

1 Exercice

On considère la fonction suivante :

```
def f(x) :
    if x <= 1 :
        return 1
    elif x % 2 == 0 :
        return 2*f(x/2)
    else :
        return 1 + f(x+1)
```

1. Montrer que cette fonction récursive se termine pour toute valeur entière de l'argument x .
2. Soient $x \in \mathbb{N}$ et $g(x) = f(x) + x$. À partir du programme précédent, donner un programme python qui calcule $g(x)$.
3. En déduire que pour tout $x \in \mathbb{N}$, $g(x)$ est une puissance de 2.

Corrigé

1. Si x est pair et > 1 , alors $f(x) = 2f(x/2)$ et $x/2 < x$. Si x est impair et > 1 , alors $f(x) = 1 + 2f((x+1)/2)$ et $(x+1)/2 < x$. Donc dans les deux cas récursifs, on appelle f en une ou deux étapes avec un argument strictement inférieur à x . Ceci prouve la terminaison du calcul de $f(x)$ par récurrence sur x .
2. Soit $g(x) = f(x) + x$. En remplaçant $f(x)$ par $g(x) - x$ dans la définition de f et en simplifiant, on obtient un programme calculant g :

```
def g(x) :
    if x <= 1 :
        return 1 + x
    elif x % 2 == 0 :
        return 2*g(x/2)
    else :
        return g(x+1)
```

3. Avec la question précédente, il est immédiat par récurrence que $g(x)$ est toujours une puissance de 2.

2 Métrologie

2.1 Mise en situation

2.2 Mise en équation de détermination du plan des moindres carrés

3 Détermination du défaut de planéité

Objectif L'objectif de cette partie est de rechercher le plan des moindres carrés c'est à dire de trouver les valeurs a , b et c qui minimisent les écarts entre le plan \mathcal{P} et un nuage de points.
Le plan des moindres carrés permettra de déterminer le défaut de planéité.

3.1 Conditionnement du problème

Question 1 Montrer que la méthode des moindres carrés permet d'aboutir aux 3 équations suivantes :

$$\sum_{i=1}^n (a x_i^2 + b x_i y_i + c x_i - x_i z_i) = 0 \quad \sum_{i=1}^n (a x_i y_i + b y_i^2 + c y_i - y_i z_i) = 0 \quad \sum_{i=1}^n (a x_i + b y_i + c - z_i) = 0$$

Correction On a :

$$\frac{\partial E(a, b, c)}{\partial a} = 0 \Leftrightarrow \sum_{i=1}^n \frac{\partial (z_i - a x_i - b y_i - c)^2}{\partial a} = 0 \Leftrightarrow \sum_{i=1}^n -2 x_i (z_i - a x_i - b y_i - c) = 0$$

$$\text{Au final : } \frac{\partial E(a, b, c)}{\partial a} = 0 \Leftrightarrow \sum_{i=1}^n (-x_i z_i + a x_i^2 + b x_i y_i + c x_i) = 0$$

On a :

$$\frac{\partial E(a, b, c)}{\partial b} = 0 \Leftrightarrow \sum_{i=1}^n \frac{\partial (z_i - a x_i - b y_i - c)^2}{\partial b} = 0 \Leftrightarrow \sum_{i=1}^n -2 y_i (z_i - a x_i - b y_i - c) = 0$$

$$\text{Au final : } \frac{\partial E(a, b, c)}{\partial b} = 0 \Leftrightarrow \sum_{i=1}^n (-y_i z_i + a x_i y_i + b y_i^2 + c y_i) = 0$$

On a :

$$\frac{\partial E(a, b, c)}{\partial c} = 0 \Leftrightarrow \sum_{i=1}^n \frac{\partial (z_i - a x_i - b y_i - c)^2}{\partial c} = 0 \Leftrightarrow \sum_{i=1}^n -2 (z_i - a x_i - b y_i - c) = 0$$

$$\text{Au final : } \frac{\partial E(a, b, c)}{\partial c} = 0 \Leftrightarrow \sum_{i=1}^n (-z_i + a x_i + b y_i + c) = 0$$

On définit les grandeurs suivantes :

$$S_{xx} = \sum_{i=1}^n x_i^2, S_{yy} = \sum_{i=1}^n y_i^2, S_{xy} = \sum_{i=1}^n x_i y_i, S_{xz} = \sum_{i=1}^n x_i z_i, S_{yz} = \sum_{i=1}^n y_i z_i, S_x = \sum_{i=1}^n x_i, S_y = \sum_{i=1}^n y_i \text{ et } S_z = \sum_{i=1}^n z_i.$$

On note $X = (a, b, c)$ le vecteur solution du problème avec $X \in \mathcal{M}_{1,3}(\mathbb{R})$.

Question 2 Montrer que le problème peut se mettre sous la forme du système linéaire suivant : $AX + B = 0$ où $A \in \mathcal{M}_{3,3}(\mathbb{R})$ et $B \in \mathcal{M}_{1,3}(\mathbb{R})$. On donnera les expressions de A et de B .

Correction On a :

$$\begin{aligned} \sum_{i=1}^n (-x_i z_i + a x_i^2 + b x_i y_i + c x_i) = 0 &\Leftrightarrow -\sum_{i=1}^n x_i z_i + \sum_{i=1}^n a x_i^2 + \sum_{i=1}^n b x_i y_i + \sum_{i=1}^n c x_i = 0 \\ &\Leftrightarrow a \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i y_i + c \sum_{i=1}^n x_i = \sum_{i=1}^n x_i z_i \Leftrightarrow a S_{xx} + b S_{xy} + c S_x = S_{xz} \end{aligned}$$

De même, on détermine que :

$$\begin{aligned} \sum_{i=1}^n (-y_i z_i + a x_i y_i + b y_i^2 + c y_i) = 0 &\Leftrightarrow a S_{xy} + b S_{yy} + c S_y = S_{yz} \\ \sum_{i=1}^n (-z_i + a x_i + b y_i + c) = 0 &\Leftrightarrow a S_x + b S_y + c n = S_z \end{aligned}$$

On a donc :

$$\begin{cases} a S_{xx} + b S_{xy} + c S_x = S_{xz} \\ a S_{xy} + b S_{yy} + c S_y = S_{yz} \\ a S_x + b S_y + c n = S_z \end{cases} \Leftrightarrow \begin{bmatrix} S_{xx} & S_{xy} & S_x \\ S_{xy} & S_{yy} & S_y \\ S_x & S_y & n \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} S_{xz} \\ S_{yz} \\ S_z \end{bmatrix} \Leftrightarrow AX = -B$$

Le signe négatif provient de la formulation du problème dans la question.

3.2 Résolution du problème

Question 3 Donner une méthode (ou le nom d'un algorithme) permettant de résoudre le système linéaire ci-dessus. Préciser les différentes étapes de cet algorithme ainsi que sa complexité.

Correction Pour résoudre ce problème, il est possible d'utiliser l'algorithme du pivot de Gauss. Cet algorithme se déroule en trois phases majeures :

1. recherche du pivot ;
2. triangularisation de la matrice A (en phase avec les transformations nécessaires sur la matrice B) ;
3. phase de remontée permettant de déterminer X .

On peut montrer que la complexité algorithmique du pivot de Gauss est en $\mathcal{O}(n^3)$.

Question 4 Donner l'implémentation de la fonction `read_file` permettant de lire un fichier de mesures formaté comme indiqué ci-dessus et permettant de retourner la liste des points mesurés.

Correction

```
def read_file(file):
    fid = open(file, 'r')
    pts = []
    for ligne in fid:
        ligne = ligne.split(",")
        print(ligne)
        pts.append([float(ligne[0]), float(ligne[1]), float(ligne[2])])
    return pts
```

Question 5 Donner l'implémentation du programme principal (main) permettant de lire le fichier de mesure appelé `mesures.txt` et de déterminer la liste des paramètres `[a, b, c]` correspondant aux paramètres du plan des moindres carrés. Vous utiliserez les fonctions précédemment définies.

Correction

```
liste_pts = read_file("mesures.txt")
plan = plan_moindres_carres(liste_pts)
```

3.3 Application – Détermination du défaut de planéité

Question 6 Donner l'implémentation de la fonction `dist_pt_plan` en Python permettant de retourner la distance algébrique entre un point et un plan. Les spécifications de la fonction sont les suivantes :

■ Python

```
def dist_pt_plan(pt, plan):
    """
    Permet de calculer une distance point - plan
    Entrées :
        * pt(list) : point de coordonnées [x,y,z]
        * plan(list) : caractéristiques du plan [a,b,c]
    Sortie :
        * d(float) : distance
    """
```

Correction

```
def dist_pt_plan(pt, pl):
    return (pt[2] - pl[0]*pt[0] - pl[1]*pt[1] - pl[2]) / (sqrt(pl[0]**2 + pl[1]**2 + 1))
```

Question 7 Donner l'implémentation de la fonction `defaut_planeite` en Python permettant de retourner le défaut de planéité. On donne les spécifications de la fonction :

■ Python

```
def defaut_planeite(pl, liste_pt):
    """
    Permet de calculer une distance point - plan
    Entrées :
        * pl(list) : caractéristiques du plan [a,b,c]
        * liste_pts (list) : liste de points de la forme [[x1,y1,z1], [x2,y2,z2], ...]
```

```
Sortie :
* d(flt) : distance
"""
```

Correction

```
def default_planeite(pl, liste_pt):
    dist_p=0
    dist_n=0
    for i in range(len(liste_pt)):
        dist=dist_pt_plan(liste_pt[i], pl)
        if dist_p<dist and dist>0 :
            dist_p=dist
        elif dist_n>dist and dist<0 :
            dist_n=dist
    return dist_p-dist_n
```

4 Construction d'une référence spécifiée associée à une surface nominale plane

Question 8 Quel est l'objectif de cette fonction ? En déduire le triplet correspondant à l'équation du plan répondant au point 2 de la méthode.

Correction Cette fonction a pour but de retrouver l'indice du point étant le plus éloigné du plan des moindres carrés. Ce point est aussi du côté libre de la matière.

Le nouveau plan a donc pour caractéristiques :

$[a, b, \text{liste_pt}[\text{ind}][2] - X[0] * \text{liste_pt}[\text{ind}][0] - X[1] * \text{liste_pt}[\text{ind}][1]]$.

Question 9 Quel est l'objectif de cette fonction ? Que retourne-t-elle ? Donner un commentaire pour chacune des instructions.

Correction

```
1. def balancage(pl, pt):
2.     npt=25                                #
3.     pas=0.000003                          #
4.     dist=100000                           #
5.     p12=[None, None, None]                #
6.     for k in range(-npt, npt):             #
7.         p12[0]=p1[0]+k*pas                 #
8.         for j in range(-npt, npt):         #
9.             p12[1]=p1[1]+j*pas             #
10.            p12[2]=pt[2]-p12[0]*pt[0]-p12[1]*pt[1] #
11.            temp=default_planeite(p12, liste_pt) #
12.            if temp<dist :                   #
13.                dist=temp                   #
14.                f=p12[0]                    #
15.                g=p12[1]                    #
16.                h=p12[2]                    #
17.            p12=[f, g, h]                   #
18.            return [p12, dist]              #
```

L'objectif de cette fonction est de déterminer une référence spécifiée en faisant varier l'orientation du plan des moindres carrés. Elle retourne le plan optimal ainsi que le défaut de planéité lié au nuage de point.

Les boucles *for* lignes 6 et 8 permettent de réaliser des petite variation autour d'un point *pt* prédéterminé. Ligne 10 on dispose alors des caractéristiques d'un plan ayant subi deux petites rotations. On détermine alors le défaut de planéité lié à ce nouveau plan.

On cherche alors le nouveau plan « minimisant le défaut de planéité » (en fait on ne minimise pas le défaut mais l'écart...).

Question 10 Quelle est la complexité de cet algorithme si on cherche à améliorer le choix du plan optimal par rapport à un seul des points du nuage de points ? Comment évolue la complexité de ce programme si on cherche à réaliser le balancage en utilisant chacun des points mesurés ?

Correction Pour avoir une « meilleure » solution du problème, on peut augmenter le nombre de pas de calcul. Les deux boucles imbriquées dépendant du nombre de points, on a une complexité en $\mathcal{O}(npt^2)$.

On note *n* le nombre points palpés. Si on réalise un balancage sur chacun des points du nuage, la complexité sera en $\mathcal{O}(n \cdot npt^2)$