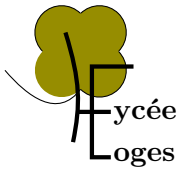
Nom : .....

Prénom : .....

2024/02/27

Classe : MP/PSI



Concours Blanc Informatique
- 3h -
type E3A/CCP

Note finale : ···· / 20

⚠ Calcuratrice interdite - Faites attention à vos explications

Assemblage et déformation de pièces automobiles

Introduction

L'objectif de cette étude est la modélisation du comportement géométrique de pièces déformables par la méthode des éléments finis. La **partie I** déterminera le posage des pièces et sera illustrée par un modèle 2 dimensions. La **partie II** permettra d'établir le maillage d'une surface plane pour pouvoir appliquer dans la **partie III** la méthode de résolution par éléments finis. La **partie IV** mettra en œuvre une base de données simplifiée.

Dans tout le sujet, il sera supposé que les modules python `numpy`, `matplotlib.pyplot` sont déjà importés dans le programme. Les fonctions et méthodes autorisées sont indiquées dans l'**annexe**. De manière générale, l'utilisation des méthodes autres que les méthodes `append` et `remove` ne sera pas acceptée. L'utilisation des fonctions `min` ou `max` sera proscrite.

I Partie : Posage des pièces

Présentation

Le posage consiste à mettre deux pièces en contact. Les pièces n'étant jamais parfaites, le nombre des points en contact n'est jamais infini comme cela pourrait être le cas lorsque l'on pose 2 surfaces parfaitement planes l'une sur l'autre. Ces points de contacts ponctuels doivent être identifiés car ils peuvent être à l'origine de variations géométriques et d'efforts particuliers.

FIGURE 1 – Illustration du problème de posage sur des surfaces réelles

Pour illustrer la question du posage, on travaille en 2 dimensions et on se limite au posage d'une pièce sur une autre. Les géométries ont été discrétisées en un nombre fini de points. Un prétraitement, qui ne sera pas abordé ici, permet d'extraire les coordonnées des points concernés par le posage. En 2 dimensions, pour chaque point d'abscisse $x_k = k\Delta x$, est associée son altitude z_k où Δx correspond à une longueur élémentaire associée à la discrétisation spatiale. Ces données sont stockées dans une liste `P` de N nombres dont l'indice k des éléments permet de définir l'abscisse des points et les valeurs leur altitude. Ainsi, $x_k = k\Delta x$ et $z_k = P[k]$ où $0 \leqslant k < N$. La variable `Delta_x` sera associée à la longueur élémentaire Δx et sera supposée déjà définie. Il sera inutile de la passer en argument des différentes fonctions.

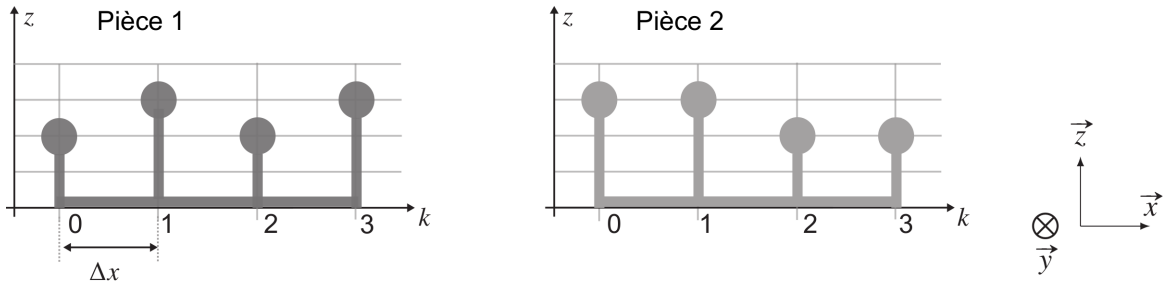


FIGURE 2 – Exemple de géométries discrètes associées à des pièces en 2D

Par exemple, sur la figure 2, on associe à la pièce de gauche la liste `P1 =[2.,3.,2.,3.]`.

Q1. Donner la liste `P2` associée à la pièce 2 sur la figure 2.

Pour répondre au problème du posage, il faut commencer par retourner la pièce en effectuant une rotation autour de l'axe $(O, \vec{y})$, pour que les points soient bien en vis-à-vis.

FIGURE 3 – Exemple de posages entre 2 pièces. À gauche : posage aux points 1 et 3. À droite : posage aux points 1 et 2 avec collision en 3

Q2. Écrire une fonction `retourne(Ls)` qui prend une liste `Ls` de nombres en argument et qui renvoie une nouvelle liste dont l'ordre des éléments est inversé et le signe de chaque élément lui aussi inversé. Par exemple, `retourne([1.,2.,3.])` renverra `[-3.,-2.,-1.]`.

On appellera posage optimal, le posage qui vérifie les deux contraintes suivantes :

- ◇ le minimum des distances algébriques entre les points en vis-à-vis doit être positif pour valider le fait qu'il n'y ait pas de collision,
- ◇ le maximum des distances entre les points en vis-à-vis doit être le plus petit possible.

Q3. À partir des deux pièces de la figure 3, représenter la pièce 2 associée au posage (contact) aux points 0 et 1, puis la pièce 2 associée au posage aux points 2 et 3. Indiquer dans chaque cas si le posage génère des collisions ou non.

Répondre sur le document réponse.

Pour caractériser le posage, la méthode mise en œuvre consiste à repérer la position des points du maillage de la deuxième pièce par rapport à ceux de la première pièce. On se placera dans le cadre d'une pièce faiblement inclinée (sur la figure 4, l'inclinaison est amplifiée). Pour chacune des deux pièces, on associe une droite passant par les 2 points de contacts. Les deux droites ainsi définies seront coïncidentes lors du contact entre les deux pièces comme l'illustre la figure 4.

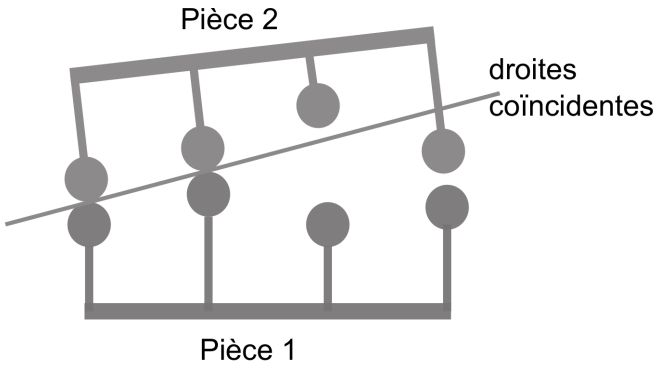


FIGURE 4 – Exemple de posage associé à deux autres pièces

On considère le posage sur les points n et p. On note z_{rn} et z_{rp} les altitudes des points de contact sur la pièce référence, et z_{pn} et z_{pp} les altitudes des points de contact sur la pièce à poser.

L'équation d'une droite qui passe par deux points de contact ayant pour coordonnées $(n\Delta x, z_n)$ et $(p\Delta x, z_p)$ est :

$$z = ax + b \tag{1}$$

avec pour coefficient directeur a et pour ordonnée à l'origine b tels que :

$$a = \frac{z_n - z_p}{(n-p)\delta x} \qquad \text{et} \qquad b = z_n - \frac{n(z_n - z_p)}{n-p}$$

Q4. Préciser les coefficients a et b de l'équation $z = ax + b$ d'une droite qui passe par deux points de contact ayant pour coordonnées $(n\Delta x, z_n)$ et $(p\Delta x, z_p)$

Q5. Écrire la fonction `droite(Ls, n, p)` qui prend en arguments la liste des altitudes `Ls` et deux entiers `n` et `p` correspondant aux indices des deux points de contact et qui renvoie une liste contenant la pente a et l'ordonnée à l'origine b de la droite associée.

On applique cette fonction au bâti (pièce 1) et à la pièce 2 retournée. Pour un point donné d'une pièce donnée, on définit son altitude comme la distance algébrique entre ce point et le point sur la droite de même abscisse. Cette altitude sera comptée positivement si le point est audessus de la droite et négativement s'il est situé endessous. Pour une abscisse donnée, la distance algébrique entre deux points en visàvis (chacun appartenant à une des deux pièces) est la différence entre leur altitude respective par rapport à la droite.

Q6. Écrire une fonction `posage(Ls1, Ls2, n, p)` qui prend en arguments 2 listes de points (`Ls1` et `Ls2`) et 2 entiers `n` et `p` qui correspondent aux indices des points de contact. La pièce associée à `Ls1` est située endessous de la pièce associée à `Ls2` (qui a été préalablement retournée). Cette fonction renvoie une liste de 2 éléments correspondant à la distance algébrique maximale et à la distance algébrique minimale entre les pièces discrétisées dans cette situation. Cette fonction devra faire appel à la fonction précédente `droite`.

Q7. Écrire une fonction `posage_opt(Ls1, Ls2)` qui prend en arguments 2 listes de même taille `Ls1` et `Ls2` (dans la position retournée) et qui renvoie une liste de 2 éléments correspondant aux abscisses des points associés au posage optimal. Cette fonction doit faire appel à la fonction définie en **Q5**. On ne traitera pas le cas où plusieurs couples de points peuvent conduire au posage optimal. Indiquer en justifiant succinctement la complexité de cette fonction en terme de nombre de comparaisons.

II Partie : Maillage de la surface

On souhaite appliquer la méthode des éléments finis pour déterminer la déformation de la surface suite aux contraintes qui découlent du posage. Cette méthode nécessite, au préalable, un découpage de la surface en surfaces élémentaires. Cette étape s'appelle le maillage.

Le maillage proposé lors de cette étude est un maillage triangulaire. Chaque surface élémentaire est alors un triangle dont les sommets sont des nœuds de la surface à étudier. Le maillage est une étape déterminante permettant d'obtenir des résultats cohérents ou non par la méthode des éléments finis. En effet, un mauvais maillage comme illustré à gauche figure 5 peut être à l'origine d'instabilités numériques ou encore d'erreurs d'arrondis. Au contraire, une triangulation de Delaunay illustrée à droite figure 5 présente de bons résultats lors d'un traitement numérique. Dans ce maillage, les triangles ont des arêtes courtes, des angles ni trop petits, ni trop grands.

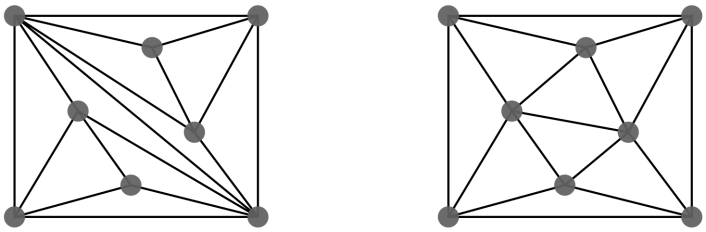


FIGURE 5 – Exemple de 2 triangulations obtenues à partir de nœuds identiques

Une triangulation est dite de **Delaunay** lorsque le cercle circonscrit à chaque triangle ne contient aucun autre point du maillage.

Une triangulation peut devenir une triangulation de Delaunay par une suite de basculements (flips) successifs : quand 2 triangles ne respectent pas la condition de Delaunay, il faut remplacer l'arête commune P_2P_4 par l'arête P_1P_3 comme l'illustre la figure 6. L'algorithme mis en œuvre pour réaliser une triangulation de Delaunay est un algorithme itératif. La construction s'appuie sur un maillage initial de Delaunay. Ce maillage constitué d'un faible nombre de nœuds est construit de telle sorte que la surface ainsi délimitée contienne l'ensemble des nœuds qui devront être ajoutés à chaque itération. Ce maillage initial peut être composé d'un ou de plusieurs triangles vérifiant le critère de Delaunay.

Chaque nœud P_k restant du maillage est alors ajouté en suivant l'algorithme :

- ◇ repérer à quel triangle T_{in} appartient le point P_k (par exemple sur la figure 7, P_4 appartient au triangle $P_0P_1P_2$),

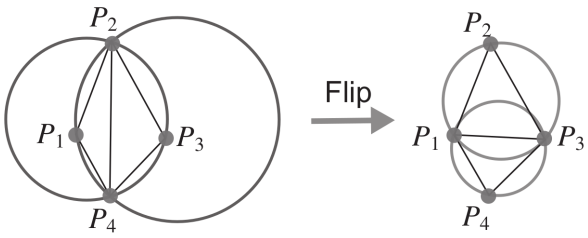


FIGURE 6 – Principe du basculement

◇ construire 3 triangles à partir du point P_k et des sommets du triangle T_{in}. Ajouter ces triangles à la liste des triangles `T`. Supprimer le triangle T_{in} de la liste `T`.

La triangulation ainsi obtenue n'est pas nécessairement de Delaunay. Pour chaque nouveau triangle :

— vérifier que les triangles voisins respectent la triangulation de Delaunay. Si ce n'est pas le cas, basculer l'arête commune (figure 6).

— répéter cette opération pour les triangles modifiés après le basculement.

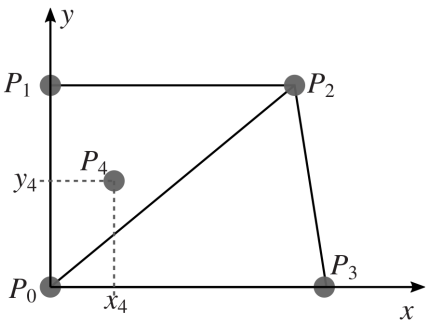


FIGURE 7 – Ajout d'un point à un maillage de Delaunay

Structure des données

On dispose de N nœuds P_k d'une surface plane repérés par leurs coordonnées cartésiennes (x_k, y_k). Ces nœuds sont stockés dans le tableau `noeud` qui est un tableau `Numpy` de flottants de taille $N \times 2$. Chaque nœud est repéré par son indice dans le tableau `noeud`. L'élément k du tableau `noeud` contient les coordonnées cartésiennes du nœud d'indice k. Ainsi `noeud[k] = array([xk,yk])`.

On souhaite construire un tableau `T` qui contient des listes à 3 éléments. Ces listes représentent l'indice des nœuds d'un même triangle. Dans le cas de la figure 7, la liste `T` s'écrit : `T = [[0,1,2],[2,3,0]]`. L'ordre des indices des nœuds est sans importance.

Q8. On considère le maillage de Delaunay de la figure 7. On souhaite ajouter le nœud P_4 à ce maillage et faire les transformations nécessaires pour obtenir une nouvelle triangulation de Delaunay. Sur les figures du document réponse, représenter les différentes transformations qui permettent d'aboutir à une triangulation de Delaunay. Indiquer la liste `T` qui résulte de cette triangulation.

L'algorithme de Delaunay doit permettre de construire la liste `T` des triangles du maillage. Cette liste est initialisée par 2 triangles contenant tous les nœuds du maillage. Ensuite, à chaque nouveau nœud ajouté, on applique l'algorithme de triangulation, pour mettre à jour la liste `T`.

L'implémentation de cet algorithme nécessite l'utilisation de plusieurs fonctions :

◇ `addT(indN, indT)` : fonction qui prend en arguments l'indice d'un nœud `indN` à l'intérieur du triangle d'indice `indT`.

Cette fonction modifie la liste `T` en insérant les nouveaux triangles et en supprimant le triangle d'indice `indT` ;

◇ `circle(T1)` : fonction qui prend en argument une liste à 3 éléments (les indices des sommets d'un triangle) et qui renvoie les coordonnées du centre du cercle circonscrit ainsi que le rayon `[[xc,yc], R]` ;

- ◇ `insideT(indN, T1)` : fonction qui prend en arguments l'indice d'un nœud `indN` et une liste correspondant aux indices des sommets d'un triangle `T1`. Cette fonction renvoie un booléen suivant si le nœud appartient ou non au triangle considéré ;
- ◇ `insideC([xc,yc], R, [x,y])` : fonction qui prend en argument les coordonnées du centre, le rayon et les coordonnées d'un point. Cette fonction renvoie un booléen suivant si le point considéré appartient ou non au cercle de centre (xc,yc) de rayon R ;
- ◇ `edge(indN, indT)` : fonction qui prend en arguments l'indice d'un nœud `indN` et l'indice d'un triangle `indT`. Cette fonction renvoie l'indice du triangle ayant pour arête commune l'arête à l'opposé du nœud considéré ;
- ◇ `flip(indT1, indT2)` : fonction qui prend en arguments 2 indices (associés à des triangles de la liste `T`). Cette fonction permet de basculer l'arête commune en modifiant en conséquence la liste `T`.

Les tableaux `noeud` et `T` sont des variables globales qu'il sera inutile de passer en arguments des fonctions et qui pourront être utilisées et modifiées à l'intérieur de cellesci. Dans la suite du sujet, on considérera que les différents nœuds sont toujours strictement inclus dans un triangle de la liste `T` et qu'ils ne sont jamais situés sur l'une des arêtes.

Q9. Écrire la fonction `addT(indN, indT)` décrite cidessus.

Q10. La fonction `flip(indT1, indT2)` qui permet de basculer l'arête commune à 2 triangles en modifiant les sommets de 2 triangles de la liste `T` est définie cidessous. Choisir l'une des 3 propositions données pour compléter les instructions manquantes (indiquées par instructions à compléter).

Fonction `flip`

```
1 def flip(indT1, indT2):
2     T1 = T[indT1]
3     T2 = T[indT2]
4     Ledge, Lflip = [], []
5
6     # instructions à compléter
7
8     T[indT1] = Lflip + [Ledge[0]]
9     T[indT2] = Lflip + [Ledge[1]]
```

Proposition 1

```
1     Ls = T1 + T2
2     for k in range(len(Ls)):
3         for j in range(k+1, len(Ls)):
4             if Ls[k] == Ls[j]:
5                 Ledge += [ Ls[k] ]
6             else:
7                 Lflip += [ Ls[k] ]
```

Proposition 2

```
1     for k in range(3):
2         if T2[k] in T1:
3             Ledge += [ T2[k] ]
4             else:
5                 Lflip += [ T2[k] ]
6     for noeud in T1:
7         if noeud not in Ledge:
8             Lflip += [ noeud ]
```

Proposition 3

```
1     for k in range(3):
2         if T2[k] == T1[k]:
3             Ledge += [ T2[k] ]
4             else:
5                 Lflip += [ T2[k] ]
6                 Lflip += [ T1[k] ]
```

Pour vérifier si un point M du plan cartésien appartient à un triangle ABC, on propose de faire le produit vectoriel $\overrightarrow{AB} \wedge \overrightarrow{AM}$ et de vérifier que le résultat est du même signe que le produit vectoriel $\overrightarrow{AB} \wedge \overrightarrow{AC}$. On procédera de la même manière pour chaque sommet en comparant les signes de $\overrightarrow{BC} \wedge \overrightarrow{BM}$ à $\overrightarrow{BC} \wedge \overrightarrow{BA}$ et enfin de $\overrightarrow{CA} \wedge \overrightarrow{CM}$ à $\overrightarrow{CA} \wedge \overrightarrow{CB}$. L'annexe du sujet rappelle la syntaxe et la définition du produit vectoriel sous `numpy`.

Q11. Écrire la fonction `insideT(indN, T1)` qui prend en argument l'indice d'un nœud `indN` et une liste `T1`, associée à un triangle. Cette fonction renvoie un booléen suivant si le nœud appartient au triangle ou non (`True` si le noeud appartient au triangle, `False` sinon).

Une fois le nœud d'indice `indN` inséré, pour se ramener à une configuration de Delaunay, on applique la fonction récursive `checkedge(indN, indT)` où `indN` est l'indice du nœud inséré, et `indT` un triangle qui a pour sommet le nouveau nœud. Cette fonction doit d'abord vérifier si le nœud inséré est inclus dans les cercles circonscrits des triangles voisins, c'estàdire, les triangles qui possèdent une arête commune aux nouveaux triangles. Par exemple, sur la figure 7, P_0P_2 est une arête commune à un nouveau triangle formé par l'insertion du nœud P_4 et au triangle défini par les nœuds P_0, P_2 et P_3. Si le nœud inséré

appartient effectivement à l'un des cercles circonscrits, il faut alors procéder à un basculement grâce à la fonction `flip`. Une fois ce basculement réalisé, il faut à nouveau s'assurer que le nœud d'indice `indN` et les triangles ainsi modifiés vérifient le critère de Delaunay en appliquant la fonction `checkedge`.

```
1 def checkedge(indN, indT):
2     indA = edge(indN, indT)
3     if indA != None:
4         C, R = cercle( T[indA] )
5         if insideC( C, R, noeud[indN] ):
6             # instruction à compléter
7             # instruction à compléter
8             # instruction à compléter
```

Q12. Indiquer l'intérêt de la condition ligne 3 de la fonction `checkedge`. Compléter les lignes 6, 7 et 8 de cette fonction.

Q13. Écrire la fonction `delaunay(noeud)` qui prend en arguments un tableau de nœuds (de dimension $N \times 2$) et qui renvoie la liste des triangles `T` correspondant à une triangulation de Delaunay. Cette fonction pourra utiliser les fonctions `insideT`, `addT` et `checkedge` déjà définies. On considère toujours que chaque nœud inséré est strictement inclus dans un triangle de la distribution. De plus, les 4 premiers éléments du tableau `noeud` permettent d'initialiser le tableau des triangles : `T = [[0,1,2],[2,3,0]]`.

III Partie : Détermination de la matrice de rigidité d'une pièce

À partir de ce maillage, l'objectif est maintenant de calculer numériquement la déformation de la pièce étudiée.

Dans la suite du sujet, on considérera que toutes les constantes `N`, `E`, `S`, `L`, `Fx`, `p0` et `n` ont été préalablement définies de manière globale.

Pour illustrer cette partie, nous travaillons sur un modèle 1D où 2 nœuds du maillage sont reliés par une barre.

1 Modélisation mécanique

Chaque barre se déforme proportionnellement à l'effort subit, comme un ressort. La raideur dépend de la section de la barre S de sa longueur L et de l'élasticité du matériau E (module d'Young) :

$$\overrightarrow{F} = k\Delta L\overrightarrow{v} = \frac{ES}{L}\Delta L\overrightarrow{v} \tag{2}$$

où $\overrightarrow{v}$ est un vecteur unitaire de même direction que la barre (figure 8).

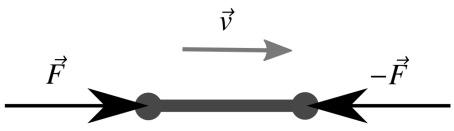


FIGURE 8 – Efforts exercés au niveau des barres

On considère que les efforts extérieurs ne s'exercent que sur les nœuds. Le problème consiste à déterminer les déplacements des nœuds.

Sur des structures complexes, le système d'équations peut atteindre plusieurs centaines de milliers voire des millions d'inconnues. Une mise en œuvre numérique est alors indispensable.

Calcul de la déformation d'une poutre en traction

Pour introduire la méthode de résolution par éléments finis, nous allons travailler sur un problème simple à une dimension : la poutre en traction.

La poutre représentée figure 9 est encastrée en O. Cette poutre de longueur L de section S est soumise à un effort $F_x = 100$N en A et à un effort réparti, également suivant $\overrightarrow{x}$, représenté par une densité linéique de force $p(x) = p_0 = 500$ N/m (non réprésenté).

La théorie de la résistance des matériaux nous fournit les équations suivantes :

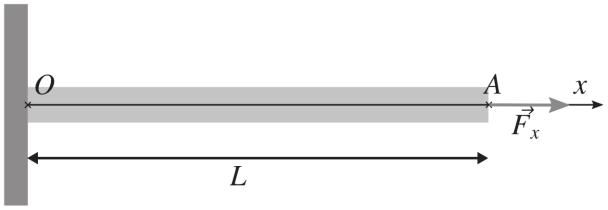


FIGURE 9 – Paramétrage du problème de la poutre

$$\begin{cases} \dfrac{\mathrm{d}N}{\mathrm{d}x} + p(x) = 0 \\ N = ES\dfrac{\mathrm{d}u}{\mathrm{d}x} \end{cases} \quad \text{avec comme conditions aux limites} \quad \begin{cases} u(0) = 0 \\ N(L) = F_x \end{cases}$$

où N correspond à l'effort normal dans la poutre et u le déplacement longitudinal dû à la traction. En combinant ces deux premières équations, on obtient :

$$ES\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = -p(x) \tag{3}$$

Après intégration, on obtient simplement l'expression analytique suivante :

$$u_{th}(x) = -\frac{p_0}{2ES}x^2 + \frac{(F_x + Lp_0)}{ES}x$$

Tracé de la solution analytique

Q14. Écrire les instructions permettant de créer un vecteur `x` de n points régulièrement répartis entre 0 et L inclus, puis permettant de créer le vecteur `uth` représentant le déplacement dans la poutre des points d'abscisse `x`. Enfin, écrire les instrutions permettant de tracer la solution analytique $u_{th}(x)$.

L'équation différentielle peut être résolue analytiquement dans ce cas simple, mais dans le cadre général des problèmes d'élasticité 2D ou 3D, ce n'est pas possible. On envisage alors une résolution numérique en appliquant la méthode d'approximation que sont les éléments finis.

Recherche de la solution par éléments finis

La méthode de résolution par éléments finis consiste à transformer l'équation différentielle en un système linéaire d'équations.

En repartant de l'équation (3), la multiplication de l'équation différentielle par une fonction test Φ, puis son intégration permettent d'aboutir à la relation suivante :

$$\int_0^L ES\frac{\mathrm{d}^2u}{\mathrm{d}x^2}\Phi(x)\mathrm{d}x = \int_0^L -p(x)\Phi(x)\mathrm{d}x$$

où la fonction test Φ (dont un exemple est donné figure 10) doit vérifier les mêmes conditions aux limites que la fonction u, soit $\Phi(0) = 0$.

Une intégration par parties nous amène au résultat suivant :

$$\int_0^L ES\frac{\mathrm{d}u}{\mathrm{d}x}\frac{\mathrm{d}\Phi}{\mathrm{d}x}\mathrm{d}x = \int_0^L p(x)\Phi(x)\mathrm{d}x + N(L)\Phi(L) \tag{4}$$

Nous allons alors chercher une solution linéaire par morceaux à l'équation (4) ce qui va nous amener à la discrétisation du problème et à son écriture matricielle.

Nous cherchons alors $u(x)$ sous la forme :

$$u(x) = \sum_{k=0}^{n} u_k\Phi_k(x) \tag{5}$$

où les fonctions Φ_k sont appelées fonctions de forme illustrées sur la figure 10. Ces fonctions ont les propriétés suivantes :

- $x_k = kdx = k\dfrac{L}{n}$ et $0 \leqslant k \leqslant n$;

- ◇ Φ_k est continue ;
- ◇ Φ_k est affine sur $[x_{k-1}, x_k]$ et sur $[x_k, x_{k+1}]$;
- ◇ $\Phi_k(x_k) = 1$;
- ◇ $\forall x \in [0,x_{k-1}] \cup [x_{k+1},L]$, $\Phi_k(x) = 0$;
- ◇ même si les fonctions Φ_k ne sont pas dérivables en tout point de $[0,L]$, on considèrera les dérivées $\dfrac{\mathrm{d}\Phi_k}{\mathrm{d}x}$ comme des fonctions continues par morceaux, en les prolongeant de manière quelconque aux points de "cassure". Le choix du prolongement n'aura de toute façon aucun impact sur les calculs d'intégrales demandés.

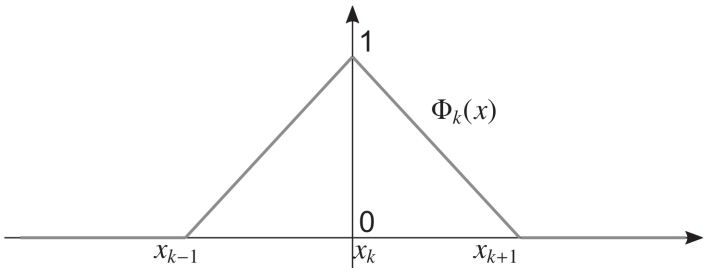


FIGURE 10 – Fonction de forme $\Phi_k(x)$

Les scalaires u_k constituent les inconnues du problème et représentent le déplacement associé à chaque nœud k. Une fois les déplacements u_k déterminés, il est possible de calculer la fonction déplacement u en n'importe quel point x de la poutre grâce à l'équation (5).

Q15. Écrire une fonction `phi(k, x)` qui prend en argument un entier k et un flottant $x \in [0, L]$ et qui renvoie la valeur $\Phi_k(x)$.

Nous pouvons alors réécrire le problème sous forme matricielle. En notant `U` et `Phi` les vecteurs respectivement associés aux déplacements u_k et aux fonctions $\Phi_k(x)$, on peut écrire la fonction u de la manière suivante :

$$U = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{pmatrix} \qquad \Phi(x) = \begin{pmatrix} \Phi_0(x) \\ \Phi_1(x) \\ \vdots \\ \Phi_n(x) \end{pmatrix} \qquad u(x) = {}^t U \Phi(x) \tag{6}$$

où ${}^tU$ correspond à la transposée du vecteur U. La dérivée de u par rapport à x s'écrit alors simplement :

$$\frac{\mathrm{d}u}{\mathrm{d}x} = {}^t U \frac{\mathrm{d}\Phi}{\mathrm{d}x} \tag{7}$$

Pour résoudre l'équation (4) il faut alors déterminer la matrice $M = \displaystyle\int_0^L \frac{\mathrm{d}\Phi}{\mathrm{d}x}^t \frac{\mathrm{d}\Phi}{\mathrm{d}x} \mathrm{d}x$.

Q16. Exprimer le produit des dérivées $\dfrac{\mathrm{d}\Phi_i}{\mathrm{d}x}^t \dfrac{\mathrm{d}\Phi_j}{\mathrm{d}x}$ pour $x \in [0, L]$ en fonction de n et de L lorsque $i = j$, $i = j = 0$, $i = j = n$, $|i - j| = 1$ et dans les autres cas. En déduire l'intégrale $\displaystyle\int_0^L \frac{\mathrm{d}\Phi_i}{\mathrm{d}x}^t \frac{\mathrm{d}\Phi_j}{\mathrm{d}x} \mathrm{d}x$ à nouveau en fonction de n et de L pour toutes les valeurs de i et j vues.

Les fonctions de forme ainsi choisies permettent d'aboutir à une matrice M tridiagonale :

$$M = C \begin{pmatrix} \alpha/2 & \beta & & & \\ \beta & \alpha/2 & \beta & & \\ & \ldots & \ldots & \ldots & \\ & & \beta & \alpha/2 & \beta \\ & & & \beta & \alpha/2 \end{pmatrix}$$

Q17. À partir des résultats de la question précédente et en posant $\beta = -1$, exprimer les coefficients C et α.
Le problème peut alors se réécrire, à partir de l'équation (4) de la manière suivante :

$$ESMU = P + F_x \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \tag{8}$$

où P est le vecteur associé à l'effort appliqué aux nœuds de la poutre. Par la suite, on considère que le vecteur P et la matrice M ont été définis comme des tableaux `numpy`.
Cette équation peut se mettre sous la forme suivante et correspond à un système linéaire de n équations à n inconnues :

$$AU = B \tag{9}$$

Q18. Exprimer la matrice A et le vecteur B en fonction de M, P et des différentes constantes.
Écrire les instructions permettant de créer A et B.
La résolution d'un tel système linéaire peut se faire à l'aide de l'algorithme du pivot partiel de Gauss.
La fonction `resolution` prend en arguments un tableau `R` de dimensions $(n \times n)$ de type flottant et un vecteur à n éléments de type flottant. Cette fonction renvoie un vecteur à n éléments solution de l'équation $RX = Y$.
Les fonctions `combinaison` et `echanger_ligne` permettent respectivement de faire une combinaison linéaire entre deux lignes du système et d'échanger deux lignes du système.

```
1  def resolution(R, Y):
2      Syst = []
3      nb = len(M)
4      for k in range(nb):
5          ligne = []
6          for j in range(nb):
7              ligne.append( R[k, j] )
8          ligne.append( Y[k] )
9          Syst.append( ligne )
10     # Mise sous forme triangulaire
11     for i in range(nb):
12         j = pivot(Syst, i)
13         echanger_lignes(Syst, i, j)
14         for k in range(i+1, nb):
15             # instruction à compléter
16             combinaison(Syst, k, i, mu)
17     # Remontée
18     X = [0]*nb
19     # instructions à compléter
20     return X
```

Q19. Indiquer l'intérêt des lignes 2 à 9 de la fonction `resolution`.

Q20. À la ligne 12, la fonction `resolution` fait appel à la fonction `pivot` qui prend en arguments le tableau `Syst` et un indice `k` et qui renvoie l'indice du pivot le plus grand en valeur absolue parmi les pivots encore disponibles dans la colonne d'indice `k`. Indiquer la structure du tableau `Syst`. Expliquer en quoi il est pertinent d'appliquer cet algorithme avec le pivot le plus grand en valeur absolue. Écrire la fonction `pivot` répondant au problème.

Q21. La fonction `combinaison` prend en arguments un tableau `Syst`, 2 entiers `k` et `i` et un flottant `x`. Cette fonction modifie la ligne `k` du tableau `Syst` en réalisant la transvection :

$$\text{Ligne k} = \text{Ligne k} + \text{mu} \times \text{Ligne i}$$

Compléter la ligne 15 du code proposé en définissant la variable `mu`. Écrire une fonction `combinaison` qui réalise l'opération souhaitée.

Une fois le problème mis sous forme triangulaire, il reste à créer le vecteur des solutions `X`. C'est la phase de remontée.

Q22. Proposer les instructions permettant de construire le vecteur `X` à partir du système triangulaire.

Q23. En utilisant la matrice A et le vecteur B définis en **Q18**, écrire l'instruction permettant de résoudre l'équation (8) et d'affecter le résultat à un vecteur `U`. À partir des fonctions précédemment définies, définir la fonction `uEF` qui construit $u(x)$le déplacement u défini à l'équation (5) et solution du problème à partir du vecteur `U` et de la fonction `phi(k, x)`.

IV Partie : Sauvegarde des résultats

Afin d'optimiser le choix des paramètres de calculs, on décide de sauvegarder dans une base de données divers paramètres de calculs.

La base de données se compose de trois tables, d'une table **composant** composée des attributs

- `id_voiture` (int) : identifiant de la voiture,
- `id_element` (int) : identifiant d'un élément composant la voiture `id`

d'une table **element** composée des attributs

- `id` (int) : identifiant de l'élément,
- `nom` (txt) : nom de l'élément (par exemple, "portière avant gauche", "aile droite", etc.)
- `prix` (int) : prix de fabrication de l'élément

et d'une table **simulation** composée des attributs

- `id` (int) : identifiant de la simulation
- `id_element` (int) : identifiant de l'élément
- `nb_noeuds` (int) : nombre de nœuds de la simulation
- `methode` (txt) : méthode utilisée pour la simulation
- `temps` (float) : temps de calculs pour effectuer la simulation
- `precision` (float) : précision moyenne du résultat obtenu (en mm)

Les clefs primaires sont

- pour la table **composant**, le couple `(id_voiture, id_element)`
- pour la table **element**, l'attribut `id`
- pour la table **simulation**, l'attribut `id`

Q24. Proposer une requête SQL qui renvoie le nombre d'éléments dans la base.

Q25. Que renvoie la requête suivante

```
1 SELECT SUM(e.prix) FROM element AS e
2 INNER JOIN composant AS c ON c.id_element = e.id
3 WHERE c.id_voiture = 21 ;
```

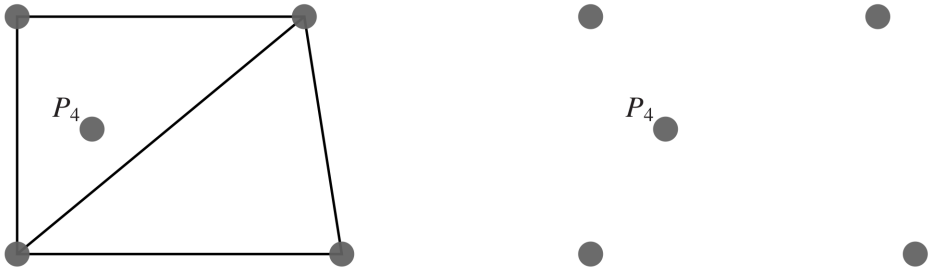
Q26. Proposer une requête qui renvoie le temps minimal de la simulation ainsi que le prix de l'élément d'identifiant 1.

Q27. Proposer une requête qui renvoie pour chaque élément coûtant strictement plus de 1 000 euro, son identifiant et la moyenne de la précision de la simulation.

Document réponse pour les Q3. et Q7.

Q3.

Q7.



ANNEXE

Rappels des syntaxes en Python

Remarque : sous Python, l'import du module `numpy` permet de réaliser des opérations pratiques sur les tableaux : `from numpy import *`. Les indices de ces tableaux commencent à 0.

	Python
tableau à une dimension	`L=[1,2,3]` (liste) `v=array([1,2,3])` (vecteur)
tableau à deux dimensions	`M=[[1,2],[3,4]]` (liste de listes) `T=array([[1,2],[3,4]])` (tableau)
accéder à un élément	`v[0]` renvoie 1 (`L[0]`, `M[0][0]` et `T[0,0]` également)
opération entre deux vecteurs de même dimension	`v-w` renvoie `array([v[0]-w[0], v[1]-w[1],v[2]-w[2]])`
ajouter un élément	`L.append(5)` uniquement sur les listes
supprimer la première occurence d'un élément d'une liste	`L.remove(element)`
calculer le produit vectoriel entre 2 vecteurs coplanaires	`cross([x1,y1],[x2,y2])` donne `x1y2-x2y1`
évaluer la racine carrée d'un nombre flottant	`sqrt(4)`
évaluer la valeur absolue d'un nombre flottant	`abs(-2)`
tableau de 0 (2 lignes, 3 colonnes)	`zeros((2,3))`
séquence équirépartie quelconque de 0 à 10.1 (exclu) par pas de 0.1	`arange(0,10.1,0.1)`
séquence de 100 valeurs équiréparties de 0 à 10.0 (inclus)	`linspace(0,10,100)`
affiche y en fonction de x où x et y sont des listes ou vecteurs de même dimension	`plot(x, y)`