# 1 Présentation

## 1.1 Mise en situation

## 1.2 Mise en équation de détermination du plan des moindre carrés

# 2 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 $\mathscr{P}$ et un nuage de points.

Le plan des moindre carrés permettra de déterminer le défaut de planéité.

## 2.1 Conditionnement du problème

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

$$\sum_{i=1}^{n}\left(ax_i^2+bx_iy_i+cx_i-x_iz_i\right)=0 \quad \sum_{i=1}^{n}\left(ax_iy_i+by_i^2+cy_i-y_iz_i\right)=0 \quad \sum_{i=1}^{n}\left(ax_i+by_i+c-z_i\right)=0$$

**Correction** On a :

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

Au final : $\dfrac{\partial E(a,b,c)}{\partial a}=0\Leftrightarrow\sum_{i=1}^{n}\left(-x_iz_i+ax_i^2+bx_iy_i+cx_i\right)=0$

On a :

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

Au final : $\dfrac{\partial E(a,b,c)}{\partial b}=0\Leftrightarrow\sum_{i=1}^{n}\left(-y_iz_i+ax_iy_i+by_i^2+cy_i\right)=0$

On a :

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

Au final : $\dfrac{\partial E(a,b,c)}{\partial b}=0\Leftrightarrow\sum_{i=1}^{n}\left(-z_i+ax_i+by_i+c\right)=0$

On définit les grandeurs suivantes :

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

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

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

**Correction** On a :

$$\sum_{i=1}^{n}\left(-x_iz_i+ax_i^2+bx_iy_i+cx_i\right)=0\Leftrightarrow-\sum_{i=1}^{n}x_iz_i+\sum_{i=1}^{n}ax_i^2+\sum_{i=1}^{n}bx_iy_i+\sum_{i=1}^{n}cx_i=0$$

$$\Leftrightarrow a\sum_{i=1}^{n}x_i^2+b\sum_{i=1}^{n}x_iy_i+c\sum_{i=1}^{n}x_i=\sum_{i=1}^{n}x_iz_i\Leftrightarrow aS_{xx}+bS_{xy}+cS_x=S_{xz}$$

De même, on détermine que :

$$\sum_{i=1}^{n}\left(-y_i z_i + a x_i y_i + b y_i^2 + c y_i\right) = 0 \Leftrightarrow a S_{xy} + b S_{yy} + c S_y = S_{yz}$$

$$\sum_{i=1}^{n}\left(-z_i + a x_i + b y_i + c\right) = 0 \Leftrightarrow a S_x + b S_y + c n = S_z$$

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.

## 2.2 Résolution du problème

**Question 3** *Donner une méthode (ou le nom d'un algorithme) permettant de résoudre le système linéaire ci-dessus. On précisera 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 deux phases majeures :

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

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

Les points mesurés par une machine à mesurer tridimensionnelle sont stockés dans un fichier texte. Dans ce fichier sont inscrits successivement les coordonnés des points puis les coordonnées de la normale de contact entre le palpeur et la surface mesurée. Le fichier est de la forme suivante :

```
-101.88340,-155.21568,-50.30434,0,0,1
-99.21040,-145.54768,-50.304844,0,0,1
-82.43090,-129.69318,-50.292844,0,0,1
-59.72540,-134.12818,-50.301844,0,0,1
...
```

**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.*

**Rappel** La fonction `split` permet de séparer les éléments d'une chaîne de caractère suivant un motif et de les stocker dans une liste :

```
>>> a = "-101.88340,-155.21568,-50.30434,0,0,1"
>>> a = a.split(",")
          ['-59.72540', '-134.12818', '-50.301844', '0', '0', '1']
```

**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.*

**Correction**

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

## 2.3 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(flt) : 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 le 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 defaut_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
```

# 3 Construction d'une référence spécifiée associée à une surface nominalement 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 coté libre de la matière.

Le nouveau plan a donc pour caractéristiques :
`[a,b,liste_pt[ind][2]-X[0]*liste_pt[ind][0]-X[1]*liste_pt[ind][1].`

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

**Correction** L'objectif de cette fonction est de déterminer une référence spécifie en faisant varier l'orientation du plan des moindres carrés. Elle retourne le plan optimal ainsi que le défaut de planéité de la surface.

```
def balancage(pl, pt):
    npt=25                          #
    pas=0.000003                    #
    dist=100000                     #
    pl2=[None,None,None]            #
    for k in range(-npt,npt):       #
        pl2[0]=pl[0]+k*pas          #
        for j in range(-npt,npt):   #
            pl2[1]=pl[1]+j*pas      #
            pl2[2]=pt[2]-pl2[0]*pt[0]-pl2[1]*pt[1] #
            temp=defaut_planeite(pl2,liste_pt)     #
            if temp<dist :          #
                dist=temp           #
                f=pl2[0]            #
                g=pl2[1]            #
                h=pl2[2]            #
    pl2=[f,g,h]                     #
    return [pl2,dist]               #
```

**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 balançage 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 balançage sur chacun des points du nuage, la complexité sera en $\mathcal{O}(n \cdot npt^2)$