I Partie : Posage des pièces

Q1. Pour la pièce 2, on propose `P2 = [3.,3.,2.,2.]`

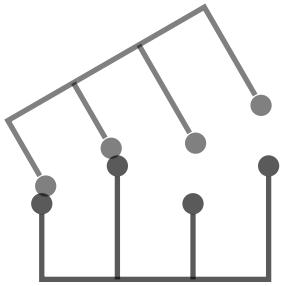
Q2.

```
def retourne(Ls: list) -> list:
    Lr = []
    for i in range( len(Ls) ):
        Lr.append( -Ls[-1-i] )
    return Lr
```

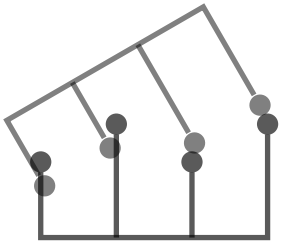
```
def retourne(Ls: list) -> list:
    return [ -e for e in Ls[::-1] ]
```

Q3.

Points de posage en 0 et 1.



Points de posage en 2 et 3.



Q4. Les coefficients sont $a = \dfrac{z_n - z_p}{(n-p)\Delta x}$ et $b = z_n - \dfrac{n(z_n - z_p)}{n-p}$

Q5.

```
def droite(Ls: list, n: int, p: int) -> list:
    return [(Ls[n]-Ls[p])/(n-p)/Delta_x, Ls[n]-n*(Ls[n]-Ls[p])/(n-p)]
```

Q6.

```
def posage(Ls1: list, Ls2: list, n: int, p: int) -> list:
    dmin, dmax = float('inf'), -float('inf')
    a1, b1 = droite(Ls1, n, p)
    a2, b2 = droite(Ls2, n, p) # a1=a2
    for i in range(len(Ls1)):
        d1 = Ls1[i] - (a1*i + b1)
        d2 = Ls2[i] - (a2*i + b2)
        h = d2-d1
        if h < dmin:
            dmin = h
        if dmax < h:
            dmax = h
    return [dmin, dmax]
```

Q7.

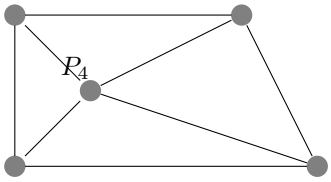
```
def posage_opt(Ls1: list, Ls2: list) -> list:
    n, p = 0, 1
    minDmax = float('inf')
    for i in range(len(Ls1)-1):
        for j in range(i+1, len(Ls1)):
            dm, dM = posage(Ls1, Ls2, i, j)
```

```
7            if dm >= 0 and dM < minDmax:
8               minDmax = dM
9               n, p = i, j
10      return [n, p]
```

En notant n la longueur d'une liste `Ls1`, la complexité temporelle asymptotique sera en $\mathcal{O}(n^3)$. En effet, la fonction `posage` est en $\mathcal{O}(n)$, les deux boucles imbriquées entraînent $n(n+1)/2$ appels à la fonction `posage`, ce qui donne une complexité cubique.

II Partie : Maillage de la surface

Q8.



Et les triangles forment une configuration de Delaunay.

Q9.

```
1  def addT(indN: int, indT: int) -> list:
2      ind1, ind2, ind3 = T[indT]
3      T.append( [ind1, ind2, indN] )
4      T.append( [ind1, ind3, indN] )
5      T.append( [ind2, ind3, indN] )
6      T.remove( [ind1, ind2, ind3] )
```

Q10. La proposition n° 1 est incorrecte car la liste `Lflip` pourra contenir 3 fois le même indice de sommet. La proposition n° 3 est incorrecte car l'ordre des sommets d'un triangle est inconnu (`T2[k] == T1[k]`). On choisit la proposition n° 2 qui ne répète aucun nœud.

Q11.

```
1  def insideT(indN: int, T1: list) -> bool:
2      M = T[indN]
3      A = T[T1[0]]
4      B = T[T1[1]]
5      C = T[T1[2]]
6      if cross(B-A, M-A) * cross(B-A, C-A) < 0:
7          return False
8      if cross(C-B, M-B) * cross(C-B, A-B) < 0:
9          return False
10     return cross(A-C, M-C) * cross(A-C, B-C) < 0:
```

Q12. La ligne 3 permet de vérifier que le triangle à l'opposé du nœud d'indice `indN` existe bien, *i.e.* que l'on ne soit pas en bordure du maillage. On complète alors ainsi

```
1  def checkedge(indN: int, indT: int) -> None:
2      indA = edge(indN, indT)
3      if indA != None:
4          C, R = cercle( T[indA] )
5          if insideC( C, R, noeud[indN] ):
6              flip(indA, indT)
7              checkedge(indN, indA)
8              checkedge(indN, indT)
```

Q13.

```python
def delaunay(noeud: list) -> list:
    if len(noeud) < 3:
        return [[]]
    if len(noeud) == 3:
        return [[0,1,2]]
    T = [[0,1,2],[2,3,0]]
    for indN in range( 4, len(noeud) ):
        indT = 0
        while not insideT(indN, T[indT]): # les triangles sont supposés couvrir tous les noeuds
            indT += 1
        addT(indN, indT)
        checkedge(indN, indT)
    return T
```

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

Q14.

```python
x = linspace(0, L, n)
uth = -p0/2/E/S*x**2 + (Fx+L*p0)/E/S*x
plot(x, uth)
```

ou sans `numpy`

```python
x = [ i*L/(n-1) for i in range(n) ]
uth = [ xi/E/S*(-p0/2*xi + (Fx+L*p0)) for xi in x ]
plot(x, uth)
```

Q15. On normalise pour que $dx = 1$,

```python
def phi(k: int, x: float) -> float:
    d = abs(x/L - k) # distance par rapport à x_k
    if d > 1:
        return 0
    return 1-d
```

Q16. $\frac{\mathrm{d}\Phi_i}{\mathrm{d}x}\frac{\mathrm{d}\Phi_j}{\mathrm{d}x}$ a pour valeur

- si $i = j$, 1 sur $[x_{i-1},x_{i+1}] \cap [0,L]$, 0 sinon
- si $|i-j| = 1$, -1 sur $[x_m, x_M]$, 0 sinon (avec m=min(i,j), M=max(i,j))
- si $|i-j| > 1$, 0 sur $[0,L]$

De là, l'intégrale vaut

- si $i = j \in \{0,n\}$, L/n
- si $i = j \notin \{0,n\}$, $2L/n$
- si $|i-j| = 1$, $-L/n$
- si $|i-j| > 1$, 0

Q17. Avec $\beta = -1$, $C = L/n$ et $\alpha = 2$.

Q18. On a la relation $A = ES * M$, $B = P + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ Fx \end{pmatrix}$

```
A = E*S*M
v = zeros((len(P),1))
v[-1]=1
B = P + Fx * v
```

ou sans `numpy`

```
A = []
for i in range(len(M)):
    ligne = []
    for j in range(len(M[0])):
        ligne.append( E*S*M[i][j] )
    A.append(ligne)
B = [p for p in P]
B[-1] += Fx
```

Q19. Les lignes 2 à 9 permettent d'initialiser la matrice `Syst` représentant le système à résoudre.

Q20. Utiliser le plus grand pivot en valeur absolue disponible permet de ne pas choisir la valeur nulle (si c'était le cas, la ligne serait nulle).
On propose donc

```
def pivot(Syst: list, k:int) -> int:
    imax, vmax = -1, 0
    for i in range( len(Syst) ):
        if abs(Syst[i][k]) > vmax:
            imax, vmax = i, abs(Syst[i][k])
    return imax
```

Q21. On propose la fonction suivante

```
def combinaison(Syst: list, k: int, i: int, x: float) -> None:
    for j in range( len(Syst)):
        Syst[k][j] += x * Syst[i][j]
```

et on complète par

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

Q22. Le code pour la remontée (ligne 18 et suivantes) :

```
    X =[0]*nb
    for i in range(nb-1, -1, -1):
        for j in range(i, nb):
            X[i] = Syst[i][j]*Y[j]
```

Q23. avec A inversible, $U = resolution(A, B)$
Question peu dirigée, on utilise les relations (6),

```
def uEF(U: list, phi: 'function', x: float) -> list:
    ux = 0
    for i in range(len(U)):
        ux += U[i]*phi(i,x)
    return ux
```

IV Partie : Sauvegarde des résultats

Q24.

```sql
SELECT COUNT(*) FROM element ;
```

Q25. Cette requête renvoie le prix coûtant des pièces formant la voiture d'identifiant 21.

Q26.

```sql
SELECT MIN(s.temps), e.prix FROM simulation AS s
INNER JOIN element AS e ON e.id = s.id_element
WHERE e.id = 1;
```

Q27.

```sql
SELECT s.id_element, AVG(s.precision) FROM simulation AS s
INNER JOIN element AS e ON e.id = s.id_element
WHERE e.prix > 1000
GROUP BY s.id_element
HAVING AVG(s.precision) < 0.15 ;
```