Assemblage et déformation de pièces automobiles

Corrigé UPSTI

I - Posage des pièces

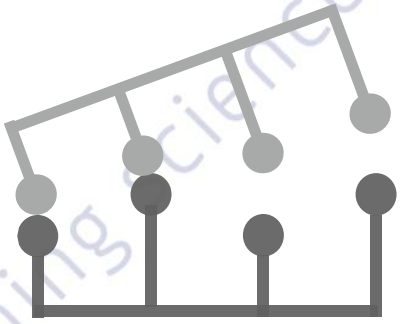
Question 1 Donner la liste `P2` associée à la pièce 2 sur la figure 2.

```
P2 = [3., 3., 2., 2.]
```

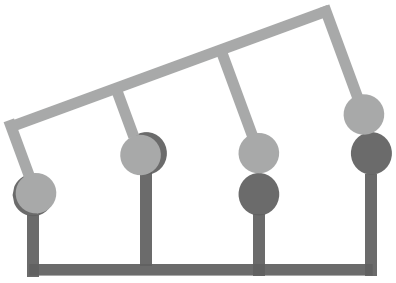
Question 2 Écrire une fonction `retourne(Ls)` qui prend une liste `Ls` de nombres en argument et qui renvoie une nouvelle liste dont l'ordre des éléments est inversé et le signe de chaque élément lui aussi inversé. Par exemple, `retourne([1.,2.,3.])` renverra `[-3.,-2.,-1.]`.

```
def retourne(Ls):
    L = []
    i = len(Ls) - 1 # dernier indice de Ls
    while i >= 0:
        L.append(-Ls[i])
        i = i -1
    return L
```

Question 3 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.



Sans collision



Avec collisions aux points 0 et 1

Question 4 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.

```
def droite(Ls, n, p):
    zn, zp = Ls[n], Ls[p]
    a = (zn - zp) / (n - p ) / Delta_x
    b = zn - n * (zn - zp) / (n - p)
    return [a, b]
```

Question 5 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 en-dessous 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.

```
def posage(Ls1, Ls2, n, p):
    a1, b1 = droite(Ls1, n, p)
    a2, b2 = droite(Ls2, n, p)

    maxi, mini = 0, 0 # distances des points de posage
    for i in range(len(Ls1)): # listes de même longueur
        altitude1 = Ls1[i] - (a1 * i * Delta_x + b1)
        altitude2 = Ls2[i] - (a2 * i * Delta_x + b2)
        distance = altitude2 - altitude1
        if distance > maxi:
            maxi = distance
        if distance < mini:
            mini = distance
    return [maxi, mini]
```

Question 6 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.

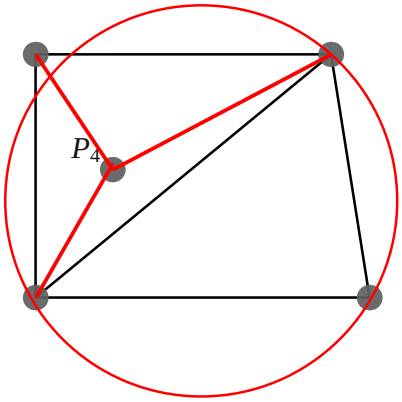
Indiquer en justifiant succinctement la complexité de cette fonction en terme de nombre de comparaisons.

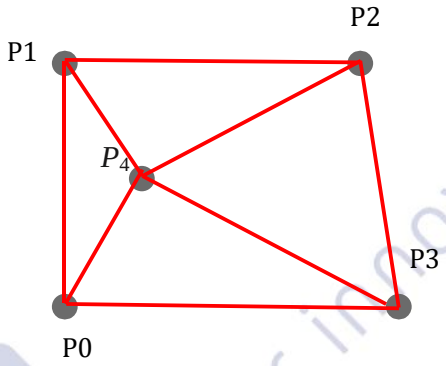
```
def posage_opt(Ls1, Ls2):
    N = len(Ls1)
    distanceMax = None # initialisation distance maximale trouvée.

    for n in range(N-1): # pour chaque point sauf le dernier
        for p in range(n+1, N): # pour chaque point après celui d'indice n
            maxi, mini = posage(Ls1, Ls2, n, p)
            if mini >= 0: # condition pour que le posage soit valide
                if distanceMax is None or maxi < distanceMax:
                    distanceMax = maxi
                    nopt, popt = n, p # points de posage optimal trouvés jusqu'ici
    return [nopt, popt]
```

La fonction comprend deux boucles imbriquées. La fonction `posage`, appelée à chaque itération, est de complexité linaire. La complexité en nombre de comparaisons de `posage_opt` est donc cubique, $O(n^3)$.

II - Maillage de surface

Question 7 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 DR, représenter les différentes transformations qui permettent d'aboutir à une triangulation de Delaunay. Indiquer la liste T qui résulte de cette triangulation.





Ajout 3 triangles, suppression (P0, P1, P2), vérification condition de Delaunay, basculement

```
T = [[0, 1, 4], [1, 2, 4], [0, 4, 3], [2, 3, 4]]
```

Question 8 Écrire la fonction `addT(indN,indT)` décrite ci-dessus.

```
def addT(indN, indT):
    p1, p2, p3 = T[indT] # récupération des indices des noeuds
    T.append([p1, p2, indN])
    T.append([p1, p3, indN])
    T[indT] = [p2, p3, indN] # modification triangle initial
                             # permet de ne pas avoir à le supprimer

def addT(indN, indT):
    p1, p2, p3 = T.pop(indT) # récupération indices et suppression du triangle initial
    T.append([p1, p2, indN])
    T.append([p1, p3, indN])
    T.append([p2, p3, indN])
```

Remarque : l'annexe du DR ne présente pas la méthode pop. La première version est sans doute plus cohérente avec la syntaxe demandée, mais la deuxième est plus cohérente avec la question 12.

Question 9 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 ci-dessous. Choisir l'une des 3 propositions données pour compléter les instructions manquantes (indiquées par *instructions à compléter*).

Les instructions à compléter doivent créer une liste `Ledge` comprenant les 2 indices des nœuds communs à `T1` et à `T2` et créer une lister `Lflip` comprenant les deux autres indices.

La première proposition va bien construire la liste `Ledge`, mais la liste `Lflip` comprendra beaucoup trop d'éléments (5 après la première itération pour k=0, dans le pire des cas, 9 après la deuxième itération…).

La troisième proposition suppose que les nœuds communs ont le même indice dans chaque triangle.

La proposition adéquate est la deuxième.

Question 10 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).

```
def insideT(indN, T1):
    A, B, C = noeud[T1[0]], noeud[T1[1]], noeud[T1[2]]
    M = noeud[indN]

    for P1, P2, P3 in ((A,B,C), (B,C,A), (C,A,B)): # 3 tests
        v1 = cross(P2-P1, P3-P1)
        v2 = cross(P2-P1, M-P1)
        if v1*v2 < 0: # les signes des produits vectoriels sont différents
            return False
    return True
```

L'utilisation des tableaux numpy permet de simplifier le calcul des coordonnées des différents vecteurs.

Question 11 Indiquer l'intérêt de la condition ligne 3 de la fonction `checkedge`. Compléter les lignes 6, 7 et 8 de cette fonction.

Si le triangle est sur le bord du maillage, il n'y a pas de triangle ayant pour arrête commune l'arrête opposée au nœud inséré. La fonction `edge` retourne l'objet `None` et la condition de Delaunay est automatiquement vérifiée.

```
5         if insideC(C, R, noeud[indN]):
6             flip(indT, indA)
7             checkedge(indN, indT)
8             checkedge(indN, indA)
```

Question 12 Écrire la fonction `delaunay(noeud)` qui prend en arguments un tableau de nœuds (de dimension $N\times2$) 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]]`.

```
def delaunay(noeud):
    T = [[0, 1, 2], [2, 3, 0]]

    for indN in range(4, len(noeud)): # parcours des indices des noeuds à insérer
        # recherche du triangle incluant le noeud à insérer
        indT = 0
        while not insideT(indN, indT):
            indT = indT + 1

        addT(indN, indT) # ajout de nouveaux triangles à la fin de T
        for iT in range(len(T)-3, len(T)): # parcours des indices des nouveaux triangles
            checkedge(indN, iT) # vérification de la condition de Delaunay

    return T
```

III - Détermination de la matrice de rigidité d'une pièce

Question 13 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 instructions permettant de tracer la solution analytique $u_{th}(x)$.

```
x = linspace(0, L, n) # ou, par cohérence avec les questions suivantes, n+1
uth = -p0 / 2 / E / S * x**2 + (Fx + L*p0) / E / S * x
plot(x, uth)
```

Question 14 Écrire une fonction phi(k,x) qui prend en argument un entier k et un flottant x ∈ [0, L] et qui renvoie la valeur $\Phi_k(x)$.

```
def phi(k, x):
    dx = L / n # n nombre d'intervalles
    xk = dx * k
    if x > xk + dx or x < xk - dx:
        return 0.
    return 1 - abs(x - xk)/dx
```

```
def phi(k, x):
    xk = k * L / n
    xkm1 = (k - 1) * L / n
    xkp1 = (k + 1) * L / n
    if (x <= xkm1) or (xkp1 <= x):
        return 0
    elif (xkm1 <= x <= xk):
        return (x - xkm1) * n / L
    else:
        return (xkp1 - x) * n / L
```

Question 15 Exprimer le produit des dérivées $\frac{d\Phi_i}{dx}\frac{d\Phi_j}{dx}$ 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 $\int_0^L \frac{d\Phi_i}{dx}\frac{d\Phi_j}{dx}dx$ à nouveau en fonction de n et L lorsque $i = j$, $i = j = 0$, $i = j = n$, $|i - j| = 1$ et dans les autres cas.

Notons $\Delta x = \frac{L}{n}$.

Pour la fonction test $\Phi_i(x)$: si $x \in [x_{i-1}, x_i]$, alors $\frac{d\Phi_i(x)}{dx} = \frac{1}{\Delta x}$

si $x \in [x_i, x_{i+1}]$, alors $\frac{d\Phi_i(x)}{dx} = -\frac{1}{\Delta x}$

sinon $\frac{d\Phi_i(x)}{dx} = 0$

d'où : si $i = j$, $\int_0^L \frac{d\Phi_i}{dx}\frac{d\Phi_j}{dx}dx = \int_0^L \left(\frac{d\Phi_i}{dx}\right)^2 dx = \int_{x_{i-1}}^{x_{i+1}} \frac{1}{(\Delta x)^2}dx = \frac{2\Delta x}{(\Delta x)^2} = \frac{2}{\Delta x}$

si $i = j = 0$, $\int_0^L \left(\frac{d\Phi_0}{dx}\right)^2 dx = \int_{x_0}^{x_1} \frac{1}{(\Delta x)^2}dx = \frac{1}{\Delta x}$

si $i = j = n$, $\int_0^L \left(\frac{d\Phi_n}{dx}\right)^2 dx = \int_{x_{n-1}}^{x_n} \frac{1}{(\Delta x)^2}dx = \frac{1}{\Delta x}$

si $j = i + 1$, $\int_0^L \frac{d\Phi_i}{dx}\frac{d\Phi_{i+1}}{dx}dx = \int_{x_i}^{x_{i+1}} \frac{d\Phi_i}{dx}\frac{d\Phi_{i+1}}{dx}dx = \int_{x_i}^{x_{i+1}} \frac{-1}{\Delta x}\frac{1}{\Delta x}dx = \frac{-1}{\Delta x}$, idem si $j = i - 1$

sinon, l'integrale est nulle.

Question 16 À partir des résultats de la question précédente et en posant $\beta = -1$, exprimer les coefficients C et α.

Par identification : $C = \frac{1}{\Delta x}$ et $\alpha = 2$.

Question 17 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.

Par identification, on obtient : $A = ES\,M$ et $B = P$ avec l'ajout de F_x sur le dernier terme.

```
A = E * S * M
B = P
B[-1] = B[-1] + Fx
```

Question 18 Indiquer l'intérêt des lignes 2 à 9 de la fonction `resolution`.

Ces lignes créent la matrice augmentée du système linéaire sous la forme d'une liste de listes en dupliquant les données. Les tableaux R et Y ne seront pas modifiés. Les opérations de transvection, permutation, dilatation, sont réalisées simultanément sur la matrice et le second membre du système.

Question 19 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.

`Syst` est une liste de `nb` lignes, chaque ligne est une liste de `(nb+1)` valeurs correspondant aux colonnes.

La recherche du pivot conduit à éviter des divisions par des nombres proches de 0.

```
def pivot(S, j):
    imax = j # recherche à partir de la diagonale
    pmax = abs(S[j][j])
    for i in range(j+1, len(S)):
        if abs(S[i][j]) > pmax:
            imax, pmax = i, abs(S[i][j])
    return imax
```

Question 20 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 :

Ligne k = Ligne k + mu × Ligne i

Compléter la ligne 16 du code proposé en définissant la variable `mu`. Écrire une fonction `combinaison` qui réalise l'opération souhaitée.

```
def combinaison(Syst, k, i, x):
    for j in range(len(Syst[k])):
        Syst[k][j] = Syst[k][j] + x * Syst[i][j]

L16 : mu = -Syst[k][i] / Syst[i][i]
```

Question 21 Proposer les instructions permettant de construire le vecteur X à partir du système triangulaire.

```
    i = nb
    while i > 0:
        i = i - 1
        xi = Syst[i][nb]
        for j in range(i+1, nb):
            xi = xi - X[j]*Syst[i][j]
        X[i] = xi / Syst[i][i]
```

ou

```
    for k in range(nb) :
        somme = 0
        for j in range(nb-k, nb):
            somme += Syst[nb-1-k][j]*X[j]
        X[nb-1-k] = (Syst[nb-1-k][nb] - somme) / Syst[nb-1-k][nb-1-k]
```

Question 22 En utilisant la matrice A et le vecteur B définis en Q17, 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)`.

```
U = resolution(A, B)

def uEF(U, x):
    ux = 0
    for k in range(len(U)):
        ux = ux + phi(k, x) * U[k]
    return ux
```

IV - Analyse et comparaison des résultats

Question 23 Indiquer en justifiant succinctement quel posage vous semble respecter les règles associées au posage optimal définies en partie I de ce sujet.

Le posage optimal est celui qui minimise la distance des points de la surface à un plan (passage en 3D) passant par les points de posage.

Si les graphiques proposés correspondent aux distances par rapport au plan de posage, la première solution est plus pertinente.

Question 24 En exploitant les données du tableau 1 et la figure 13, déterminer l'ordre de grandeur du nombre minimal d'éléments du maillage pour que le résultat du calcul soit satisfaisant (précision des résultats inférieure à 1 mm).

En supposant que le maillage fin donne un résultat ayant convergé, de valeur 13,120 mm (probable inversion des déplacements dans le tableau 1), la valeur obtenue avec un maillage grossier, de 12,154 mm est dans la précision demandée.

Cependant, la discontinuité de précision observée figure 13, conduit à proposer un nombre de nœuds supérieurs afin d'assurer le résultat, soit autour de 8000 nœuds.

Question 25 En exploitant la forme spécifique de la géométrie de la portière, justifier le seuil observé pour 5000 éléments environ dans le tableau 1 et figure 13.

Le maillage ne doit pas prendre en compte correctement les parties fines de la géométrie tout autour de la vitre.