Programmazione e Calcolo Scientifico - a.a. 2018/2019
Progetto PYTHON

Abstract

Per la realizzazione di questo progetto, si suggerisce di scaricare dalla pagina del corso i seguenti moduli python:

- `matlab_clones.py`
- `distributions.py`

Si consiglia inoltre l'installazione dei seguenti pacchetti:

- `numpy` e `scipy` (per il calcolo scientifico);
- `matplotlib` e `plotly` (per realizzare grafici).

1 Descrizione del Problema

Si supponga di voler generare un inseme di N poligoni nello spazio $\mathbb{R}^3$ partendo dai seguenti dati:

- Per ogni $i = 0, \dots, N-1$, l'i-esimo poligono P_i deve essere inizialmente inscritto in un'ellisse

$$\frac{x^2}{r_{i_x}^2} + \frac{y^2}{r_{i_y}^2} = 1\,, \tag{1}$$

  con r_{i_x} semiasse maggiore e r_{i_y} semiasse minore, tale che r_{i_x} ed il fattore di forma (*Aspect Ratio*) $AR = \frac{r_{i_x}}{r_{i_y}}$ siano determinati secondo due rispettive leggi di distribuzione:

$$\begin{aligned} p_{r_{i_x}}(\rho) &= \frac{1-a}{r_u^{1-a} - r_\ell^{1-a}}\,\rho^{-a} && \textit{(pdf power-law con } a > 1 \textit{ e valori in } [r_\ell,\ r_u]\textit{)} \\ AR &\sim\ \mathcal{U}[1,3] && \textit{(dist. uniforme in } [1,\ 3]\textit{)} \end{aligned} \tag{2}$$

  Inoltre, per ogni $i = 0, \dots, N-1$, il numero di lati m_i di P_i deve essere determinato secondo una legge di distribuzione

$$m_i \sim\ \mathcal{U}_{\mathbb{Z}}[8, 16] \quad \textit{(dist. uniforme di interi tra 8 e 16)} \tag{3}$$

  oppure impostato ad un valore costante $m_i = m \geq 8$;

- Per ogni i, P_i viene successivamente *ruotato in senso anti-orario* di un angolo α_i attorno alla normale al poligono $\boldsymbol{n}_i$ (che assumiamo essere il versore dell'asse z, cioè $[0, 0, 1]^\top = \boldsymbol{e}_3$). L'angolo α_i viene determinato secondo la legge di distribuzione

$$\alpha_i \sim\ \mathcal{U}(0, 2\pi] \quad \textit{(dist. uniforme in } (0,\ 2\pi]\textit{)} \tag{4}$$

- Per ogni i, P_i deve essere ruotato fino ad allineare la sua normale $\boldsymbol{n}_i$ ad un versore $\boldsymbol{v}_i$ determinato casualmente secondo la legge di distribuzione di *Von Mises-Fisher* sulla sfera unitaria;

- Per ogni i, P_i deve essere traslato fino a portare il suo baricentro $\boldsymbol{b}_i$ (che coincide inizialmente con l'origine) in un punto dello spazio determinato casualmente secondo una *distribuzione uniforme* all'interno di una regione $\mathbb{D} = [x_{\min},\ x_{\max}] \times [y_{\min},\ y_{\max}] \times [z_{\min},\ z_{\max}]$;

2 Progetto

Si realizzi un progetto Python che definisca una classe *DiscreteFractureNetwork* per rappresentare un insieme di poligoni come quello sopra descritto e rappresentati a loro volta tramite una classe *Fracture*. I contenuti (*attributi* e *metodi*) di queste due classi, se non esplicitamente indicati di seguito, sono lasciati alla discrezione degli studenti (per esempio il metodo di inizializzazione *__init__*).

classe Fracture: Questa classe, deve essere costruita secondo le seguenti indicazioni.

1. deve avere come *attributi* il numero di vertici del poligono e le lunghezze dei semiassi dell'ellisse dentro la quale il poligono è stato inscritto;
2. deve avere come *attributi* il suo versore normale ed il suo baricentro;
3. deve avere come *attributi* un contenitore di dati (eventualmente anche una matrice) che contenga tutti i vertici del poligono, ordinati in senso antiorario a partire dal vertice ottenuto con $t = 0$ (si veda (7)).
4. deve avere come *attributi* gli angoli α_i, φ_i e θ_i (si veda (9));
5. deve avere come *attributo/i* gli estremi che caratterizzano il "Bounding Box"

$$[x_{i_{\min}}, x_{i_{\max}}] \times [y_{i_{\min}}, y_{i_{\max}}] \times [z_{i_{\min}}, z_{i_{\max}}] \tag{5}$$

   contentente il poligono. In alternativa può disporre di un *metodo* che li calcola quando chiamato.

classe DiscreteFractureNetwork: Questa classe, deve essere costruita secondo le seguenti indicazioni.

1. deve avere come *attributo/i* gli estremi che caratterizzano il dominio $\mathbb{D}$ del DFN

$$[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}] \,. \tag{6}$$

2. deve avere come *attributo* un oggetto `PowerLawBounded` (modulo `distributions.py`) rappresentante la legge di distribuzione per i semiassi maggiori dei poligoni.
3. deve avere come *attributo* un oggetto `VonMisesFisher` (modulo `distributions.py`) rappresentante la legge di distribuzione per le normali ai poligoni.
4. deve avere come *attributo* un contenitore di dati che contenga i vari oggetti di tipo *Fracture* rappresentanti gli N poligoni.
5. deve disporre di un *metodo* per la visualizzazione dei poligoni e di $\mathbb{D}$ in $\mathbb{R}^3$;
6. deve disporre di un *metodo* per scrivere un file di testo rappresentante l'insieme dei poligoni nella seguente forma:
   - nella prima riga deve essere indicato il numero N di poligoni generati;
   - nelle righe successive, per ogni poligono, si deve avere:
     - una riga con l'indice intero del poligono, seguto dall'intero indicante il numero di vertici del poligono;
     - una riga per ogni vertice del poligono che ne indichi le coordinate (i vertici devono essere ordinati seguendo l'ordinamento specificato nella classe *Fracture*).

   Schema di esempio del file di testo:

   `N`

   `0,` m_0

   $x_0, \; y_0, \; z_0$

   $\vdots$

   $x_{m_0-1}, \; y_{m_0-1}, \; z_{m_0-1}$

   `1,` m_1

   $\vdots$

7. deve disporre di un *metodo* per scrivere un file di testo rappresentante l'insieme dei poligoni nella seguente forma:
   - nella prima riga deve essere indicato il numero N di poligoni generati;
   - nella seconda riga deve essere indicato il numero totale $M = \sum_{i=0}^{N-1} m_i$ di vertici;
   - nelle N righe successive, una per ogni poligono P_i, si deve indicare l'indice del poligono, il numero di vertici del poligono e l'indice del primo vertice del poligono nell'elenco di tutti i vertici di tutti i poligoni;
   - nelle M righe successive si deve indicare le coordinate di tutti i vertici di tutti i poligoni.

   Schema di esempio del file di testo:
   `N`
   `M`
   `0`, m_0, `first_vertex_index_P(0)`
   $\vdots$
   `N-1`, m_{N-1}, `first_vertex_index_P(N-1)`
   $x_0,\ y_0,\ z_0$
   $\vdots$
   $x_{M-1},\ y_{M-1},\ z_{M-1}$

8. deve disporre di un *metodo* che, guardando ai "*Bounding Box*" (BB) dei vari poligoni, attraverso il criterio (11) restituisca una matrice di connettività $A \in \mathbb{R}^{N \times N}$ tale che $a_{ij} = 1$ se e solo se il BB di P_i interseca il BB di P_j, ed $a_{ij} = 0$ altrimenti.
9. deve disporre di *metodi* che permettano di aggiungere e rimuovere poligoni dall'oggetto *DiscreteFractureNetwork*.
10. deve disporre di un *metodo* che generi uno o più nuovi poligoni da aggiungere all'oggetto *DiscreteFractureNetwork*.
11. deve disporre di un *metodo* che salvi come file *.pkl* l'oggetto *DiscreteFractureNetwork*, coerentemente con le sue distribuzioni.

Suggerimenti

Generazione dei Vertici dei Poligoni

I vertici iniziali del poligono P_i inscritto nell'ellisse (1) si possono generare nel seguente modo:

$$[x_t,\ y_t]^\top = \left[r_{i_x} \cos\left(t \cdot \frac{2\pi}{m_i} \right),\ r_{i_y} \sin\left(t \cdot \frac{2\pi}{m_i} \right) \right]^\top, \quad \forall\ t = 0, \dots, m_i - 1 \tag{7}$$

Rotazione dei Poligoni

Ricordiamo di seguito le matrici di rotazione in $\mathbb{R}^3$ attorno agli assi x, y e z:

$$\begin{aligned}
\mathbf{R}_x(\omega) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\omega & -\sin\omega \\ 0 & \sin\omega & \cos\omega \end{bmatrix} \quad \textit{(rotazione attorno asse x)} \\
\mathbf{R}_y(\omega) &= \begin{bmatrix} \cos\omega & 0 & \sin\omega \\ 0 & 1 & 0 \\ -\sin\omega & 0 & \cos\omega \end{bmatrix} \quad \textit{(rotazione attorno asse y)} \\
\mathbf{R}_z(\omega) &= \begin{bmatrix} \cos\omega & -\sin\omega & 0 \\ \sin\omega & \cos\omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textit{(rotazione attorno asse z)}
\end{aligned} \tag{8}$$

Si ricordi che un angolo $\omega > 0$ implica una rotazione *anti-oraria*, mentre un angolo $\omega < 0$ implica una rotazione oraria.

Consideriamo ora un versore in *coordinate cartesiane* $\boldsymbol{v}_i = [x_i, y_i, z_i]^\top \in \mathbb{R}^3$ e con *coordinate polari* $[\theta_i, \varphi_i, 1]^\top$ (ricavabili con la funzione `cart2sph` del modulo python `matlab_clones.py`); gli angoli θ_i e φ_i rappresentano rispettivamente[1]:

$\theta_i =$ **azimuth:** l'angolo tra l'asse x e la proiezione di $\boldsymbol{v}_i$ sul piano (x, y), misurato in senso *anti-orario* rispetto l'asse z. Assume valori in $[0, 2\pi)$.

$\varphi_i =$ **elevation:** l'angolo tra $\boldsymbol{v}_i$ ed il piano (x, y). Assume valori in $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

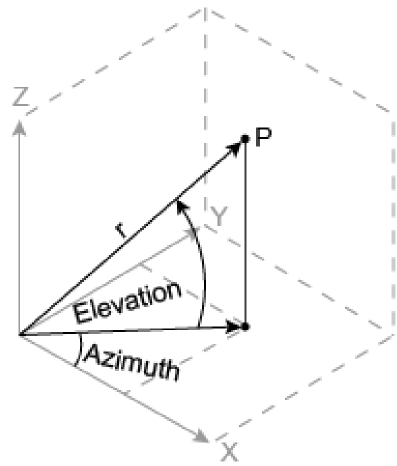

Figure 1: Convenzione coordinate polari di `matlab_clones.py` e Matlab.

Avremo quindi che, dato il poligono P_i inizialmente inscritto nella propria ellisse, per portare $\boldsymbol{n}_i$ a coincidere con $\boldsymbol{v}_i$, dopo la prima rotazione di P_i di un angolo α_i attorno alla sua normale, dovremo compiere le seguenti operazioni:

1. applicare una prima rotazione di α_i attorno all'asse z in senso anti-orario (funzione `rotz` in `matlab_clones.py`);
2. ricavare θ_i e φ_i da $\boldsymbol{v}_i = [x_i, y_i, z_i]^\top$ tramite `cart2sph` in `matlab_clones.py`;
3. applicare una seconda rotazione di $\frac{\pi}{2} - \varphi_i$ attorno all'asse y in senso anti-orario (funzione `roty` in `matlab_clones.py`);
4. applicare una terza rotazione di θ_i attorno all'asse z in senso anti-orario (funzione `rotz` in `matlab_clones.py`).

Si ha quindi:

$$\boldsymbol{v}_i = \mathbf{R}_z(\theta)\mathbf{R}_y\left(\frac{\pi}{2} - \varphi_i\right)\mathbf{R}_z(\alpha_i)\boldsymbol{n}_i \tag{9}$$

In generale, per applicare la rotazione a tutto il poligono, basterà applicare le stesse rotazioni alle coordinate di tutti i suoi vertici ottenuti con (7).

Traslazione dei Poligoni

Per ogni i, sia $\boldsymbol{b}_i'$ il vettore contenuto in $\mathbb{D} \subset \mathbb{R}^3$ dove deve essere traslato il baricentro iniziale $\boldsymbol{b}_i = [0, 0, 0]^\top$ del poligono P_i.

Per traslare tutto il poligono P_i come desiderato, dopo aver effettuato le dovute rotazioni indicate in (9), basterà sommare $\boldsymbol{b}_i'$ alle coordinate di tutti i suoi vertici.

[1]secondo la stessa convenzione di Matlab.

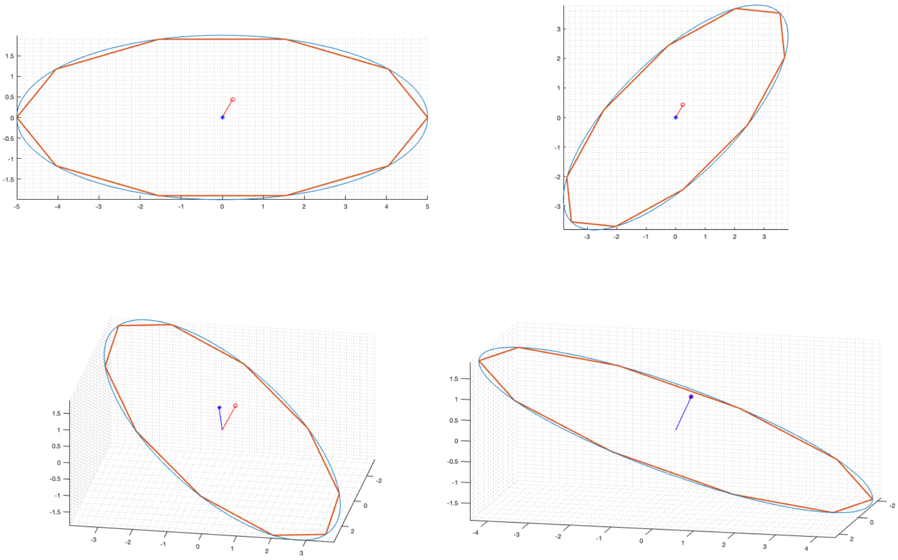

Figure 2: Inscrizione del poligono, rotazione di α_i, rotazione di $\frac{\pi}{2} - \varphi_i$, rotazione di θ_i (in rosso $\boldsymbol{v}_i$, in blu $\boldsymbol{n}_i$).

Leggi di Distribuzione

Forniamo di seguito le indicazioni per utilizzare in python le varie leggi di distribuzione richieste per la generazione dei poligoni e del DFN.

Von Mises - Fisher (distribuzione per $\boldsymbol{n}_i$): la legge di distribuzione delle normali ai poligoni dovrebbe seguire una distribuzione chiamata Von Mises-Fisher (VMF) che fornisce dei punti sulla sfera unitaria distribuiti in modo da avere un assegnato punto medio[2] rappresentato dal vettore $\boldsymbol{\mu} \in \mathbb{R}^3$ e un assegnato parametro di concentrazione $k \in \mathbb{R}$.

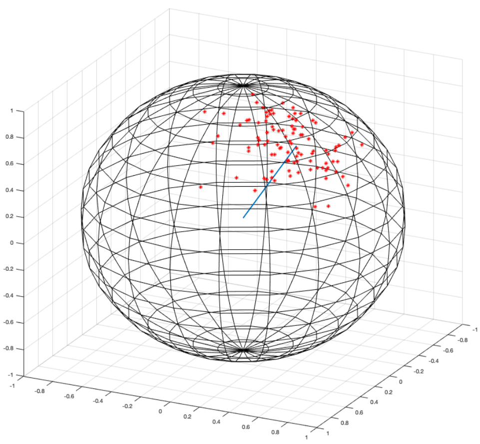

Figure 3: Esempio di campionamento con distribuzione di Von Mises - Fisher. Il segmento blu rappresenta il vettore modale $\boldsymbol{\mu}$.

Per creare una distribuzione di VMF, si faccia utilizzo della classe `VonMisesFisher` in `distributions.py`. In particolare, dato un vettore colonna `mi` $\in \mathbb{R}^3$ (rappresentato come *numpy ndarray* per il punto medio ed un parametro di concentrazione `k` $\in \mathbb{R}$:

- **Creazione oggetto distribuzione VMF:**
  `my_vmf = VonMisesFisher(mi, k)`

[2] più precisamente un punto modale.

- **Estrazione di nsamples campioni da my_vmf:**

  `vmf_samples = my_vmf.sample(nsamples)`

  In questo caso l'output `vmf_samples` è una matrice (*numpy ndarray*) $\mathbb{R}^{3\times \texttt{nsamples}}$ dove ogni colonna rappresenta un campionamento sulla sfera unitaria.

Power-Law Limitata in un Intervallo (distribuzione per r_{i_x}): la legge di distribuzione dei semiassi maggiori delle ellissi per l'inscrizione dei poligoni dovrebbe seguire la distribuzione Power-Law Limitata (PLL) in un intervallo $[r_\ell, r_u]$ con esponente $a > 1$ (pdf indicata in (2)).

Per creare una distribuzione PLL, si faccia utilizzo della classe `PowerLawBounded` in `distributions.py`. In particolare, dato un esponente $\texttt{a} \in \mathbb{R}_{>1}$ e dei raggi $\texttt{r\_l}, \texttt{r\_u} \in \mathbb{R}_{>0}$:

- **Creazione oggetto distribuzione PLL:**

  `my_pll = PowerLawBounded(alpha=a, radius_l=r_l, radius_u=r_u)`

- **Estrazione di nsamples campioni da my_pll:**

  `pll_samples = my_pll.sample(nsamples)`

  In questo caso l'output `pll_samples` è un *numpy ndarray* con shape `(nsamples, )` dove ogni elemento rappresenta un campionamento per il semiasse maggiore.

Distribuzioni Uniformi in $\mathbb{R}$ ed $\mathbb{R}^3$: si rimanda all'utilizzo di `numpy.random.uniform` (documentazione: https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.uniform.html).

Distribuzione Uniforme in $\mathbb{Z}$: per semplificare, si applichi la funzione `numpy.round` alle distribuzioni uniformi in $\mathbb{R}$.

Criterio di esclusione delle possibili intersezioni tra BB

Il criterio va applicato a ciascuna direzione, qui descritto solo per x. Intervalli $I_i(x) = [x_{i_{\min}}, x_{i_{\max}}]$, $I_j(x) = [x_{j_{\min}}, x_{j_{\max}}]$. Definiamo

$$\begin{aligned} MaxXmin &= \max\{x_{i_{\min}}, x_{j_{\min}}\}, \\ minXMax &= \min\{x_{i_{\max}}, x_{j_{\max}}\}. \end{aligned} \tag{10}$$

Allora vale

$$MaxXmin > minXMax \Rightarrow I_i(x) \cap I_j(x) = \emptyset \, . \tag{11}$$