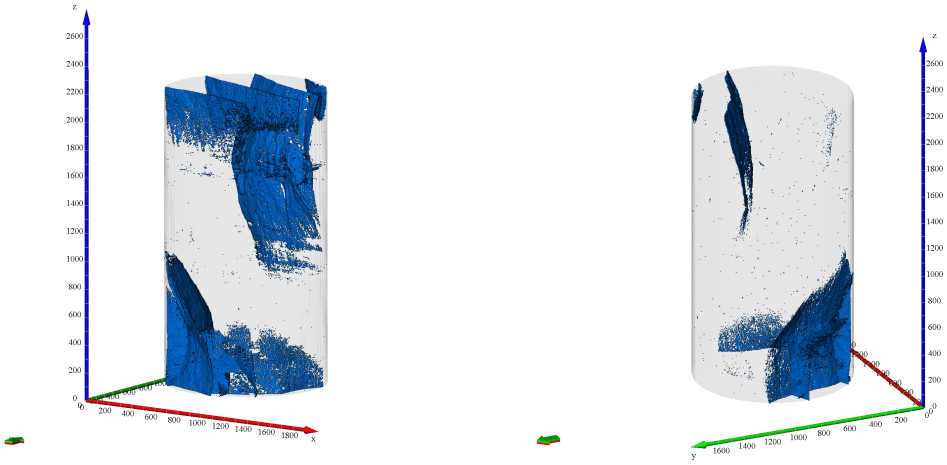
TD Modélisation - Analyse d'images avec Gdal

Microstructure d'un échantillon cylindrique de roche argileuse (1h30)

Contexte

Dans le domaine de la science des matériaux (p.ex. la mécanique des milieux poreux et des matériaux composites), le traitement d'images constitue un outil fondamental pour la caractérisation microstructurale à partir d'images numériques issues de techniques de microscopie ou de tomographie. S'agissant des géomatériaux, cette approche permet d'extraire des paramètres de texture (porosité, taille et distribution des pores, orientation des fissures, anisotropies, etc.), afin d'établir un lien entre l'information microstructurale et les propriétés multi-physiques macroscopiques (perméabilité, conductivité, résistance mécanique, etc.).



(a) Avant l'essai

(b) Après l'essai

Figure 1: Microstructures de l'échantillon avant et après l'essai d'auto-colmatage

Dans cette partie du TD, on dispose certaines images issues de coupes 2D d'un échantillon de roche argileuse, obtenues par imagerie tomographique à rayons X. Chaque image correspond à une section transversale d'un cylindre de roche, ce qui donne accès à la microstructure interne du matériau.

(a)

(b)

Figure 2: Exemples de coupes 2D

Objectif du TD

Développer un traitement automatique permettant :

- Identifier le cylindre de roche sur chaque coupe.
- Détecter les fissures présentes à l'intérieur.

Les images sont fournies sous format .tif (par exemple : `00492.tif`).

Étape 1 — Chargement et visualisation

Dans cette première étape, il s'agit de charger une coupe fournie et de l'afficher afin de visualiser la structure de la roche. À faire :

1. Charger une image .tif à partir des fichiers fournis.
2. Extraire la première bande et afficher la coupe.
3. Vérifier les dimensions de l'image avec np.shape(): `https://numpy.org/doc/stable/reference/generated/numpy.shape.html`

Étape 2 — Détection du cylindre

L'objectif de cette étape est de repérer automatiquement la zone correspondant au cylindre dans une coupe, puis d'estimer son centre et son rayon afin de délimiter la région d'intérêt. À faire :

1. 2.1 Effectuer un seuillage et une binarisation adaptés pour séparer la roche du fond (p.ex. on attribuera des valeurs nulles au fond).
2. Identifier les coordonnées des pixels appartenant à la roche.
   Dans la suite du TD, on pourra utiliser les routines suivante permettant de manipuler des tableaux:
   - `np.where(condition)` : renvoie les indices (lignes, colonnes) des éléments où la condition est vraie. `https://numpy.org/doc/2.3/reference/generated/numpy.where.html`
   - `np.nonzero(array)` : renvoie les indices (lignes, colonnes) où le tableau est non nul / True. `https://numpy.org/doc/2.3/reference/generated/numpy.nonzero.html`
   - `np.indices(shape)` : renvoie une grille complète d'indices pour tous les éléments d'un tableau de taille `shape`. `https://numpy.org/doc/2.2/reference/generated/numpy.indices.html`
   - `np.column_stack()` : combine des tuples en un tableau à deux colonnes. `https://numpy.org/devdocs/reference/generated/numpy.column_stack.html`
3. Déterminer le centre du cylindre à partir de ces coordonnées.
   Note : On pourra utiliser *mean(axis=0)* pour calculer la moyenne des coordonnées des pixels de la roche. L'argument *axis=0* signifie que la moyenne est faite par colonne.
4. Estimer le rayon correspondant à la limite externe de la roche.
5. Visualiser sur l'image le centre et le contour détecté du cylindre.
   - On peut utiliser `plt.Circle((x,y), r, ...)` pour créer une nouvelle forme graphique, avec `x,y` les coordonnées du centre et `r` le rayon du cercle.
   - `Circle = plt.Circle(...)` crée juste une forme (une instance de `matplotlib.patches.Circle`). Contrairement aux autres commandes utilisées jusqu'à présent (`imshow`, `hist`, `plot`, ...), il faut ensuite rajouter cette forme à un axe :
     - `ax = plt.gca()` # récupère l'axe courant
     - `ax.add_patch(circle)` # ajoute le cercle à cet axe

Étape 3 — Détection et affichage des fissures

L'objectif de cette étape est de détecter les fissures présentes à l'intérieur du cylindre et de les afficher sur l'image. Cela implique de traiter l'image pour réduire le bruit et mettre en évidence les fissures détectées. À faire :

- Déterminer un masque binaire des fissures en utilisant un seuil basé sur un percentile des valeurs d'intensité (sur le tableau de l'image originale).
- Afficher le résultat. Que remarquez vous?

Afin d'éviter ce problème:

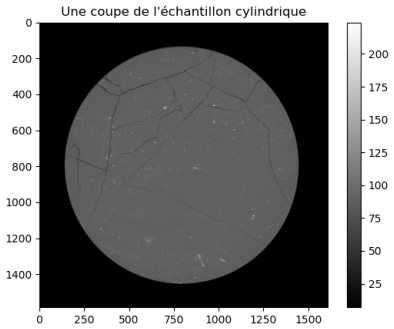
- Créer un masque définissant la zone intérieure du cylindre, en excluant une marge près du bord (par exemple $marge\_bord = mb$, avec $mb \ll rayon$). On calcule d'abord les distances de chaque pixel par rapport au centre détecté du cylindre :

  ```
  y_indices, x_indices = np.indices(image_array.shape)
  distances_image = np.sqrt((y_indices - centre[0])**2 + (x_indices - centre[1])**2)
  ```

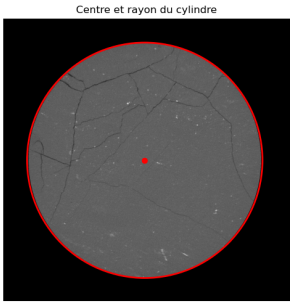
  Puis on crée un tableau booléen *masque_interieur* en comparant ces distances avec $rayon - marge\_bord$.

- Afficher le résultat : les fissures apparaissent en valeur non nulle (ou en couleur), tandis que le reste de l'image (roche et fond) reste sombre selon le masque. Que remarquez-vous ?

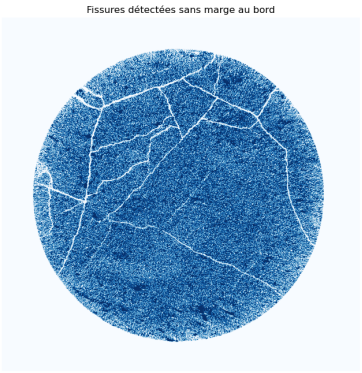
Exemple de résultats attendus



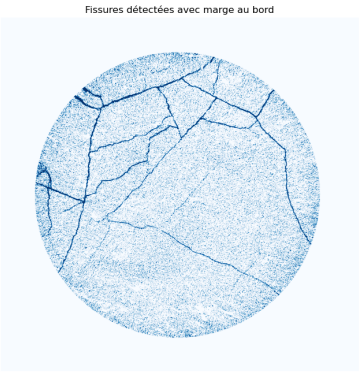
(a)



(b)



(c)



(d)

Étape optionnelle — Suppression des bruits

Cette partie du programme utilise des méthodes de traitement d'image pour nettoyer le masque de fissures obtenu précédemment, en supprimant les petits « bruits », puis visualiser les fissures sous forme colorée.

```
from scipy.ndimage import uniform_filter, binary_opening

# Nettoyage simple : flou + ouverture morphologique
masque_fissure_flou = uniform_filter(masque_fissure.astype(float), size=3)
masque_fissure = masque_fissure_flou > 0.5

# Ouverture morphologique pour éliminer les petits pixels isolés
# masque_fissure = binary_opening(masque_fissure, structure=np.ones((3, 3)))

# Image RGBA fissures bleues
image_rgba = np.zeros((hauteur, largeur, 4), dtype=np.uint8)
image_rgba[masque_fissure] = [0, 0, 255, 255]

# Affichage final
plt.figure(figsize=(8, 8))
plt.imshow(image_rgba, interpolation="none")
plt.title("Fissures détectées (masque bleu amélioré)")
plt.axis("off")
plt.show()
```

- *scipy.ndimage* est une bibliothèque Python spécialisée dans le traitement d'images.
- *uniform_filter* rend l'image plus lisse en remplaçant chaque pixel par la moyenne des pixels autour de lui dans une petite zone. Cela permet de réduire le bruit et d'atténuer les variations trop brusques.
- *binary_opening* effectue une ouverture morphologique, c'est-à-dire qu'on commence par rétrécir les zones blanches (érosion) pour supprimer les petits points isolés, puis on les regrossit légèrement (dilatation) afin de retrouver la forme principale. Cela permet de nettoyer le bruit sans modifier les grandes fissures.