POLITECNICO
DI TORINO

Dipartimento
di Scienze Matematiche

Programmazione e Calcolo scientifico
Prof. Stefano Berrone, Francesco Della Santa

Angelo Battaglia, Roberto Marchello, Lucio Moro

Anno Accademico 2018/2019

Indice

1 Introduzione

Gli obiettivi del progetto sono creare un network di fratture, ovvero un insieme di poligoni contenuti in un dominio di $\mathbb{R}^3$, e trovare i segmenti generati dalle intersezioni tra essi. Per fare ciò sono state create tre classi: Fracture, Trace, DFN.
Moduli e pacchetti utilizzati:

- matlab_clones.py
- distributions.py
- numpy
- matplotlib
- plotly
- dill

2 Fracture

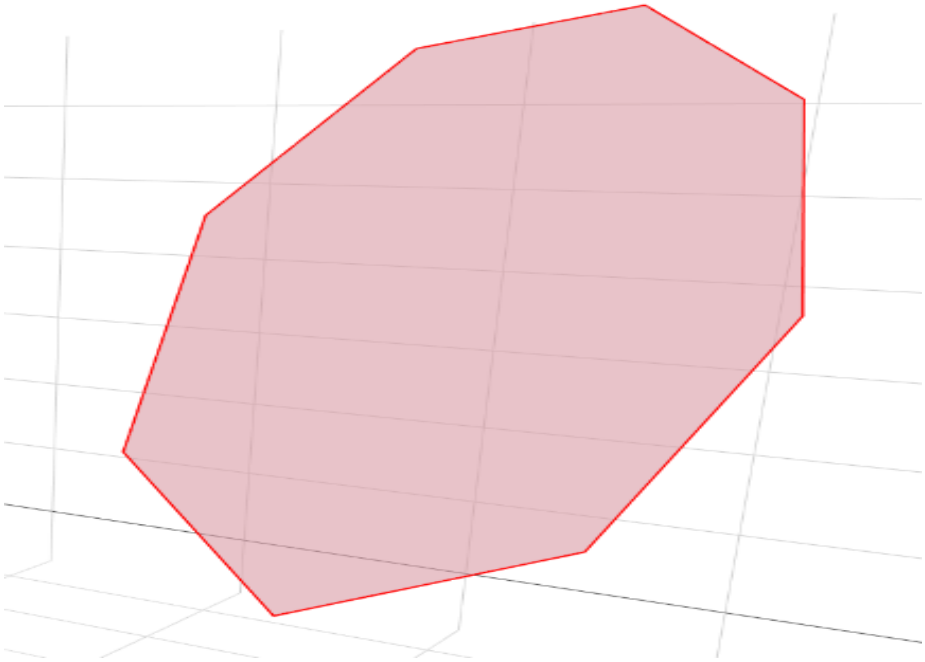
La classe Fracture descrive le fratture del network. E' di tipo deterministico, ovvero per costruirne un'istanza riceve come parametri il numero di vertici, baricentro, lunghezza del semiasse delle ascisse dell'ellisse in cui è inscritta, rapporto tra semiasse delle ascisse e semiasse delle ordinate dell'ellisse, versore normale e angolo di prima rotazione attorno l'asse z. La frattura viene inizialmente generata sul piano xy centrata nell'origine, successivamente viene effettuata la prima rotazione attorno l'asse z, le dovute rotazioni per far coincidere il versore normale e infine una traslazione per far coincidere il baricentro. Ciascuna frattura ha un suo bounding box, ovvero il più piccolo parallelepipedo che lo contiene.

2.1 Strutture dati

Le strutture dati presenti sono:

vertici
Matrice contenente i vertici della frattura come vettori colonna

xmin, xmax, ymin, ymax, zmin e zmax
Attributi che rappresentano gli estremi del bounding box di una frattura



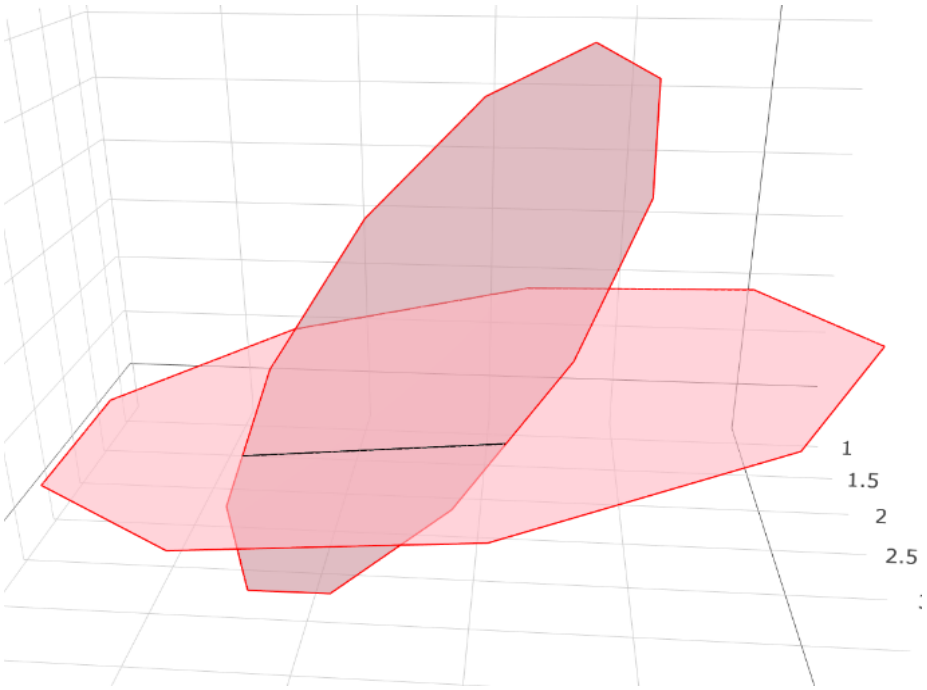
3 Trace

La classe Trace descrive le tracce, ovvero i segmenti generati dall'intersezione tra fratture. Ogni traccia ha come attributi i poligoni che la generano (oggetti della classe Fracture), i loro indici all'interno del DFN e le coordinate degli estremi inseriti in una matrice 3x2.

3.1 Metodo

Il metodo implementato è:

direzione
Restituisce la direzione della retta su cui giace la traccia.



4 DiscreteFractureNetwork

La classe Discrete Fracture Network (DFN) descrive il network di fratture. Ha natura probabilistica, ovvero per crearne un'istanza bisogna passare come parametri le informazioni sul numero iniziale di fratture N, sul dominio in $\mathbb{R}^3$ in cui sono contenuti i baricentri delle stesse e sui vari parametri caratterizzanti le distribuzioni necessarie a generare le fratture.
Due parametri opzionali possono essere inseriti:

- **tol**: tolleranza con cui vengono effettuati i calcoli, necessaria in ottica dell'aritmetica finita del calcolatore. Se non specificata, viene usata una tolleranza standard pari a 1.0e-10.
- **fixed_n_edges**: se inserito, indica il numero di vertici (fisso e >=8) delle fratture. Altrimenti le fratture sono generato con un numero casuale di vertici tra 8 a 16.

Nel momento dell'istanziazione vengono create in particolare le distribuzioni PowerLawBounded per i semiassi maggiori dei poligoni e Von Mises-Fisher per i versori normali; successivamente vengono create tutte le strutture dati (4.1) come liste vuote e infine viene chiamato il metodo genfrac (4.2.5), di cui parleremo nello specifico in seguito.

4.1 Strutture dati

Le strutture dati utilizzate sono:

fractures
Lista degli oggetti di tipo Fracture del DFN

poss_intersezioni
Lista di liste dove l'i-esima lista, corrispondente all'i-esimo poligono, contiene tutti gli indici dei poligoni con cui questo potrebbe intersecarsi

intersezioni
Lista di liste dove l'i-esima lista, corrispondente all'i-esimo poligono, contiene tutti gli indici dei poligoni con cui questo si interseca realmente

traces
Lista degli oggetti di tipo Trace del DFN

frac_traces
Lista di liste dove l'i-esima lista, corrispondente all'i-esimo poligono, contiene tutte le tracce generate da esso

4.2 Metodi

Di seguito sono illustrati i metodi che gestiscono il network.

4.2.1 inters_BB

(Codice → DFN.py → linea 163)
Controlla se i bounding box di due fratture si intersecano. Prende in input due oggetti di classe Fracture e restituisce un booleano che informa circa l'avvenuta intersezione.

4.2.2 inters_2D

(Codice → DFN.py → linea 215)
Data una retta in forma parametrica , dove t è la direzione e r_0 un punto qualsiasi della retta,

$$r : X(s) = r_0 + st$$

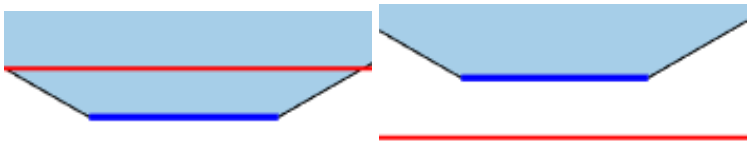
e una frattura, restituisce i valori di $\boldsymbol{s}$ trovati, ovvero calcola l'intersezione. Prima di procedere, sia la retta che la frattura vengono riportate sul piano facendo la rotazione inversa rispetto a quella fatta al momento della generazione del poligono. Il metodo scorre tutti i lati della frattura fino a quando non trova due valori di $\boldsymbol{s}$ oppure fino all'ultimo lato. Per ognuno di essi si scrive il sistema

$$\underset{A}{\begin{pmatrix} t_0 & P_{0_x} - P_{1_x} \\ t_1 & P_{0_y} - P_{1_y} \end{pmatrix}} \underset{x}{\begin{pmatrix} s \\ u \end{pmatrix}} = \underset{b}{\begin{pmatrix} P_{0_x} - r_{0_x} \\ P_{0_y} - r_{0_y} \end{pmatrix}}$$

dove: $\mathbf{P}_0$ e $\mathbf{P}_1$ sono gli estremi del lato preso in esame.
Si distinguono due casi:

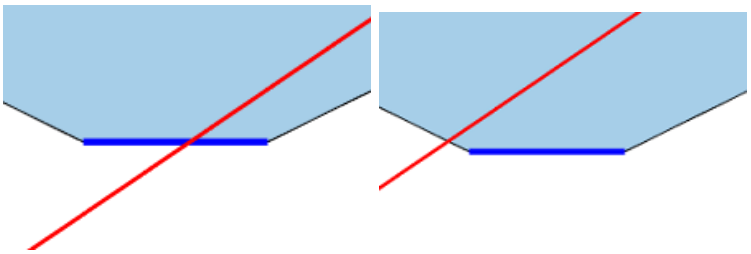
- det(A) = 0 → segmento e retta paralleli
  - $P_0 \in$ r

– $P_0 \notin$ r



Nel caso di sovrapposizione tra retta e segmento viene restituita la lista con i valori di s corrispondenti agli estremi del lato. In caso non siano sovrapposti si passa al controllo del lato successivo.

- det(A) $\neq 0 \rightarrow$ segmento e retta non paralleli



(a) $\boldsymbol{u}$ compresa tra 0 e 1 (b) $\boldsymbol{u}$ non compresa tra 0 e 1

Quando la retta e il lato non sono paralleli, si procede alla risoluzione del sistema iniziale e viene controllato il valore di $\boldsymbol{u}$. Se non è compreso tra 0 e 1, quindi non c'è intersezione, si passa al controllo del lato successivo, altrimenti si appende alla lista da restituire il valore corrispondente di $\boldsymbol{s}$.

4.2.3 gen_trace

(Codice → DFN.py → linea 181)
Dati in ingresso i parametri ***i1*** e ***i2***, che rappresentano gli indici dei poligoni, restituisce un oggetto di classe Trace (se la traccia esiste), altrimenti ***None*** se la traccia non esiste. Per fare ciò calcola inizialmente i parametri necessari al metodo (4.2.2), facendo il prodotto vettoriale tra le normali dei due poligoni (per trovare **t**, direzione della retta) e risolvendo il seguente sistema

$$\underset{A}{\begin{pmatrix} n_1 & n_2 & n_3 \\ m_1 & m_2 & m_3 \\ t_1 & t_2 & t_3 \end{pmatrix}} \underset{x}{\begin{pmatrix} x \\ y \\ z \end{pmatrix}} = \underset{b}{\begin{pmatrix} p_1 n_1 + p_2 n_2 + p_3 n_3 \\ q_1 m_1 + q_2 m_2 + q_3 m_3 \\ 0 \end{pmatrix}}$$

$$\mathbf{n} = \begin{pmatrix} n_1 & n_2 & n_3 \end{pmatrix} \textit{ versore normale del poligono } i1$$

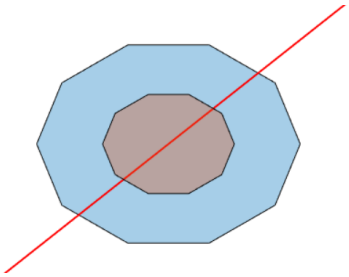
$$\mathbf{m} = \begin{pmatrix} m_1 & m_2 & m_3 \end{pmatrix} \textit{ versore normale del poligono } i2$$

$$\mathbf{p} = \begin{pmatrix} p_1 & p_2 & p_3 \end{pmatrix} \textit{ punto del poligono } i1$$

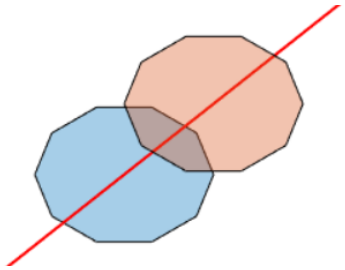
$$\mathbf{q} = \begin{pmatrix} q_1 & q_2 & q_3 \end{pmatrix} \textit{ punto del poligono } i2$$

$$\mathbf{x} = \textit{ punto } r_0 \textit{ generico della retta}$$

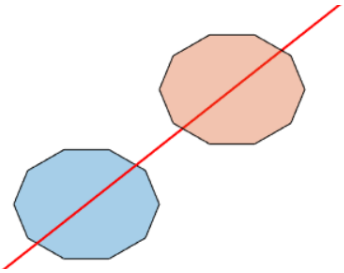
Successivamente controlla di aver ricevuto una lista contenente 4 elementi. Nel caso fosse soddisfatta la condizione, bisogna distinguere un caso sfavorevole (c), nel quale i valori di $\boldsymbol{s}$ corrispondenti ad ogni poligono restano vicini entrambi, a differenza dei casi (a) e (b).



(a) $s_{11} \leqslant s_{21} \leqslant s_{22} \leqslant s_{12}$



(b) $s_{11} \leqslant s_{21} \leqslant s_{12} \leqslant s_{22}$



(c) $s_{11} \leqslant s_{12} \leqslant s_{21} \leqslant s_{22}$

Dove s_{11}, s_{12} sono le s del primo poligono e s_{21}, s_{22} sono le s del secondo poligono. Nei casi (a) e (b) procede con l'ordinamento dei 4 valori e prende quelli centrali per generare la traccia.

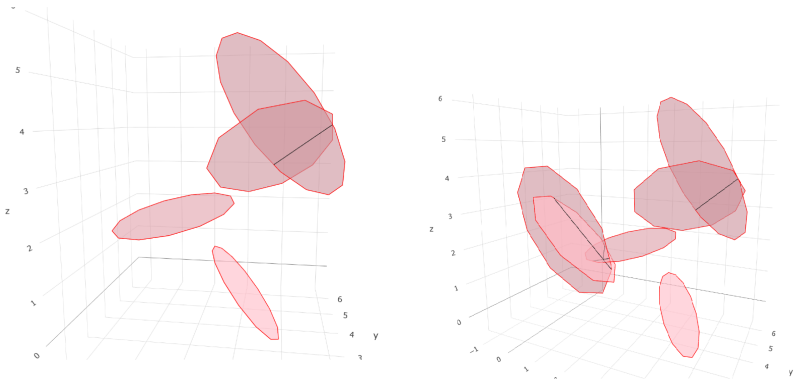
4.2.4 aggiorna_int

(Codice → DFN.py → linea 262)
Aggiorna le strutture dati (4.1) nel caso di possibile e reale intersezione tra due poligoni facendo uso dei metodi (4.2.3) e (4.2.1).

4.2.5 gen_frac

(Codice → DFN.py → linea 65)
Inizialmente genera tutti i parametri necessari alla creazione di $\boldsymbol{n}$ oggetti di classe Fracture, usando le opportune distribuzioni di probabilità del DFN; successivamente genera le fratture stesse e infine aggiorna opportunamente tutte le strutture dati del DFN, distinguendo il caso in cui ci siano già dei poligoni o meno, utilizzando il metodo (4.2.4).

Aggiunta di nuovi poligoni in un DFN già esistente:

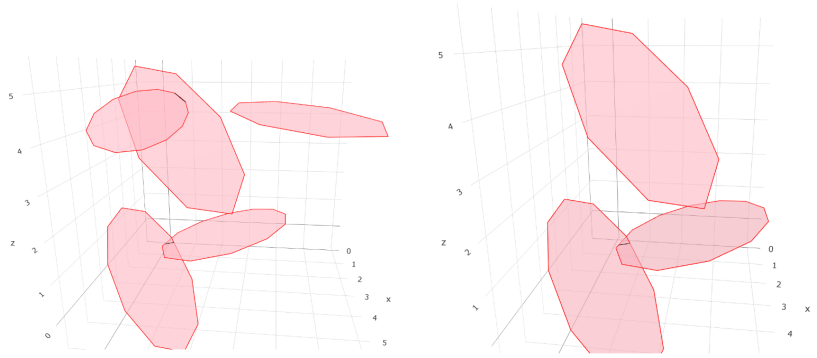


Prima Dopo

4.2.6 rimuovi

(Codice → DFN.py → linea 114)
Data una lista di indici, rimuove tutte le fratture corrispondenti a tali indici. Procede con la rimozione delle occorrenze dell' i-esimo poligono nelle strutture dati e rinumera gli indici corrispondenti a tutte le fratture che lo seguono in ***fractures***.

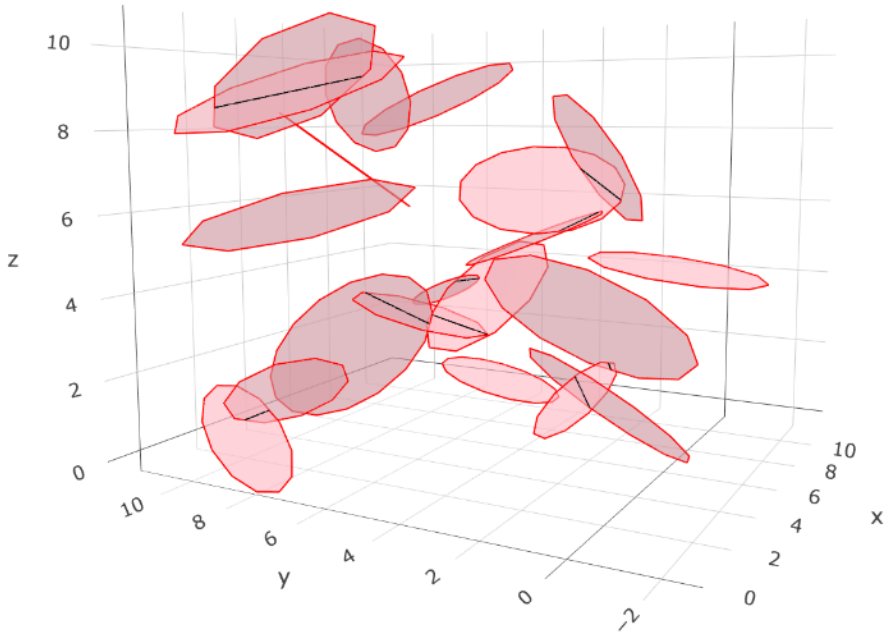
Rimozione di fratture dal DFN:



Prima Dopo

4.2.7 visual3D

(Codice → DFN.py → linea 282)
Stampa a video il DFN contenente le fratture e le tracce.



Viene fatto un opportuno controllo per stampare anche i poligoni paralleli ad uno dei piani xy, yz e xz, che altrimenti non verrebbero stampati.

4.2.8 scrittura1

(Codice → DFN.py → linea 320)
Scrive su file i dati relativi alle fratture contenute nel DFN come richiesto nella consegna del progetto al punto **7** della classe DiscreteFractureNetwork:
N
0, m_0
x_0, y_0, z_0
.
:
x_{m_0-1}, y_{m_0-1}, z_{m_0-1}
1, m_1
.
:

4.2.9 scrittura2

(Codice → DFN.py → linea 334)
Scrive su file i dati relativi alle fratture contenute nel DFN come richiesto nella consegna del progetto al punto **8** della classe DiscreteFractureNetwork:
N
M
0, m_0, first_vertex_index_P(0)
.
:
N-1, m_{N-1}, first_vertex_index_P(N-1)
x_0, y_0, z_0
.
:
x_{M-1}, y_{M-1}, z_{M-1}

4.2.10 save

(Codice → DFN.py → linea 358)
Salva l'oggetto DiscreteFractureNetwork in un file .pkl con le sue distribuzioni che lo caratterizzano.