# Épreuve d'admissibilité de modélisation d'un système, d'un procédé ou d'une organisation

## A. Présentation de l'épreuve

*Arrêté du 28 décembre 2009 modifié*

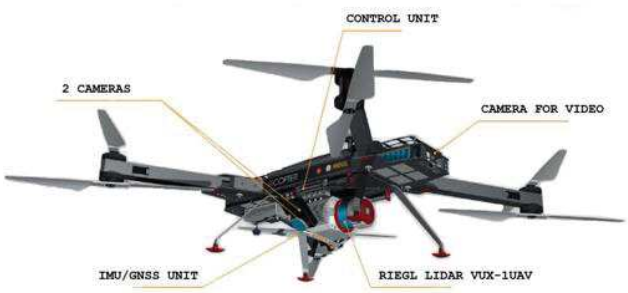
- Durée totale de l'épreuve : 6 heures
- Coefficient 1

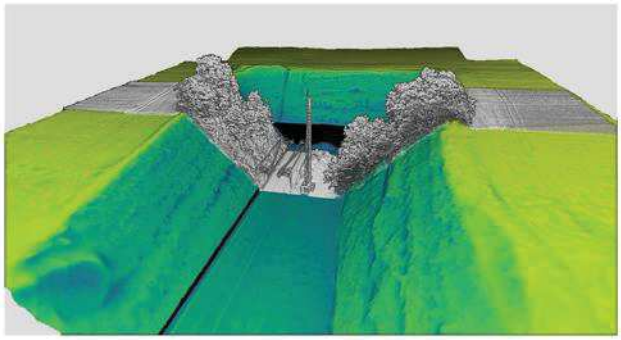
L'épreuve est spécifique à l'option choisie.

À partir d'un dossier technique comportant les éléments nécessaires à l'étude, l'épreuve a pour objectif de vérifier que le candidat est capable de synthétiser ses connaissances pour modéliser un système technique dans le domaine de la spécialité du concours dans l'option choisie en vue de prédire ou de vérifier son comportement et ses performances.

## B. Sujet

Le sujet est disponible en téléchargement sur le site du ministère à l'adresse :
https://media.devenirenseignant.gouv.fr/file/agregation_externe/99/8/s2021_agreg_externe_sii_informatique_2_1389998.pdf



## C. Éléments de correction

## PARTIE 1 : Acquisition 3D par scanner-lidar

**Question 1**
En utilisant la relation :

$$d = \frac{c \times T}{2}$$

Distance de la caténaire : $d = 3 \times 10^8 \times 10 \times 10^{-9}/2 = 1.5\ m$
Distance de l'arbre : $d = 3 \times 10^8 \times 12 \times 10^{-9}/2 = 1.8\ m$
Distance du sol : $d = 3 \times 10^8 \times 20 \times 10^{-9}/2 = 3\ m$

**Question 2**
D'après le croquis :

$$\tan\left(\frac{FOV}{2}\right) = \frac{SW}{2H}$$

$$SW = 2H \tan\left(\frac{FOV}{2}\right)$$

La surface balayée durant $\Delta T$ :

$$A = SW \times \Delta x = SW \times v\Delta T$$

**Question 3**
Le nombre de lignes est égal au temps de vol divisé par le temps de balayage d'une ligne, i.e. $dt = 1/SR$ :

$$N_l = \frac{\Delta t}{dt} = \frac{\Delta t}{1/SR} = \Delta t \times SR$$

La rotation angulaire du faisceau durant 1 s :

$$\Omega = FOV \times SR$$

Le pas angulaire correspond à l'angle parcouru par le faisceau durant la période d'un pulse, i.e. $dt = 1/PRR$ :

$$\Delta\theta = FOV \times SR \times dt = FOV \times \frac{SR}{PRR}$$

**Question 4**
Le nombre d'empreintes est égal à l'angle de champ de vision divisé par le pas angulaire :

$$N_f = \frac{FOV}{\Delta\theta} = \frac{PRR}{SR}$$

La densité de points est le produit du nombre d'empreintes par le nombre de lignes divisé par la surface balayée durant le temps de vol $\Delta t$.

$$d_p = \frac{N_l \times N_f}{SW \times v\Delta t} = \frac{PRR}{SW \times v}$$

L'espace inter-empreinte correspond à la largeur de bande divisée par le nombre d'empreintes :

$$a = \frac{SW}{N_f - 1} \cong \frac{SW \times \Delta\theta}{FOV} \cong \frac{SW \times SR}{PRR} = 2H \times \frac{SR}{PRR} \times \tan\left(\frac{FOV}{2}\right)$$

L'espace interligne correspond à la distance parcourue durant la période du balayage, i.e. $dt = 1/SR$

$$b = vdt = \frac{v}{SR}$$

**Question 5**
Pour $L = 80\ m, H = 16\ m, D = 30\ m, a = 2\ cm, b = 2cm, SR = 200\ Hz$
De l'expression de la largeur de bande, le champ de vision est déduit :

$$FOV = 2 \times \tan^{-1}\left(\frac{SW}{2D}\right) = 2 \times \tan^{-1}\left(\frac{16}{60}\right) \cong 30°$$

De l'expression de l'espace inter-empreinte, la fréquence $PRR$ est déduite :

$$PRR = \frac{SW \times SR}{a} = \frac{16 \times 200}{0.02} = 160\ kHz$$

De l'expression de l'espace interligne, la vitesse du drone est déduite :

$$v = SR \times b = 200 \times 0.02 = 4\ m/s$$

**Question 6**

Un point 3D est codé sur 16 octets et il y a $N = N_l \times N_f$ points de mesure, ce qui donne un volume de données en octets :

$$C = 16 \times N_l \times N_f = 16 \times \Delta t \times PRR = 16 \times 20 \times 160000 = 51.2\ Mo$$

**Question 7**

```
class CLidar:
    def __init__(self,p_PRR,p_SR,p_FOV,p_gamma):
        self.__PRR=p_PRR
        self.__SR=p_SR
        self.__FOV=p_FOV*math.pi/180.0
        self.__gamma=p_gamma
        self.__deltaTheta=0.
        self.__omega=0.0
        self.__Nf=0.
        self.__lMatrixPoint=[]
    def getPRR(self):
        return(self.__PRR)
    def getSR(self):
        return(self.__SR)
    def getFOV(self):
        return(self.__FOV)
    def getGamma(self):
        return(self.__gamma)
    def getOmega(self):
        self.__omega=self.__SR*self.__FOV
        return(self.__omega)
    def getDeltaTheta(self):
        self.__deltaTheta=self.getOmega()/self.__PRR
        return(self.__deltaTheta)
    def getNf(self):
        self.__Nf= self.__FOV/self.getDeltaTheta()
        return(self.__Nf)
    def setFOV(self,p_FOV):
        self.__FOV=p_FOV
```

**Question 8**

```
class CDrone:
    def __init__(self,p_h,p_v,p_deltaT,p_lidar):

self.__h=p_h;self.__v=p_v;self.__deltaT=p_deltaT;self.__Lidar=p_lidar
        self.__SW=0.;self.__A=0.;self.__Nl=0;self.__Nf=0;
        self.__density=0;self.__widthSpace=0.;self.__lengthSpace=0.
        self.__pointX=0;self.__pointY=0
        self.__lPoints=[]
    def getSW(self):
        self.__SW=2*self.__h*math.tan(self.__Lidar.getFOV()/2.0)
        return(self.__SW)
    def getA(self):
        self.__A=self.__SW*self.__v*self.__deltaT
        return(self.__A)
```

```
    def getNl(self):
        self.__Nl=self.__deltaT*self.__Lidar.getSR()
        return(self.__Nl)
    def getDensity(self):
        self.__density=self.__Nl*self.__Lidar.getNf()/self.getA()
        return(self.__density)
    def getWidthSpace(self):

self.__widthSpace=self.__SW*self.__Lidar.getSR()/self.__Lidar.getPRR()
        return(self.__widthSpace)
    def getLengthSpace(self):
        self.__lengthSpace=self.__v/self.__Lidar.getSR()
        return(self.__lengthSpace)
```

**Question 9**

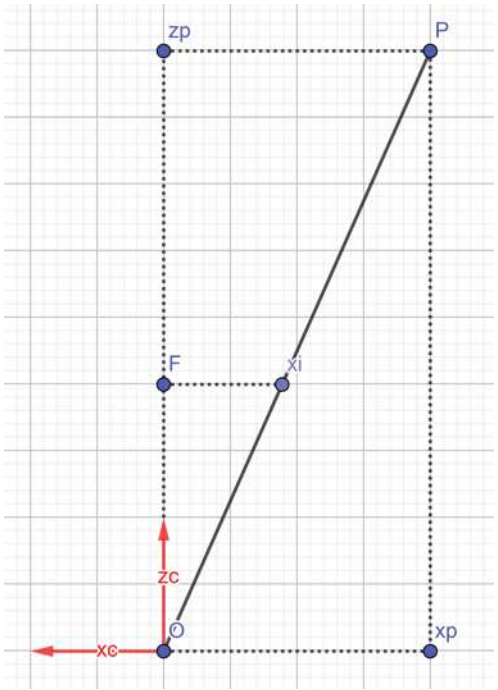
```
def listeFootPrint(self):
        for i in range(int(self.getNl())):
            for j in range(int(self.__Lidar.getNf())):
                self.__pointX=self.__lengthSpace*j
                self.__pointY=self.__widthSpace*i
                self.__lPoints.append((self.__pointX,self.__pointY))
        return(self.__lPoints)
```

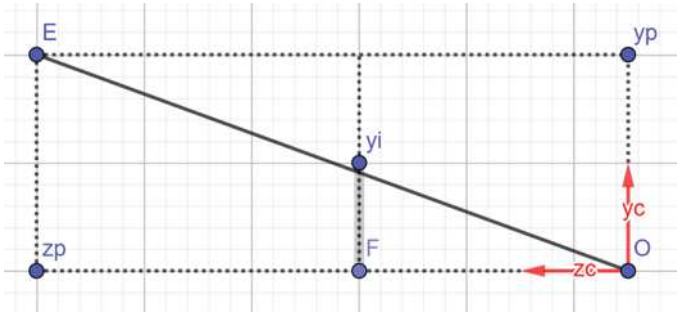
**Question 10**

Ce terme signifie que l'acquisition des points 3D s'effectue avec une résolution spatiale fine. Par pulse du lidar, il est possible d'acquérir plusieurs échos par points 3D.
Si la répartition spatiale n'est pas uniforme il faut procéder à un traitement de « rastering » dont le but est de construire une grille régulière (maillage) à partir des points de mesure.

## PARTIE 2 : Modélisation 3D par lasergrammétrie

**Question 11**



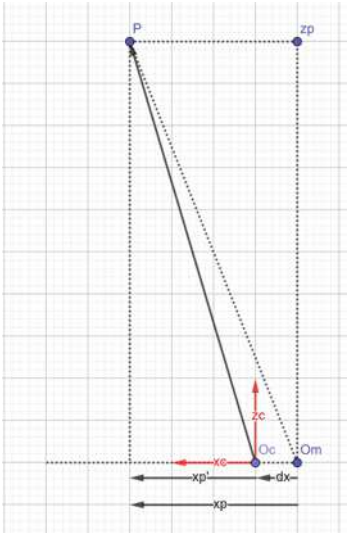

$$\frac{f}{z_p} = -\frac{x_i}{x_p} \rightarrow x_i = -x_p \frac{f}{z_p}$$

$$\frac{f}{z_p} = \frac{y_i}{y_p} \rightarrow y_i = y_p \frac{f}{z_p}$$

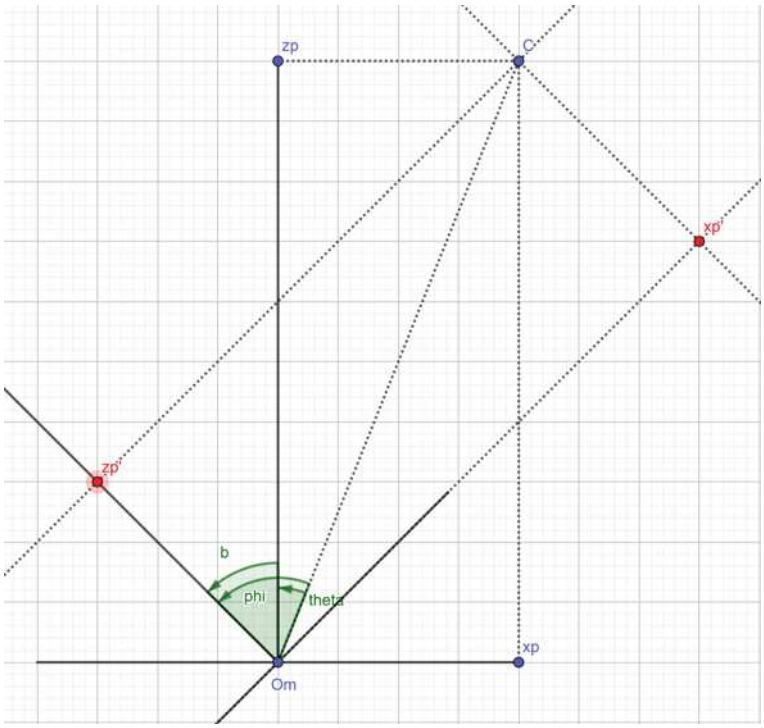
**Question 12**

$$x_i' = -fx_p \quad \text{donc} \quad x_i = \frac{x_i'}{z_i} = \frac{-fx_p}{z_p}$$
$$y_i' = fy_p \qquad y_i = \frac{y_i'}{z_i} = \frac{fy_p}{z_p}$$
$$z_i = z_p$$

**Question 13**




$$x_p' = x_p - d_x$$
$$y_p' = y_p - d_y$$
$$z_p' = z_p - d_z$$




$$\varphi = \beta + \theta$$
$$z_p = \rho\cos\theta$$
$$x_p = -\rho\sin\theta$$
$$x_p' = -\rho\sin(\beta+\theta) = -\rho\sin\beta\cos\theta - \rho\cos\beta\sin\theta = -z_p\sin\beta + x_p\cos\beta$$
$$z_p' = \rho\cos(\beta+\theta) = \rho\cos\beta\cos\theta - \rho\sin\beta\sin\theta = z_p\cos\beta + x_p\sin\beta$$
$$\begin{bmatrix} x_p' \\ y_p' \\ z_p' \end{bmatrix} = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}$$

**Question 14**

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \cong FRT \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = FP \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \cong F \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -d_x \\ 0 & 1 & 0 & -d_y \\ 0 & 0 & 1 & -d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = FP \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

$$P = \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -d_x \\ 0 & 1 & 0 & -d_y \\ 0 & 0 & 1 & -d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & -(R_{11}d_x + R_{12}d_y + R_{13}d_z) \\ R_{21} & R_{22} & R_{23} & -(R_{21}d_x + R_{22}d_y + R_{23}d_z) \\ R_{31} & R_{32} & R_{33} & -(R_{31}d_x + R_{32}d_y + R_{33}d_z) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} R & -Rd \\ 0_3^T & 1 \end{bmatrix}$$

**Question 15**

$$u = \frac{L}{2} + x_i$$

$$v = \frac{H}{2} + y_i$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = SD \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{L_{cmos}}{2} \\ 0 & 1 & \frac{H_{cmos}}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \cong SDFP \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{L_{cmos}}{2} \\ 0 & 1 & \frac{H_{cmos}}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & -Rd \\ 0_3^T & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

**Question 16**

$$\begin{bmatrix} R & -Rd \\ 0_3^T & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & -(R_{11}d_x + R_{12}d_y + R_{13}d_z) \\ R_{21} & R_{22} & R_{23} & -(R_{21}d_x + R_{22}d_y + R_{23}d_z) \\ R_{31} & R_{32} & R_{33} & -(R_{31}d_x + R_{32}d_y + R_{33}d_z) \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} R & -Rd \\ 0_3^T & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11}(x_p - d_x) + R_{12}(y_p - d_y) + R_{13}(z_p - d_z) \\ R_{21}(x_p - d_x) + R_{22}(y_p - d_y) + R_{23}(z_p - d_z) \\ R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \\ 1 \end{bmatrix}$$

$$F \begin{bmatrix} R & -Rd \\ 0_3^T & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{11}(x_p - d_x) + R_{12}(y_p - d_y) + R_{13}(z_p - d_z) \\ R_{21}(x_p - d_x) + R_{22}(y_p - d_y) + R_{23}(z_p - d_z) \\ R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \\ 1 \end{bmatrix}$$

$$F \begin{bmatrix} R & -Rd \\ 0_3^T & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} -f\left(R_{11}(x_p - d_x) + R_{12}(y_p - d_y) + R_{13}(z_p - d_z)\right) \\ f\left(R_{21}(x_p - d_x) + R_{22}(y_p - d_y) + R_{23}(z_p - d_z)\right) \\ R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \end{bmatrix}$$

$$DF \begin{bmatrix} R & -Rd \\ 0_3^T & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{L_{cmos}}{2} \\ 0 & 1 & \frac{H_{cmos}}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -f\left(R_{11}(x_p - d_x) + R_{12}(y_p - d_y) + R_{13}(z_p - d_z)\right) \\ f\left(R_{21}(x_p - d_x) + R_{22}(y_p - d_y) + R_{23}(z_p - d_z)\right) \\ R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \end{bmatrix}$$

$$= \begin{bmatrix} -f\left(R_{11}(x_p - d_x) + R_{12}(y_p - d_y) + R_{13}(z_p - d_z)\right) + \frac{L_{cmos}}{2}\left(R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z)\right) \\ f\left(R_{21}(x_p - d_x) + R_{22}(y_p - d_y) + R_{23}(z_p - d_z)\right) + \frac{H_{cmos}}{2}\left(R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z)\right) \\ R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \end{bmatrix}$$

$$SDFP \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} -S_x f\left(R_{11}(x_p - d_x) + R_{12}(y_p - d_y) + R_{13}(z_p - d_z)\right) + S_x \frac{L_{cmos}}{2} R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \\ S_y f\left(R_{21}(x_p - d_x) + R_{22}(y_p - d_y) + R_{23}(z_p - d_z)\right) + S_y \frac{H_{cmos}}{2} R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \\ R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \end{bmatrix}$$

Selon le principe des coordonnées homogènes :

$$[u \quad v \quad 1] \cong [wu \quad wv \quad w] = [u' \quad v' \quad w]$$

$$\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = \begin{bmatrix} -S_x f\left(R_{11}(x_p - d_x) + R_{12}(y_p - d_y) + R_{13}(z_p - d_z)\right) + S_x \frac{L_{cmos}}{2} R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \\ S_y f\left(R_{21}(x_p - d_x) + R_{22}(y_p - d_y) + R_{23}(z_p - d_z)\right) + S_y \frac{H_{cmos}}{2} R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \\ R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z) \end{bmatrix}$$

$$\begin{bmatrix} u = \frac{u'}{w} \\ v = \frac{v'}{w} \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \left[\frac{L_{cmos}}{2} - f\frac{\left(R_{11}(x_p - d_x) + R_{12}(y_p - d_y) + R_{13}(z_p - d_z)\right)}{\left(R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z)\right)}\right] \\ s_y \left[\frac{H_{cmos}}{2} + f\frac{\left(R_{21}(x_p - d_x) + R_{22}(y_p - d_y) + R_{23}(z_p - d_z)\right)}{\left(R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z)\right)}\right] \\ 1 \end{bmatrix}$$

**Question 17**

```
def read3dPointTextFile(pointFileName):
    fileNuagePoints=open(pointFileName,'r')
    linesNuage=fileNuagePoints.readlines()
    fileNuagePoints.close()
    N_Nuage=len(linesNuage)
    tabNuage=np.zeros((N_Nuage,4))
    tabVecNorm=np.zeros((N_Nuage,3))
    for i in range(len(linesNuage)):
        oneLine=linesNuage[i]
        tabNuage[i,0]=float(oneLine.split(' ')[0])
        tabNuage[i,1]=float(oneLine.split(' ')[1])
        tabNuage[i,2]=float(oneLine.split(' ')[2])
        tabNuage[i,3]=1.0
        tabVecNorm[i,0]=float(oneLine.split(' ')[3])
        tabVecNorm[i,1]=float(oneLine.split(' ')[4])
        tabVecNorm[i,2]=float(oneLine.split(' ')[5])
    return(tabNuage,tabVecNorm,N_Nuage)
```

**Question 18**

```
def matRotation(camOrien):
    matA=np.zeros((3,3))
    matA[0,]=np.array([1,0,0])
    matA[1,]=np.array([0,math.cos(camOrien[0,0]),math.sin(camOrien[0,0])])
    matA[2,]=np.array([0,-math.sin(camOrien[0,0]),math.cos(camOrien[0,0])])
    matB=np.zeros((3,3))
    matB[0,]=np.array([math.cos(camOrien[1,0]),0,-math.sin(camOrien[1,0])])
    matB[1,]=np.array([0,1,0])
    matB[2,]=np.array([math.sin(camOrien[1,0]),0,math.cos(camOrien[1,0])])
    matG=np.zeros((3,3))
    matG[0,]=np.array([math.cos(camOrien[2,0]),math.sin(camOrien[2,0]),0])
    matG[1,]=np.array([-math.sin(camOrien[2,0]),math.cos(camOrien[2,0]),0])
    matG[2,]=np.array([0,0,1])
    matR=np.zeros((3,3))
    matR=np.matmul(matB,matA)
    matR=np.matmul(matG,matR)
    return(matR)

def matProjection(matR,camPos):
    matRt=np.zeros((3,1))
    matRt=-np.matmul(matR,camPos)
    matP=np.zeros((4,4))
    matP[0,0:3]=matR[0,0:3]
    matP[1,0:3]=matR[1,0:3]
```

```
    matP[2,0:3]=matR[2,0:3]
    matP[0:3,3]=matRt[0:3,0]
    matP[3,3]=1
    return(matP)
```

**Question 19**

```
tabImage=np.zeros((N_Nuage,3))
```

Type : tableau de réels de N_Nuage lignes et de 3 colonnes car calcul matriciel
N_Nuage : nombre de lignes du tableau : nombre de points du nuage
Colonne 0 : composante $u'$
Colonne 1 : composante $v'$
Colonne 2 : composante $w$

**Question 20**

| Instruction Python | Traitement réalisé |
|---|---|
| `tabImage[i,]=np.matmul(matF,np.matmul(matP,tabNuage[i,]))` | $\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = FP \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$ |
| `tabImage[i,]=np.matmul(matS,np.matmul(matD,tabImage[i,]))` | $\begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = SDFP \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$ |
| `u[i,0]=round(tabImage[i,0]/tabImage[i,2])` | $u = s_x \left[ \frac{L_{cmos}}{2} - f \frac{\left(R_{11}(x_p - d_x) + R_{12}(y_p - d_y) + R_{13}(z_p - d_z)\right)}{\left(R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z)\right)} \right]$ |
| `v[i,0]=round(tabImage[i,1]/tabImage[i,2])` | $v = s_y \left[ \frac{H_{cmos}}{2} + f \frac{\left(R_{21}(x_p - d_x) + R_{22}(y_p - d_y) + R_{23}(z_p - d_z)\right)}{\left(R_{31}(x_p - d_x) + R_{32}(y_p - d_y) + R_{33}(z_p - d_z)\right)} \right]$ |

**Question 21**

| Instruction Python | Traitement réalisé |
|---|---|
| `dab[i,0]=round(math.sqrt((camPos[0,0]-tabNuage[i,0])**2+(camPos[1,0]-tabNuage[i,1])**2+(camPos[2,0]-tabNuage[i,2])**2),3)` | $dab = \sqrt{(x_c - x_p)^2 + (y_c - y_p)^2 + (z_c - z_p)^2}$ |

Rôle de la structure conditionnelle :

```
if(u[i,0]<L and u[i,0]>=0 and v[i,0]<H and v[i,0]>=0):
      lpixels_dab.append([int(u[i,0]),int(v[i,0]),dab[i,0],index])
      index=index+1
```

Permet de garder les valeurs de $u$ et de $v$ positives ou nulles et inférieures aux dimensions de l'image.

**Question 22**

On pose n la taille de la liste à trier.
Complexité en espace mémoire : O(n) et tri sur place
Complexité en temps : O(nlog(n)) dans le meilleur des cas

```
def partition_ag_simple(ldata,G,D):
    pivot=ldata[G];i=G;X=ldata[i]
    for j in range(G+1,D+1):
        if(ldata[j]>=pivot):
            ldata[j],ldata[i+1]=ldata[i+1],ldata[j]
            i=i+1
    if(i!=G):
        ldata[i],X=X,ldata[i]
    return(i)

def quickSort_ag_simple(ldata,G,D):
    if(G>=D):
        return(None)
    else:
```

```
        p=partition_ag_simple(ldata,G,D)
        quickSort_ag_simple(ldata,G,p-1)
        quickSort_ag_simple(ldata,p+1,D)
```

**Question 23**

```
def partition_ag(ldata,G,D):
    pivot=ldata[G][2];i=G;X=ldata[i]
    for j in range(G+1,D+1):
        if(ldata[j][2]>=pivot):
            i=i+1
            X=ldata[j]
            ldata[j]=ldata[i]
            ldata[i]=X
    ldata[i]=ldata[G]
    ldata[G]=X
    return(i)

def quickSort_ag(ldata,G,D):
    if(G>=D):
        return(None)
    else:
        p=partition_ag(ldata,G,D)
        quickSort_ag(ldata,G,p-1)
        quickSort_ag(ldata,p+1,D)
```

**Question 24**

Expression de k :

$$k = u + v \times L$$

Pseudo-code :
Entier v : index des lignes 0 à H-1
Entier u : index des colonnes 0 à L-1
Entrée : liste contenant des doublons de pixels « l_pixels_dab=[v,u,dab,index] »
Création d'une liste « ltempo » de taille égale au nombre de cases d'une image (LxH),

Dans une première boucle FOR sur la taille de « l_pixels_dab »
-calcul de l'index k unique à partir des pixels de « l_pixels_dab » : k=u+v*L
-Remplissage de la nouvelle liste avec l'élément correspondant de l_pixels_dab
Si un doublon est trouvé, la liste étant triée selon dab dans l'ordre décroissant, c'est la valeur de dab la plus petite qui est gardée.

Dans une deuxième boucle FOR
Nouvelle liste avec suppression des éléments égaux à [0,0,0,0]

```
def suppression_doublons(lpixels_dab,H,L):
    NCase=H*L
    ltempo=[[0,0,0,0]]*NCase
    l=[]
    for i in range(len(lpixels_dab)):
        index=lpixels_dab[i][0]+L*lpixels_dab[i][1]
        ltempo[index]=lpixels_dab[i]
    for i in range(NCase):
        if(ltempo[i]!=[0,0,0,0]):
            l.append(ltempo[i])
    return l
```

Deux boucles FOR successives sont utilisées : complexité en temps = O(n)
En revanche, le traitement n'est pas effectué uniquement avec la liste de départ. Au moins une liste supplémentaire de taille (L x H) est nécessaire.

**Question 25**

```
quickSort_ag(lpixels_dab,0,len(lpixels_dab)-1)
l1=suppression_doublons(lpixels_dab,H,L)
```

**Question 26**
La variable `img` est une matrice à deux dimensions de taille H x L, chaque cellule contient 4 réels de 32 bits. La taille totale est égale à H x L x 4 x 4 octets.

```
imSrc=cv2.imread("IMG_4000.jpg")
plt.figure(0)
plt.imshow(imSrc)

#Stockage de l'image solide dans un tableau numpy à 4 éléments
dt=np.dtype((np.float32,(4)))
img=np.zeros((H,L),dtype=dt)
for i in range(len(l1)):
      img[l1[i][1]][l1[i][0]][0]=float(imSrc[l1[i][1]][l1[i][0]][0]/255.0)
      img[l1[i][1]][l1[i][0]][1]=float(imSrc[l1[i][1]][l1[i][0]][1]/255.0)
      img[l1[i][1]][l1[i][0]][2]=float(imSrc[l1[i][1]][l1[i][0]][2]/255.0)
      img[l1[i][1]][l1[i][0]][3]=l1[i][2]
```

**Question 27**
Complexité en temps du programme principal en additionnant chaque coût (boucle FOR + fonction de tri + fonction de suppression des doublons + remplissage de l'image solide) :

$$Coût = n + nlog(n) + n + n \approx nlog(n)$$

Complexité en espace du programme principal en additionnant la taille des variables du type tableau et liste :
-variable `tabImage` : tableau 2D de taille N_Nuage x 3
-variable `lpixels_dab` : liste de 4 éléments x N_Nuage

-fonction de tri : tri sur place : 1 x N_Nuage
-fonction suppression des doublons : liste auxiliaire : 2 x N_Nuage
-variable `img` : tableau 3D de taille H x L x 4 : indépendant de N_Nuage

$$Coût = n + n + n + n \approx n$$

## PARTIE 3 : Solution de photogrammétrie

**Question 28**
En identifiant les triangles semblables :

$$\Delta PO_1N \propto \Delta I_1O_1x_1 \rightarrow \frac{Z}{f} = \frac{X}{x_1} \qquad \Delta PO_2N \propto \Delta I_1O_2x_2 \rightarrow \frac{Z}{f} = \frac{X-B}{x_2}$$

Par soustraction :

$$\frac{Z}{f}x_1 - \frac{Z}{f}x_2 = \frac{Z}{f}(x_1 - x_2) = X - X + B = B$$

Donc pour Z, X et Y :

$$Z = \frac{Bf}{(x_1 - x_2)} = \frac{Bf}{d}$$

$$\frac{X}{x_1} = \frac{Z}{f} \rightarrow X = \frac{Zx_1}{f} = \frac{Bx_1}{d} \qquad \frac{Y}{y_1} = \frac{Z}{f} \rightarrow Y = \frac{Zy_1}{f} = \frac{By_1}{d}$$

Ce qui permet de retrouver l'expression matricielle :

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{B}{d}\begin{bmatrix} x_1 \\ y_1 \\ f \end{bmatrix}$$

**Question 29**

$$d = \frac{Bf}{Z} = \frac{10 \times 0.035}{30} = 0.011$$

$$\Delta X = \frac{B}{d}\Delta x_1 = \frac{10}{0.011} \times \frac{0.0359}{7360} = 4\ mm$$

$$\Delta Y = \frac{B}{d}\Delta y_1 = \frac{10}{0.011} \times \frac{0.024}{4912} = 4\ mm$$

**Question 30**

$$d = \frac{Bf}{Z}$$

$$\Delta d = \frac{d}{dZ}\left(\frac{Bf}{Z}\right) = -\frac{Bf}{Z^2}\Delta Z$$

$$|\Delta Z| = \frac{Z^2}{Bf}\Delta d = -\frac{30^2}{10 \times 0.035} \times \frac{0.0359}{7360} = 1.2\ cm$$

Amélioration de la résolution en profondeur en augmentant la distance $B$.

**Question 31**

Si deux pixels d'une même ligne sont en correspondance, i.e. sont égaux : $L(k,j) = R(k,i)$

Or : $d = j - i$

Donc : $L(k, i+d) - R(k,i) = 0$

Ou bien : $L(k,j) - R(k, j-d) = 0$

**Question 32**

En exécutant pas à pas l'algorithme donné et en initialisant la table des coûts :

Calcul de la distance : $D(0,0) = |L(0) - R(0)| = |1-4| = 3$

Calcul des choix possibles :

$$g(1,0) = min\begin{cases} g(0,-1) + D(0,0) = 53 \\ g(-1,-1) + D(0,0) = 3 \\ g(-1,0) + D(0,0) = 53 \end{cases}$$

En choisissant le min : $g(0,0) = g(-1,-1) + D(0,0) = 3$

Ce qui correspond à un déplacement en diagonal. La résolution minimale est égale à 1 pixel car les déplacements s'effectuent de pixel en pixel.

Table des coûts et chemin :

| | $j=-1$ | $j=0$ | $j=1$ | $j=2$ | $j=3$ | $j=4$ |
|---|---|---|---|---|---|---|
| $i=-1$ | 0 | 50 | 50 | 50 | 50 | 50 |
| $i=0$ | 50 | 3 | 6 | 6 | 7 | 9 |
| $i=1$ | 50 | 7 | 7 | 7 | 6 | 7 |
| $i=2$ | 50 | 12 | 12 | 9 | 7 | 6 |
| $i=3$ | 50 | 19 | 19 | 13 | 10 | 8 |
| $i=4$ | 50 | 26 | 26 | 17 | 13 | 10 |

**Question 33**

```
import numpy as np

L=[1,1,4,5,6]
R=[4,5,6,8,8]
N=len(L)
g=np.zeros((N+1,N+1))
P=np.zeros((N+1,N+1))
#Coût maximal
vmax=50
for i in range(N+1):
    g[0,i]=vmax
    g[i,0]=vmax

g[1,1]=abs(L[0]-R[0])
```

```
for i in range(1,N+1):
    for j in range(1,N+1):
        D=abs(L[j-1]-R[i-1])
        g[i,j]=D+np.amin([g[i,j-1],g[i-1,j-1],g[i-1,j]])
        P[i,j]=np.argmin([g[i,j-1],g[i-1,j-1],g[i-1,j]])+1
```

**Question 34**

```
i=N
j=N
d=[]
while(i>0 and j >0):
    #print("i=",i,"j:",j)
    if(P[i,j]==2):
        #print("diag")
        if(g[i,j]==g[i-1,j-1]):
            d.append((j-i,j-1))
        i=i-1
        j=j-1
    elif(P[i,j]==1):
        #print("hori")
        if(g[i,j]==g[i,j-1]):
            d.append((j-i,j-1))
        j=j-1
    else:
        #print("vert")
        if(g[i,j]==g[i-1,j]):
            d.append((j-i,j-1))
        i=i-1
```

**Question 35**

$L$ : Largeur d'une image en pixels
$H$ : Hauteur d'une image en pixels

| Brique algorithmique | Complexité en mémoire | Complexité en temps |
|---|---|---|
| Initialisation | $O(L^2)$ | $O(L)$ |
| DP table | $O(L^2)$ | $O(L^2)$ |
| Liste des appariements | $O(L^2)$ | $O(L^2)$ |
| Pour une paire d'images | $O(HL^2)$ | $O(HL^2)$ |

**Question 36**

| **Critère** | **Lasergrammétrie** | **Photogrammétrie** |
|---|---|---|
| Plage d'utilisation | Distance importante | Distance courte/moyenne |
| Résolution spatiale | 2 cm | 4 mm |
| Résolution en profondeur | 1 cm | 1 cm |
| Intégration matérielle | Caméra+Lidar | 2 caméras |
| Rasterisation du nuage de points | Oui | Oui |
| Coût temps de calcul | $nlog(n)$ =1.7e6 x log(1.7e6)= 10 591 763 | $HL^2$ =800 x 1200 x 1200= 1 152 000 000 |
| Intégration embarquée des traitements numériques | Oui possible en estimant un calcul élémentaire à 10 ns, de l'ordre de la centaine de millisecondes | Non, sur la même base de temps élémentaire, de l'ordre de la dizaine de secondes pour une seule paire d'images |

## PARTIE 4 : Analyse des images par segmentation

**Question 37**

```
def conv_y(image,h):
    H,L=np.shape(image)
    imgF=np.zeros((H-2,L-2))
    for y in range(0,H-2):
        for x in range(0,L-2):
            for i in range(0,3):
                for j in range(0,3):
                    imgF[y,x]=imgF[y,x]+h[i,j]*image[y+i,x+j]
    return(imgF)
```

$y =$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **55.44** | **67.33** | **63** | 53.55 | 45.33 | 53.11 | 63.44 | 64.11 | 57.11 | 50.55 |
| **59.44** | **71.33** | **61** | 51.11 | 48.22 | 63.88 | 71.11 | 67.33 | 52.22 | 50.33 |
| **44.88** | **57.33** | **53.44** | 57.88 | 58.44 | 66 | 58.22 | 60.11 | 38.44 | 48.55 |
| 53.33 | 51.77 | 53.77 | 55 | 59.44 | 62.22 | 52 | 52.66 | 39.11 | 49.33 |
| 45.11 | 34.88 | 36.11 | 48.22 | 58.33 | 63.22 | 51.22 | 53 | 42.66 | 50.22 |
| 53.44 | 53.55 | 53.66 | 56.88 | 56.11 | 50.55 | 42.66 | 51 | 48.22 | 55.88 |
| 51 | 52.77 | 55.66 | 55.66 | 48.77 | 50.33 | 46.66 | 54.77 | 52.55 | 50.77 |

**Question 38**

Par distributivité du produit de convolution :

$$DoG(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma) = I(x,y) * \big(G(x,y,k\sigma) - G(x,y,\sigma)\big)$$

**Question 39**

$$\Delta L(x,y,\sigma) = \frac{x^2+y^2-2\sigma^2}{2\pi\sigma^6} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$$

Filtre LoG : L'opérateur laplacien est appliqué uniquement à la fonction Gaussienne 2D en raison des propriétés du produit de convolution vis-à-vis de la dérivée.

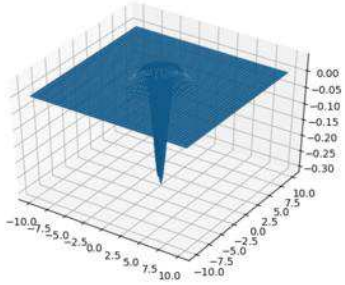
**Question 40**

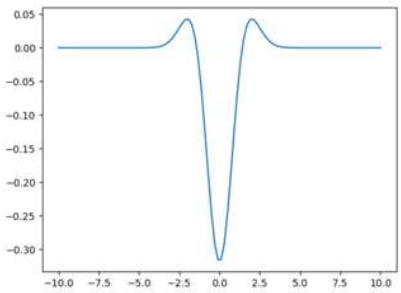
En utilisant la formule de Taylor d'un accroissement de $\sigma(k-1)$ autour du point $\sigma$ à l'ordre 1 :

$$\frac{\partial G}{\partial \sigma} = \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{\sigma(k-1)} = \frac{x^2+y^2-2\sigma^2}{2\pi\sigma^6} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$$

$$DoG = \sigma(k-1)\Delta L(x,y,\sigma)$$

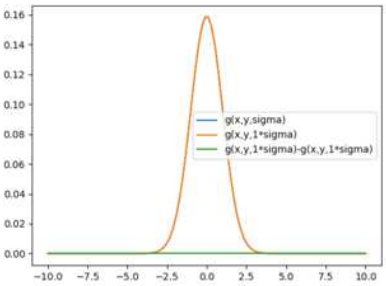
**Question 41**

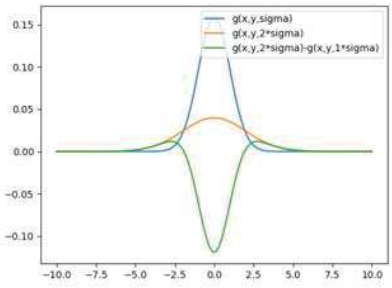


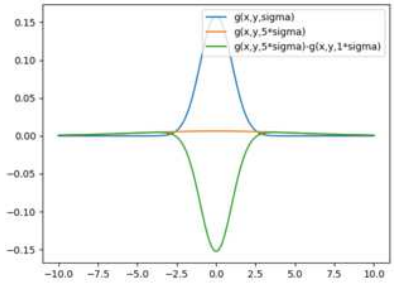


Pour k=1 DoG et LoG sont semblables.

Le paramètre k va influencer l'amplitude du pic mais pas sa localisation.








**Question 42**

```
def g2D(x,y,sigma):
    return((1.0/(2*np.pi*sigma**2))*np.exp(-(x**2+y**2)/(2*sigma**2)))

def masque2D(p,sigma):
    h=np.zeros((2*p+1,2*p+1))
    somme=0
    #Calcul de -p à p de la Gaussienne
    #Pour symétrie dans le masque 2D
    for i in range(-p,p+1):
        for j in range(-p,p+1):
            h[i+1,j+1]=g2D(i,j,sigma)
    for i in range(0,2*p+1):
            for j in range(0,2*p+1):
                somme=somme+h[i,j]
    return(h/somme)
```

**Question 43**

Depuis la définition du produit de convolution en 2D :

$$G^{\sigma}(x,y) * I(x,y) = G^{\sigma}(x)G^{\sigma}(y) * I(x,y)$$

$$= \iint G^{\sigma}(x,y)I(x-x',y-y')dx'dy' = \int G^{\sigma}(x)\left(\int G^{\sigma}(y)I(x-x',y-y')dy'\right)dx'$$

$$= G^{\sigma}(x) * G^{\sigma}(y) * I(x,y)$$

**Question 44**

Une Convolution 2D : pour 1 pixel et un masque 2D (matrice) de dimension $(2p+1)$ :
$N_c = (2p+1) \times (2p+1)$
Deux convolutions 1D successives pour un 1 pixel et deux masques 1D (vecteur) de dimension $(2p+1)$ :
$N_c = (2p+1) + (2p+1)$ d'où le gain : $gain = \frac{(2p+1)\times(2p+1)}{(2p+1)+(2p+1)}$

**Question 45**

```
def g1D(x,sigma):
    """Calcul de la Gaussienne pour x et sigma données"""
    return((1.0/(np.sqrt(2*np.pi)*sigma))*np.exp(-(x**2)/(2*sigma**2)))

def masque1D(p,sigma):
    """Calcul du masque de -p à p de la Gaussienne pour symétrie
    Taille du masque : 2p+1"""
    h=np.zeros((1,2*p+1))
    somme=0
    for i in range(-p,p+1):
        h[0,i+1]=g1D(i,sigma)
    for i in range(0,2*p+1):
        somme=somme+h[0,i]
    return(h/somme)

def conv_L(image,sigma,p):
    """Produit de convolution avec séparabilité filtre gaussien
    Associé à la fonction de convolution L(x,y,sigma)
```

```
    Sans gestion des bords"""
    H,L=np.shape(image)
    conv_tab=np.zeros((H,L),dtype=np.float32)
    conv_tab2=np.zeros((H,L),dtype=np.float32)
    #Masque 1D Gaussien
    h1D=masque1D(p,sigma)
    print(h1D)
    #Convolution suivant la largeur de l'image
    for x in range(0,H-p):
        for y in range(p,L-p):
            for j in range(0,2*p+1):
                conv_tab[x,y]=conv_tab[x,y]+h1D[0,j]*image[x,y+j-p]
    #Convolution suivant la hauteur de l'image
    for x in range(p,H-p):
        for y in range(0,L-p):
            for i in range(0,2*p+1):
                conv_tab2[x,y]=conv_tab2[x,y]+h1D[0,i]*conv_tab[x+i-p,y]
    return(conv_tab2)

def DoG(image,k,sigma,p):
    """Calcul de la différence de Gaussiennes
    Dog=L(x,y,ksigma)-L(x,y,sigma)"""
    #Soustraction pixel par pixel
    dog_tab=abs(conv_L(image,k*sigma,p)-conv_L(image,sigma,p))
    return(dog_tab)
```

**Question 46**

```
def DoGN(image,N,k,sigma,p):
    """Calcul de l'image filtrée pour 1 octave de N-1 Dog"""
    H,L=np.shape(image)
    maxi=np.zeros((H,L))
    dog_liste=[]
    old=conv_L(image,sigma,p)
    for i in range(N):
        #Soustraction pixel par pixel
        new=conv_L(image,k**(i+1)*sigma,p)
        dog=abs(new-old)
        dog_liste.append(dog)
        old=new
    #Recherche maxi pixel dans chaque DoG
    for x in range(H):
        for y in range(L):
            #Valeur maximale de référence prise sur le premier dog à pixel
donné
            maxi[x,y]=dog_liste[0][x,y]
            for k in range(N):
                if(dog_liste[k][x,y]>maxi[x,y]):
                    maxi[x,y]=dog_liste[k][x,y]
    return(maxi)
```

**Question 47**

```
def DoGMR(image,N,k,sigma,p,nO):
    """Calcul sur nO octaves du DoGN
    Sigma multiplié par 2 à chaque octave
    Recherche du maximum parmi les DoGN de chaque octave
    Parcours pixel par pixel sur l'image d'origine (H,L)
    Pour les images sous échantillonnées, les indices sont recalculés"""
    H,L=np.shape(image)
    dogmr_liste=[]
    maxi=np.zeros((H,L))
    for i in range(0,nO):
```

```
        imgSub=image[::2**(i),::2**(i)]
        #A chaque octave, multiplication par 2 de sigma
        dogn=DoGN(imgSub,N,k,sigma*2**(i),p)
        print(np.shape(dogn))
        print(dogn)
        dogmr_liste.append(dogn)
    #Recherche du maxi parmi les maximums de chaque octave
    maxi=dogmr_liste[0]
    for x in range(H):
        for y in range(L):
            for i in range(1,nO):

valMax=max(dogmr_liste[i][x//(2**i),y//(2**i)],dogmr_liste[i-1][x//(2**(i-
1)),y//(2**(i-1))])
                if(valMax>maxi[x,y]):
                    maxi[x,y]=valMax

  return(maxi)
```

**Question 48**

$$(1-w^n)^k = 1-P$$

Soit $k = \frac{\log(1-P)}{\log(1-w^n)}$

**Question 49**

```
from IPython import get_ipython
get_ipython().magic('reset -sf')
from sklearn import linear_model, datasets
import csv
def contour_ransac(X,Y):
    ransac = linear_model.RANSACRegressor()
    ransac.fit(X,Y)
    a=ransac.estimator_.coef_
    b=ransac.estimator_.intercept_
    angle=np.arctan(a)
    writer=csv.writer(open("angle.csv","a"))
    writer.writerow(a,b,angle)
    return(angle)

# Prédiction
model,angle=contour_ransac(X,y)
```

**Question 50**

Le principe de clustering ou de classification automatique est de répartir un ensemble d'informations en groupes (clusters) en fonction de leurs similitudes. Les K-means (K-moyennes) permettent de diviser les éléments en K groupes en minimisant une fonction de distance entre les éléments au sein de chaque groupe. On évalue la distance d'un point par rapport à la distance moyenne des objets de son groupe.

```
from IPython import get_ipython
get_ipython().magic('reset -sf')
from sklearn import linear_model, datasets
from sklearn.cluster import KMeans
import csv
data=csv.reader(open("angle.csv","r"))
x=pd.read_csv('angle.csv')
```

```
model=KMeans(n_clusters=18)
model.fit(x)
print(model.labels_)
```

**Question 51**

$$M(\mathrm{A_i}, \mathrm{B_i}, \mathrm{C_i}) = \sum_{j=1}^{n} \left(A_i Xj + B_i Y_j + C_i - Z_j\right)^2$$

$$\frac{\partial M(\mathrm{A_i}, \mathrm{B_i}, \mathrm{C_i})}{\partial A_i} = 0$$

$$\leftrightarrow \frac{\partial}{\partial A_i} \sum_{j=1}^{n} \left(A_i X_j + B_i Y_j + C_i - Z_j\right)^2 = 0$$

$$\leftrightarrow \sum_{j=1}^{n} 2X_j \left(A_i X_j + B_i Y_j + C_i - Z_j\right) \quad = 0$$

$$\leftrightarrow \sum_{j=1}^{n} \left(A_i X_j^2 + X_j B_i Y_j + X_j C_i - X_j Z_j\right) \quad = 0$$

$$\leftrightarrow \left(\sum_{j=1}^{n} X_j^2\right) A_i + \left(\sum_{j=1}^{n} X_j \ \ Y_j\right) B_i + \left(\sum_{j=1}^{n} X_j \ \ \right) C_i = \sum_{j=1}^{n} Z_j X_j$$

De même :

$$\frac{\partial M(\mathrm{A_i}, \mathrm{B_i}, \mathrm{C_i})}{\partial B_i} = 0$$

$$\leftrightarrow \left(\sum_{j=1}^{n} X_j \ \ Y_j\right) A_i + \left(\sum_{j=1}^{n} Y_j^2\right) B_i + \left(\sum_{j=1}^{n} Y_j \ \ \right) C_i = \sum_{j=1}^{n} Z_j Y_j$$

De même :

$$\frac{\partial M(\mathrm{A_i}, \mathrm{B_i}, \mathrm{C_i})}{\partial C_i} = 0$$

$$\leftrightarrow \left(\sum_{j=1}^{n} X_j \ \ \right) A_i + \left(\sum_{j=1}^{n} Y_j \ \ \right) B_i + \left(\sum_{j=1}^{n} 1\right) C_i = \sum_{j=1}^{n} Z_j$$

Le système des trois équations issues des dérivées partielles s'écrit alors sous forme matricielle :

$$\begin{bmatrix} \sum_{j=1}^{n} \mathrm{x_j^2} & \sum_{j=1}^{n} \mathrm{x_j}\,.\,\mathrm{y_j} & \sum_{j=1}^{n} \mathrm{x_j} \\ \sum_{j=1}^{n} \mathrm{x_j}\,.\,\mathrm{y_j} & \sum_{j=1}^{n} \mathrm{y_j^2} & \sum_{j=1}^{n} \mathrm{y_j} \\ \sum_{j=1}^{n} \mathrm{x_j} & \sum_{j=1}^{n} \mathrm{y_j} & \mathrm{n} \end{bmatrix} . \begin{bmatrix} \mathrm{A_i} \\ \mathrm{B_i} \\ \mathrm{C_i} \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} \mathrm{x_j}\,.\,\mathrm{z_j} \\ \sum_{j=1}^{n} \mathrm{y_j}\,.\,\mathrm{z_j} \\ \sum_{j=1}^{n} \mathrm{z_j} \end{bmatrix}$$

**Question 52**

$$F = \begin{bmatrix} x_1 & y_1 & 1 \\ \cdots & & \\ x_j & y_j & 1 \\ \cdots & & \\ x_n & y_n & 1 \end{bmatrix} \qquad D = \begin{bmatrix} z_1 \\ \cdots \\ z_j \\ \cdots \\ z_n \end{bmatrix} \qquad E = \begin{bmatrix} A_i \\ B_i \\ C_i \end{bmatrix}$$

**Question 53**

```
def moindre_carre(x,y,z):

    tmp_F = []
    tmp_D = []
    for i in range(len(xs)):
        tmp_F.append([xs[i], ys[i], 1])
        tmp_D.append(zs[i])
    D = np.matrix(tmp_D).T
    F = np.matrix(tmp_F)
    fit = (F.T * F).I * F.T * D
    erreur = D - A * fit

    print ("erreur:")
    print (erreur)
    return fit
```

**Question 54**

$$\theta = atan\left(\frac{B_i}{A_i}\right) \qquad \varphi = atan\left(\frac{\left(\frac{C_i}{A_i}\right)}{\sqrt{1+\left(\frac{B_i}{A_i}\right)^2}}\right)$$

**Question 55**

K =18*18=324

```
from IPython import get_ipython
get_ipython().magic('reset -sf')
from sklearn import linear_model, datasets
from sklearn.cluster import KMeans
import csv

data=csv.reader(open("angle3D.csv","r"))

def estime_plan(data):
  x=pd.read_csv('angle.csv')
  model=KMeans(n_clusters=324)
  model.fit(x)
  print(model.labels_)
  return(model.labels_)
```

**Question 56**

| | $\theta, \varphi$ | $\theta$ | $\varphi$ |
|---|---|---|---|
| ***Erreur moyenne Laser*** | 2.9 | 1.28 | 4.5 |
| ***Erreur moyenne photogrammétrie*** | 8.13 | 4.8 | 11.4 |