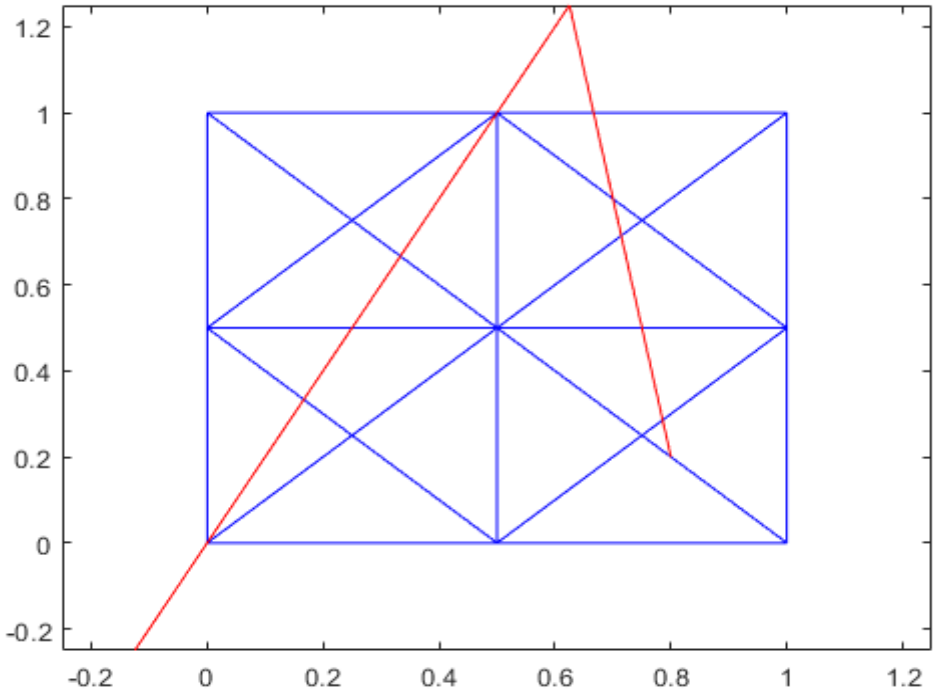
Programmazione e calcolo scientifico

Triangolazione

Gabriele Fioravanti

Gianluigi Lopardo

Triangolazione



Introduzione e main

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 *quad.1.ele* (contenente le informazioni sui triangoli) 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.

Come prima cosa, viene inizializzata la matrice *vertcom*, avente nella colonna i-esima tutti i triangoli contenenti il vertice i. Servirà nel punto 2) per trovare i triangoli vicini ad un triangolo tagliato.

vertcom

8x13 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	7	15	8	3	1	2	5	3	2	1	3	8	1
2	9	16	10	4	2	4	6	8	6	10	4	10	5
3	0	0	0	0	5	9	7	11	7	13	11	13	15
4	0	0	0	0	6	12	16	14	9	15	12	14	16
5	0	0	0	0	11	0	0	0	0	0	0	0	0
6	0	0	0	0	12	0	0	0	0	0	0	0	0
7	0	0	0	0	13	0	0	0	0	0	0	0	0
8	0	0	0	0	14	0	0	0	0	0	0	0	0

Dopo l'inizializzazione delle variabili necessarie si esegue un ciclo che, per ogni traccia, richiama le funzioni *trian*, *vicinilato* e *vicinivertice*, che eseguono le operazioni necessarie per tutto il progetto.

Tutti i dati richiesti vengono infine stampati su un file .txt.

Nello specifico i vari punti richiesti dal progetto:

1) Per ciascuna traccia individuare l'elenco dei triangoli tagliati dalla traccia.

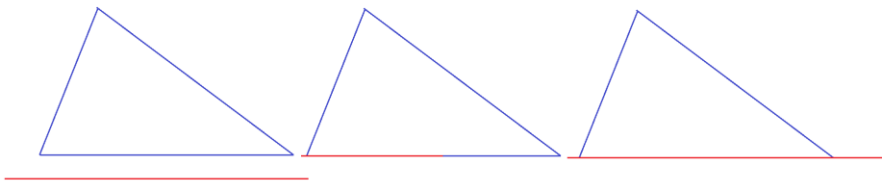
Dal main viene chiamata la funzione *trian*, che gestisce le funzioni per le intersezioni e la sottotriangolazione.

Viene inizializzato *triangle* come vettore nullo di dimensione *n_ele* (numero dei triangoli).

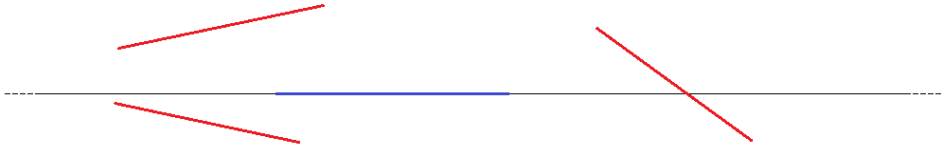
Inizia il controllo in ordine crescente, dal primo triangolo. Quando si arriva ad una intersezione il programma entra nella funzione *Coda.m*. Appena trovo una intersezione in un triangolo, verifico tra i suoi vicini e metto in coda a loro volta i vicini dei vicini nel caso in cui ci sia intersezione. Dopo aver verificato che il vicino j-esimo esista (`if neigh(i,j) ~= -1`), ci assicuriamo di non aver già controllato il triangolo in esame (`if triangle(neigh(i,j)) == 0`) e che non sia già stato inserito nella coda (`if neigh(i,j)==coda(z)`. In caso affermativo inserisco il triangolo nella coda dei triangoli da controllare.
Così facendo, invece di eseguire il ciclo su ogni triangolo, eseguo i controlli solo sui triangoli inseriti nella coda, in cui potrebbe esserci intersezione, aggiornando il rispettivo vettore ogniqualvolta si trova una nuova intersezione, fino ad esaurire la coda.
Per ogni triangolo salvo una matrice contenente i suoi nodi e li passo a due a due alla funzione *intersezione*, in modo da identificarne un lato. Questa funzione restituisce un parametro *alfa* che, per ogni lato, identifica un caso di intersezione (ad es. *alfa=-2* se la retta su cui giace la traccia e quella su cui giace il lato sono paralleli). Attraverso *intersezione* esaurisco tutti i casi possibili per l'intersezione, dopo aver trovato il vettore tangente il lato e quello tangente la traccia.

- Vettori paralleli: utilizzo il prodotto vettoriale [*alfa=-2*]
    - Lato coincidente: verifico sia per l'asse x che y [*alfa=-2*]
        - Ricopre solo parte del lato (da intendersi tagliato). Controllo se gli estremi della traccia sono compresi tra 0 e 1, gli estremi del segmento del lato parametrizzato. [*alfa=2*]

```
if (alfa==-2 && ((t1>0 && t1<1) || (t2>0 && t2<1)))
```



Escludendo i casi paralleli vi è sicuramente intersezione tra le rette su cui giacciono rispettivamente la traccia ed il lato. Viene individuata la disequazione che divide il piano in due parti: sopra e sotto la retta su cui giace la traccia. Quindi escludiamo i casi in cui il lato si trova 'tutto sopra' o 'tutto sotto' la retta su cui giace la traccia.



Diversamente viene considerato il caso in cui uno degli estremi del lato si trovi sopra e uno si trovo sotto la retta.
Per trovare il punto di intersezione tra i due segmenti inizialmente vengono esauriti i casi in cui la traccia sia parallela all'asse delle ascisse o all'asse delle ordinate. Ad esempio, consideriamo il caso in cui sia parallelo all'asse x, allora:

$$\hat{x} = \frac{\hat{y}-y_1}{y_2-y_1}(x_2 - x_1) + x_1,$$

dove $y = \hat{y} = costante$, essendo la traccia parallela all'asse x. Sostituendo questo valore costante nell'equazione della retta su cui giace il lato, ricaviamo l'ascissa del punto di intersezione, che avranno coordinate $(\hat{x}, \hat{y})$.
Esclusi i casi paralleli, si arriva all'equazione generale, inizialmente parametrizzando le due rette come segue:

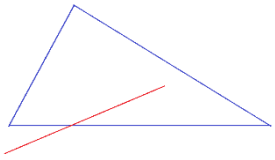
$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}$$

E riconducendo ad un sistema lineare risolubile attraverso il metodo di Gauss:

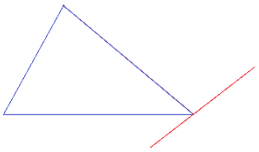
```
A=[-1/(XE2-XE1) 1/(YE2-YE1); -1/(XT2-XT1) 1/(YT2-YT1)];
b=[YE1/(YE2-YE1)-XE1/(XE2-XE1); YT1/(YT2-YT1)- XT1/(XT2-XT1)];
P=A\b;
```

Andiamo a visualizzare i possibili casi di intersezione:

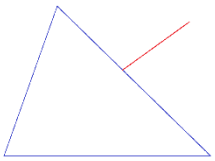
- Intersezione tra il lato e la traccia in un punto P generico esclusi i vertici



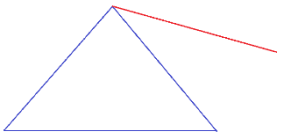
- Intersezione tra un vertice del lato e la traccia



- Intersezione tra punto generico del triangolo e tra un estremo della traccia



- Intersezione tra un vertice del triangolo e tra un estremo della traccia



- Nessuna intersezione

Dopo aver verificato l'intersezione per ogni lato verifico per ogni triangolo se ha almeno un lato intersecato e salvo un parametro *beta* avente valore 1 in caso affermativo.
Infine genero la matrice *triangle* avente per ogni traccia (righe) il valore *beta* di ogni triangolo *i* nella colonna *i-esima*.

Traccia/Triangolo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	1	1	0	1	1	0	0	0	1	0	1	0
2	1	0	1	1	0	1	0	0	0	1	0	0	1	0	0

2) Per ciascuna traccia individuare l'elenco dei triangoli che condividono almeno un vertice con un triangolo tagliato e per questi memorizzare anche quali vertici o lati sono condivisi con un triangolo tagliato.

Triangoli che condividono un lato con un triangolo tagliato

Per ogni triangolo, verifico se questo è tagliato (*triangle(i)==1*). In caso affermativo controllo per ognuno dei tre possibili vicini (se esistono) se non è a sua volta stato intersecato dalla traccia (e quindi appartiene già a *triangle* e non va considerato tra i vicini dei triangoli tagliati). Salvo infine i due vertici del lato in comune.

In stampa avrò:

```
Triangoli che condividono un lato con un triangolo tagliato: [6 7 3 14 4]  che condivi-
              dono rispettivamente i lati:      [9 5;1 9;8 11;5 8;6 11]
```

Triangoli che condividono un vertice con un triangolo tagliato

Per ogni triangolo, verifico se questo è tagliato *triangle(i)==1*. In caso affermativo ciclo finché non esaurisco la matrice *vertcom* generata nel *main*.

Per ogni triangolo verifico nelle colonne corrispondenti ai suoi vertici controllo che il triangolo considerato non sia tagliato dalla traccia e che non sia già stato inserito nella lista. Escludo i triangoli che hanno due vertici in comune, in quanto già considerati.

In stampa avrò:

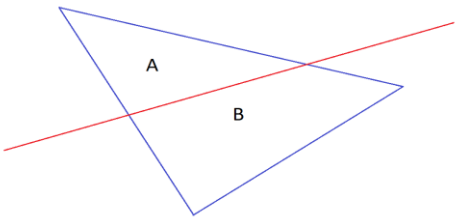
```
Triangoli che condividono un vertice con un triangolo tagliato:
```

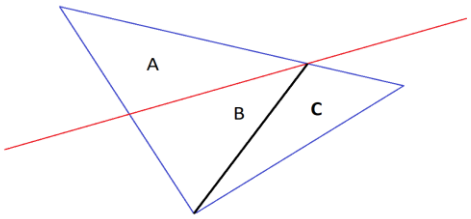
```
[1 5 13 8]     che condividono rispettivamente i vertici: [5 5 5 8]
```

3) Per ciascun triangolo tagliato creare i sottopoligoni ottenuti tagliando il triangolo con la traccia.

Vengono distinti diversi casi:

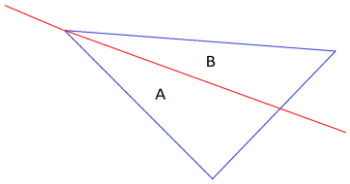
- Traccia che attraversa il triangolo in punti diversi dai vertici:





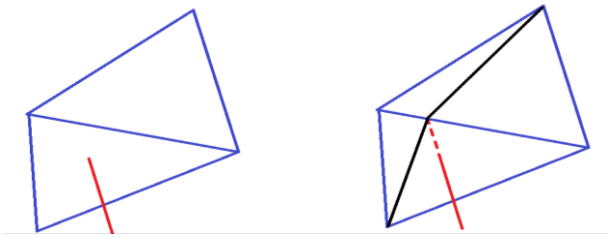
Genera il triangolo A e il quadrilatero B. Vengono salvati i punti di intersezione e il quadrilatero viene sottotriangolato.

- Traccia passante per un vertice e un punto interno al lato:



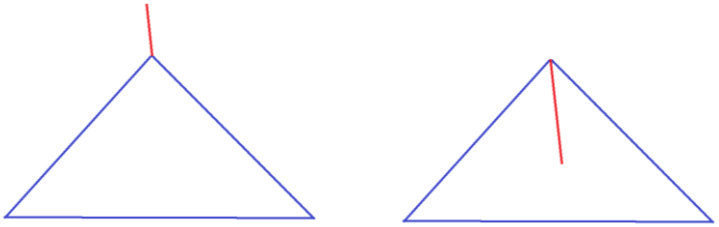
Genera automaticamente due triangoli, salvo il punto di intersezione.

- Traccia che termina all'interno del triangolo:



La traccia va prolungata fino a toccare un lato del triangolo. Viene eseguita la sottotriangolazione come nei punti precedenti, ma va inoltre considerata l'intersezione con l'eventuale triangolo vicino che condivide col triangolo tagliato il lato intersecato.

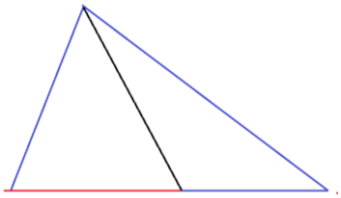
- Estremo della traccia in un vertice:



Se un estremo della traccia si trova in un vertice del triangolo vanno differenziati i due casi in figura: nel primo caso il triangolo non si intende tagliato, nel secondo prolungo la traccia e procedo come visto nei punti precedenti.

L'appartenenza del secondo vertice al triangolo viene verificata tramite la funzione *interno.*

- Traccia che ricopre parzialmente il lato:



Viene salvato l'estremo che cade sul lato, sarà quindi un vertice dei nuovi triangoli generati dalla sottotriangolazione.

4) Salvare le coordinate curvilinee delle intersezioni della traccia con i segmenti o punti della triangolazione.

La traccia viene parametrizzata come $\gamma(t) = T_1 + t(T_2 - T_1)$, con $t \in [0,1]$, dove T_1 e T_2 sono gli estremi della traccia. Dopo aver calcolato l'intersezione con i diversi lati dei triangoli, viene calcolata la coordinata curvilinea t e salvata in un vettore quando sussiste l'intersezione (ovvero quando $t \in [0,1]$), così da poter risalire al punto di intersezione sostituendo t nella parametrizzazione della traccia.

Codice

index.m

```
%inizializzazione
tic
global edge ele neigh node punti tracce
global n_edge n_ele n_neigh n_node n_punti n_tracce


[edge, n_edge] = filetomatr('triangolazione_e_tracce/quad.1.edge');
[ele, n_ele] = filetomatr('triangolazione_e_tracce/quad.1.ele');
[neigh, n_neigh] = filetomatr('triangolazione_e_tracce/quad.1.neigh');
[node, n_node] = filetomatr('triangolazione_e_tracce/quad.1.node');
[punti, n_punti, tracce, n_tracce] = traccetomatr('triangola-
zione_e_tracce/trace.trace');

%salvo triangoli per ogni vertice
vertcom = [];
    for i = 1:n_node
        s = 1;
        for j = 1:n_ele
            for k = 2:4
                if i == ele(j,k)
                    vertcom(s,i) = ele(j,1);
                    s = s+1;
                end
            end
        end
    end
    fid = fopen ('result.txt', 'wt');
%per ogni traccia

title = 'Programmazione e Calcolo Scientifico - Problema 1';
fprintf (fid, '%s\n\n\n', title);
for i_traccia=1:n_tracce
    [triangle,sottopoligoni3,sottopoligoni4,triangolazione,curvilinee] =
trian(i_traccia);
    [lati,tagl] = vicinilato(triangle);
    [vert,tagv] = vicinivertice(triangle,tagl,vertcom);

    % scrittura su file

fprintf(fid, '\n\t\t%s %d', 'Risultati per la traccia numero', i_traccia);

%punto 1
title1 = '(1) Elenco dei triangoli tagliati dalla traccia (il numero del trian-
golo corrisponde alla colonna, la colonna corrisponde alla traccia).';
legend1_1 = ' 1 : intersezione';
legend1_0 = ' 0 : nessuna intersezione';
legend1_m1 = '-1 : nessuna intersezione, triangolo controllato';
legend1_m5 = '1/2: triangolo non intersecato ma sottotriangolato';
fprintf (fid, '\n\n%s\n\t%s\n\t%s\n\t%s\n\t%s\n \n\n', title1, legend1_1, leg-
end1_0, legend1_m1, legend1_m5);
fprintf (fid, [repmat(' %d\t', [1, 16]) '\n'], triangle);
%fprintf (fid, '\n\n%s\n\t%s\n\t%s\n\t%s\n\t%s\n', title1, legend1_1, legend1_0,
legend1_m1, mat2str(triangle));
```

```
%punto 2
title2 = '(2) Elenco dei triangoli che condividono almeno un vertice con un
triangolo tragliato.';
legend2_v = ' - Triangoli che condividono solo un vertice con un triangolo ta-
gliato: ';
legend2_v2 = ' che condividono rispettivamente i vertici: ';
legend2_l = ' - Triangoli che condividono un lato con un triangolo tagliato: ';
legend2_l2 = ' che condividono rispettivamente i lati: ';
fprintf (fid, '\n\n%s\n\t%s\t%s\t%s%s\n\t%s\t%s\t%s\t%s \n\n', title2, leg-
end2_v, mat2str(tagv), legend2_v2, mat2str(vert), legend2_l, mat2str(tagl),leg-
end2_l2, mat2str(lati));
    % da stampare anche i lati e i vertici in corrispondenza

%punto 3
title3 = '(3) Elenco dei si sottopoligoni ottenuti tagliando il triangolo con la
traccia.';
fprintf (fid, '\n\n%s\n\t%s\n\t%s\n\n\n', title3, mat2str(sottopoligoni3,2),
mat2str(sottopoligoni4,2));
    %stamp sottopoligoni3, sottopoligoni4

%punto 4
title4 = '(4) Sottotriangolazione conforme alla traccia ed al suo eventuale pro-
lungamento.';
fprintf (fid, '\n\n%s\n\t%s\n\t%s\n\n\n', title4, mat2str(triangolazione,2),
mat2str(sottopoligoni3,2));
    %stamp triangolazione [sottopoligoni3 fa parte della triangolazione]

%punto 5
title5 = '(5) Coordinate curvilinee delle intersezioni della traccia con i seg-
menti o punti della triangolazione.';
fprintf (fid, '\n\n%s\t%s\n', title5, mat2str(curvilinee,2));
    %stamp curvilinee


fprintf(fid, '\n\n\n\n');



end
fprintf(fid, '\n\n\n\n%s\n%s', 'Gabriele Fioravanti','Gianluigi Lopardo');

fclose (fid);
toc
```

filetomatr.m

```
%funzione per leggere tutti i file dei triangoli

function [res,n] = filetomatr(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])'; %salvo la matrice con le info necessarie
end
```

traccetomatr.m

```
%funzione per leggere tutti i file della traccia

function [res,n,res2,n2] = traccetomatr(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',[3,n])'; %salvo la matrice con le info necessarie
n2 = fscanf(f,'%d',1)'; %salvo il primo valore, che indica il numero di righe
fgetl(f); %tolgo la prima riga inutile
res2 = fscanf(f,'%f',[3,n2])'; %salvo la matrice con le info necessarie
end
```

trian.m

```
%funzione per leggere i triangoli

function [triangle,sottopoligoni3,sottopoligoni4,triangolazione,curvilinee] =
trian(i_traccia)

global n_ele

triangle = zeros(1,n_ele);
sottopoligoni3=[];
sottopoligoni4=[];
triangolazione=[];
%s = zeros(3,2);
coda=[];
curvilinee=[];

for i=1:n_ele

    [triangle(i),sottopoligoni3,sottopoligoni4,triangolazione,t1,t2,t3,prolungo]
= intersezione_trian(i,i_traccia,sottopoligoni3,sottopoligoni4,triangolazione);
    if triangle(i)==-1
                t1=inf;
                t2=inf;
                t3=inf;
    end
    curvilinee=CC(curvilinee,t1,t2,t3);

    if prolungo>0
         triangle(prolungo)=0.5;
    end

    if triangle(i) == 1 %appena trovata la prima intersezione inizio il meccani-
smo di coda

         coda=Coda(coda,triangle,i);
         lun=length(coda);
         i_coda=0;
```

```
        while i_coda~=lun

            i_coda=i_coda+1;

            [triangle(coda(i_coda)),sottopoligoni3,sottopoligoni4,triangola-
zione,t1,t2,t3,prolungo]=intersezione_trian(coda(i_coda),i_traccia,sottopoli-
goni3,sottopoligoni4,triangolazione);
            if triangle(coda(i_coda))==-1
                t1=inf;
                t2=inf;
                t3=inf;
            end
            curvilinee=CC(curvilinee,t1,t2,t3);

            if prolungo>0
                triangle(prolungo)=0.5;
            end

            %calcola l'intersezione con i triangoli d'interesse nel
            %vettore coda

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

                coda=Coda(coda,triangle,coda(i_coda)); %aggiorna la coda
                %coda(i_coda)=-1;
                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
```

intersezione.m

```
function  [alfa,P,tT,t1,t2]  = intersezione( XE1,YE1,XE2,YE2,i_traccia )

global punti tracce
P=0;
tT=inf;
t1=inf;
t2=inf;
TE1=[XE2-XE1 YE2-YE1]; %vettore tangente il lato


XT1=punti(tracce(i_traccia,2),2); %ascissa primo nodo della traccia
XT2=punti(tracce(i_traccia,3),2); %ascissa secondo nodo della traccia

YT1=punti(tracce(i_traccia,2),3); %ordinata primo nodo della traccia
YT2=punti(tracce(i_traccia,3),3); %ordinata sedcondo nodo della traccia
```

```
TT=[XT2-XT1 YT2-YT1]; %vettore tangente la traccia

alfa=0;


if norm(cross([TE1 0],[TT 0]))<=eps %vettori paralleli
    if XE1==XE2 % lato parallelo asse x
        if XT1==XE1 %lato e traccia sono coincidenti

            t1=(YT1-YE1)/(YE2-YE1);
            t2=(YT2-YE1)/(YE2-YE1);
            alfa=-2;

        end
    end

    if YE1==YE2 % lato parallelo asse y
        if YT1==YE1 %lato e traccia sono coincidenti

            t1=(XT1-XE1)/(XE2-XE1);
            t2=(XT2-XE1)/(XE2-XE1);
            alfa=-2;

        end
    end

    if abs((YT1-YE1)/(YE2-YE1)-(XT1-XE1)/(XE2-XE1))<=eps %lato e traccia coinci-
denti nel caso generale

        t1=(XT1-XE1)/(XE2-XE1);
        t2=(XT2-XE1)/(XE2-XE1);
        alfa=-2;

    end

    if (alfa==-2 && ((t1>0 && t1<1) || (t2>0 && t2<1))) % la traccia ricopre
solo parzialemente il lato, dunque vi è intersezione

        alfa=2;
        %potrebbe essere importante sapere quale (o quali) sono i punti di
        %intersezione, ovvero dove termina la traccia

    end

end


if norm(cross([TE1 0],[TT 0]))>0 %esclude i vettori paralleli

    if (XT1==XT2 && (XE1-XT1)*(XE2-XT1)<=0) %posta la traccia parallela l'asse
x, esclude gli "estremi concordi"

        P=[XT1 YE1+(YE2-YE1)*(XT1-XE1)/(XE2-XE1)]'; %punto di intersezione tra
la
                                                    %retta tengente il lato e
quella tangente la traccia

        alfa=0.2;
    end
```

```
    if (YT1==YT2 && (YE1-YT1)*(YE2-YT1)<=0) %posta la traccia parallela l'asse
y, esclude gli "estremi concordi"

        P=[XE1+(XE2-XE1)*(YT1-YE1)/(YE2-YE1) YT1]'; %punto di intersezione tra
la retta
                                                    %tengente il lato e quella
tangente la traccia
        alfa=0.2;

    end

    if (XT1~=XT2 && YT1~=YT2 && ((YE1-YT1)/(YT2-YT1)-(XE1-XT1)/(XT2-XT1))*((YE2-
YT1)/(YT2-YT1)-(XE2-XT1)/(XT2-XT1))<=0) %posta la traccia non parallela gli assi
esclude gli estremi concordi

        %distinguo tre casi: lato parallelo asse x, lato parallelo
        %asse y e lato non parallelo gli assi

        if XE1==XE2
            P=[XE1 YT1+(XE1-XT1)*(YT2-YT1)/(XT2-XT1)]';
            alfa=0.2;
        end

        if YE1==YE2
            P=[XT1+(YE1-YT1)*(XT2-XT1)/(YT2-YT1) YE2]';
            alfa=0.2;
        end

        %imposto il sistema per l'intersezione tra le due rette(quella
        %tangente il lato e quella tangente la traccia)
       if (XE1~=XE2 && YE1~=YE2)

           A=[-1/(XE2-XE1) 1/(YE2-YE1); -1/(XT2-XT1) 1/(YT2-YT1)]; %matrice per
risolvere il sistema
           b=[YE1/(YE2-YE1)-XE1/(XE2-XE1); YT1/(YT2-YT1)- XT1/(XT2-XT1)]; %vet-
tore per la risoluzione
           P=A\b; %trovo il punto di intersezione
           alfa=0.2;
       end
    end
end

if alfa==0.2 %ho trovato una possibile intersezione tra traccia e lato

    tE=(P(1)-XE1)/(XE2-XE1); %tE è il valore per cui P appartiene alla retta pa-
rallela al lato
    tT=(P(1)-XT1)/(XT2-XT1); %tt è il valore per cui P appartiene alla retta pa-
rallela alla traccia

    if isnan(tE)
        tE=(P(2)-YE1)/(YE2-YE1);
    end

    if isnan(tT)
        tT=(P(2)-YT1)/(YT2-YT1);
    end

end
```

```
%il valore tE è sempre compreso tra 0 ed 1, poiché si escludono le
%intersezione in cui tale valore potrebbe essere minore di 0 o maggiore di 1

if (alfa==0.2 && tT>=0 && tT<=1 && tE>0 && tE<1) % vi è intersezione tra il lato
e la traccia in un punto P generico esclusi i vertici

    alfa=1;

end

if (alfa==0.2 && tT>=0 && tT<=1 && (tE==0 || tE==1)) %vi è intersezione tra un
vertice del lato e la traccia

    alfa=0.5;

end


if (alfa==1 && (tT==0 || tT==1)) %l'intersezione avviene tra punto generico del
triangolo e tra un vertice della traccia

    alfa=1.25;

end

if (alfa==0.5 && (tT==0 || tT==1)) %l'intersezione avviene tra un vertice del
triangolo e tra un vertice della traccia

    alfa=0.6;

end

if alfa<0.2 %non vi è né intersezione né possibile intersezione tra lato e trac-
cia

    P=0;

end
```

intersezionetrian.m

```
function [ beta,sottopoligoni3,sottopoligoni4,triangolazione,tT_1,tT_2,tT_3,pro-
lungo ] = intersezione_trian( i_trian,i_traccia,sottopoligoni3,sottopoli-
goni4,triangolazione)

global ele node neigh punti tracce

s = zeros(3,2);
s(1,:) = node(ele(i_trian, 2), 2:3);
s(2,:) = node(ele(i_trian, 3), 2:3);
s(3,:) = node(ele(i_trian, 4), 2:3);
prolungo=0;
```

```
[a1,P1,tT_1,tP1,tP2] = intersezione(s(1,1), s(1,2), s(2,1), s(2,2), i_traccia);
%calcola il valore a1 della possibile intersezione col primo lato

if a1~=2 %se la traccia e il primo lato non sono paralleli

    [a2,P2,tT_2,tP1,tP2] = intersezione(s(2,1), s(2,2), s(3,1), s(3,2),i_trac-
cia); %%calcola il valore a2 possibile intersezione col secondo lato

    if a2~=2    %se la traccia non è parallela al secondo lato

       [a3,P3,tT_3,tP1,tP2] = intersezione(s(3,1), s(3,2), s(1,1),
s(1,2),i_traccia); %calcola il valore a3 della possibile interesezione col terzo
lato

    else
        a3=0;

    end

else
    a2=0;
    a3=0;

end

%considero prima i casi di intersezione "classica": punto-punto o
%punto-vertice

beta=-1; %che contrassegna una non intersezione

if ((a1==1||a1==1.25) && (a2==1||a2==1.25) && a3==0) %l'intersezione avviene tra
un punto del primo lato ed uno del secondo

    beta=1;

    lun4=size((sottopoligoni4),1);
    lun3=size((sottopoligoni3),1);
    lunsot=size((triangolazione),1);

    sottopoli-
goni4(lun4+1,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3),P2',P1'];
    sottopoligoni3(lun3+1,:)=[P1',P2',node(ele(i_trian,3),2:3)];

    triangolazione(lunsot+1,:)=[node(ele(i_trian,2),2:3),P1',P2'];
    triangolazione(lun-
sot+2,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3),P2'];
    return

end

if ((a1==1||a1==1.25) && a2==0 && (a3==1||a3==1.25)) %l'intersezione avviene tra
un punto del primo lato ed uno del terzo

    beta=1;

    lun4=size((sottopoligoni4),1);
    lun3=size((sottopoligoni3),1);
    lunsot=size((triangolazione),1);
```

```
    sottopoli-
goni4(lun4+1,:)=[node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3),P3',P1'];
    sottopoligoni3(lun3+1,:)=[P1',P3',node(ele(i_trian,2),2:3)];

    triangolazione(lunsot+1,:)=[node(ele(i_trian,3),2:3),P1',P3'];
    triangolazione(lun-
sot+2,:)=[node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3),P3'];
    return

end

if (a1==0 && (a2==1||a2==1.25) && (a3==1||a3==1.25)) %l'intersezione avviene tra
un punto del secondo lato ed uno del terzo

    beta=1;

    lun4=size((sottopoligoni4),1);
    lun3=size((sottopoligoni3),1);
    lunsot=size((triangolazione),1);

    sottopoli-
goni4(lun4+1,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),P2',P3'];
    sottopoligoni3(lun3+1,:)=[P2',P3',node(ele(i_trian,4),2:3)];

    triangolazione(lunsot+1,:)=[node(ele(i_trian,2),2:3),P2',P3'];
    triangolazione(lun-
sot+2,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),P2'];
    return

end


if((a1==0.5||a1==0.6) && (a2==1||a2==1.25) && (a3==0.5||a3==0.6)) %intersezione
con secondo lato e secondo vertice

    beta=1;

    lun3=size((sottopoligoni3),1);
    sottopoli-
goni3(lun3+1,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),P2'];
    sottopoli-
goni3(lun3+2,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3),P2'];
    return

end

if((a1==1||a1==1.25) && (a2==0.5||a2==0.6) && (a3==0.5||a3==0.6)) %intersezione
con primo lato e primo vertice

    beta=1;

    lun3=size((sottopoligoni3),1);
    sottopoli-
goni3(lun3+1,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3),P1'];
    sottopoli-
goni3(lun3+2,:)=[node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3),P1'];
    return

end
```

```
if((a1==0.5||a1==0.6) && (a2==0.5||a2==0.6) && (a3==1||a3==1.25)) %intersezione
con terzo lato e terzo vertice

    beta=1;

    lun3=size((sottopoligoni3),1);
    sottopoli-
goni3(lun3+1,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),P3'];
    sottopoli-
goni3(lun3+2,:)=[node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3),P3'];
    return

end


%studio i casi in cui vi è intersezione tra la traccia ed il triangolo, ma
%un estremo rimane all'interno del triangolo e bisogna prolungare la
%traccia


if (a1==1 || (a1==1.25 && ((tT_1==0 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,3),2:3))==1) ...
                          || (tT_1==1 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,2),2:3))==1))))  %vi è intersezione tra un punto della
traccia e il primo lato

    if (a2==0.2 && a3==0) %il possibile punto d'intersezione avviene col secondo
lato

        beta=1;
        lunsot=size((triangolazione),1);

        %creo la sottotriangolazione interna

        triangolazione(lunsot+1,:)=[P1',P2',node(ele(i_trian,2),2:3)];
        triangolazione(lunsot+2,:)=[P1',P2',node(ele(i_trian,3),2:3)];
        triangolazione(lun-
sot+3,:)=[P2',node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3)];

        %inoltre devo andare anche a sottotriangolizzare il triangolo
        %adiacente al punto di intersezione per mantenere la
        %sottotriangolarizzazione conforme

       if neigh(i_trian,2)==-1

           return

       else

           prolungo=neigh(i_trian,2);
           for j=2:4

               if i_trian==neigh(neigh(i_trian,2),j)

                   lunsot=size((triangolazione),1);

                   triangolazione(lun-
sot+1,:)=[node(ele(neigh(i_trian,2),j),:),node(ele(i_trian,3),2:3),P2'];
```

```
                    triangolazione(lun-
sot+2,:)=[node(ele(neigh(i_trian,2),j),:),node(ele(i_trian,4),2:3),P2'];

                    return
                end
            end
        end


    end

    if (a2==0 && a3==0.2) %il possibile punto d'intersezione avviene col terzo
lato

        beta=1;
        lunsot=size((triangolazione),1);

        %creo la sottotriangolazione interna

        triangolazione(lunsot+1,:)=[P1',P3',node(ele(i_trian,2),2:3)];
        triangolazione(lunsot+2,:)=[P1',P3',node(ele(i_trian,3),2:3)];
        triangolazione(lun-
sot+3,:)=[P3',node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)];

        %inoltre devo andare anche a sottotriangolizzare il triangolo
        %adiacente al punto di intersezione per mantenere la
        %sottotriangolarizzazione conforme

       if neigh(i_trian,3)==-1

           return

       else

           prolungo=neigh(i_trian,3);
           for j=2:4

               if i_trian==neigh(neigh(i_trian,3),j)

                   lunsot=size((triangolazione),1);

                   triangolazione(lun-
sot+1,:)=[node(ele(neigh(i_trian,3),j),:),node(ele(i_trian,2),2:3),P3'];
                   triangolazione(lun-
sot+2,:)=[node(ele(neigh(i_trian,3),j),:),node(ele(i_trian,4),2:3),P3'];

                   return
               end
           end
       end
    end

    if (a2==0.2 && a3==0.2) %il possibile punto d'intersezione avviene col terzo
vertice

        beta=1;
        lunsot=size((triangolazione),1);

        triangolazione(lun-
sot+1,:)=[P1',node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3)];
```

```
        triangolazione(lun-
sot+2,:)=[P1',node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)];

        return

    end
end


if (a2==1 || (a2==1.25 && ((tT_2==0 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,3),2:3))==1) ...
                        || (tT_2==1 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,2),2:3))==1)))) %vi è intersezione tra un punto generico
della traccia e il secondo lato
    if (a1==0.2 && a3==0) %il possibile punto d'intersezione avviene col primo
lato

        beta=1;
        lunsot=size((triangolazione),1);

        %creo la sottotriangolazione interna

        triangolazione(lunsot+1,:)=[P1',P2',node(ele(i_trian,2),2:3)];
        triangolazione(lunsot+2,:)=[P1',P2',node(ele(i_trian,3),2:3)];
        triangolazione(lun-
sot+3,:)=[P2',node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3)];

        %inoltre devo andare anche a sottotriangolare il triangolo
        %adiacente al punto di intersezione per mantenere la
        %sottotriangolarizzazione conforme

       if neigh(i_trian,4)==-1

           return

       else

           prolungo=neigh(i_trian,4);
           for j=2:4

               if i_trian==neigh(neigh(i_trian,4),j)

                   lunsot=size((triangolazione),1);

                   triangolazione(lun-
sot+1,:)=[node(ele(neigh(i_trian,4),j),:),node(ele(i_trian,2),2:3),P1'];
                   triangolazione(lun-
sot+2,:)=[node(ele(neigh(i_trian,4),j),:),node(ele(i_trian,3),2:3),P1'];

                   return
               end
           end
       end
    end

    if (a1==0 && a3==0.2) %il possibile punto d'intersezione avviene col terzo
lato
```

```
        beta=1;
        lunsot=size((triangolazione),1);

        %creo la sottotriangolazione interna

        triangolazione(lunsot+1,:)=[P2',P3',node(ele(i_trian,2),2:3)];
        triangolazione(lunsot+2,:)=[P2',P3',node(ele(i_trian,4),2:3)];
        triangolazione(lun-
sot+3,:)=[P2',node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3)];

        %inoltre devo andare anche a sottotriangolare il triangolo
        %adiacente al punto di intersezione per mantenere la
        %sottotriangolazazione conforme

        if neigh(i_trian,3)==-1

            return

        else

            prolungo=neigh(i_trian,3);
            for j=2:4

                if i_trian==neigh(neigh(i_trian,3),j)

                    lunsot=size((triangolazione),1);

                    triangolazione(lun-
sot+1,:)=[node(ele(neigh(i_trian,3),j),:),node(ele(i_trian,2),2:3),P3'];
                    triangolazione(lun-
sot+2,:)=[node(ele(neigh(i_trian,3),j),:),node(ele(i_trian,4),2:3),P3'];

                    return
                end
            end
        end
     end

    if (a1==0.2 && a3==0.2) %il possibile punto d'intersezione avviene col primo
vertice

        beta=1;
        lunsot=size((triangolazione),1);

        triangolazione(lun-
sot+1,:)=[P2',node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3)];
        triangolazione(lun-
sot+2,:)=[P2',node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3)];

        return

    end
end


if (a3==1 || (a3==1.25 && ((tT_3==0 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,3),2:3))==1) ...
                        || (tT_3==1 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
```

```
,punti(tracce(i_traccia,2),2:3))==1)))) %vi è intersezione tra un punto generico
della traccia e il terzo lato
    if (a1==0.2 && a2==0) %il possibile punto d'intersezione avviene col primo
lato

        beta=1;
        lunsot=size((triangolazione),1);

        %creo la sottotriangolazione interna

        triangolazione(lunsot+1,:)=[P1',P3',node(ele(i_trian,2),2:3)];
        triangolazione(lunsot+2,:)=[P1',P3',node(ele(i_trian,3),2:3)];
        triangolazione(lun-
sot+3,:)=[P3',node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)];

        %inoltre devo andare anche a sottotriangolizzare il triangolo
        %adiacente al punto di intersezione per mantenere la
        %sottotriangolarizzazione conforme

       if neigh(i_trian,4)==-1

           return

       else

           prolungo=neigh(i_trian,4);
           for j=2:4

               if i_trian==neigh(neigh(i_trian,4),j)

                   lunsot=size((triangolazione),1);

                   triangolazione(lun-
sot+1,:)=[node(ele(neigh(i_trian,4),j),:),node(ele(i_trian,2),2:3),P1'];
                   triangolazione(lun-
sot+2,:)=[node(ele(neigh(i_trian,4),j),:),node(ele(i_trian,3),2:3),P1'];

                   return
               end
           end
       end

    end

    if (a1==0 && a2==0.2) %il possibile punto d'intersezione avviene col secondo
lato

        beta=1;
        lunsot=size((triangolazione),1);

        %creo la sottotriangolazione interna

        triangolazione(lunsot+1,:)=[P2',P3',node(ele(i_trian,2),2:3)];
        triangolazione(lunsot+2,:)=[P2',P3',node(ele(i_trian,4),2:3)];
        triangolazione(lun-
sot+3,:)=[P2',node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3)];

        %inoltre devo andare anche a sottotriangolizzare il triangolo
        %adiacente al punto di intersezione per mantenere la
        %sottotriangolarizzazione conforme
```

```
      if neigh(i_trian,2)==-1

          return

      else

           prolungo=neigh(i_trian,2);
           for j=2:4

               if i_trian==neigh(neigh(i_trian,2),j)

                   lunsot=size((triangolazione),1);

                   triangolazione(lun-
sot+1,:)=[node(ele(neigh(i_trian,2),j),:),node(ele(i_trian,3),2:3),P2'];
                   triangolazione(lun-
sot+2,:)=[node(ele(neigh(i_trian,2),j),:),node(ele(i_trian,4),2:3),P2'];

                   return
               end
           end
       end

    end

    if (a1==0.2 && a2==0.2) %il possibile punto d'intersezione avviene col se-
condo vertice

        beta=1;
        lunsot=size((triangolazione),1);

        triangolazione(lun-
sot+1,:)=[P3',node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3)];
        triangolazione(lun-
sot+2,:)=[P3',node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)];

        return

    end
end



if ((a1==0.5 && a2==0.5 && a3==0.2) || ((a1==0.6 && a2==0.6 &&
a3==0.2)&&((tT_1==0 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,3),2:3))==1) ...
                                                                    ||
(tT_1==1 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,2),2:3))==1)))) %intersezione tra la traccia ed il se-
condo vertice + possibile intersezione con il terzo lato, il che equivale a dire
che la traccia è interna al triangolo

    beta=1;
    lunsot=size((triangolazione),1);

    triangolazione(lun-
sot+1,:)=[node(ele(i_trian,3),2:3),node(ele(i_trian,2),2:3),P3'];
```

```
    triangolazione(lun-
sot+2,:)=[node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3),P3'];

    %inoltre devo andare anche a sottotriangolizzare il triangolo
    %adiacente al punto di intersezione per mantenere la
    %sottotriangolarizzazione conforme

    if neigh(i_trian,3)==-1

        return

    else

        prolungo=neigh(i_trian,3);
        for j=2:4

            if i_trian==neigh(neigh(i_trian,3),j)

                lunsot=size((triangolazione),1);

                triangolazione(lun-
sot+1,:)=[node(ele(neigh(i_trian,3),j),:),node(ele(i_trian,2),2:3),P3'];
                triangolazione(lun-
sot+2,:)=[node(ele(neigh(i_trian,3),j),:),node(ele(i_trian,4),2:3),P3'];

                return
            end
        end
    end

end

if ((a1==0.5 && a2==0.2 && a3==0.5) || ((a1==0.6 && a2==0.2 &&
a3==0.6)&&((tT_1==0 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,3),2:3))==1) ...
                                                                    ||
(tT_1==1 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,2),2:3))==1)))) %intersezione tra la traccia ed il primo
vertice + possibile intersezione con il secondo lato, il che equivale a dire che
la traccia è interna al triangolo

    beta=1;
    lunsot=size((triangolazione),1);

    triangolazione(lun-
sot+1,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),P2'];
    triangolazione(lun-
sot+2,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3),P2'];

    %inoltre devo andare anche a sottotriangolizzare il triangolo
    %adiacente al punto di intersezione per mantenere la
    %sottotriangolarizzazione conforme

    if neigh(i_trian,2)==-1

          return

    else
```

```
            prolungo=neigh(i_trian,2);
            for j=2:4

                if i_trian==neigh(neigh(i_trian,2),j)

                    lunsot=size((triangolazione),1);

                    triangolazione(lun-
sot+1,:)=[node(ele(neigh(i_trian,2),j),:),node(ele(i_trian,3),2:3),P2'];
                    triangolazione(lun-
sot+2,:)=[node(ele(neigh(i_trian,2),j),:),node(ele(i_trian,4),2:3),P2'];

                    return
                end
            end
    end

end

if ((a1==0.2 && a2==0.5 && a3==0.5) || ((a1==0.2 && a2==0.6 &&
a3==0.6)&&((tT_2==0 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,3),2:3))==1) ...
                                                                    ||
(tT_2==1 && in-
terno(node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)
,punti(tracce(i_traccia,2),2:3))==1)))) %intersezione tra la traccia ed il terzo
vertice + possibile intersezione con il primo lato, il che equivale a dire che
la traccia è interna al triangolo

    beta=1;
    lunsot=size((triangolazione),1);

    triangolazione(lun-
sot+1,:)=[node(ele(i_trian,4),2:3),node(ele(i_trian,2),2:3),P1'];
    triangolazione(lun-
sot+2,:)=[node(ele(i_trian,4),2:3),node(ele(i_trian,3),2:3),P1'];

    %inoltre devo andare anche a sottotriangolizzare il triangolo
    %adiacente al punto di intersezione per mantenere la
    %sottotriangolarizzazione conforme

    if neigh(i_trian,4)==-1

            return

    else

            prolungo=neigh(i_trian,4);
            for j=2:4

                if i_trian==neigh(neigh(i_trian,4),j)

                    lunsot=size((triangolazione),1);

                    triangolazione(lun-
sot+1,:)=[node(ele(neigh(i_trian,4),j),:),node(ele(i_trian,2),2:3),P1'];
                    triangolazione(lun-
sot+2,:)=[node(ele(neigh(i_trian,4),j),:),node(ele(i_trian,3),2:3),P1'];
```

```
                    return
                end
            end
    end

end



if a1==1.25 %intersezione tra estremo traccia e punto generico triangolo, in cui
la traccia è esterna, poiché ho già esauirito i casi in cui la traccia è interna
            %o in cui vi sia un intersezione classica


            beta=1;
            lunsot=size((triangolazione),1);

             triangolazione(lun-
sot+1,:)=[node(ele(i_trian,4),2:3),node(ele(i_trian,2),2:3),P1'];
             triangolazione(lun-
sot+2,:)=[node(ele(i_trian,4),2:3),node(ele(i_trian,3),2:3),P1'];

            return
end


if a2==1.25 %intersezione tra estremo traccia e punto generico triangolo, in cui
la traccia è esterna, poiché ho già esauirito i casi in cui la traccia è interna
            %o in cui vi sia un intersezione classica


            beta=1;
            lunsot=size((triangolazione),1);

            triangolazione(lun-
sot+1,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3),P2'];
            triangolazione(lun-
sot+2,:)=[node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3),P2'];

            return
end


if a3==1.25 %intersezione tra estremo traccia e punto generico triangolo, in cui
la traccia è esterna, poiché ho già esauirito i casi in cui la traccia è interna
            %o in cui vi sia un intersezione classica


            beta=1;
            lunsot=size((triangolazione),1);

            triangolazione(lun-
sot+1,:)=[node(ele(i_trian,3),2:3),node(ele(i_trian,2),2:3),P3'];
            triangolazione(lun-
sot+2,:)=[node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3),P3'];

            return
```

```
end

%bisogna studiare i casi paralleli

if a1==2 %la traccia è parallela al primo lato e vi è intersezione

    beta=1;

    %analizzo i casi in cui solo un estremo della traccia è interno al lato

    if ((tP1>0 && tP1<1)&&(tP2<=0 || tP2>=1)) %il primo estremo della traccia è
interno al lato, mentre il secondo no

        lunsot=size((triangolazione),1);
        tT_1=0;
        triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3)];
        triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)];

        return

    end

    if ((tP1<=0 || tP1>=1)&&(tP2>0 && tP2<1)) %il secondo estremo della traccia
è interno al lato, mentre il primo no

        lunsot=size((triangolazione),1);
        tT_2=0;
        triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3)];
        triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)];

        return

    end

    %adesso analizzo invece i casi in cui entrambi gli estremi sono interni
    %alla traccia

    if ((tP1>0 && tP1<1)&&(tP2>0 && tP2<1))

        if tP1<tP2

            lunsot=size((triangolazione),1);
            tT_1=0;
            tT_2=0;
            triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3)];
            triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)];
            triangolazione(lunsot+3,:)=[punti(tracce(i_trac-
cia,2),2:3),punti(tracce(i_traccia,3),2:3),node(ele(i_trian,4),2:3)];

            return

        else

            lunsot=size((triangolazione),1);
```

```
            triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,2),2:3),node(ele(i_trian,4),2:3)];
            triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,4),2:3)];
            triangolazione(lunsot+3,:)=[punti(tracce(i_trac-
cia,2),2:3),punti(tracce(i_traccia,3),2:3),node(ele(i_trian,4),2:3)];

            return

        end
    end


end

if a2==2 %la traccia è parallela al secondo lato e vi è intersezione

    beta=1;

    if ((tP1>0 && tP1<1)&&(tP2<=0 || tP2>=1)) %il primo estremo della traccia è
interno al lato, mentre il secondo no

        lunsot=size((triangolazione),1);
        tT_1=0;
        triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,2),2:3)];
        triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,4),2:3),node(ele(i_trian,2),2:3)];

        return

    end

    if ((tP1<=0 || tP1>=1)&&(tP2>0 && tP2<1)) %il secondo estremo della traccia
è interno al lato, mentre il primo no

        lunsot=size((triangolazione),1);
        tT_2=0;
        triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,2),2:3)];
        triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,4),2:3),node(ele(i_trian,2),2:3)];

        return

    end

    %adesso analizzo invece i casi in cui entrambi gli estremi sono interni
    %alla traccia

    if ((tP1>0 && tP1<1)&&(tP2>0 && tP2<1))

        if tP1<tP2

            lunsot=size((triangolazione),1);
            tT_1=0;
            tT_2=0;
            triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,2),2:3)];
```

```
            triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,4),2:3),node(ele(i_trian,2),2:3)];
            triangolazione(lunsot+3,:)=[punti(tracce(i_trac-
cia,2),2:3),punti(tracce(i_traccia,3),2:3),node(ele(i_trian,2),2:3)];

            return

        else

            lunsot=size((triangolazione),1);

            triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,3),2:3),node(ele(i_trian,2),2:3)];
            triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,4),2:3),node(ele(i_trian,2),2:3)];
            triangolazione(lunsot+3,:)=[punti(tracce(i_trac-
cia,2),2:3),punti(tracce(i_traccia,3),2:3),node(ele(i_trian,2),2:3)];

            return

        end
    end

end

if a3==2 %la traccia è parallela al terzo lato e vi è intersezione

    beta=1;

    if ((tP1>0 && tP1<1)&&(tP2<=0 || tP2>=1)) %il primo estremo della traccia è
interno al lato, mentre il secondo no

        lunsot=size((triangolazione),1);
        tT_1=0;
        triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3)];
        triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,4),2:3),node(ele(i_trian,3),2:3)];

        return

    end

    if ((tP1<=0 || tP1>=1)&&(tP2>0 && tP2<1)) %il secondo estremo della traccia
è interno al lato, mentre il primo no

        lunsot=size((triangolazione),1);
        tT_2=0;
        triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3)];
        triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,4),2:3),node(ele(i_trian,3),2:3)];

        return

    end

    %adesso analizzo invece i casi in cui entrambi gli estremi sono interni
    %alla traccia
```

```
    if ((tP1>0 && tP1<1)&&(tP2>0 && tP2<1))

        if tP1<tP2

            lunsot=size((triangolazione),1);
            tT_2=0;
            triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,4),2:3),node(ele(i_trian,3),2:3)];
            triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3)];
            triangolazione(lunsot+3,:)=[punti(tracce(i_trac-
cia,2),2:3),punti(tracce(i_traccia,3),2:3),node(ele(i_trian,3),2:3)];

            return

        else

            lunsot=size((triangolazione),1);
            tT_1=0;
            tT_2=0;
            triangolazione(lunsot+1,:)=[punti(tracce(i_trac-
cia,3),2:3),node(ele(i_trian,4),2:3),node(ele(i_trian,3),2:3)];
            triangolazione(lunsot+2,:)=[punti(tracce(i_trac-
cia,2),2:3),node(ele(i_trian,2),2:3),node(ele(i_trian,3),2:3)];
            triangolazione(lunsot+3,:)=[punti(tracce(i_trac-
cia,2),2:3),punti(tracce(i_traccia,3),2:3),node(ele(i_trian,3),2:3)];

            return

        end
    end

end
```

Coda.m

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

global neigh

for j=2:4
    flag=0;
    x=length(coda);
     if neigh(i,j) ~= -1
         if triangle(neigh(i,j)) == 0 %il triangolo non è stato già controllato

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

```
        end
    end
end
```

interno.m

```
function [ int ] = interno( N1,N2,N3,P )

int=0;
A=[N2'-N1' N3'-N1'];
y=A\(P'-N1');
if (y(1)>=-eps && y(1)<=1+eps && y(2)>=-eps && y(2)<=1+eps)
    if ((1-y(1)-y(2)>=-eps)&&(1-y(1)-y(2)<=1+eps))
        int=1;
    end
end

end
```

CC.m

```
function [ curvilinee ] = CC( curvilinee,t1,t2,t3 )

if t1~=inf
    x=length(curvilinee);
    flag=0;
    for z=x:-1:1
        if t1==curvilinee(z)
            flag=1;
        end
    end
    if flag==0
        curvilinee(x+1)=t1;
    end
end

if t2~=inf
    x=length(curvilinee);
    flag=0;
    for z=x:-1:1
        if t2==curvilinee(z)
            flag=1;
        end
    end
    if flag==0
        curvilinee(x+1)=t2;
    end
end
```

```
if t3~=inf
    x=length(curvilinee);
    flag=0;
    for z=x:-1:1
        if t3==curvilinee(z)
            flag=1;
        end
    end
    if flag==0
        curvilinee(x+1)=t3;
    end
end

curvilinee=sort(curvilinee);

end
```

near.m

```
% vicini

function tocheck = near(triangle)

    global edge ele neigh node punti tracce
    global n_edge n_ele n_neigh n_node n_punti n_tracce

        tocheck = [];

    for i = 1:n_ele
        if triangle(i) == 1
                for j = 2:4
                    if neigh(i,j) ~= -1
                        if triangle(neigh(i,j)) == -1
                            x = length(tocheck);
                            tocheck(x+1) = neigh(i,j);

                        end
                    end
                end
        end
    end
end
```

vicinilato.m

```
% triangoli che condividono lato con triangoli tagliati

function [lati,tagl] = vicinilato(triangle)
global n_ele neigh ele

tagl=[];
lati=[];
k=1;
for i=1:n_ele
```

```
    if triangle(i)==1
        for j=2:4
            m=1;
            if neigh(i,j)~=-1
                if triangle(neigh(i,j))~=1
                    tagl(k)=(neigh(i,j));
                    for l=2:4
                        if l~=j
                            lati(k,m)=ele(i,l);
                            m=m+1;
                        end
                    end
                    k=k+1;
                end
            end
        end
    end
end
```

vicinivertice.m

```
% triangoli che condividono lato con triangoli tagliati

function [vert,tagv] = vicinivertice(triangle,tagl,vertcom)
global n_ele ele

tagv=[];
vert=[];
k=0;

for i=1:n_ele
    if triangle(i)==1
       for j=2:4
           x=1;
           while (x<(size(vertcom,1)+1) && vertcom(x,ele(i,j)) ~= 0)
               flag=0;
               if triangle(vertcom(x,ele(i,j))) ~= 1

                    for z=k:-1:1
                        if tagv(z)==vertcom(x,ele(i,j))
                             flag=1;
                             break
                        end
                    end
                  if flag==0

                      for z=1:length(tagl)
                          if vertcom(x,ele(i,j))==tagl(z) %il triangolo ha un
lato in comune
                               flag=1;
                          end
                      end

                      if flag==0
```

```
                        k=k+1;
                        tagv(k)=vertcom(x,ele(i,j));
                        vert(k)=ele(i,j);
                    end
                end



            end
            x=x+1;
        end

    end
  end
end
```