Détection et représentation de fissures

Sylvia Feld-Payet

1 Objectif et données

Le principal objectif de ce TP sur la détection et la représentation de fissures est de comprendre les grandes étapes de la détermination d'un chemin de fissure discret à partir d'un résultat de calcul pour un modèle à endommagement diffus. Pour cela, partons du calcul par éléments finis de l'éprouvette utilisée dans l'article de S. Feld-Payet, J. Besson and F. Feyel, *Finite element analysis of damage in ductile structures using a nonlocal model combined with a three-field formulation* (International Journal of Damage Mechanics, 2011), déjà abordé au TP8 et étudions le champ d'endommagement régularisé : voir figure 1.

1.1. Où vous attendez-vous à voir une fissure ? Vous pouvez utiliser les fonction d'Adobe pour l'indiquer sur le pdf afin de pouvoir comparer le résultat final à votre prédiction ou utiliser les outils de graphisme sur une capture d'écran. ?

L'analyse de ce champ 2D se fera à l'aide d'un script en Python. Pour démarrer, ouvrir le fichier `TP9_a_completer.py`.

2 Charger les données

A l'aide d'un script de post-processing, les valeurs du champs d'endommagement et du champ de plasticité cumulée non locale à partir duquel ce dernier a été calculé ont été stockés dans le fichier `calcul.post`. Ce fichier se présente sous la forme suivante :

```
# === time 1.0000000000e-02
# elem ip X Y epcum_bar_max wpmax
1 1 2.0374861000e+00 8.0624543167e+00 0.0000000000e+00 0.0000000000e+00
1 2 2.0541407500e+00 7.9675034167e+00 0.0000000000e+00 0.0000000000e+00
1 3 2.1614351500e+00 8.0264306667e+00 0.0000000000e+00 0.0000000000e+00
# elem ip X Y epcum_bar_max wpmax
2 1 1.2305356833e+00 2.8549687500e+00 0.0000000000e+00 0.0000000000e+00
2 2 1.3302389833e+00 2.8376767000e+00 0.0000000000e+00 0.0000000000e+00
2 3 1.2638439333e+00 2.9345601500e+00 0.0000000000e+00 0.0000000000e+00
```

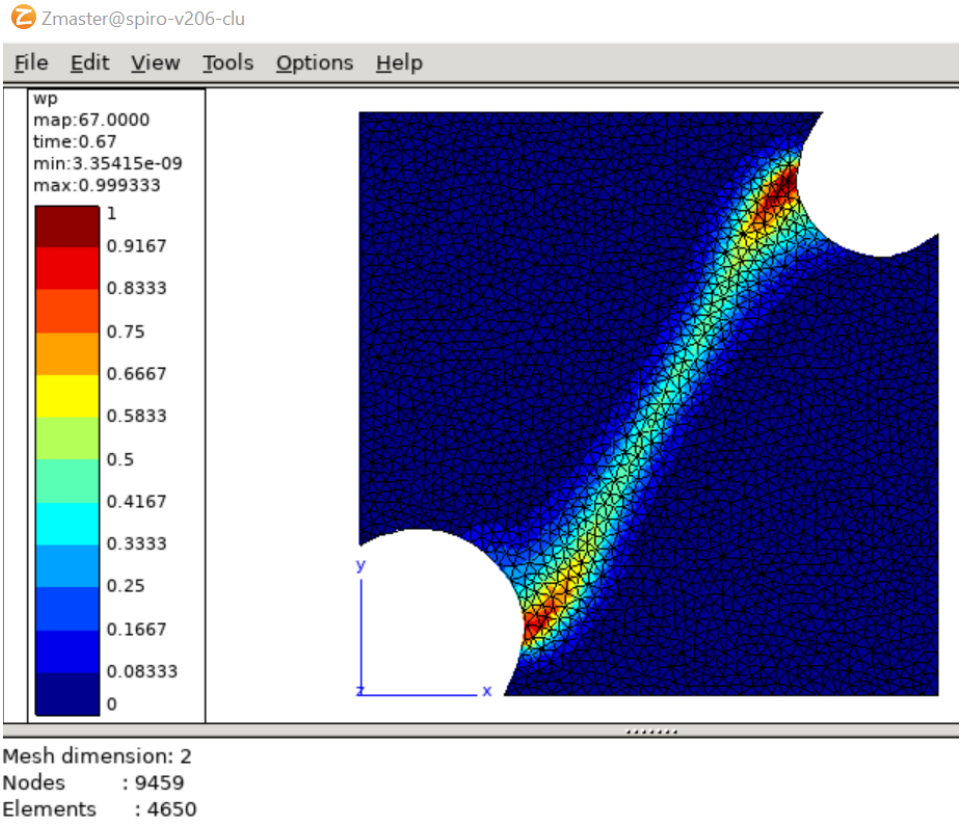

FIGURE 1 – Champ d'endommagement (dénoté ω_p) obtenu avec Zset au temps 0.67s.

```
# elem ip X Y epcum_bar_max wpmax
[...]
```

Dans ce fichier, `wpmax` désigne l'endommagement et `epcum_bar_max` désigne la plasticité cumulée non locale. Pour chaque élément `elem`, il y a trois points d'intégration pour lesquels les positions suivant les axes `X` et `Y` sont spécifiées. S'agissant d'un calcul mené avec le formalisme des petites déformations, ce sont les positions dans la configuration de référence qui sont précisées. Notons qu'il n'est donc utile de charger les positions des points d'intégration qu'une seule fois.

2.1. A partir de l'exemple fourni pour le temps 0.50s, charger les données pour le temps 0.67s et vérifier à l'aide du graphique que la carte obtenue correspond bien à ce qui est présenté en figure 1.

3 Distinguer les différentes zones d'amorçage possibles

Pour ce calcul, deux fissures peuvent apparaître : une au niveau de chaque encoche. Il est plus simple de traiter chaque fissure indépendamment tant qu'elles ne sont pas

près de se rencontrer. En effet, un traitement générique peut alors être appliqué à chaque fissure en ne considérant que les points fortement endommagés qui l'entourent.

3.1. En enlevant le mode commentaire pour le bloc """Select only the higly damaged areas""", tester différentes valeurs de seuil pour déterminer celles qui vous permettent d'obtenir deux groupes distincts de points dont l'endommagement dépasse ce seuil. Attention à faire en sorte d'avoir suffisamment de points dans chaque groupe pour pouvoir exploiter les données. Pour la suite, il est recommandé de prendre la valeur threshold_damage = 0.6.

> **Solution :** Si la valeur de seuil est trop basse, il n'y a qu'un seul groupe. Si la valeur est trop élevée, il n'y aura pas assez de données dans chaque groupe.

Pour le moment, cette distinction en deux groupes n'est que visuelle. Il reste encore à la programmer pour obtenir deux listes de points distinctes. Au sein d'un code de calcul par éléments finis, il est possible d'utiliser la connectivité des éléments via les nœuds pour déterminer des ensembles de points endommagés voisins. Ne disposant pas de cette notion ici, il est possible de faire appel à une fonction de regroupement disponible dans la librairie Python `sklearn.cluster` : DBSCAN (Density-Based Spatial Clustering of Applications with Noise). Cette fonction va attribuer un `label` à chaque groupe sous forme d'entier à partir de 0 (-1 étant réservé aux points n'appartenant pas à un groupe particulier et pouvant donc être considéré comme du bruit). Pour l'utiliser, il faut enlever le mode commentaire pour le bloc "'Use DBSCAN to separate the damaged areas"'.

3.2. En considérant les valeurs dans `labels`, combien de groupes avez-vous ?

> **Solution :** Il faut obtenir deux groupes distincts. Il ne devrait pas y avoir de bruit, donc que deux labels : 0 et 1.

A partir de maintenant, chacune des deux zones sera considérée indépendamment et toutes les analyses suivantes seront effectuées dans une boucle abordant une zone à la fois.

4 Sélectionner les informations utiles dans chaque zone

Afin de traiter plus facilement chaque groupe de points, il est utile de définir des listes propres à chaque zone de positions de points d'intégration et de valeur d'endommagement (`local_X_position, local_Y_position` et `local_damage`).

4.1. En vous aidant de `labels`, compléter le bloc "'Consider each damaged area independently"' pour remplir les listes de données propres à chaque zone ci-dessus.

Solution :

```
for i in indices[0]:
    local_X_position.append(X_position_with_high_damage[i])
    local_Y_position.append(Y_position_with_high_damage[i])
    local_damage.append(high_damage[i])
```

5 Où insérer le premier incrément de fissure ?

La prochaine étape consiste ensuite à trouver le support du premier incrément de fissure.

5.1 Détermination d'un premier point sur la fissure

La plupart des méthodes commencent par déterminer un premier point censé appartenir à la future fissure. Le plus simple, en dimension 2, est de choisir le point avec le maximum d'endommagement dans la zone.

5.1. Dans le bloc "'Find the point with maximum damage"', sélectionner le point avec l'endommagement maximal dans la zone et stocker ses coordonnées dans (`X_max_damage, Y_max_damage`). Vérifiez visuellement sa position sur la carte d'endommagement.

Solution : Il faut utiliser la fonction `argmax` de numpy.

```
index_max = np.argmax(local_damage)
X_max_damage = local_X_position[index_max]
Y_max_damage = local_Y_position[index_max]

plt.scatter(local_X_position, local_Y_position, c=local_damage, cmap="jet")
plt.colorbar(orientation="vertical")
plt.title("Maximum damage point marked with a star for area with label %d"%(label))
plt.scatter(X_max_damage, Y_max_damage, color = "black", marker="*", s=100,
      label='maximum damage')
plt.axis("equal")
plt.legend()
plt.show()
```

Notez que la position du point avec le maximum d'endommagement ne se situe pas sur le bord, là où elle serait attendue. Cela est dû à l'utilisation d'une méthode

de régularisation pour calculer le champ étudié, ici la méthode non locale avec gradient implicite. Vu le faible écart, ce n'est pas un problème tant que le premier incrément de fissure intersecte également le bord de l'éprouvette.

5.2 Détermination d'un autre point sur la fissure

Positionnement de points d'échantillonnage

Pour déterminer un premier incrément (ou segment en 2D) de fissure, il faut un autre point sur la fissure. Il faut donc chercher un autre point où l'endommagement est maximal. Différentes possibilités existent. Si le point d'endommagement maximal se situe sur un bord, alors il suffit de chercher *le* prochain point où l'endommagement est maximal à une certaine distance (via un calcul de barycentre de points suffisamment endommagés ou la recherche du maximum de la projection de l'endommagement sur une droite). Or, dans le cas présent, le premier point n'est pas sur un bord mais à l'intérieur du domaine. Il faut donc déterminer non pas une, mais deux direction de fissuration possibles. Nous allons donc utiliser le même principe que celui du Marching Ridges afin de déterminer plusieurs directions simultanément : évaluer dans un repère cylindrique les angles correspondant à des maxima relatifs d'endommagement.

5.2. En enlevant le mode commentaire pour la partie "'Position evaluation points on a circle"', faire varier la valeur du rayon où sont positionnés les points d'évaluation autour de la valeur recommandée. Pourquoi faut-il éviter de choisir un rayon beaucoup plus grand ? Quels sont les inconvénients à choisir un rayon beaucoup plus petit ?

Solution : La valeur proposée (0.15), permet d'aller le plus loin possible tout gardant le cercle dans la zone de points sélectionnée pour une valeur seuil d'endommagement de 0.6. Au-delà, on va manquer d'informations pour évaluer l'endommagement. On risque également d'approcher un chemin de fissure un peu courbe par une droite : il faut faire attention à prendre un rayon suffisamment petit pour capturer la courbure locale. Il ne faut cependant pas choisir un rayon trop petit pour éviter de rester dans une zone de très fort endommagement où on ne pourra pas facilement distinguer la crête.

Évaluation de l'endommagement aux points d'échantillonnage

L'objectif des points d'échantillonnage est de permettre une analyse de l'évolution de l'endommagement. Afin d'évaluer l'endommagement, la fonction `second_order_polynomial_approximation` a été définie pour fournir une approximation par moindres carrés de la valeur de l'endommagement à un point donné. Cette fonction se sert de la valeur de l'endommagement aux 13 points les plus proches pour estimer au mieux les coefficients d'un polynôme de second ordre.

5.3. Reprendre la valeur proposée du rayon (0.15) et enlever le mode commentaire du bloc "'Evaluate damage on the circle"'. Visualiser pour un point d'échantillonnage donné les points les plus proches utilisés pour évaluer la valeur et imprimer les valeurs d'endommagement correspondantes ainsi que la valeur approchée au point d'échantillonnage. Cette dernière vous semble-t-elle correcte ?

Solution :

```
if eval_point == 0:
    print("damage_sample", damage_sample)
    print("estimated_damage", estimated_damage)
    plt.scatter(local_X_position, local_Y_position, c=local_damage, cmap="jet")
    plt.colorbar(orientation="vertical")
    plt.scatter(
        X_max_damage,
        Y_max_damage,
        color="black",
        marker="*",
        s=100,
        label="maximum damage",
    )
    plt.scatter(
        X_evaluation_point_position[eval_point],
        Y_evaluation_point_position[eval_point],
        color="gray",
        marker="x",
        s=50,
        label="evaluation point",
    )
    plt.scatter(X_sample, Y_sample, color="blue", marker="x", s=50, label="sampling points")
    plt.axis("equal")
    plt.legend()
    plt.title("Sampling points are used to estimate damage at each evaluation point")
    plt.show()
```

5.4. Vérifier visuellement les valeurs d'endommagement obtenues pour tous les points d'évaluation par rapport aux valeurs des points avoisinants. Ces valeurs vous semblent-t-elles correctes ?

5.5. Tracer l'évolution de l'endommagement en fonction de l'angle. Selon vous, quelles

sont les directions correspondant à la crête d'endommagement ?

Solution :

```
plt.plot(evaluation_angles, approx_damage, "*-", color="blue")
plt.title(
    "Evolution of damage with the angular position between %d" % evaluation_angles[0]
    + " and %d degrees" % evaluation_angles[-1]
)
plt.show()
```

Les tendances globales montrent deux directions correspondant à la crête. Cependant, on peut déjà remarquer que le signal est bruité et qu'il y a plus de maxima locaux que voulu.

5.6. Enlever le mode commentaire sur le bloc """Damage evolution analysis to find relative max values""" pour trouver les valeurs relatives maximales données par la fonction `argrelmax` de `scipy.signal`. Est-ce que cela correspond à vos attentes ? Pourquoi avoir pris la peine de définir le nouvel array `wrappped_approx_damage` ?

Solution : Avec les valeurs estimées, on se retrouve avec des maxima locaux non désirés dûs au fait que la courbe n'est pas suffisamment lisse. L'array `wrappped_approx_damage` sert à prendre en compte le cas où il y aurait un maximum relatif pour la direction zéro degrés.

Réduire la dépendance à la discrétisation

Avec les données brutes, il peut y avoir trop de bruit pour trouver les 2 directions attendues pour chaque zone. On voit ici l'influence de la discrétisation du calcul par éléments finis qui explique que le recours à un pré-traitement de lissage des données avant analyse par les algorithmes de *crack path tracking*. D'une manière générale, la solution consiste à lisser le champ scalaire étudié en entrée et à utiliser des moyennes sur plusieurs rayons pour déterminer les directions. Pour ce TP, une solution plus économique est simplement de lisser la courbe, par exemple à l'aide d'un filtre de Savitzky-Golay.

5.7. Enlever le mode commentaire sur le bloc """Damage evolution analysis on smoothed curved to find relative max values""" pour trouver les valeurs relatives maximales données par la fonction `argrelmax` de `scipy.signal`. Est-ce que cela correspond plus à vos attentes ?

5.8. Tracer les segments correspondants pour chaque zone et vérifier la cohérence par rapport aux points endommagés sélectionnés.

Solution :

```
for num_branch in np.arange(0, len(max_indices)):
    X_ridge_point = X_max_damage + radius * math.cos(
        wrapped_evaluation_angles[max_indices[num_branch]] * math.pi / 180.0
    )
    Y_ridge_point = Y_max_damage + radius * math.sin(
        wrapped_evaluation_angles[max_indices[num_branch]] * math.pi / 180.0
    )
```

6 Jusqu'où insérer le premier incrément de fissure ?

Une fois le support d'une fissure déterminé, il reste à savoir quelle partie de ce support correspond réellement à un incrément de fissure à insérer. On fait pour cela appel à un critère d'insertion qui compare la valeur de l'endommagement à une valeur critique.

6.1 Évaluer l'endommagement le long du support

6.1. Pour chaque direction de fissure possible, distribuer des points pour y évaluer l'endommagement. Vous pouvez choisir de distribuer une vingtaine de points avec un pas égal à 20% du rayon considéré. Stocker leurs coordonnées dans les listes `X_on_segment` et `Y_on_segment` et les valeurs d'endommagement correspondantes dans une liste nommée `damage_on_segment`.

Solution :

```
for multip in np.arange(0, 20):
    new_radius = multip * delta_L
    X_point = X_max_damage + new_radius * math.cos(
        wrapped_evaluation_angles[index_angle] * math.pi / 180.0
    )
    Y_point = Y_max_damage + new_radius * math.sin(
        wrapped_evaluation_angles[index_angle] * math.pi / 180.0
    )
    X_sample, Y_sample, damage_sample = find_closest_points(
        X_point, Y_point, local_X_position, local_Y_position, sample_size, local_damage
    )
    estimated_damage = second_order_polynomial_approximation(
        X_point, Y_point, X_sample, Y_sample, damage_sample
    )
    X_on_segment.append(X_point)
```

```
    Y_on_segment.append(Y_point)
    damage_on_segment.append(estimated_damage)
```

6.2 Utiliser un critère d'insertion

Il faut enfin choisir une valeur d'endommagement suffisamment proche de la borne supérieure (1) qui corresponde à l'apparition d'une fissure : considérons par exemple 0.95.

6.2. Tracer l'évolution de l'endommagement en fonction des coordonnées suivant l'axe X des points d'échantillonnage et ainsi que la droite correspondant à la valeur limite dans une même figure pour chacun des supports de fissure envisagé.

Solution :

```
plt.plot(X_on_segment, damage_on_segment, color="black", label="damage")
plt.plot(
    X_on_segment,
    limit_damage * np.ones(len(X_on_segment)),
    color="red",
    label="limit value of %f" % limit_damage,
)
plt.title(
    "Evolution of damage along the X-axis for angle: %d deg"
    % wrapped_evaluation_angles[index_angle]
)
plt.legend()
plt.show()
```

6.3. Combien de segments de fissure sont à insérer ?

Solution : Pour la première zone, en bas à gauche, les valeurs d'endommagement sont en-dessous du seuil. Pour la seconde zone, en haut à droite, une partie de chacun des deux segments correspond à un endommagement suffisamment élevé pour être insérée.

6.4. En comparant la valeur d'endommagement à chacun des points d'échantillonnage à la valeur critique, ne sélectionner que les points sur la partie du support à insérer et stocker leurs coordonnées dans les listes `selected_X_on_segment` et `selected_Y_on_segment`. Une précision de l'ordre de 20% du rayon d'évaluation sera suffisante ici pour comprendre les grands principes.

Solution :

```
for index_point in np.arange(0,len(X_on_segment)):
        estimated_damage = damage_on_segment[index_point]
        if estimated_damage >= limit_damage:
            selected_X_on_segment.append(X_on_segment[index_point])
            selected_Y_on_segment.append(Y_on_segment[index_point])
```

6.5. Tracer les parties à insérer pour chaque support.

Solution :

```
plt.scatter(local_X_position, local_Y_position, c=local_damage, cmap="jet")
plt.colorbar(orientation="vertical")
plt.scatter(
    X_max_damage, Y_max_damage, color="black", marker="*", s=100, label="maximum
        damage"
)
plt.plot(selected_X_on_segment, selected_Y_on_segment, "-*", c="black")
plt.axis("equal")
plt.legend()
plt.title(
    "Part of the segments above limit damage for angle: %d deg"
    % wrapped_evaluation_angles[index_angle]
)
plt.show()
```

6.6. Stocker les coordonnées des points extrémités des segments dans les listes `X_all_segments` et `Y_all_segments` définies au début du fichier. Tracer les parties à insérer pour chaque support.

Solution :

```
if len(selected_X_on_segment) > 0:
    X_all_segments.append([X_max_damage, selected_X_on_segment[-1]])
    Y_all_segments.append([Y_max_damage, selected_Y_on_segment[-1]])
```

7 Résultat final

7.1. En vous aidant des listes `X_all_segments` et `Y_all_segments`, tracer tous les segments de fissure à insérer sur la carte d'endommagement initiale pour la comparer qualitativement avec votre prédiction.

Solution :

```
plt.scatter(X_position, Y_position, c=damage, cmap="jet", marker=".")
plt.colorbar(orientation="vertical")
for index in np.arange(0, len(X_all_segments)):
    plt.plot(X_all_segments[index], Y_all_segments[index], "-*", c="black")
plt.axis("equal")
plt.title("Crack increments ready for insertion for time %f" % selected_time)
plt.show()
```

8 Pour aller plus loin

La combinaison de choix qui vous ont été proposés a permis d'obtenir une discrétisation de la fissure (ou des fissures) à insérer dans le modèle. Comme illustré dans le cours, d'autres options sont possibles pour chaque étape. Si vous en avez le temps, voici quelques options à explorer :

- Reprendre la recherche d'un premier point sur la fissure (5.1) en cherchant le barycentre de points suffisamment endommagés.
- Reprendre la détermination d'un autre point sur la fissure (5.2) en utilisant une régression linéaire sur la position des points les plus endommagés pour approcher le premier incrément de fissure par un segment de droite.

De la même manière, le code obtenu permet de traiter un cas particulier de manière satisfaisante. Pour traiter plus de cas de manière plus robuste, des petites modifications sont possibles et peuvent être également explorées :

- utiliser le champ (non borné) de plasticité cumulée non locale pour la direction et le champ d'endommagement pour le critère d'insertion ;
- ajouter une fonction poids dans l'approximation par moindres carrés (5.2) pour donner moins de poids aux points d'échantillonnage les plus éloignés ;
- déterminer plus précisément la position de la pointe de fissure pour 6.2 grâce à une interpolation linéaire en se servant des coordonnées et des valeurs aux points avant et après intersection avec la valeur critique.

Enfin, seule l'étape d'amorçage a été traitée. Il est possible d'aller plus loin en :

- programmant la recherche du prochain point sur la crête à partir des extrémités pertinentes pour la même carte (*exhaustion method*) ;
- programmant la recherche pour tous les pas de temps en exploitant les segments déjà trouvés (propagation de fissure).

N'hésitez pas à tester les pistes qui vous inspirent !