi è formato da *i_taglio*, due triangoli e due quadrilateri. Questi ultimi due andranno a condividere il lato formato dai due nodi. Allora, essendo i quadrilateri nella posizione pari di *i_facce*, devo controllare che il primo lato di *Facce_Pol*(*i_facce(2k)*), poiché quando vado ad eseguire la funzione *Crea_facce_1* nella prima posizione vado a salvare il lato formato da due nodi. Allora vi sono tre casistiche:

```
Facce_pol(i_facce(2),1) = Facce_pol(i_facce(4),1)
Facce_pol(i_facce(2),1) = Facce_pol(i_facce(6),1)
Facce_pol(i_facce(2),1) = Facce_pol(i_facce(8),1)
```

Che vanno ad esaurire i vari casi.

Tramite dei controlli andrò allora a formare i poliedri che si sono andati a creare con l'intersezione.

Vicini

Tetra_vicini è la funzione che serve ad individuare i tetraedri che condividono almeno un vertice con un tetraedro tagliato e memorizza i vertici, i lati e le facce che sono condivise con tali tetraedri. In input viene passato solamente la variabile *tetra_inters*, un vettore dove ricavo quali sono i tetraedri intersecati e quali non lo sono (tramite i valori 1, 1.5, 0, -1).

Tale funzione agisce come segue: ciclo sui tetraedri e appena trovo la prima intersezione

```
if (tetra_inters(i)==1 || tetra_inters(i)==1.5)
```

Vado a confrontare i nodi del tetraedro *i* con tutti gli altri tetraedri che non sono stati interesecati, ovvero

```
if (tetra_inters(j)~=1 && tetra_inters(i)~=1.5)
```

Quando trovo che questi hanno dei nodi in comune li salvo in *punti_provv*, ad esempio per il primo nodo avrò:

```
if ele(i,1)==ele(j,k)
    x=length(punti_provv);
    punti_provv(x+1)=ele(i,1);
```

```
                end
```
Dove *k* cicla da 1 a 4, su tutti i nodi del tetraedro j.

In seguito ordino i nodi tramite la funzione *Ordina_punti_vicini* e li salvo nella matrice *punti_vicini*, dove vi sono degli indici che si riferiscono alla variabile *global node*.

Salvo quindi il tetraedro *j* nel vettore vicini, controllando che esso non appaia già tramite un flag. Dopo, se *l=length(punti_provv)=2* vuol dire che *j* condivide un lato che salvo in *lati_vicini* dopo aver controllato che esso non vi sia tramite *Ordina_lati*. Se *l=3* so che il tetraedro *j* condivide una faccia, allora salvo i tre lati come in precedenza in lati vicini e salvo la faccia in *facce_vicine* dopo averle ordinate con *Ordina_facce_vicini*.

Avrò in output:

- *vicini:* i tetraedri vicini
- *punti_vicini*: i nodi condivisi individuati da indici di riferimento a *node*
- *lati_vicin*i: i lati condivisi, ognuno individuato da due indici con riferimento a *node*.
- *facce_vicini*: le facce condivise, individuate da tre indici con riferimento a *node*.

Codice

index.m

```
global node n_node n_ele ele neigh fract punti n_punti

[edge, n_edge] = edgetomatr('tetgen_e_fratture/barra.1.edge');
[ele, n_ele] = filetomatr('tetgen_e_fratture/barra.1.ele');
[neigh, n_neigh] = filetomatr('tetgen_e_fratture/barra.1.neigh');
[node, n_node] = filetomatr('tetgen_e_fratture/barra.1.node');
[face, n_face] = facetomatr('tetgen_e_fratture/barra.1.face');
[punti, n_punti, fract, n_fract, n_vert] = fracttomatr('tetgen_e_frat-
ture/fract.pol');


for i=1:n_fract
    [n,d]=trovapiano(n_vert(i),fract(i,:),punti);
    posizionenodi = posnodi(n,d);
    [tetra_inters,Punti_pol,Lati_pol,Facce_pol,Poliedri,Punti_frat-
tura,Sotto_frattura]=tetra(posizionenodi,n,d,i);
    [vicini,punti_vicini,lati_vicini,facce_vicini] = Tetra_vicini(tetra_inters);

end
```

filetomatr.m

```
function [res,nr] = filetomatr(file)

f = fopen(file,'r');
n = fscanf(f,'%d',2)'; %salvo il primo valore, che indica il numero di righe
nr = n(1); nc = n(2);
fgetl(f); %tolgo la prima riga inutile
res = fscanf(f,'%f',[nc+1,nr])';
res = res(:,2:nc+1);%salvo la matrice con le info necessarie
end
```

fracttomatr.m

```
%read fract file

function [res,n,res2,n2,nc] = fracttomatr(file)

f = fopen(file,'r');
n = fscanf(f,'%d',1)'; %salvo il primo valore, che indica il numero di righe
fgetl(f); %tolgo la prima riga inutile
res = fscanf(f,'%f',[4,n])'; res=res(:,2:4); %salvo la matrice con le info ne-
cessarie
n2 = fscanf(f,'%d',1)'; %salvo il primo valore, che indica il numero di righe
fgetl(f); %tolgo la prima riga inutile
mat=fscanf(f,'%f',[6,n2])';
nc = mat(:,2)';  %salvo il primo valore, che indica il numero di righe
res2 = mat(:,3:6); %salvo la matrice con le info necessari
```

```
end
```

facetomatr.m

```
%funzione per leggere il file face

function [res,n] = facetomatr(file)

f = fopen(file,'r');
n = fscanf(f,'%d',1)'; %salvo il primo valore, che indica il numero di righe
fgetl(f); %tolgo la prima riga inutile
res = fscanf(f,'%f',[7,n])'; %salvo la matrice con le info necessarie
res = res(:,2:7);
end
```

edgetomatr.m

```
%funzione per leggere il file edge

function [res,n] = edgetomatr(file)

f = fopen(file,'r');
n = fscanf(f,'%d',1)'; %salvo il primo valore, che indica il numero di righe
fgetl(f); %tolgo la prima riga inutile
res = fscanf(f,'%f',[5,n])'; %salvo la matrice con le info necessarie
res = res(:,2:5);
end
```

trovapiano.m

```
function [ n,d ] = trovapiano( n_punti,indice_punti,punti )

n=[0 0 0]; %sarà il vettore medio normale al piano
d=0;       %parametrizzando il piano come ax+by+cz+d=0


for i=1:n_punti

    for j=i+1:n_punti

        for k=j+1:n_punti

            V1=punti(indice_punti(j),:)-punti(indice_punti(i),:);
            V2=punti(indice_punti(k),:)-punti(indice_punti(i),:);

            %V1 e V2 sono i vettori formati dai vertici della frattura che
            %servono ad individuare il piano

            n=cross(V1,V2)+n;

        end
    end
end
```

```
n=n/nchoosek(n_punti,2); %si fa una media di tutti i versoli normali considerati

for i=1:n_punti
    d=-n(1)*punti(indice_punti(i),1)-n(2)*punti(indice_punti(i),2)-
n(3)*punti(indice_punti(i),3)+0;
    %si ottiene d sostituendo i vertici della frattura nell'equazione del
    %piano
end

d=d/n_punti; %si fa una media dei parametri d ricavati

end
```

posnodi.m

```
% posizione punti rispetto alla frattura

% valore 1 se il punto sta sopra al piano
% valore -1 se il punto sta sotto al piano
% valore 0 se il punto giace sul piano

function [posizionenodi] = posnodi(n,d)
global node n_node

posizionenodi=zeros(1,n_node);

%per ogni nodo si usa la disequazione data dall'equazione cartesiana del
%piano ax+by+cz+d>0 sostituendo le coordinate (x,y,z) di ogni nodo

for i=1:n_node

    r = node(i,:)*n'+d;
    if r > eps
        posizionenodi(i) = 1;
    elseif r < -eps
        posizionenodi(i) = -1;
    end
end

end
```

tetra.m

```
function [ tetra_inters,Punti_pol,Lati_pol,Facce_pol,Poliedri,Punti_frat-
tura,Sotto_frattura] = tetra( posizionenodi,n,d,i_fract )

global n_ele n_punti fract punti

tetra_inters=zeros(1,n_ele);
Punti_pol=[];
```

```
Lati_pol=[];
Facce_pol=[];
Poliedri=[];
Punti_frattura=[];
Sotto_frattura=[];
coda=[];

Fra_punti=zeros(n_punti,3);

for i=1:n_punti(i_fract)

    %salvo i punti della frattura poiché mi serviranno nel valutare
    %l'intersezione tra essa e i vari tetraedri
    Fra_punti(i,:)=punti(fract(i_fract,i),:);

end

for i=1:n_ele
    [punti_intersezione,pos_int,pos_nodi] = Intersezione_punti(posizio-
nenodi,n,d,i); %calcola l'interseione tra piano della frattura e tetraedro
    [tetra_inters(i),Punti_frattura,Sotto_frattura(1,:)]=Poliedri_intersezione(
punti_intersezione,Fra_punti,posizionenodi,i,Punti_frattura,n); %verifico se vi
è intersezione tra tetraedro e frattura

    if tetra_inters(i)==1 %appena trovo un intersezione si inizierà il procedi-
mento di coda

        [Poliedri,Facce_pol,Lati_pol,Punti_pol]=Sottopoliedri(punti_interse-
zione,pos_int,i,pos_nodi,Poliedri,Facce_pol,Lati_pol,Punti_pol); %creo i sotto-
poliedri

        coda=Coda(coda,tetra_inters,i); %aggiorno la coda
        lun=length(coda);
        i_coda=0;

        while i_coda~=lun %ciclo fino ad aver finito la coda

            i_coda=i_coda+1;
            [punti_intersezione,pos_int,pos_nodi] = Intersezione_punti(posizio-
nenodi,n,d,coda(i_coda)); %calcola l'interseione tra piano della frattura e te-
traedro inserito in coda
            LL=size(Sotto_frattura,1);
            [tetra_inters(coda(i_coda)),Punti_frattura,Sotto_frat-
tura(LL+1,:)]=Poliedri_intersezione( punti_intersezione,Fra_punti,posizio-
nenodi,coda(i_coda),Punti_frattura,n);%verifico che l'intersezione sussiste

            if tetra_inters(coda(i_coda))==1 %se vi è intersezione con il
triangolo della coda

                [Poliedri,Facce_pol,Lati_pol,Punti_pol]=Sottopoliedri(punti_in-
tersezione,pos_int,i,pos_nodi,Poliedri,Facce_pol,Lati_pol,Punti_pol); %creo i
sottopoliedri
                coda=Coda(coda,tetra_inters,coda(i_coda)); %aggiorna la coda
                lun=length(coda); %aggiorno la lunghezza della coda
            end

        end

    end
```

```
    if ~isempty(coda)  %se è iniziato il meccanismo di coda esci dal ciclo gene-
rale
        break
    end


end
```

Coda.m

```
function coda  = Coda(coda,tetra_inters,i)

global neigh

for j=1:4
    flag=0;
    x=length(coda);
     if neigh(i,j) ~= -1
         if tetra_inters(neigh(i,j)) == 0 %il tetraedro non è stato già control-
lato

                  for z=x:-1:1
                        if neigh(i,j)==coda(z) %il tetraedro appartiene già alla
coda
                             flag=1;
                             break
                        end
                  end
                  if flag==0
                        coda(x+1)=neigh(i,j); %aggiungo il vicino alla coda
                  end
         end
     end
end
```

intersezione_punti.m

```
function [punti_intersezione,pos_int,pos_nodi ] = Intersezione_punti( posizio-
nenodi,n,d,i_tetra )

global ele node

punti_intersezione=[];
pos_int=[];
pos=0;
pos_nodi=zeros(1,4);
%individuo i lati con gli indici 1...6 rispettivamente per i lati
%formati dai nodi 1-2,1-3,1-4,2-3,2-4,3-4.

%individuo le facce con gli indici 1...4 rispettivamente per le facce
%che contengono i lati (4,5,6),(2,3,6),(1,3,5),(1,2,4), ovvero che non
```

```
%contengono il vertice nella rispettiva posizione: ad esmpio la faccia 1 non
contiene in vertice 1.
for j=1:4
   pos_nodi(j)=posizionenodi(ele(i_tetra,j)); %salvo la posizione dei nodi del
relativo tetraedro
end


for j=1:3
        for k=j+1:4
            pos=pos+1;
            if posizionenodi(ele(i_tetra,j))*posizionenodi(ele(i_tetra,k))==-1
%in tal caso vi è sicuramente intersezione tra il piano su cui giace la traccia
e il lato considerato

                P1=node(ele(i_tetra,j),:);
                P2=node(ele(i_tetra,k),:);
                x=size((punti_intersezione),1);
                y=length(pos_int);
                t=-(P1*n'+d)/((P2-P1)*n'); %trovo l'ascissa curvilinea dell'in-
tersezione parametrizzando i lato P1-P2 in funzione del parametro t
                punti_intersezione(x+1,:)=P1+t*(P2-P1); %dall'ascissa curvilinea
ricavo le coordinate cartesiane del punto d'intersezione
                pos_int(y+1)=pos; %salvo la posizione di tale punto come indi-
cato prima
            end
        end
end

end
```

Poliedri_intersezione.m

```
function [ int,punti_fratt,sotto_fratt ] = Poliedri_intersezione(
Facc_punti,Fra_punti,posnodi,i_tetra,punti_fratt,n)

%Facc_punti sono le coordinate dei punti di intersezione
%Fra_punti sono le coordinate dei punti della frattura

sotto_fratt=zeros(6,1);
int=0;
Punti_int_facc=[];
Punti_int_fra=[];
Punti_fratt_taglio=[];
global ele node n_punti

%sommo i valori delle posizioni dei nodi in pos_tot
pos_tot=posnodi(ele(i_tetra,1))+posnodi(ele(i_tetra,2))+posnodi(ele(i_te-
tra,3))+posnodi(ele(i_tetra,4));

% nei primi due casi (pos_tot=4 o pos_tot=-4) i punti sono tutti sopra o sotto
% il teraedro, nel terzo e nel quarto (pos_tot=3 o pos_tot=-3) vi sarà un ver-
tice
% che non apperterrà al piano e gli altri tre sopra o sotto, dunque in questi
% casi non può esservi intersezione
```

```
if (pos_tot==4 || pos_tot==-4 || pos_tot==3 || pos_tot==-3)
    int=-1;
    return
end

flag=1;

if size(Facc_punti,1)==0 %ovvero non vi sono intersezioni classiche

    if (pos_tot==1 || pos_tot==-1) %ciò vuol dire che tre vertice staranno sul
piano, e devo considerare il caso dell'intersezione parallela

    flag=0;

    else

    int=-1;

    return

    end
end

for i=1:4 % salvo i nodi che giacciono sul piano della frattura in Fac_punti
    if posnodi(ele(i_tetra,i))==0

        L=size(Facc_punti,1);
        Facc_punti(L+1,:)=node(ele(i_tetra,i));

    end
end

% si definiscono tre versore che appartengono al piano considerato per
% valutare l'appartenenza del punto alla faccia

E1=(Fra_punti(2,:)-Fra_punti(1,:))/norm(Fra_punti(2,:)-Fra_punti(1,:));
E2=cross(E1,n)/norm(n);
E3=(E1+E2)/norm(E1+E2);


L=size(Facc_punti,1);

for i=1:L
    s1=0;
    s2=0;
    s3=0;

    for j=1:n_punti-1
        pos_int=punti_appartiene_seg-
mento(Facc_punti(i,:),Fra_punti(j,:),Fra_punti(j+1,:)); %controllo se il punto
considerato appartenga ad un lato

        if pos_int==1 %in questo caso appartiene
            s1=10;
            s2=10;
            s3=10;
            break
        else
```

```
            k1=interno(Facc_punti(i,:),Fra_punti(j,:),Fra_punti(j+1,:),E1);
            k2=interno(Facc_punti(i,:),Fra_punti(j,:),Fra_punti(j+1,:),E2);
            k3=interno(Facc_punti(i,:),Fra_punti(j,:),Fra_punti(j+1,:),E3);
            s1=s1+k1;
            s2=s2+k2;
            s3=s3+k3;
        end
    end

   if s1~=10 %il punto appartiene a qualche segmento
        pos_int=punti_appartiene_segmento(Facc_punti(i,:),Fra_punti(n_punti,:),Fra_punti(1,:));

        if pos_int==1
            s1=10;
            s2=10;
            s3=10;
        else

        k1=interno(Facc_punti(i,:),Fra_punti(n_punti,:),Fra_punti(1,:),E1);
        k2=interno(Facc_punti(i,:),Fra_punti(n_punti,:),Fra_punti(1,:),E2);
        k3=interno(Facc_punti(i,:),Fra_punti(n_punti,:),Fra_punti(1,:),E3);
        s1=s1+k1;
        s2=s2+k2;
        s3=s3+k3;

        end
   end

   if (s1==s2 && s2==s3)
        if ((s1==1 || s1==10) && flag~=0)%il punto è interno
            int=1;
            l=size(Punti_int_facc,1);
            Punti_int_facc(l+1,:)=Facc_punti(i,:);
        elseif (flag==0 && (s1==1 || s1==10)) %siamo nel caso in cui la frattura è parallela ad una faccia del tetraedro
            int=1.5;
            l=size(Punti_int_facc,1);
            Punti_int_facc(l+1,:)=Facc_punti(i,:);
        end
   end
end

if size(Punti_int_facc,1)==L
    if flag==0 %nel caso di intersezione parallela se tutti i punti della faccia appartengono alla frattura non vi è intersezione
        int=0;
    end
else %se tutti i punti della faccia appartengono alla frattura allora sicuramente i punti della frattura non possono appartenere alla faccia

    %il modo di procedere è anologo a quanto fatto sopra

    E1=(Facc_punti(2,:)-Facc_punti(1,:))/norm(Facc_punti(2,:)-Facc_punti(1,:));
    E2=cross(E1,n)/norm(n);
```

```
    E3=(E1+E2)/norm(E1+E2);

    for i=1:n_punti
        s1=0;
        s2=0;
        s3=0;

        for j=1:L-1
            pos_int=punti_appartiene_seg-
mento(Fra_punti(i,:),Facc_punti(j,:),Facc_punti(j+1,:));

            if pos_int==1
                s1=10;
                s2=10;
                s3=10;
                break
            else
                k1=inters_in-
terno(Fra_punti(i,:),Facc_punti(j,:),Facc_punti(j+1,:),E1);
                k2=inters_in-
terno(Fra_punti(i,:),Facc_punti(j,:),Facc_punti(j+1,:),E2);
                k3=inters_in-
terno(Fra_punti(i,:),Facc_punti(j,:),Facc_punti(j+1,:),E3);
                s1=s1+k1;
                s2=s2+k2;
                s3=s3+k3;
            end
        end
        if s1~=10

            pos_int=punti_appartiene_seg-
mento(Fra_punti(i,:),Facc_punti(L,:),Facc_punti(1,:));
            if pos_int==1
                s1=10;
                s2=10;
                s3=10;
            else
                k1=inters_in-
terno(Fra_punti(i,:),Facc_punti(L,:),Facc_punti(1,:),E1);
                k2=inters_in-
terno(Fra_punti(i,:),Facc_punti(L,:),Facc_punti(1,:),E2);
                k3=inters_in-
terno(Fra_punti(i,:),Facc_punti(L,:),Facc_punti(1,:),E3);
                s1=s1+k1;
                s2=s2+k2;
                s3=s3+k3;
            end
        end

        if (s1==s2 && s2==s3)
            if ((s1==1 || s1==10) && flag~=0)%il punto è interno
                int=1;
                l=size(Punti_int_fra,1);
                Punti_int_fra(l+1,:)=Fra_punti(i,:);
            elseif (flag==0 && (s1==1 || s1==10))
                int=1.5;
                l=size(Punti_int_fra,1);
                Punti_int_fra(l+1,:)=Fra_punti(i,:);
            end
        end
```

```
    end
end

%adesso bidogna calcolare i punti di intersezione tra i lati della faccia e
%i lati della frattura ciclando su tutti i lati

if (size(Punti_int_facc,1)~=L && size(Punti_int_fra,1)~=n_punti) %escludo il
caso in cui tutti punti della faccia sono contenuti nella frattura e viceversa
    for i=1:n_punti-1
        for j=1:L-1
            P=intersezioni_seg-
menti(Fra_punti(i,:),Fra_punti(i+1,:),Facc_punti(j,:),Facc_punti(j+1,:));
            if P~=0
               l=size(Punti_fratt_taglio,1);
               Punti_fratt_taglio(l+1,:)=P;
            end
        end
    end
    P=intersezioni_seg-
menti(Fra_punti(i,:),Fra_punti(i+1,:),Facc_punti(L,:),Facc_punti(1,:));
    if P~=0
    l=size(Punti_fratt_taglio,1);
    Punti_fratt_taglio(l+1,:)=P;
    end

    for j=1:L-1
        P=intersezioni_seg-
menti(Fra_punti(n_punti,:),Fra_punti(1,:),Facc_punti(j,:),Facc_punti(j+1,:));
        if P~=0
        l=size(Punti_fratt_taglio,1);
        Punti_fratt_taglio(l+1,:)=P;
        end
    end

    P=intersezioni_seg-
menti(Fra_punti(n_punti,:),Fra_punti(1,:),Facc_punti(L,:),Facc_punti(1,:));
    if P~=0
    l=size(Punti_fratt_taglio,1);
    Punti_fratt_taglio(l+1,:)=P;
    end
end

%dopo aver salvato i punti di intersezione nella matrice Punti_fratt_taglio
%vado a creare la sottofrattura

if int>=1 %se avviene l'intersezione

    %in sotto_fratt il primo indice sarà quello del tetraedro con cui
    %avviene l'intersezione, il secondo è un -1 di default e gli altri
    %elementi saranno degli indici che individuano i punti di punti_fratt
    %che vanno a formare la sottofrattura

    sotto_fratt(1:2)=[i_tetra,-1];

    %con ordina_frattura salvo i punti in una matrice e li inddividuo con
    %degli indici

    [indici_1,punti_fratt]=ordina_frattura(punti_fratt,Punti_int_facc);
    [indici_2,punti_fratt]=ordina_frattura(punti_fratt,Punti_int_fra);
    [indici_3,punti_fratt]=ordina_frattura(punti_fratt,Punti_fratt_taglio);
```

```
    L1=length(indici_1);
    L2=length(indici_2);
    L3=length(indici_3);
    for j=1:L1
        flag=1;
        for i=3:6
            if sotto_fratt(i)==0
                break
            elseif sotto_fratt(i)==indici_1(j) %controllo che il punto di inte-
resse non appartenga già alla sottofrattura
                flag=0;
            end
        end
        if flag==1
            sotto_fratt(i)=indici_1(j); %nel caso in cui il punto non appartiene
lo vado a salvare
        end
    end

    for j=1:L2
        flag=1;
        for i=3:6
            if sotto_fratt(i)==0
                break
            elseif sotto_fratt(i)==indici_2(j)  %controllo che il punto di inte-
resse non appartenga già alla sottofrattura
                flag=0;
            end
        end
        if flag==1
            sotto_fratt(i)=indici_2(j); %nel caso in cui il punto non appartiene
lo vado a salvare
        end
    end

    for j=1:L3
        flag=1;
        for i=3:6
            if sotto_fratt(i)==0
                break
            elseif sotto_fratt(i)==indici_3(j)  %controllo che il punto di inte-
resse non appartenga già alla sottofrattura
                flag=0;
            end
        end
        if flag==1
            sotto_fratt(i)=indici_3(j); %nel caso in cui il punto non appartiene
lo vado a salvare
        end
    end

end
```

inters_interno.m

```
function [int] = inters_interno(P,A,B,tanP)
```

```
tanAB=(B-A);
int=0;
if norm(cross(tanAB,tanP))>eps %escludo i vettori paralleli

    %voglio trovare l'equazione cartesiana della retta AB e della retta che
    %passa per P con direzione tanP

    if tanP(1)==0
        if tanP(2)==0
            A_mat=[1 0 0; 0 1 0];
            b=[P(1) P(2)];
        elseif tanP(3)==0
            A_mat=[1 0 0; 0 0 1];
            b=[P(1) P(3)];
        else
            A_mat=[1 0 0;0 1/(tanP(2)) -1/(tanP(3))];
            b=[P(1) P(2)/tanP(2)-P(3)/tanP(3)];
        end
    elseif tanP(2)==0
        if tanP(3)==0
            A_mat=[0 1 0; 0 0 1];
            b=[P(2) P(3)];
        else
            A_mat=[0 1 0;1/(tanP(1)) 0 -1/(tanP(3))];
            b=[P(2) P(1)/tanP(1)-P(3)/tanP(3)];
        end
    elseif tanP(3)==0
        A_mat=[0 0 1; 1/(tanP(1)) -1/(tanP(2)) 0];
        b=[P(3) P(1)/tanP(1)-P(2)/tanP(2)];
    else
        A_mat=[1/(tanP(1)) -1/(tanP(2)) 0; 1/(tanP(1)) 0 -1/(tanP(3))];
        b=[P(1)/tanP(1)-P(2)/tanP(2) P(1)/tanP(1)-P(3)/tanP(3)];
    end

    if tanAB(1)==0
        if tanAB(2)==0
            A_mat(3:4,:)=[1 0 0; 0 1 0];
            b(3:4)=[A(1) A(2)];
        elseif tanAB(3)==0
            A_mat(3:4,:)=[1 0 0; 0 0 1];
            b(3:4)=[A(1) A(3)];
        else
            A_mat(3:4,:)=[1 0 0;0 1/(tanAB(2)) -1/(tanAB(3))];
            b(3:4)=[A(1) A(2)/tanAB(2)-A(3)/tanAB(3)];
        end
    elseif tanAB(2)==0
        if tanAB(3)==0
            A_mat(3:4,:)=[0 1 0; 0 0 1];
            b(3:4)=[A(2) A(3)];
        else
            A_mat(3:4,:)=[0 1 0;1/(tanAB(1)) 0 -1/(tanAB(3))];
            b(3:4)=[A(2) A(1)/tanAB(1)-A(3)/tanAB(3)];
        end
    elseif tanAB(3)==0
        A_mat(3:4,:)=[0 0 1; 1/(tanAB(1)) -1/(tanAB(2)) 0];
        b(3:4)=[A(3) A(1)/tanAB(1)-A(2)/tanAB(2)];
    else
        A_mat(3:4,:)=[1/(tanAB(1)) -1/(tanAB(2)) 0; 1/(tanAB(1)) 0 -1/(ta-
nAB(3))];
        b(3:4)=[A(1)/tanAB(1)-A(2)/tanAB(2) A(1)/tanAB(1)-A(3)/tanAB(3)];
    end
```

```
    %si imposta un sistema di quattro equazioni in tre incognite, il quale è ben
definito, dove ogni
    %equazione è quella di un piano

    PP=A_mat\b'; %trovo il punto di intersezione tra le due rette, le quali ho
espresso in forma parametrica

    %t sarà l'ascissa curvilinea del punto di intersezione rispetto alla
    %semiretta

    if tanP(1)~=0
        t=(PP(1)-P(1))/tanP(1);
    elseif tanP(2)~=0
        t=(PP(2)-P(2))/tanP(2);
    elseif tanP(3)~=0
        t=(PP(3)-P(3))/tanP(3);
    end

    %tAB sarà l'ascissa curvilinea del punto di intersezione rispetto al lato
    %AB

    if tanAB(1)~=0
        tAB=(PP(1)-A(1))/tanAB(1);
    elseif tanAB(2)~=0
        tAB=(PP(2)-A(2))/tanAB(2);
    elseif tanAB(3)~=0
        tAB=(PP(3)-A(3))/tanAB(3);
    end

    if (t>0 && tAB>=0 && tAB<=1) %t deve essere maggiore di zero perché sto con-
siderando una semiretta. tAB compreso tra 0 ed 1 poiché il punto deve apparte-
nere al lato
        if (norm(PP-A)<eps || norm(PP-B)<eps) %è il caso in cui il punto di in-
tersezione è un vertice del lato AB
            int=0.5;
        else
            int=1;
        end
    end

end

end
```

punti_appartiene_segmento.m

```
function [ int ] = punti_appartiene_segmento( P,A,B )

%per valutare l'appartenenza lavoro sulle tre coordinate x,y,z e trovo
%delle ascissa curvilinee per tali coordinate, facendo attenzione ai casi
%paralleli dove si imposta un valore di default inf. se t1,t2 e t3
%coincidono e il valore è compreso tra 0 ed 1 allora il punto appartiene al
%segmento.

int=0;
```

```
if A(1)==B(1)
    if P(1)==A(1)
        t1=inf;
    else
        return
    end
else
    t1=(P(1)-A(1))/(B(1)-A(1));
end

if A(2)==B(2)
    if P(2)==A(2)
        t2=inf;
    else
        return
    end
else
    t2=(P(2)-A(2))/(B(2)-A(2));
end

if A(3)==B(3)
    if P(3)==A(3)
        t3=inf;
    else
        return
    end
else
    t3=(P(3)-A(3))/(B(3)-A(3));
end

if t1==inf
    if t2==inf
        if (t3>=0 && t3<=1)
            int=1;
        end

    elseif t3==inf
        if (t2>=0 && t2<=1)
            int=1;
        end
    else

        if ((t2==t3)&&(t2>=0 && t2<=1))
            int=1;
        end
    end
elseif t2==inf
    if t3==inf
        if (t1>=0 && t1<=1)
            int=1;
        end
    else

        if ((t1==t3) && (t1>=0 && t1<=1))
            int=1;
        end
    end
elseif t3==inf
    if ((t1==t2) && (t1>=0 && t1<=1))
        int=1;
    end
```

```
else
    if ((t1==t2) && (t2==t3) && (t1>=0 && t1<=1))
        int=1;
    end
end

end
```

intersezione_segmenti.m

```
function [ PP ] = intersezione_segmenti( P1,P2,Z1,Z2)

%analogo a inters_interno, con la differenza che qui considera due segmenti
%e lì un segmento ed una semiretta

Tan1=(P2-P1);
Tan2=(Z2-Z1);
int=0;

if norm(cross(Tan1,Tan2))>eps %escludo i vettori paralleli
    if tan1(1)==0
       if tan1(2)==0
           A_mat=[1 0 0; 0 1 0];
           b=[P1(1) P1(2)];
        elseif Tan1(3)==0
            A_mat=[1 0 0; 0 0 1];
            b=[P1(1) P1(3)];
       else
            A_mat=[1 0 0;0 1/(Tan1(2)) -1/(Tan1(3))];
            b=[P1(1) P1(2)/Tan1(2)-P1(3)/Tan1(3)];
       end
    elseif Tan1(2)==0
        if Tan1(3)==0
            A_mat=[0 1 0; 0 0 1];
            b=[P1(2) P1(3)];
        else
            A_mat=[0 1 0;1/(Tan1(1)) 0 -1/(Tan1(3))];
            b=[P1(2) P1(1)/Tan1(1)-P1(3)/Tan1(3)];
        end
    elseif Tan1(3)==0
        A_mat=[0 0 1; 1/(Tan1(1)) -1/(Tan1(2)) 0];
        b=[P1(3) P1(1)/Tan1(1)-P1(2)/Tan1(2)];
    else
        A_mat=[1/(Tan1(1)) -1/(Tan1(2)) 0; 1/(Tan1(1)) 0 -1/(Tan1(3))];
        b=[P1(1)/Tan1(1)-P1(2)/Tan1(2) P1(1)/Tan1(1)-P1(3)/Tan1(3)];
    end

    if Tan2(1)==0
        if Tan2(2)==0
            A_mat(3:4,:)=[1 0 0; 0 1 0];
            b(3:4)=[Z1(1) Z1(2)];
        elseif Tan2(3)==0
            A_mat(3:4,:)=[1 0 0; 0 0 1];
            b(3:4)=[Z1(1) Z1(3)];
        else
            A_mat(3:4,:)=[1 0 0;0 1/(Tan2(2)) -1/(Tan2(3))];
```

```
            b(3:4)=[Z1(1) Z1(2)/Tan2(2)-Z1(3)/Tan2(3)];
        end
    elseif Tan2(2)==0
        if Tan2(3)==0
            A_mat(3:4,:)=[0 1 0; 0 0 1];
            b(3:4)=[Z1(2) Z1(3)];
        else
            A_mat(3:4,:)=[0 1 0;1/(Tan2(1)) 0 -1/(Tan2(3))];
            b(3:4)=[Z1(2) Z1(1)/Tan2(1)-Z1(3)/Tan2(3)];
        end
    elseif Tan2(3)==0
        A_mat(3:4,:)=[0 0 1; 1/(Tan2(1)) -1/(Tan2(2)) 0];
        b(3:4)=[Z1(3) Z1(1)/Tan2(1)-Z1(2)/Tan2(2)];
    else
        A_mat(3:4,:)=[1/(Tan2(1)) -1/(Tan2(2)) 0; 1/(Tan2(1)) 0 -1/(Tan2(3))];
        b(3:4)=[Z1(1)/Tan2(1)-Z1(2)/Tan2(2) Z1(1)/Tan2(1)-Z1(3)/Tan2(3)];
    end

    %si è difenito il sistema di quattro equzioni in tre incognite

    PP=A_mat\b'; %trovo il punto di intersezione tra le due rette, le quali ho
espresso in forma parametrica

    %trovo t1 e t2 che sono le ascisse curvilinee del punto PP rispetto ai due
    %lati considerati

    if tanP(1)~=0
        t1=(PP(1)-P1(1))/Tan1(1);
    elseif tanP(2)~=0
        t1=(PP(2)-P1(2))/Tan1(2);
    elseif tanP(3)~=0
        t1=(PP(3)-P1(3))/Tan1(3);
    end

    if tanAB(1)~=0
        t2=(PP(1)-Z1(1))/Tan2(1);
    elseif tanAB(2)~=0
        t2=(PP(2)-Z1(2))/Tan2(2);
    elseif tanAB(3)~=0
        t2=(PP(3)-Z1(3))/Tan2(3);
    end

    if (t1>=0 && t1<=1) && (t2>=0 && t2<=1) %l'intersezione avviene effettiva-
mente, poiché il punto appartiene ad i lati considerati
        int=1;
    end
end

if int~=1 %non avviene intersezione
    PP=0; %valore di default di PP in tale caso
end

end
```

ordina_frattura.m

```
function [ indice_punti,punti_fratt ] = ordina_frattura( punti_fratt,punti_provv
)

%punti_provv sono i punti provvisori che andrò ad inserire in quelli
%definitivi punti_fratt

lun1=size(punti_fratt,1);
lun2=size(punti_provv,1);
indice_punti=[];

for i=1:lun2
    flag=0;
    for j=1:lun1
        if punti_provv(i,:) == punti_fratt(j,:) %se il punto appartiene già a
punti_fratt, salvo l'indice del punto
            lun_indice=length(indice_punti);
            indice_punti(lun_indice+1)=j;
            flag=1;
        end
    end
    if flag==0 %nel caso in cui il punto non appartiene invece a punti_fratt lo
inserisco e salvo il nuovo indice L+1

            L=size(punti_fratt,1);
            punti_fratt(L+1,:)=punti_provv(i,:);
            lun_indice=length(indice_punti);
            indice_punti(lun_indice+1)=L+1;
    end
end


end
```

Sottopoliedri.m

```
function [ Poliedri,Facce_pol,Lati_pol,Punti_pol ] = Sottopoliedri( punti_inter-
sezione,pos_int,i_tetra,pos_nodi,Poliedri,Facce_pol,Lati_pol,Punti_pol )

[Punti_pol,indice_punti] = Ordina_punti( Punti_pol,punti_intersezione,i_tetra);
%ordino i punti di intersezione in Punti_pol e salvo gli indici
[pos_int_facc,facce_inters,n_inters] = posizioneinterna_facceinterse-
cate(pos_int); %ricavo la posixione interna dei punti di intersezione e quali
sono le facce del tetraedro intersecate

if n_inters == 4 %4 punti di intersezione
    [Poliedri,Facce_pol,Lati_pol ] = Crea_poliedri_4( in-
dice_punti,pos_int_facc,pos_int,Poliedri,Facce_pol,Lati_pol);
end

if n_inters == 3 %3 punti di intersezione
    [Poliedri,Facce_pol,Lati_pol] = Crea_poliedri_3(indice_punti,facce_in-
ters,pos_int_facc,Poliedri,Facce_pol,Lati_pol);
end

if n_inters == 2 %2 punti di intersezione
```

```
    [Poliedri,Facce_pol,Lati_pol] = Crea_poliedri_2(i_punti,pos_nodi,facce_in-
ters,pos_int,Poliedri,Facce_pol,Lati_pol);
end

if n_inters ==1 %1 punti di intersezione
    [Poliedri,Facce_pol,Lati_pol] = Crea_poliedri_1(i_punti,facce_in-
ters,pos_nodi,Poliedri,Facce_pol,Lati_pol);
end

end
```

ordina_punti.m

```
function [ punti_pol,indice_punti ] = Ordina_punti( punti_pol,punti_interse-
zione,i_tetra )

global node ele
lun1=size(punti_intersezione,1);
lun2=size(punti_pol,1);
indice_punti=[];

%inizialmente ordino i 4 vertici del tetraedro e ricavo dalle veriabili
%globali

for i=1:4
    flag=0;

    for j=1:lun2
        if node(ele(i_tetra,i),:) == punti_pol(j,:)
            lun_indice=length(indice_punti);
            indice_punti(lun_indice+1)=j;
            flag=1;
        end
    end
    if flag==0
            L=size(punti_pol,1);
            punti_pol(L+1,:)=node(ele(i_tetra,i),:);
            lun_indice=length(indice_punti);
            indice_punti(lun_indice+1)=L+1;
    end
end

%dopoché ordino i punti di intersezione

for i=1:lun1
    flag=0;
    for j=1:lun2
        if punti_intersezione(i,:) == punti_pol(j,:)
            lun_indice=length(indice_punti);
            indice_punti(lun_indice+1)=j;
            flag=1;
        end
    end
    if flag==0

            L=size(punti_pol,1);
            punti_pol(L+1,:)=punti_intersezione(i,:);
```

```
            lun_indice=length(indice_punti);
            indice_punti(lun_indice+1)=L+1;
    end
end
```

posizioneinterna_facceintersecate.m

```
function [ pos_int_facc,facce_inters,n_inters ] = posizioneinterna_facceinter-
secate( pos_int )

n_inters=length(pos_int);
facce_inters=zeros(1,4);

for j=1:n_inters %calcola con quali lati avvengono le intersezioni, ordinando i
lati
    if pos_int(j)==1
        pos_int_facc(:,j)=[-1 -1 1 1];
    elseif pos_int(j)==2
        pos_int_facc(:,j)=[-1 1 -1 3];
    elseif pos_int(j)==3
        pos_int_facc(:,j)=[-1 3 3 -1];
    elseif pos_int(j)==4
        pos_int_facc(:,j)=[1 -1 -1 2];
    elseif pos_int(j)==5
        pos_int_facc(:,j)=[3 -1 2 -1];
    else
        pos_int_facc(:,j)=[2 2 -1 -1];
    end
end

for j=1:n_inters %calcola con quali facce avviene l'intersezione
    if (pos_int(j)==4 || pos_int(j)==5 || pos_int(j)==6)
        facce_inters(1)=1;
    end
    if (pos_int(j)==2 || pos_int(j)==3 || pos_int(j)==6)
        facce_inters(2)=1;
    end
    if (pos_int(j)==1 || pos_int(j)==3 || pos_int(j)==5)
        facce_inters(3)=1;
    end
    if (pos_int(j)==1 || pos_int(j)==2 || pos_int(j)==4)
        facce_inters(4)=1;
    end
end
end
```

Crea_poliedri_1.m

```
function [ Poliedri,Facce_pol,Lati_pol ] = Crea_poliedri_1( i_punti,facce_in-
ters,pos_nodi,Poliedri,Facce_pol,Lati_pol )

%siamo nel caso in cui vi è un solo punto di intersezione e due vertici
```

```
%giacciono sulla frattura
x=0;
i_base=[];
pos_int_vert=zeros(2,4);
i_vert=zeros(1,2);
for i=1:4
    if pos_nodi(i)==0
        x=x+1;
        i_vert(x)=i; %salvo i vertici che giacciono sulla frattura
    end
end
for i=1:2 %salvo la posizione interna dei vertici con riferimento alla struttura
interna
    if i_vert(i)==1
        pos_int_vert(i,:)=[0 1 1 1];
    end
    if i_vert(i)==2
        pos_int_vert(i,:)=[1 0 2 2];
    end
    if i_vert(i)==3
        pos_int_vert(i,:)=[2 2 0 3];
    end
    if i_vert(i)==4
        pos_int_vert(i,:)=[3 3 3 0];
    end
end

%creo la faccia di taglio composta dai i due vertici sopra definiti ed il
%punto di intersezione classico

Lati=[i_punti(5) i_punti(i_vert(1));i_punti(i_vert(1))
i_punti(i_vert(2));i_punti(i_vert(2)) i_punti(5)];
[Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
[Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
i_taglio=i_facce;

%vi saranno due facce del tetraedro che non saranno intersecate,allora
%salvo tali facce poiché formeranno un sottopoliedro

if facce_inters(1)==0 %se la prima faccia non è intersecata
    Lati=[i_punti(2) i_punti(3);i_punti(3) i_punti(4);i_punti(4) i_punti(2)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
    x=length(i_base);
    i_base(x+1)=i_facce;
end
if facce_inters(2)==0 %se la seconda faccia non è intersecata
    Lati=[i_punti(1) i_punti(3);i_punti(3) i_punti(4);i_punti(4) i_punti(1)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
    x=length(i_base);
    i_base(x+1)=i_facce;
end
if facce_inters(3)==0 %se la terza faccia non è intersecata
    Lati=[i_punti(1) i_punti(2);i_punti(2) i_punti(4);i_punti(4) i_punti(1)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
```

```
    x=length(i_base);
    i_base(x+1)=i_facce;
end
if facce_inters(4)==0 %se la quarta faccia non è intersecata
    Lati=[i_punti(1) i_punti(2);i_punti(2) i_punti(3);i_punti(3) i_punti(1)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
    x=length(i_base);
    i_base(x+1)=i_facce;
end

%inoltre vi saranno due facce intersecate e devo andare a creare le
%sottofacce di tali

if facce_inters(1)==1
    x=length(i_facce);
    if pos_int_vert(1,1)==0 %il primo vertice non appartiene alla faccia consi-
derata

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(2),i_punti(3),i_punti
(4),i_punti(5),pos_int_vert(2,1),Lati_pol,Facce_pol);
    else

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(2),i_punti(3),i_punti
(4),i_punti(5),pos_int_vert(1,1),Lati_pol,Facce_pol);
    end
end

if facce_inters(2)==1
    x=length(i_facce);
    if pos_int_vert(1,2)==0 %il primo vertice non appartiene alla faccia consi-
derata

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(3),i_punti
(4),i_punti(5),pos_int_vert(2,2),Lati_pol,Facce_pol);
    else

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(3),i_punti
(4),i_punti(5),pos_int_vert(1,2),Lati_pol,Facce_pol);
    end
end

if facce_inters(3)==1
    x=length(i_facce);
    if pos_int_vert(1,3)==0 %il primo vertice non appartiene alla faccia consi-
derata

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(2),i_punti
(4),i_punti(5),pos_int_vert(2,3),Lati_pol,Facce_pol);
    else

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(2),i_punti
(4),i_punti(5),pos_int_vert(1,3),Lati_pol,Facce_pol);
    end
end

if facce_inters(4)==1
    x=length(i_facce);
    if pos_int_vert(1,4)==0 %il primo vertice non appartiene alla faccia consi-
derata
```

```
[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(2),i_punti
(3),i_punti(5),pos_int_vert(2,4),Lati_pol,Facce_pol);
    else

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(2),i_punti
(3),i_punti(5),pos_int_vert(1,4),Lati_pol,Facce_pol);
    end
end

flag1=0;
flag2=0;

%il primo elemento di Facce_pol delle facce create da Crea_facce_2 è
%formato dal lato costituito da due nodi. Tale lato sarà condiviso con una
%delle facce non intersecate. Inoltre si ha che se il lato è in
%condivisione con i_facce(1) non può esserlo con i_facce(2) e viceversa.
%Allora salvo in dei flag valutando con quale faccia vi è la condivisione.

%i_base(1) condividerà un lato con una tra i_facce(1) e i_facce(2) e
%condividerà un altro lato con una tra i_facce(3) e i_facce(4). Allora
%controllo con quali di tali facce il lato è condiviso.
for j=1:3
    if Facce_pol(i_base(1),j)==Facce_pol(i_facce(1),1)
        flag1=1;
    end
    if Facce_pol(i_base(1),j)==Facce_pol(i_facce(3),1)
        flag2=1;
    end
end

y=size(Poliedri,1);

if (flag1==1 && flag2==1) %i_base(1) forma il sottopoliedro con i_facce(1) e
i_facce(3)
    Poliedri(y+1)=[i_taglio,i_base(1),i_facce(1),i_facce(3),-1];
    Poliedri(y+2)=[i_taglio,i_base(2),i_facce(2),i_facce(4),-1];
end
if (flag1==1 && flag2==0) %i_base(1) forma il sottopoliedro con i_facce(1) e
i_facce(4)
    Poliedri(y+1)=[i_taglio,i_base(1),i_facce(1),i_facce(4),-1];
    Poliedri(y+2)=[i_taglio,i_base(2),i_facce(2),i_facce(3),-1];
end
if (flag1==0 && flag2==1) %i_base(1) forma il sottopoliedro con i_facce(2) e
i_facce(3)
    Poliedri(y+1)=[i_taglio,i_base(1),i_facce(2),i_facce(3),-1];
    Poliedri(y+2)=[i_taglio,i_base(2),i_facce(1),i_facce(4),-1];
end
if (flag1==0 && flag2==0) %i_base(1) forma il sottopoliedro con i_facce(2) e
i_facce(4)
    Poliedri(y+1)=[i_taglio,i_base(1),i_facce(2),i_facce(4),-1];
    Poliedri(y+2)=[i_taglio,i_base(2),i_facce(1),i_facce(3),-1];
end
end
```

Crea_poliedri_2.m

```
function [ Poliedri,Facce_pol,Lati_pol ] = Crea_poliedri_2(
i_punti,pos_nodi,facce_inters,pos_int,Poliedri,Facce_pol,Lati_pol )

%vi sono due punti di intersezione ed un vertice appartiene alla frattura

%inizialmente salvo il vertice in questione e la relativa posizione interna

for i=1:4
    if pos_nodi(i)==0
        i_vert=i;
        break
    end
end

if i_vert==1
    pos_int_vert=[0 1 1 1];
end
if i_vert==2
    pos_int_vert=[1 0 2 2];
end
if i_vert==3
    pos_int_vert=[2 2 0 3];
end
if i_vert==4
    pos_int_vert=[3 3 3 0];
end


facce_inters(i_vert)=2; %do un valore diverso alla faccia intersecata opoosta al
vertice poiché avrà un intersezione diversa rispetto le altre facce

%adesso si va a formare i_taglio, la faccia di taglio. costituita da un
%vertice e due punti di intersezione

Lati=[i_punti(5) i_punti(6);i_punti(6) i_punti(i_vert);i_punti(i_vert)
i_punti(5)];
[Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
[Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
i_taglio=i_facce;

%creo le sottofacce relative alla faccia opposta al vertice che giace sulla
%frattura

if i_vert==1
    x=length(i_facce);

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
(4),i_punti(5),i_punti(6),pos_int(1,1),pos_int(1,2),Lati_pol,Facce_pol);
end
if i_vert==2
    x=length(i_facce);

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(5),i_punti(6),pos_int(2,1),pos_int(2,2),Lati_pol,Facce_pol);
end
if i_vert==3
```

```
    x=length(i_facce);

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(5),i_punti(6),pos_int(3,1),pos_int(3,2),Lati_pol,Facce_pol);
end
if i_vert==4
    x=length(i_facce);

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(3),i_punti(5),i_punti(6),pos_int(4,1),pos_int(4,2),Lati_pol,Facce_pol);
end

%creo le altre sottofacce

if facce_inters(1)==1
    x=length(i_facce);
    if pos_int(1,1)==-1 %il primo punto di intersezione non appartiene alla fac-
cia considerata

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(2),i_punti(3),i_punti
(4),i_punti(6),pos_int_vert(1),Lati_pol,Facce_pol);
    else

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(2),i_punti(3),i_punti
(4),i_punti(5),pos_int_vert(1),Lati_pol,Facce_pol);
    end
elseif facce_inters(1)==0 %la faccia non intersecata e la vado a salvare in
i_base
    Lati=[i_punti(2) i_punti(3);i_punti(3) i_punti(4);i_punti(4) i_punti(2)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
    i_base=i_facce;
end

if facce_inters(2)==1
    x=length(i_facce);
    if pos_int(2,1)==-1 %il primo punto di intersezione non appartiene alla fac-
cia considerata

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(3),i_punti
(4),i_punti(6),pos_int_vert(2),Lati_pol,Facce_pol);
    else

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(3),i_punti
(4),i_punti(5),pos_int_vert(2),Lati_pol,Facce_pol);
    end
elseif facce_inters(2)==0 %la faccia non intersecata e la vado a salvare in
i_base
    Lati=[i_punti(1) i_punti(3);i_punti(3) i_punti(4);i_punti(4) i_punti(1)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
    i_base=i_facce;
end

if facce_inters(3)==1
    x=length(i_facce);
    if pos_int(3,1)==-1 %il primo punto di intersezione non appartiene alla fac-
cia considerata
```

```
[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(2),i_punti
(4),i_punti(6),pos_int_vert(3),Lati_pol,Facce_pol);
    else

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(2),i_punti
(4),i_punti(5),pos_int_vert(3),Lati_pol,Facce_pol);
    end
elseif facce_inters(3)==0 %la faccia non intersecata e la vado a salvare in
i_base
    Lati=[i_punti(1) i_punti(2);i_punti(2) i_punti(4);i_punti(4) i_punti(1)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
    i_base=i_facce;
end

if facce_inters(4)==1
    x=length(i_facce);
    if pos_int(4,1)==-1 %il primo punto di intersezione non appartiene alla fac-
cia considerata

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(2),i_punti
(3),i_punti(6),pos_int_vert(4),Lati_pol,Facce_pol);
    else

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_2(i_punti(1),i_punti(2),i_punti
(3),i_punti(5),pos_int_vert(4),Lati_pol,Facce_pol);
    end
elseif facce_inters(4)==0 %la faccia non intersecata e la vado a salvare in
i_base
    Lati=[i_punti(1) i_punti(2);i_punti(2) i_punti(3);i_punti(3) i_punti(1)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,i_facce] = Ordina_facce(Facce,Facce_pol);
    i_base=i_facce;
end

y=size(Poliedri,1);

% se consideriamo le facceche vanno da 3 a 6 avremmo che 2 di esse
% condivideranno un lato formato da due nodi. inoltre la faccia 3 non può
% condividerla con la 5 e la faccia 5 non può condividerlo con la 6.
% Controllando tale lato in condivisione andremo a creare i sottopoliedri.

if Facce_pol(i_facce(3),1)==Facce_pol(i_facce(5),1)
    Poliedri(y+1,:)=[i_facce(1) i_facce(3) i_facce(5) i_taglio -1];
    Poliedri(y+2,:)=[i_facce(2) i_facce(4) i_facce(6) i_taglio i_base];
    return
end
if Facce_pol(i_facce(3),1)==Facce_pol(i_facce(6),1)
    Poliedri(y+1,:)=[i_facce(1) i_facce(3) i_facce(6) i_taglio -1];
    Poliedri(y+2,:)=[i_facce(2) i_facce(4) i_facce(5) i_taglio i_base];
    return
end
if Facce_pol(i_facce(4),1)==Facce_pol(i_facce(5),1)
    Poliedri(y+1,:)=[i_facce(1) i_facce(4) i_facce(5) i_taglio -1];
    Poliedri(y+2,:)=[i_facce(2) i_facce(3) i_facce(6) i_taglio i_base];
    return
end
if Facce_pol(i_facce(4),1)==Facce_pol(i_facce(6),1)
```

```
    Poliedri(y+1,:)=[i_facce(1) i_facce(4) i_facce(6) i_taglio -1];
    Poliedri(y+2,:)=[i_facce(2) i_facce(4) i_facce(5) i_taglio i_base];
    return
end
end
```

Crea_facce_1.m

```
function [Lati_pol,Facce_pol,indice_facce ] = Crea_facce_1(
N1,N2,N3,P1,P2,ind_P1,ind_P2,Lati_pol,Facce_pol )

if ind_P1==1 %P1 appartiene al lato formato da N1 ed N2

    if ind_P2==2 %P2 appartiene al lato formato da N2 ed N3

        Lati=[N2 P1; P1 P2; P2 N2; N1 N3; N1 P1; N3 P2];
        [Lati_pol,i_lati]=Ordina_lati(Lati_pol,Lati);
        Facce=[i_lati(1) i_lati(2) i_lati(3) -1; i_lati(4) i_lati(5) i_lati(6)
i_lati(2)];
        [Facce_pol,indice_facce]=Ordina_facce(Facce,Facce_pol);
        return
    end

    if ind_P2==3 %%P2 appartiene al lato formato da N3 ed N1

       Lati=[N1 P1; P1 P2; P2 N1; N2 N3; N2 P1; N3 P2];
       [Lati_pol,i_lati]=Ordina_lati( Lati_pol,Lati );
       Facce=[i_lati(1) i_lati(2) i_lati(3) -1; i_lati(4) i_lati(5) i_lati(6)
i_lati(2)];
       [Facce_pol,indice_facce]=Ordina_facce(Facce,Facce_pol);
       return
    end
end

if ind_P1==2 %P1 appartiene al lato formato da N2 ed N3

    if ind_P2==1 %P2 appartiene al lato formato da N1 ed N3

        Lati=[N2 P1; P1 P2; N2 P2; N1 N3; N1 P2; N3 P1];
        [Lati_pol,i_lati]=Ordina_lati( Lati_pol,Lati );
        Facce=[i_lati(1) i_lati(2) i_lati(3) -1; i_lati(4) i_lati(5) i_lati(6)
i_lati(2)];
        [Facce_pol,indice_facce]=Ordina_facce(Facce,Facce_pol);
        return
    end

    if ind_P2==3 %%P3 appartiene al lato formato da N3 ed N1

       Lati=[N3 P1; P1 P2; N3 P2; N1 N2; N1 P2; N2 P1];
       [Lati_pol,i_lati]=Ordina_lati( Lati_pol,Lati );
       Facce=[i_lati(1) i_lati(2) i_lati(3) -1; i_lati(4) i_lati(5) i_lati(6)
i_lati(2)];
       [Facce_pol,indice_facce]=Ordina_facce(Facce,Facce_pol);
       return
```

```
    end
end

if ind_P1==3 %P1 appartiene al lato formato da N3 ed N1

    if ind_P2==1 %P2 appartiene al lato formato da N1 ed N2

        Lati=[N1 P1; P1 P2; N1 P2; N2 N3; N2 P2; N3 P1];
        [Lati_pol,i_lati]=Ordina_lati( Lati_pol,Lati );
        Facce=[i_lati(1) i_lati(2) i_lati(3) -1; i_lati(4) i_lati(5) i_lati(6)
i_lati(2)];
        [Facce_pol,indice_facce]=Ordina_facce(Facce,Facce_pol);
        return
    end

    if ind_P2==2 %%P3 appartiene al lato formato da N2 ed N3

       Lati=[N3 P1; P1 P2; N3 P2; N1 N2; N1 P1; N2 P2];
       [Lati_pol,i_lati]=Ordina_lati( Lati_pol,Lati );
       Facce=[i_lati(1) i_lati(2) i_lati(3) -1;i_lati(4) i_lati(5) i_lati(6)
i_lati(2) ];
       [Facce_pol,indice_facce]=Ordina_facce(Facce,Facce_pol);
       return
    end
end
```

Crea_facce_2.m

```
function [ Lati_pol,Facce_pol,indice_facce ] = Crea_facce_2(
N1,N2,N3,P,i_vert,Lati_pol,Facce_pol )

if i_vert==1 %il vertice intersecato è nella prima posizione
    Lati=[N1 N2; N2 P; N1 P; N1 N3; N3 P];
    [Lati_pol,i_lati]=Ordina_lati(Lati_pol,Lati);
    Facce=[i_lati(1) i_lati(2) i_lati(3) -1; i_lati(4) i_lati(5) i_lati(3) -1];
    [Facce_pol,indice_facce]=Ordina_facce(Facce,Facce_pol);
    return
end

if i_vert==2 %il vertice intersecato è nella seconda posizione
    Lati=[N1 N2; N1 P; N2 P; N2 N3; N3 P];
    [Lati_pol,i_lati]=Ordina_lati(Lati_pol,Lati);
    Facce=[i_lati(1) i_lati(2) i_lati(3) -1; i_lati(4) i_lati(5) i_lati(3) -1];
    [Facce_pol,indice_facce]=Ordina_facce(Facce,Facce_pol);
    return
end

if i_vert==3 %il vertice intersecato è nella terza posizione
    Lati=[N1 N3; N1 P; N3 P; N2 N3; N2 P];
    [Lati_pol,i_lati]=Ordina_lati(Lati_pol,Lati);
    Facce=[i_lati(1) i_lati(2) i_lati(3) -1; i_lati(4) i_lati(5) i_lati(3) -1];
    [Facce_pol,indice_facce]=Ordina_facce(Facce,Facce_pol);
    return
end
```

Crea_poliedri_3.m

```
function [ Poliedri,Facce_pol,Lati_pol ] = Crea_poliedri_3( i_punti,facce_in-
ters,pos_int,Poliedri,Facce_pol,Lati_pol )

%siamo nel caso in cui vi sono 3 punti di intersezione

i_facce=[];

%formo la faccia del taglio formata dai tre punti di intersezione

Lati=[i_punti(5) i_punti(6);i_punti(6) i_punti(7);i_punti(7) i_punti(5)];
[Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
[Facce_pol,indice_facce] = Ordina_facce(Facce,Facce_pol);
i_taglio=indice_facce;

if facce_inters(1)==1

    x=length(i_facce);

    if pos_int(1,1)==-1 %il primo punto non appartiene alla prima faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
(4),i_punti(6),i_punti(7),pos_int(1,2),pos_int(1,3),Lati_pol,Facce_pol);
    end
    if pos_int(1,2)==-1 %il secondo punto non appartiene alla prima faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
(4),i_punti(5),i_punti(7),pos_int(1,1),pos_int(1,3),Lati_pol,Facce_pol);
    end
    if pos_int(1,3)==-1 %il terzo punto non appartiene alla prima faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
(4),i_punti(5),i_punti(6),pos_int(1,1),pos_int(1,2),Lati_pol,Facce_pol);
    end

else %se la faccia non è intersecata, la salvo in i_base
    Lati=[i_punti(2) i_punti(3);i_punti(3) i_punti(4);i_punti(4) i_punti(2)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,indice_facce] = Ordina_facce(Facce,Facce_pol);
    i_base=indice_facce;
end



if facce_inters(2)==1

    x=length(i_facce);

    if pos_int(2,1)==-1 %il primo punto non appartiene alla seconda faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(6),i_punti(7),pos_int(2,2),pos_int(2,3),Lati_pol,Facce_pol);
```

```
    end
    if pos_int(2,2)==-1 %il secondo punto non appartiene alla seconda faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(5),i_punti(7),pos_int(2,1),pos_int(2,3),Lati_pol,Facce_pol);
    end
    if pos_int(2,3)==-1 %il terzo punto non appartiene alla seconda faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(5),i_punti(6),pos_int(2,1),pos_int(2,2),Lati_pol,Facce_pol);
    end
else %se la faccia non è intersecata, la salvo in i_base
    Lati=[i_punti(1) i_punti(3);i_punti(3) i_punti(4);i_punti(4) i_punti(1)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,indice_facce] = Ordina_facce(Facce,Facce_pol);
    i_base=indice_facce;
end



if facce_inters(3)==1

    x=length(i_facce);

    if pos_int(3,1)==-1 %il primo punto non appartiene alla terza faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(6),i_punti(7),pos_int(3,2),pos_int(3,3),Lati_pol,Facce_pol);
    end
    if pos_int(3,2)==-1 %il secondo punto non appartiene alla terza faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(5),i_punti(7),pos_int(3,1),pos_int(3,3),Lati_pol,Facce_pol);
    end
    if pos_int(3,3)==-1 %il terzo punto non appartiene alla terza faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(5),i_punti(6),pos_int(3,1),pos_int(3,2),Lati_pol,Facce_pol);
    end
else %se la faccia non è intersecata, la salvo in i_base
    Lati=[i_punti(1) i_punti(2);i_punti(2) i_punti(4);i_punti(4) i_punti(1)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,indice_facce] = Ordina_facce(Facce,Facce_pol);
    i_base=indice_facce;
end



if facce_inters(4)==1

    x=length(i_facce);

    if pos_int(4,1)==-1 %il primo punto non appartiene alla prima faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(3),i_punti(6),i_punti(7),pos_int(4,2),pos_int(4,3),Lati_pol,Facce_pol);
    end
    if pos_int(4,2)==-1 %il secondo punto non appartiene alla prima faccia
```

```
[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(3),i_punti(5),i_punti(7),pos_int(4,1),pos_int(4,3),Lati_pol,Facce_pol);
    end
    if pos_int(4,3)==-1 %il terzo punto non appartiene alla prima faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(3),i_punti(5),i_punti(6),pos_int(4,1),pos_int(4,2),Lati_pol,Facce_pol);
    end
else %se la faccia non è intersecata, la salvo in i_base
    Lati=[i_punti(1) i_punti(2);i_punti(2) i_punti(3);i_punti(3) i_punti(1)];
    [Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
    Facce=[indice_lati(1),indice_lati(2),indice_lati(3),-1];
    [Facce_pol,indice_facce] = Ordina_facce(Facce,Facce_pol);
    i_base=indice_facce;
end

x=size(Poliedri,1);

%formo i poliedri

Poliedri(x+1,:)=[i_facce(1),i_facce(3),i_facce(5),i_taglio,-1];
Poliedri(x+2,:)=[i_facce(2),i_facce(4),i_facce(6),i_taglio,i_base];

end
```

Crea_poliedri_4.m

```
function [ Poliedri,Facce_pol,Lati_pol ] = Crea_poliedri_4(
i_punti,pos_int_facce,pos_int,Poliedri,Facce_pol,Lati_pol)


[Facce_pol,Lati_pol,i_taglio ] = Taglio_4(
i_punti(5),i_punti(6),i_punti(7),i_punti(8),pos_int,Facce_pol,Lati_pol );

i_facce=[];
x=length(i_facce);

%creo i lati e le relative facce per ogni triangolo che forma il tetraedro

if (pos_int_facce(1,1)~=-1 && pos_int_facce(1,2)~=-1) %il primo ed il secondo
punto di intersezione appartengono alla prima faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
(4),i_punti(5),i_punti(6),pos_int_facce(1,1),pos_int_facce(1,2),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(1,1)~=-1 && pos_int_facce(1,3)~=-1) %primo  e terzo

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
(4),i_punti(5),i_punti(7),pos_int_facce(1,1),pos_int_facce(1,3),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(1,1)~=-1 && pos_int_facce(1,4)~=-1) %primo e quarto

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
```

```
(4),i_punti(5),i_punti(8),pos_int_facce(1,1),pos_int_facce(1,4),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(1,2)~=-1 && pos_int_facce(1,3)~=-1) %secondo e terzo

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
(4),i_punti(6),i_punti(7),pos_int_facce(1,2),pos_int_facce(1,3),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(1,2)~=-1 && pos_int_facce(1,4)~=-1) %secondo e quarto

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
(4),i_punti(6),i_punti(8),pos_int_facce(1,2),pos_int_facce(1,4),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(1,3)~=-1 && pos_int_facce(1,4)~=-1) %terzo e quarto

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(2),i_punti(3),i_punti
(4),i_punti(7),i_punti(8),pos_int_facce(1,3),pos_int_facce(1,4),Lati_pol,Facce_p
ol);
end

x=length(i_facce);
if (pos_int_facce(2,1)~=-1 && pos_int_facce(2,2)~=-1) %il primo ed il secondo
punto di intersezione appartengono alla seconda faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(5),i_punti(6),pos_int_facce(2,1),pos_int_facce(2,2),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(2,1)~=-1 && pos_int_facce(2,3)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(5),i_punti(7),pos_int_facce(2,1),pos_int_facce(2,3),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(2,1)~=-1 && pos_int_facce(2,4)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(5),i_punti(8),pos_int_facce(2,1),pos_int_facce(2,4),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(2,2)~=-1 && pos_int_facce(2,3)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(6),i_punti(7),pos_int_facce(2,2),pos_int_facce(2,3),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(2,2)~=-1 && pos_int_facce(2,4)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(6),i_punti(8),pos_int_facce(2,2),pos_int_facce(2,4),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(2,3)~=-1 && pos_int_facce(2,4)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(3),i_punti
(4),i_punti(7),i_punti(8),pos_int_facce(2,3),pos_int_facce(2,4),Lati_pol,Facce_p
ol);
end

x=length(i_facce);
```

```
if (pos_int_facce(3,1)~=-1 && pos_int_facce(3,2)~=-1) %il primo ed il secondo
punto di intersezione appartengono alla terza faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(5),i_punti(6),pos_int_facce(3,1),pos_int_facce(3,2),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(3,1)~=-1 && pos_int_facce(3,3)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(5),i_punti(7),pos_int_facce(3,1),pos_int_facce(3,3),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(3,1)~=-1 && pos_int_facce(3,4)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(5),i_punti(8),pos_int_facce(3,1),pos_int_facce(3,4),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(3,2)~=-1 && pos_int_facce(3,3)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(6),i_punti(7),pos_int_facce(3,2),pos_int_facce(3,3),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(3,2)~=-1 && pos_int_facce(3,4)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(6),i_punti(8),pos_int_facce(3,2),pos_int_facce(3,4),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(3,3)~=-1 && pos_int_facce(3,4)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(4),i_punti(7),i_punti(8),pos_int_facce(3,3),pos_int_facce(3,4),Lati_pol,Facce_p
ol);
end

x=length(i_facce);
if (pos_int_facce(4,1)~=-1 && pos_int_facce(4,2)~=-1) %il primo ed il secondo
punto di intersezione appartengono alla quarta faccia

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(3),i_punti(5),i_punti(6),pos_int_facce(4,1),pos_int_facce(4,2),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(4,1)~=-1 && pos_int_facce(4,3)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(3),i_punti(5),i_punti(7),pos_int_facce(4,1),pos_int_facce(4,3),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(4,1)~=-1 && pos_int_facce(4,4)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(3),i_punti(5),i_punti(8),pos_int_facce(4,1),pos_int_facce(4,4),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(4,2)~=-1 && pos_int_facce(4,3)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
```

```
(3),i_punti(6),i_punti(7),pos_int_facce(4,2),pos_int_facce(4,3),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(4,2)~=-1 && pos_int_facce(4,4)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(3),i_punti(6),i_punti(8),pos_int_facce(4,2),pos_int_facce(4,4),Lati_pol,Facce_p
ol);
end
if (pos_int_facce(4,3)~=-1 && pos_int_facce(4,4)~=-1)

[Lati_pol,Facce_pol,i_facce(x+1:x+2)]=Crea_facce_1(i_punti(1),i_punti(2),i_punti
(3),i_punti(7),i_punti(8),pos_int_facce(4,3),pos_int_facce(4,4),Lati_pol,Facce_p
ol);
end

%ho creato otto facce in tale tipologia di intersezione

L=size(Poliedri,1);

%adesso per creare i poliedri controllo le facce salvate in i_facce di
%indice pari, e ho che due di esse condivideranno un lato formato da due
%nodi(che si trova nalla prima posizione), allora avrò le seguenti
%possibilità: 2-4 e 6-8, 2-6 e 4-8, 2-8 e 4-6. Con dei controlli allora
%andrò a formare i poliedri.

if Facce_pol(i_facce(2),1)== Facce_pol(i_facce(4),1)
    Poliedri(L+1,:)=[i_facce(1),i_facce(3),i_facce(6),i_facce(8),i_taglio];
    Poliedri(L+2,:)=[i_facce(2),i_facce(4),i_facce(5),i_facce(7),i_taglio];
    return
end

if Facce_pol(i_facce(2),1)== Facce_pol(i_facce(6),1)
    Poliedri(L+1,:)=[i_facce(1),i_facce(4),i_facce(5),i_facce(8),i_taglio];
    Poliedri(L+2,:)=[i_facce(2),i_facce(3),i_facce(6),i_facce(7),i_taglio];
    return
end

if Facce_pol(i_facce(2),1)== Facce_pol(i_facce(8),1)
    Poliedri(L+1,:)=[i_facce(1),i_facce(4),i_facce(6),i_facce(7),i_taglio];
    Poliedri(L+2,:)=[i_facce(2),i_facce(3),i_facce(5),i_facce(8),i_taglio];
    return
end
end
```

Tetra_vicini.m

```
function [vicini,punti_vicini,lati_vicini,facce_vicini] = Tetra_vicini( te-
tra_inters )
global ele n_ele
vicini=[];
punti_vicini=[];
lati_vicini=[];
facce_vicini=[];
```

```
% serve per individuare i tetraedri vicini ai tetraedri intersecati e per
% salvare quali punti, lati e facce sono condivisi con tali tetraedri

for i=1:n_ele
    if (tetra_inters(i)==1 || tetra_inters(i)==1.5) %se il teraedro è interse-
cato
        for j=1:n_ele %ciclo nuovamente sui tetraedri
            punti_provv=[];
            if (tetra_inters(j)~=1 && tetra_inters(i)~=1.5) %verifico che il te-
traedro non isa intersecato
                for k=1:4 %ciclo sui nodi del tetraedro

                    %salvo i vertici in comune

                    if ele(i,1)==ele(j,k) %se i tetraedri hanno il primo nodo in
comune
                        x=length(punti_provv);
                        punti_provv(x+1)=ele(i,1);
                    end
                    if ele(i,2)==ele(j,k) %se i tetraedri hanno il secondo nodo
in comune
                         x=length(punti_provv);
                         punti_provv(x+1)=ele(i,2);

                    end
                    if ele(i,3)==ele(j,k) %se i tetraedri hanno il terzo nodo in
comune
                         x=length(punti_provv);
                         punti_provv(x+1)=ele(i,3);

                    end
                    if ele(i,4)==ele(j,k) %se i tetraedri hanno il quarto nodo
in comune
                         x=length(punti_provv);
                         punti_provv(x+1)=ele(i,4);

                    end
                end

                l=length(punti_provv);
                if l>0 %i tetraedri condiderati hanno almeno un nodo in comune

                    punti_vicini=Ordina_punti_vicini(punti_vicini,punti_provv);
%ricavo gli indici dei nodi vicini

                    x=length(vicini);
                    flag=1;

                    for i_c=1:x
                        if vicini(i_c)==j %controllo se il tetraedro non appar-
tiene già ai vicini
                            flag=0;
                        end
                    end
                    if flag==1
                        vicini(x+1)=j; %salvo il vicino
                    end
```

```
                    if l==2 %i tetraedri condividono un lato

                        lati_vicini=Ordina_lati(lati_vicini,punti_provv);

                    elseif l==3 % i tetraedri condividono una faccia
                        lati_vicini=Ordina_lati(lati_vicini,punti_provv(1:2));
%salvo il primo lato 1-2
                        lati_vicini=Ordina_lati(lati_vicini,punti_provv(2:3));
%salvo il secondo lato 2-3
                        lati_vicini=Ordina_lati(lati_vicini,punti_provv(1:2:3));
%salvo il terzo lato 1-3
                        facce_vicini=Ordina_facce_vicini(facce_vi-
cini,punti_provv); %salvo la faccia
                    end
                end
            end
        end
    end
end
```

Ordina_punti_vicini.m

```
function [ punti_vicini] = Ordina_punti_vicini( punti_vicini,punti_provv )
lun1=length(punti_provv);
lun2=length(punti_vicini);

%controllo che il nodo in condivisione tra un vicino e il tetraedro
%intersecato non appartenga già a punti_vicini:
%se appartiene ne salvo l'indice, se non appartiene lo aggiungo

for i=1:lun1
    flag=0;
    for j=1:lun2
        if punti_provv(i) == punti_vicini(j)
            flag=1;
        end
    end
    if flag==0

            L=length(punti_vicini);
            punti_vicini(L+1)=punti_provv(i);

    end
end
end
```

Ordina_lati.m

```
function [ Lati_pol,indice_lati ] = Ordina_lati( Lati_pol,Lati )

lun1=size(Lati,1);
lun2=size(Lati_pol,1);
indice_lati=[];
```

```
%controllo che il lato non appartenga già a Lati_poli:
%se appartiene ne salvo l'indice, se non appartiene lo aggiungo e salvo il nuovo
indice L+1

for i=1:lun1
    flag=0;
    for j=1:lun2
        if sort(Lati(i,:))==sort(Lati_pol(j,:))
            lun_indice=length(indice_lati);
            indice_lati(lun_indice+1)=j;
            flag=1;
        end
    end

    if flag==0
            L=size(Lati_pol,1);
            Lati_pol(L+1,:)=Lati(i,:);
            lun_indice=length(indice_lati);
            indice_lati(lun_indice+1)=L+1;
    end

end
```

Ordina_facce.m

```
function [ Facce_pol,indice_facce ] = Ordina_facce( Facce,Facce_pol )

lun1=size(Facce,1);
lun2=size(Facce_pol,1);
indice_facce=[];

for i=1:lun1
    flag=0;
    for j=1:lun2
        if sort(Facce(i,:))==sort(Facce_pol(j,:))
            lun_indice=length(indice_facce);
            indice_facce(lun_indice+1)=j;
            flag=1;
        end
    end

    if flag==0
            L=size(Facce_pol,1);
            Facce_pol(L+1,:)=Facce(i,:);
            lun_indice=length(indice_facce);
            indice_facce(lun_indice+1)=L+1;
    end

end
```

```
end
```

Ordina_facce_vicini.m

```
function [ facce_vicini ] = Ordina_facce_vicini( facce_vicini,facce )
lun1=size(facce,1);
lun2=size(facce_vicini,1);

%controllo se la faccia appartiene già, se non appartiene la aggiungo

for i=1:lun1
    flag=0;
    for j=1:lun2
        if sort(facce(i,:))==sort(facce_vicini(j,:))
            flag=1;
        end
    end

    if flag==0
            L=size(facce_vicini,1);
            facce_vicini(L+1,:)=facce(i,:);
    end

end
end
```

Taglio_4.m

```
function [Facce_pol,Lati_pol,i_taglio ] = Taglio_4(
P1,P2,P3,P4,pos,Facce_pol,Lati_pol )

Lati=[];

%tale funzione serve a formare i lati della faccia del taglio quando
%questa è formata da 4 punti di intersezione.

%considero i 6 possibili lati formati da quattro vertici ed escludo quelli
%che non possono essere presi, ovvero, secondo la struttura interna delle
%facce, non può esservi un lato quando i due vertici considerati hanno le
%seguenti posizioni: 1-6, 2-5, 3-4, 4-3, 5-2, 6-1. In caso contrario salvo
%i lati.

if ((pos(1)==1 && pos(2)~=6)||(pos(1)==2 && pos(2)~=5) ||(pos(1)==3 &&
pos(2)~=4) ||(pos(1)==4 && pos(2)~=3) ||(pos(1)==5 && pos(2)~=2) ||(pos(1)==6 &&
pos(2)~=1))
    x=size(Lati,1);
    Lati(x+1,:)=[P1 P2];
end
if ((pos(1)==1 && pos(3)~=6)||(pos(1)==2 && pos(3)~=5) ||(pos(1)==3 &&
pos(3)~=4) ||(pos(1)==4 && pos(3)~=3) ||(pos(1)==5 && pos(3)~=2) ||(pos(1)==6 &&
pos(3)~=1))
    x=size(Lati,1);
    Lati(x+1,:)=[P1 P3];
end
```

```
if ((pos(1)==1 && pos(4)~=6)||(pos(1)==2 && pos(4)~=5) ||(pos(1)==3 &&
pos(4)~=4) ||(pos(1)==4 && pos(4)~=3) ||(pos(1)==5 && pos(4)~=2) ||(pos(1)==6 &&
pos(4)~=1))
    x=size(Lati,1);
    Lati(x+1,:)=[P1 P4];
end


if ((pos(2)==1 && pos(3)~=6)||(pos(2)==2 && pos(3)~=5) ||(pos(2)==3 &&
pos(3)~=4) ||(pos(2)==4 && pos(3)~=3) ||(pos(2)==5 && pos(3)~=2) ||(pos(2)==6 &&
pos(3)~=1))
    x=size(Lati,1);
    Lati(x+1,:)=[P2 P3];
end
if ((pos(2)==1 && pos(4)~=6)||(pos(2)==2 && pos(4)~=5) ||(pos(2)==3 &&
pos(4)~=4) ||(pos(2)==4 && pos(4)~=3) ||(pos(2)==5 && pos(4)~=2) ||(pos(2)==6 &&
pos(4)~=1))
    x=size(Lati,1);
    Lati(x+1,:)=[P2 P4];
end


if ((pos(3)==1 && pos(4)~=6)||(pos(3)==2 && pos(4)~=5) ||(pos(3)==3 &&
pos(4)~=4) ||(pos(3)==4 && pos(4)~=3) ||(pos(3)==5 && pos(4)~=2) ||(pos(3)==6 &&
pos(4)~=1))
    x=size(Lati,1);
    Lati(x+1,:)=[P3 P4];
end

%dopo aver trovato i lati di interesse, salvo la faccia e ne ricavo
%l'indice.

[Lati_pol,indice_lati] = Ordina_lati( Lati_pol,Lati );
Facce=[indice_lati(1),indice_lati(2),indice_lati(3),indice_lati(4)];
[Facce_pol,i_taglio] = Ordina_facce(Facce,Facce_pol);

end
```