Richard CAUCHY
n° de candidat : 24981

2022

Titre : Observer Et Etudier Les Objets Spatiaux a L'Aide Des points De Lagrange

Problématique

Comment peut-on prédire les éventuelles chutes d'asteroïdes sur la Terre a l'aide des points de Lagrange ?

Plan

Objectif :

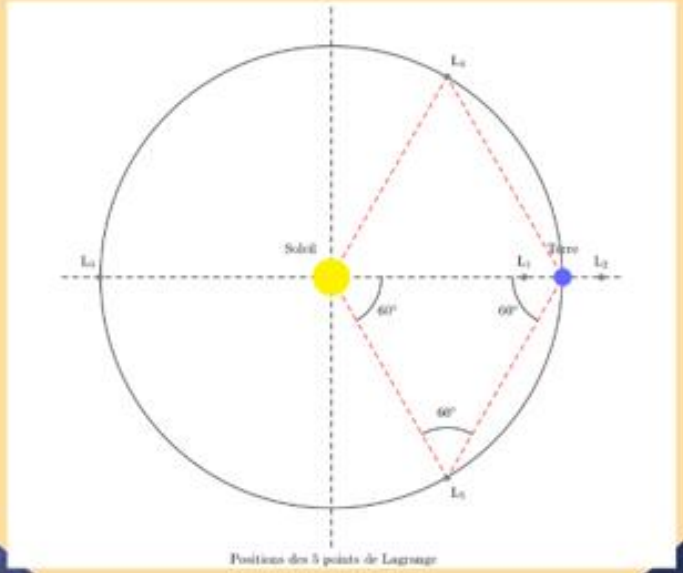
- Déterminer la position des 5 points de Lagrange

- Concevoir un algorithme permettant de déterminer la taille, la position, la vitesse et la direction d'un objet pris en photo sur les 5 points de Lagrange.

- Déterminer la trajectoire d'un objet étudié a l'aide des informations élémentaires récoltées.

- Déterminer si l'objet va rentrer en collision ou non avec la Terre

Les points de Lagrange



Positions des 5 points de Lagrange

#1

#2

#3

Equation de L1, L2 et L3

Méthode :

- Repère Galiléen centré sur le barycentre O

- Projeter le PFD sur (Soleil-Terre)

- On exprime l'equation en fonction du rapport de masse k, en utilisant la masse réduite.

Equations de la forme :

$$L1 : (1+k)x - 1 + \frac{1}{(1-x)^2} - \frac{k}{x^2} = 0$$

$$L2 : (1+k)x + 1 - \frac{1}{(1+x)^2} - \frac{k}{x^2} = 0$$

$$L3 : (1+k)x + k - \frac{k}{(1+x)^2} - \frac{1}{x^2} = 0$$

avec : $k = \frac{M}{M_0}$ où M et M_0 sont la masse de la Terre et du soleil

$x = \frac{LM}{a}$ avec LM : distance points de Lagrange-Terre; et a : distance M_0M

Solution des positions L4 et L5

Méthode :

- Exprimer les 3 forces en fonction de k

- Appliquer le PFD dans le repère tournant centré sur O (Barycentre)

- Projeter selon x et y

On obtient alors le système :

$$(1) : a^3 \frac{k}{1+k} * \frac{1}{(r_A)^3} * (\frac{a}{1+k} - x) - a^3 \frac{1}{k+1} * \frac{1}{(r_B)^3} * (x + \frac{ka}{1+k}) + x = 0$$
$$(2) : -a^3 \frac{k}{1+k} * \frac{1}{(r_A)^3} - a^3 \frac{1}{k+1} * \frac{1}{(r_B)^3} + 1 = 0$$

avec : $k = \frac{M}{M_0}$ où M et M_0 sont la masse de la Terre et du soleil

$x = \frac{LM}{a}$ avec LM : distance points de Lagrange-Terre; et a : distance M_0M

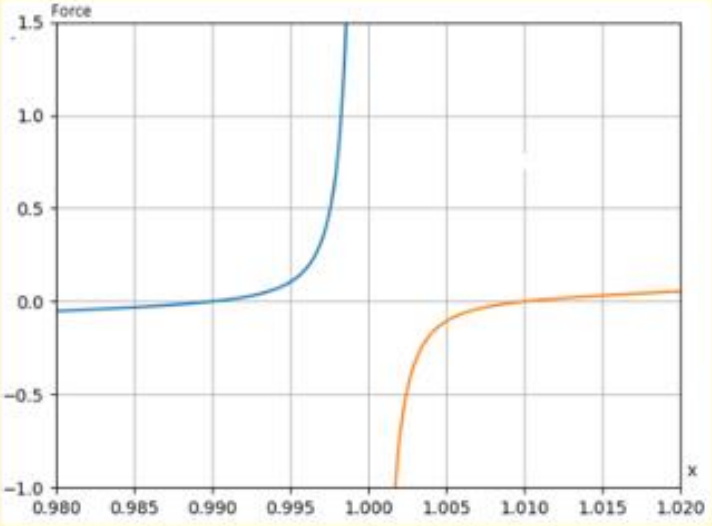
r_A: Distance Soleil-Points de Lagrange

r_B: Distance Terre-Points de Lagrange

On a alors $r_A = r_B = a$ solution du système

Donc les triangles (Soleil-L4-Terre) et (Soleil-L5-Terre) sont équilateraux

Résolution par dichotomie



Courbes représentatives des équations de L1 et L2

```
Le points de Lagrange L1 se situe a  0.009969329705336816 ua, (soit 1495399 km) de la Terre

Le points de Lagrange L2 se situe a  0.010036032619913815 ua, (soit -1505404 km) de la Terre

Le points de Lagrange L3 se situe a  2.0002000000000004 ua, (soit 300030000 km) de la Terre

Le points de Lagrange L4 se situe a  1 ua (soit 150000000 km) de la Terre

Le points de Lagrange L5 se situe a  1 ua (soit 150000000 km) de la Terre
```

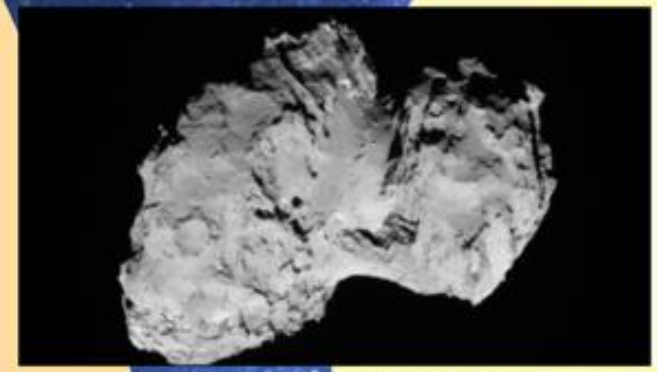
Détermination des caractéristques de l'asteroïde éudié

Hypothèse : Il y a un seul objet sur l'image et il est suffisemment proche

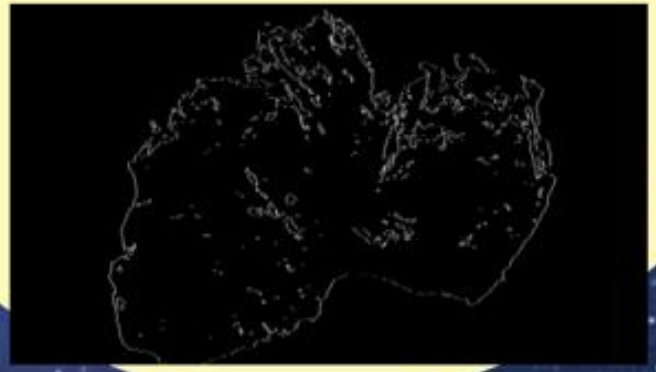
Transformation Image

Caractéristiques de l'asteroïde

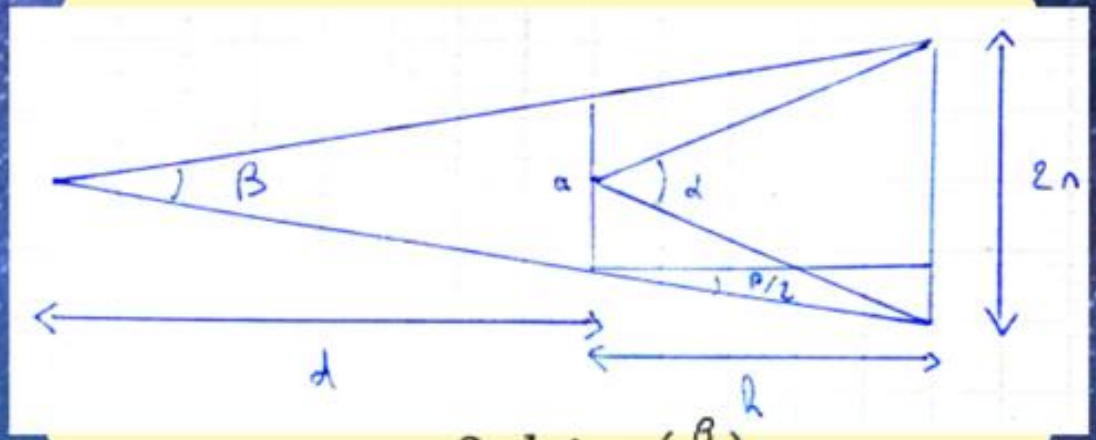
Transformation d'image en utilisant le gradient



Comète Tchouri publiée sur france24.com

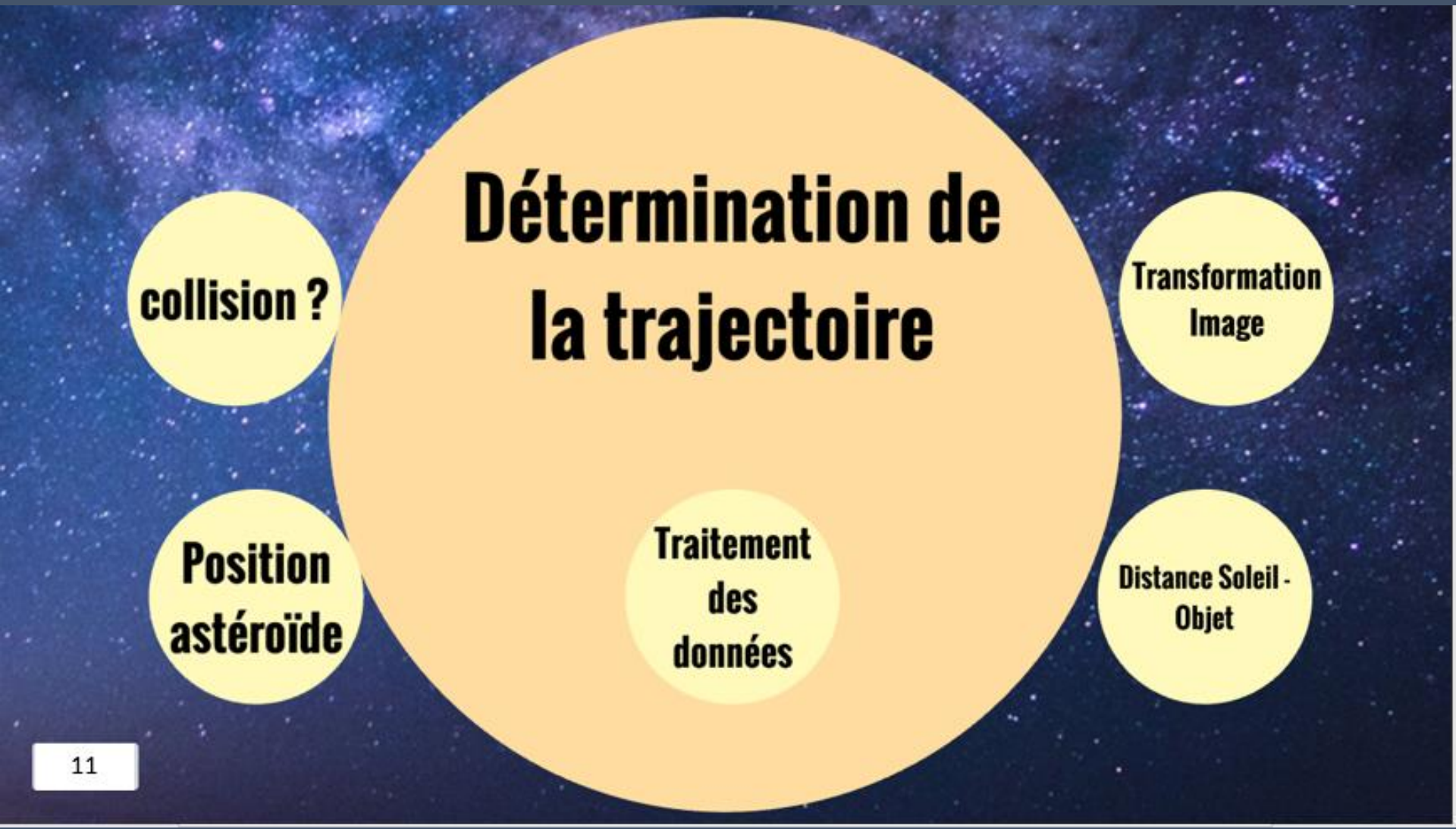


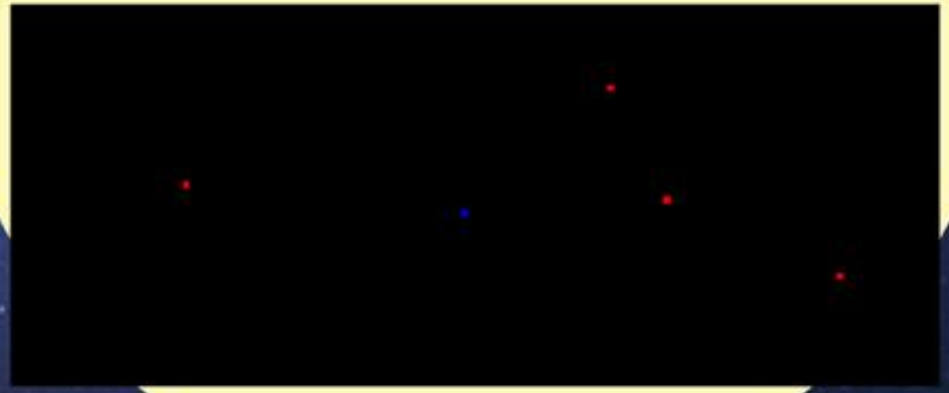
Caractèristiques de l'astéroïde



$$r = \frac{2.d.\tan(\frac{\beta}{2})}{1 - 2.\frac{\tan(\frac{\beta}{2})}{\tan(\frac{\alpha}{2})}}$$

```
L'objet étudié mesure 4.5 cm, et l'algorithme mesure 4.789299460030746 cm.
Soit une precision de : 0.9395946187025673
```


Détermination de la trajectoire
collision ?
Position astéroïde
Traitement des données
Transformation Image
Distance Soleil - Objet



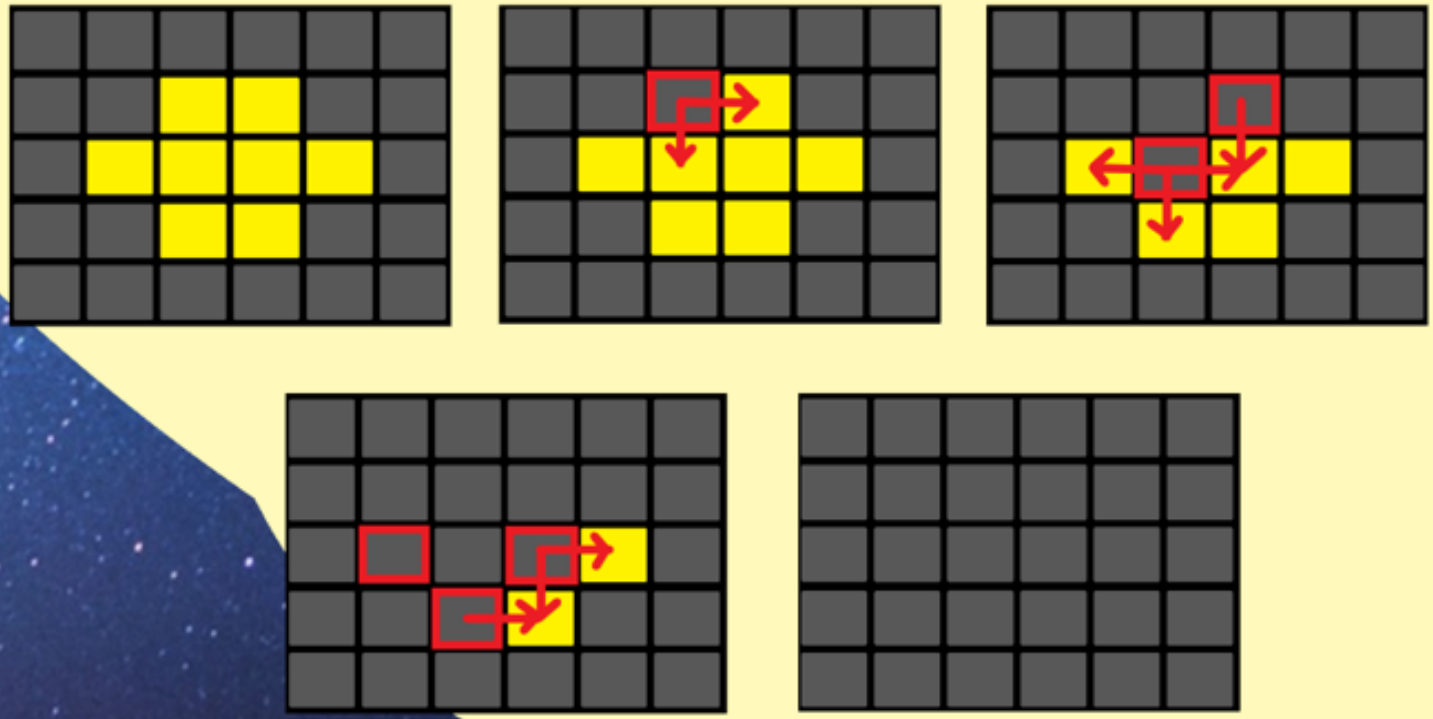
Fonctions

```python
def paquet(img, pix, L):
    """renvoie la liste des coordonnées des pixels voisins noirs de pix"""
    x, y = pix
    l, h, t = img.shape
    if not est_noir(img[x, y]):
        L.append(pix)
        img[x, y] = 0
    else:
        return None
    liste_voisin = [(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)]
    for i in range(len(liste_voisin)):
        x,y = liste_voisin[i]
        if 0 <= x < l and 0 <= y < h :
            paquet(img, liste_voisin[i], L)


def paquet_all(img):
    """renvoie une liste de liste de paquet de chaque objet celeste"""
    M = [] # liste final
    l, h, t = img.shape
    for i in range(l):
        for j in range(h):
            L = []
            paquet(img, (i, j), L)
            if L != [None] and L != []:
                M.append(L)
    return M
```

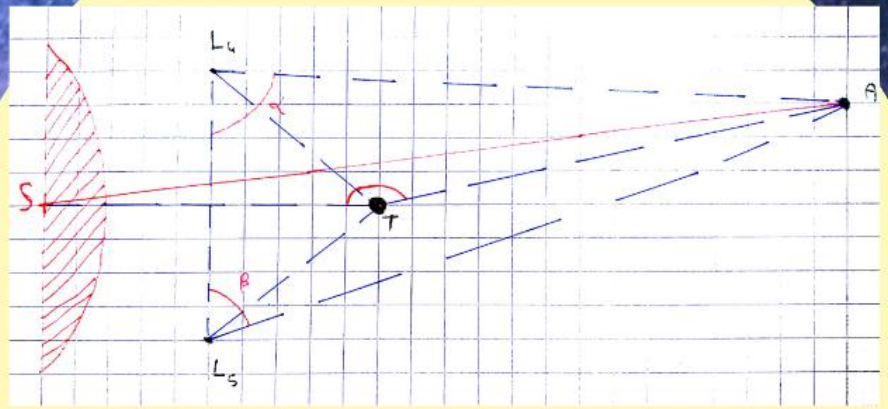
```python
def Trouve_terre(L):
    """prend en argument la liste renvoyé par paquet_all et renvoie la position de l'objet : Terre dans cette liste"""
    L_len = [] # liste verifiant pour tout i dans [|0, len(L)|] L_len[i] = len(L[i])
    for i in range(len(L)):
        L_len.append(len(L[i]))
    maxi = max(L_len) # taille de la liste la plus grande (donc de la liste Terre)
    for i in range(len(L_len)):
        if L_len[i] == maxi:
            return i
```

```python
def moyenne_paquet(L):
    """prend en argument la liste renvoyé par paquet_all et renvoie une liste des points centraux de chaque paquet"""
    M = []
    for i in range(len(L)):
        sx = 0 # somme des coord en x
        sy = 0 # somme des coord en y
        for j in range(len(L[i])):
            x, y = L[i][j]
            sx, sy = sx + x, sy + y
        moy_paquet = (arrondi(sx / len(L[i])), arrondi(sy / len(L[i])))
        M.append(moy_paquet)
    return M

def pointe_image(img):
    """prend en argument l'iamge en noir et blanc et renvoie une image blanche avec un pixel rouge sur la postion des etoiles et un pixel bleu sur la position de la Terre"""
    L = paquet_all(img)
    k = Trouve_terre(L)
    L = moyenne_paquet(L)
    for i in range(len(L)):
        x, y = L[i]
        if i == k:
            img[x, y, 2] = 255
        else:
            img[x, y, 0] = 255
    return img
```

Shema :



Distance Soleil - Objet



$$(SA)^2 = (ST)^2 + \left[(ST)^2 + ((L_4L_5)\frac{sin(\widehat{L_4L_5A})}{sin(\alpha + \widehat{L_4L_5A})})^2 - 2(ST)(L_4L_5)\frac{sin(\widehat{L_4L_5A})cos(\gamma)}{sin(\alpha + \widehat{L_4L_5A})}\right]$$

$$-2(ST)\left[(ST)^2 + \left((L_4L_5)\frac{sin(\widehat{L_4L_5A})}{sin(\alpha + \widehat{L_4L_5A})}\right)^2 - 2(ST)(L_4L_5)\frac{sin(\widehat{L_4L_5A})cos(\gamma)}{sin(\alpha + \widehat{L_4L_5A})}\right]^{\frac{1}{2}} . cos\left(Arccos\left(\frac{(L_4T)^2 + (ST)^2 + \left((L_4L_5)\frac{sin(\widehat{L_4L_5A})}{sin(\alpha + \widehat{L_4L_5A})}\right)^2 - 2(ST)(L_4L_5)\frac{sin(\widehat{L_4L_5A})cos(\gamma)}{sin(\alpha + \widehat{L_4L_5A})} - (L_4L_5)\frac{sin(\widehat{L_4L_5A})}{sin(\alpha + \widehat{L_4L_5A})}}{2(ST)\sqrt{(ST)^2 + \left((L_4L_5)\frac{sin(\widehat{L_4L_5A})}{sin(\alpha + \widehat{L_4L_5A})}\right)^2 - 2(ST)(L_4L_5)\frac{sin(\widehat{L_4L_5A})cos(\gamma)}{sin(\alpha + \widehat{L_4L_5A})}}}\right) + \frac{\pi}{3}\right)$$

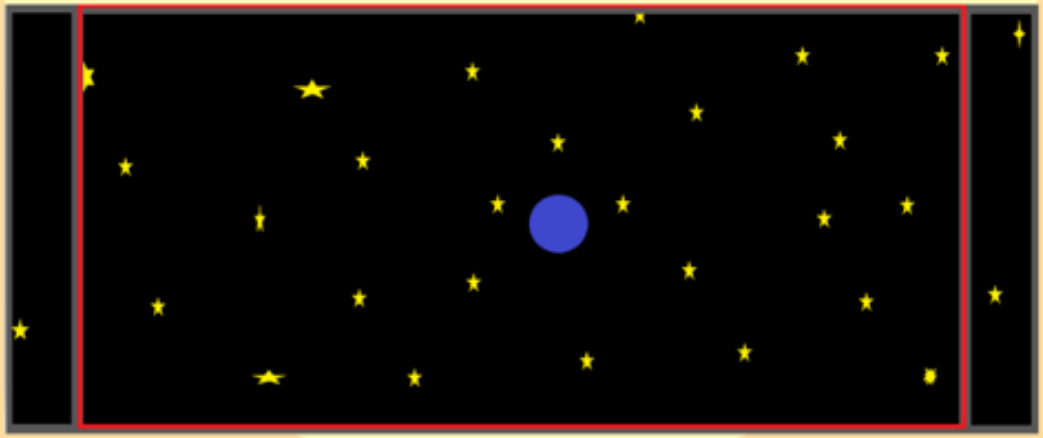
```
a = 0.01
b = 0.01
l'objet étudié est a 82493539 km de la Terre et  223885418 km du soleil
```

Traitement des données

- Convertir pixel-angle grâce à l'angle de prise de la photo

- Détermine l'angle L4-L5-Objet et L5-L4-Objet

- Simplification du problème : On considère qu'il y a un unique Objet



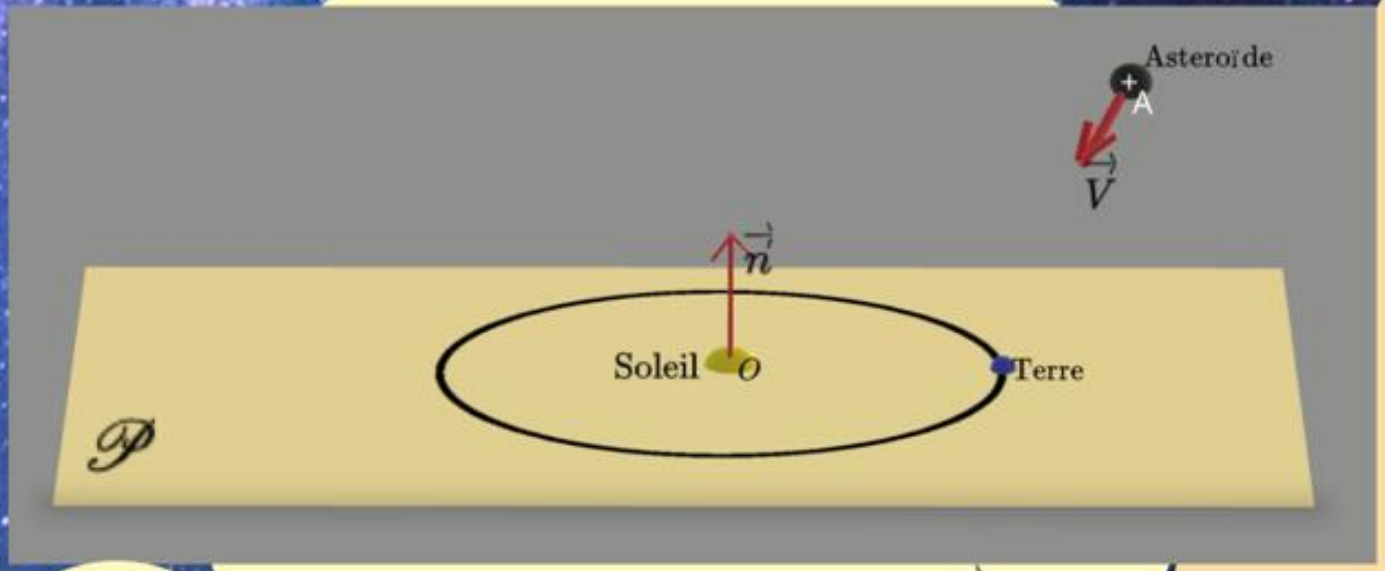
Position astéroïde

-Détermine la vitesse et la position de l'astéroïde grâce à 2 séries de photos

- Convertit les coordonnées sphériques en coordonnées cartésiennes grâce à :

$$x = r.sin(\theta).cos(\varphi)$$
$$y = r.sin(\theta).sin(\varphi)$$
$$z = r.cos(\theta)$$

Collision ?



Programmes

Soit D la demi-droite dirigée par $\vec{V}$

$\forall M \in D,\ \exists t \in \mathbb{R} \mid M = A + t.V$

On a alors : $M \in \mathscr{P} \iff \overrightarrow{OM}.\vec{n} = 0$

```python
def position_M(pos_carthesienne, V_carthesienne, t):
    """Renvoie la position d'un objet suivant la droite
(pos_carthesienne, V_carthesienne) au temps t"""
    pos = pos_carthesienne
    for i in range (len(V_carthesienne)):
        pos[i] = pos[i] + t * V_carthesienne
    return pos

def produit_scalaire(vect1, vect2):
    """Renvoie le produit scalaire des vecteurs vect1 et
vect2"""
    res = 0
    for i in range(len(vect1)):
        res = res + vect1[i] * vect2[i]
    return res

def vecteur_MP(M, P):
    """Prend en argument 2 liste definissant la position
des points M et P et renvoie le vecteur Vect(MP)"""
    vect_MP = [P[0] - M[0], P[1] - M[1], P[2] - M[2]]
    return vect_MP

def equation_intersection(pos_carthesienne, V_carthesienne,
n, t):
    """renvoie Vect(MP).Vect(n)"""
    return
produit_scalaire(vecteur_MP(position_M(pos_carthesienne,
V_carthesienne, t), P), n)
```

```python
def Trouve_t(pos_carthesienne, V_carthesienne, b):
    """Trouve le temps t de l'intersection entre l'objet et
le plan constituant la trajectoire de la Terre en resolvant
par dichotomie equation_intersection = 0"""
    prec = 3600   # 1h
    if (equation_intersection(pos_carthesienne,
V_carthesienne, a) < 0 and
equation_intersection(pos_carthesienne, V_carthesienne, b)
> 0) or (equation_intersection(pos_carthesienne,
V_carthesienne, a) > 0 and
equation_intersection(pos_carthesienne, V_carthesienne, b)
< 0):
        while abs(b - a) < prec:
            m = 0.5 * (a + b)
            if equation_intersection(pos_carthesienne,
V_carthesienne, m) < 0:
                a = m
            else:
                b = m
    else:
        print("Pas de solution")
    sol = (a + b) / 2
    a = 0   # Pour eviter les conflits si on execute
plusieurs fois la fonction
    return sol

def point_intersection(pos_carthesienne, V_carthesienne,
b):
    """renvoie le points d'intersection entre le plan et la
trajectoire de l'objet"""
    t = Trouve_t(pos_carthesienne, V_carthesienne, b)
    return position_M(pos_carthesienne, V_carthesienne, t)
```

```python
def collision(pos_carthesienne, V_carthesienne, b,
marge_erreur):
    """Renvoie True si l'objet rentrera en collision avec
la Terre et False sinon, marge erreur etant une liste de
taille 3 avec la marge d'erreur sur x, y et z"""
    pos = point_intersection(pos_carthesienne,
V_carthesienne, b)
    if (abs(pos[0] - distance_T_S) < marge_erreur[0]) and
(abs(pos[1] - distance_T_S) < marge_erreur[1]) and
(abs(pos[2] - rayon_Terre) < marge_erreur[2]):
        return True
    else:
        return False
```

Conclusion et annexes

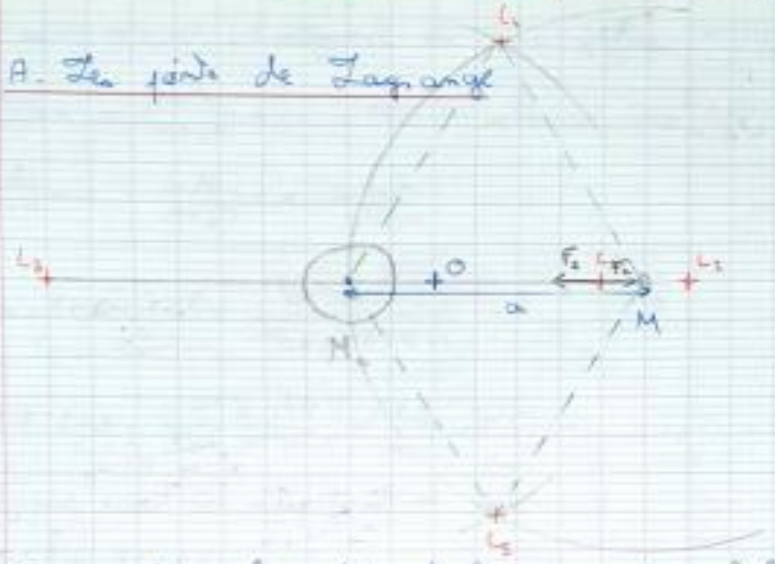
- Beaucoup de simplifications
- Fonctionne sur des cas simples

On peut alors encore chercher à:

Déterminer la trajectoire précise de l'objet.
Étudier la stabilité des points de Lagrange et éventuellement trouver un système qui stabilise les satellites.

Détermination des positions des points de Lagrange

A. Les points de Lagrange



On a : M_0 : la position et la masse du soleil.
M : la position et la masse de la Terre.
O : le barycentre des masses M_0 et M
m : la masse de l'astéroïde étudié.
a : la distance M_0M.

Dans toute la suite, on néglige m par rapport à M_0 et M.
On suppose que la Terre a une trajectoire circulaire autour de la Terre.

B. Points de Lagrange L_1, L_2 et L_3.

1. Point L_2

On pose $d = L_2M$, $x = \frac{d}{a}$ et $k = \frac{M}{M_0}$.

On étudie le mouvement dans un repère galiléen

centré sur le barycentre O des deux masses.

On a alors : $OM = a \times \frac{M_0}{M_0+M} = a \times \frac{1}{1+\frac{M}{M_0}} = \frac{a}{1+k}$

$OM_0 = a \times \frac{M}{M_0+M} = a \times \frac{k}{1+k}$

A.N : $k = \frac{5,9722 \times 10^{24}}{1,9884 \times 10^{30}} = 3,0035.10^{-6}$

$OM = \frac{149\,597\,870}{1+3,0035.10^{-6}} = 149\,597\,420,7$ km

remarque le barycentre se trouve dans le soleil.

$OM_0 = 149\,597\,870 \times \frac{3,0035.10^{-6}}{1+3,0035.10^{-6}} = 449,3$ km

On suppose que l'astéroïde de masse m se situe en L_2.
m subit une force F_0 vers M_0 et une force F_2 vers M.
On suppose L_2 entre le soleil et la Terre.

On a : $F_2 = \frac{Gm M_0}{(a-d)^2} = \frac{Gm M_0}{(a-ax)^2}$

$F_2 = \frac{Gm M}{d^2} = \frac{Gm M}{(ax)^2}$

D'après le PFD.

$m\vec{a} = \vec{F_0} - \vec{F_2}$

$$\vec{a} = \frac{1}{m}\left(\vec{F}_2 - \vec{F}_1\right)$$

On projette sur l'axe M_0M ; et on a : $a = \omega^2 OL_2$

Donc, $\omega^2 OL_2 = \frac{F_2 - F_1}{m}$

$$= \frac{1}{m}\left(\frac{GmM_0}{(a-ax)^2} - \frac{GmM}{(ax)^2}\right)$$

$$\omega^2 OL_2 = G\left(\frac{M_0}{(a-ax)^2} - \frac{M}{(ax)^2}\right)$$

D'après la relation de Chasle, $OL_2 + L_2M = OM$

Donc $OL_2 = OM - L_2M$

Donc $OL_2 = OM - d$

Donc $OL_2 = OM - ax$

D'où : $\omega^2(OM - ax) = G\left(\frac{M_0}{(a(1+\frac{x}{\xi}))^2} - \frac{M}{(ax)^2}\right)$

On a vu que $OM = \frac{a}{1+\xi}$

Donc, $\omega^2\left(a\left(\frac{1}{1+\xi} - x\right)\right) = G\left(\frac{1}{a^2}\left(\frac{M_0}{(1-x)^2} - \frac{M}{x^2}\right)\right)$

$\Leftrightarrow \omega^2\left(\frac{1}{1+\xi} - x\right) = \frac{G}{a^3}\left(\frac{M_0}{(1-x)^2} - \frac{M}{x^2}\right)$ (*)

D'autre part, l'objet est placé au point L_2, donc sa Vitesse angulaire est égal à la Vitesse angulaire des objets M et M_0 autour de O, qu'on notera ω_0.

On cherche alors à déterminer ω_0 dans le cas général :

$$\vec{F}_{1/2} = -G\frac{m_1m_2}{r^2}\vec{u}_r \quad ; \quad \vec{F}_{2/1} = +G\frac{m_1m_2}{r^2}\vec{u}_r$$

$$m_2\ddot{\vec{a}}_2 = G\frac{m_1m_2}{r^2}\vec{u}_r$$

$$m_1\ddot{\vec{a}}_1 = -G\frac{m_1m_2}{r^2}\vec{u}_r$$

$$\vec{r} = \vec{a}_2 - \vec{a}_1$$

$$\ddot{\vec{a}} = -\frac{Gm_1}{r^2}\vec{u}_r - \left(\frac{Gm_2}{r^2}\vec{u}_r\right)$$

$$\ddot{\vec{r}} = -\frac{G}{r^2}(m_1+m_2)\vec{u}_r$$

On multiplie par $\frac{1}{m_1m_2}$:

$$\frac{\ddot{\vec{r}}}{m_1m_2} = -\frac{G}{r^2}\frac{m_1+m_2}{m_1m_2}\vec{u}_r$$

$$\frac{m_1m_2}{m_1+m_2}\ddot{\vec{r}} = -\frac{Gm_1m_2}{r^2}\vec{u}_r$$

On pose alors $\mu = \frac{m_1m_2}{m_1+m_2}$ la masse réduite

On a alors : $\ddot{\vec{r}} = r\omega_0^2\vec{e}_r$

$$\Leftrightarrow \mu\ddot{\vec{r}} = r\mu\omega_0^2\vec{e}_r$$

$$\Leftrightarrow r\mu\omega_0^2\vec{e}_r = G\frac{m_1m_2}{r^2}\vec{e}_r$$

$$\Leftrightarrow \omega_0^2 = G\frac{m_1m_2}{\mu r^3}$$

$\Leftrightarrow \omega_0^2 = G\frac{m_0+m_2}{a^3}$

Donc, pour ce cas, on a : $\omega_0^2 = G\frac{M_0+M}{a^3}$

Revenons à (*) :

On a : $\omega_0^2\left(\frac{1}{1+k}-x\right) = \frac{G}{a^3}\left(\frac{M_0}{(1-x)^2}-\frac{M}{x^2}\right)$

$\Leftrightarrow G\frac{M_0+M}{a^3}\left(\frac{1}{1+k}-x\right) = \frac{G}{a^3}\left(\frac{M_0}{(1-x)^2}-\frac{M}{x^2}\right)$

$\Leftrightarrow (M_0+M)\left(\frac{1}{1+k}-x\right) = \frac{M_0}{(1-x)^2}-\frac{M}{x^2}$

$\Leftrightarrow M_0\left(\frac{M_0+M}{M_0}\right)\left(\frac{1}{1+k}-x\right) = \frac{M_0}{(1-x)^2}-\frac{M}{M_0 x^2}M_0$

$\Leftrightarrow M_0\left(1+\frac{M}{M_0}\right)\left(\frac{1}{1+k}-x\right) = M_0\left(\frac{1}{(1-x)^2}-\frac{k}{x^2}\right)$

$\Leftrightarrow (1+k)\left(\frac{1}{1+k}-x\right) = \frac{1}{(1-x)^2}-\frac{k}{x^2}$

$\Leftrightarrow 1-(1+k)x = \frac{1}{(1-x)^2}-\frac{k}{x^2}$

Le but étant de faire une équation en fonction de k.
On a alors :

$$(1+k)x - 1 + \frac{1}{(1-x)^2} - \frac{k}{x^2} = 0$$

Par une méthode numérique, on obtient : $x = 0,99$
On a alors $d = x \times a$

A.N. : $L_2M = d = 1\,495\,399$ km

Intérêt de L_2 : Observation ininterrompue du Soleil. C'est dans ce but qu'on y a installé le satellite SOHO.

2. Point L_3 :

On pose $a = MM_0$; $d = L_3M$; $x = \frac{d}{a}$ et $k = \frac{M}{M_0}$

On étudie le mouvement dans un repère galiléen centré sur le barycentre O des deux masses M_0 et M.

De la même manière qu'en 1. on a :

$OM = \frac{a}{1+k}$ et $OM_0 = \frac{ak}{1+k}$

De même, en L_3, on subit une force F_1 vers M_0 et une force F_2 vers M.

$F_1 = G\frac{mM_0}{(a+ax)^2}$ et $F_2 = G\frac{mM}{(ax)^2}$

D'après 2ème loi de Newton

$$\omega^2 OL_3 = \frac{1}{m}(F_1+F_2)$$

$$\omega^2 OL_3 = G\left(\frac{M_0}{(a+ax)^2}+\frac{M}{(ax)^2}\right)$$

$$\omega^2(OM+ax) = G\left(\frac{M_0}{(a+ax)^2}+\frac{M}{(ax)^2}\right)$$

$$\omega^2\left(\frac{1}{1+k}+x\right) = \frac{G}{a^3}\left(\frac{M_0}{(1+x)^2}+\frac{M}{x^2}\right)$$

on devant être fixe p/r à M et M_0, il faut que $\omega = \omega_0$.

On a alors : $\omega_0^2 = G\frac{M_0+M}{a^3}$

Donc $(M_0+M)\left(\frac{x}{x+k} + x\right) = \frac{M_0}{(x+x)^2} + \frac{M}{x^2}$

$\Leftrightarrow (x+k)\left(\frac{1}{x+k} + x\right) = \frac{1}{(x+x)^2} + \frac{k}{x^2}$

$\Leftrightarrow x + (x+k)x = \frac{1}{(x+x)^2} + \frac{k}{x^2}$

D'où $x + (1+k)x - \frac{1}{(1+x)^2} - \frac{k}{x^2} = 0$

Par résolution numérique, on trouve : $x = 1,0x$

On a alors $d = x \times a$

A.N : $L_1M = d = 1505\,404$ km

Intérêt de L_1 : Observation ininterrompue de l'univers

3. Point L_3.

On pose $a = MM_0$; $d = L_3M_0$; $x = \frac{d}{a}$ et $k = \frac{M}{M_0}$

On étudie le mouvement dans un repère galiléen centré sur le barycentre O des deux masses M_0 et M. $OM_0 = a\frac{M}{M_0+M}$ et $OM = a\frac{M_0}{M_0+M}$

$OM = \frac{a}{1+k}$ et $OM_0 = \frac{ak}{1+k}$

En L_3, on subit une force F_1 vers M_0 et une force F_2 vers M.

$F_1 = G\frac{mM_0}{(a+x)^2}$

$F_2 = G\frac{mM}{(a+a+x)^2}$

D'après la 2ème loi de Newton, $\omega^2 OL_3 = \frac{F_1+F_2}{m}$

$= G\left(\frac{M_0}{(a+x)^2} + \frac{M}{(a+a+x)^2}\right)$

$\omega^2(OM_0 + a+x) = \frac{F_1+F_2}{m} = G\left(\frac{M_0}{(a+x)^2} + \frac{M}{(a+a+x)^2}\right)$

$\omega^2\left(\frac{ak}{1+k} + x\right) = \frac{G}{a^2}\left(\frac{M_0}{x^2} + \frac{M}{(1+x)^2}\right)$

on devant être fixe par rapport M et M_0, il faut que $\omega = \omega_0$. (ω_0 étant la vitesse angulaire des masses M et M_0 autour de O)

Pour ce système à 2 corps, $\omega_0^2 = \frac{G}{a^3}(M_0+M)$

Donc $(M_0+M)\left(\frac{k}{1+k} + x\right) = \frac{M_0}{x^2} + \frac{M}{(1+x)^2}$

$(1+k)\left(\frac{k}{1+k} + x\right) = \frac{1}{x^2} + \frac{k}{(1+x)^2}$

$k + (1+k)x = \frac{1}{x^2} + \frac{k}{(1+x)^2}$

D'où $(1+k)x + k - \frac{k}{(1+x)^2} - \frac{1}{x^2} = 0$

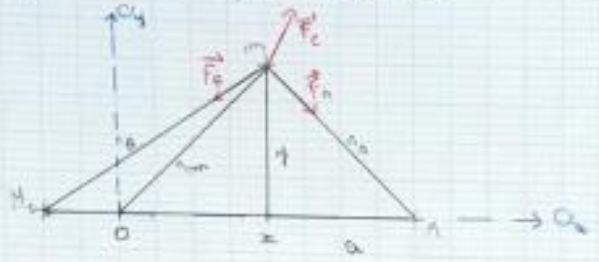
Par résolution numérique :

A.N : $L_3M = 300\,339\,000$ km

Intérêt de L_3 : Très peu au vu de la distance de la Terre.

4. Points L_4 et L_5.

Dans cette partie, nous allons directement déterminer les positions des points L_4 et L_5



$u = \frac{M_0 M}{M_0}$, $k = \frac{M}{M_0}$

On a : $M = k M_0 = \frac{k}{1+k}(M_0 + M)$

$M_0 = \frac{1}{1+k}(M_0 + M)$

$OM_0 = \frac{ka}{1+k}$

$OM = \frac{a}{1+k}$

et $\omega^2 = \frac{G}{a^3}(M_0 + M)$

Pour qu'il y ait une position d'équilibre dans le repère tournant avec les masses M_0 et M, il faut que la somme des forces dans ce repère soit nulle.

Dans ce repère, la masse m est soumise à 3 forces : - les deux forces d'attraction gravitationnelle
- la force centrifuge

La masse m étant immobile dans le repère tournant, il n'y a pas de force de Coriolis.

On appelle C le point de la position de m.

$F_a = G\frac{mM}{(CM)^2} = G\frac{mM}{(r_a)^2}$

$= Gm\frac{M}{M_0}\cdot\frac{M_0}{(r_a)^2}$

$= Gm\,k\,\frac{1}{(r_a)^2}$ / M_0

$= Gm\,k\,\frac{M_0 + M}{(M_0 + M)(r_a)^2}$ / M_0

$= Gm\,k\,\frac{M_0 + M}{\left(k + \frac{M}{M_0}\right)(r_a)^2}$

$= Gm\,k\,\frac{M_0 + M}{(k+k)(r_a)^2}$

$= m\,\underline{G(M_0 + M)}\,\frac{k}{(k+k)(r_a)^2}$

$\omega^2 = \frac{k}{(k+k)(r_a)^2}$

$F_a = m\,\omega^2\,u$

$$F_B = \frac{G m M_0}{(CM_0)^2} = G \frac{m M_0}{(r_B)^2}$$

$$= G m \frac{1}{\frac{(r_B)^2}{M_0}}$$

$$= G m \frac{M_0 + M}{\frac{(M_0+M)(r_B)^2}{M_0}}$$

$$= m \underbrace{G \frac{M_0 + M}{(z+R)(r_B)^2}}_{\omega^2 a^3}$$

$$\underline{F_B = m \frac{\omega^2 a^3}{(z+R)(r_B)^2}}$$

$$F_c = m \omega^2 (OC) = m \omega^2 r_m$$

On assimile les 2 masses M et M_0 à une seule masse située de centre O.

Donc d'après le PFD, à l'équilibre :

$$F_A + F_B + F_C = 0$$

$$\begin{cases} /O_x : F_{Ax} + F_{Bx} + F_{Cx} = 0 \\ /O_y : F_{Ay} + F_{By} + F_{Cy} = 0 \end{cases}$$

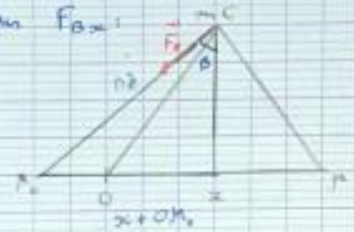
Pour F_{Ax} :

$$F_{Ax} = F_A \sin(\alpha) = F_A \frac{OM - x}{r_A}$$

$$= m \omega^2 a^3 \frac{1}{z+R} \frac{1}{(r_A)^3} (OM - x)$$

$$\underline{F_{Ax} = m \omega^2 a^3 \frac{1}{z+R} \frac{1}{(r_A)^3} \left(\frac{a}{z+R} - x \right)}$$

Pour F_{Bx} :



$$F_{Bx} = - F_B \sin(\beta) = - F_B \frac{x + OM_0}{r_B}$$

$$= - m \omega^2 a^3 \frac{1}{R+z} \frac{1}{(r_B)^3} \frac{x + OM_0}{r_B}$$

$$\underline{F_{Bx} = - m \omega^2 a^3 \frac{1}{R+z} \frac{1}{(r_B)^3} \left(x + \frac{Ra}{z+R} \right)}$$

Pour F_{Cx} :

On a alors (OC) une droite directrice de $\vec{F_C}$

$$F_{Cx} = F_C \sin(\delta) = F_C \frac{x}{r_m}$$

$$= m\omega^2 r_m \frac{x}{r_m}$$

$$\underline{F_{Cx} = m\omega^2 x}$$

En simplifiant par $m\omega^2$, on obtient :

(1) $$a^3 \frac{k}{2+k} \frac{1}{(r_A)^3} \left(\frac{a}{2+k} - x\right) = a^3 \frac{1}{k+2} \frac{1}{(r_B)^3} \left(x + \frac{k\,a}{2+k}\right) + x = 0$$

On fait de même sur O_y :

Pour F_{Ay} :

$$F_{Ay} = -F_A \cos(\alpha) = -F_A \frac{y}{r_A}$$

$$= -m\omega^2 a^3 \frac{k}{2+k} \frac{1}{(r_A)^3} \frac{y}{r_A}$$

$$\underline{F_{Ay} = -m\omega^2 a^3 \frac{k}{2+k} \frac{1}{(r_A)^3} y}$$

Pour F_{By} :

$$F_{By} = -F_B \cos(\beta) = -F_B \frac{y}{r_B}$$

$$= -m\omega^2 a^3 \frac{1}{k+2} \frac{1}{(r_B)^3} \frac{y}{r_B}$$

$$\underline{F_{By} = -m\omega^2 a^3 \frac{1}{k+2} \frac{1}{(r_B)^3} y}$$

Pour F_{Cy} :

$$F_{Cy} = F_C \cos(\delta) = F_C \frac{y}{r_m}$$

$$= m\omega^2 r_m \frac{y}{r_m}$$

$$\underline{F_{Cy} = m\omega^2 y}$$

En simplifiant par $m\omega^2 y$, on obtient :

(2) $$-a^3 \frac{k}{2+k} \frac{1}{(r_A)^3} - a^3 \frac{1}{k+2} \frac{1}{(r_B)^3} + 1 = 0$$

On remarque que pour $r_A = r_B = a$, on a :

$$-a^3 \frac{k}{2+k} \frac{1}{a^3} - a^3 \frac{1}{k+2} \frac{1}{a^3} + 1$$

$$= -\frac{k+1}{k+2} + 1$$

$$= -1 + 1 = 0$$

Donc, $\underline{r_A = r_B = a}$ est solution de (2)

Cette solution indique que le triangle MCM_0 forme un triangle équilatéral.

D'après le Th. de Pythagore :

$$a^2 = y^2 + \left(\frac{a}{2}\right)^2$$

$$y = \sqrt{a^2 - \frac{a^2}{4}}$$

$$y = \sqrt{a^2\left(1 - \frac{1}{4}\right)}$$

$$y = a\sqrt{\frac{3}{4}}$$

$$\underline{y = \frac{\sqrt{3}}{2}a}$$

Et $x = \frac{a}{2} - O M_0$

$$= \frac{a}{2} - \frac{ka}{2+k}$$

$$= a\left(\frac{1}{2} - \frac{k}{2+k}\right)$$

$$= a\left(\frac{2+k-2k}{2(2+k)}\right)$$

$$\underline{x = a\left(\frac{2-k}{2(2+k)}\right)}$$

Vérifions que les solutions trouvés sont valables dans (a)

$$a^3\frac{k}{2+k} - \frac{2}{a^2}\left(\frac{a}{2+k} - a\left(\frac{2-k}{2(2+k)}\right)\right) - a^2\frac{a}{k+2} - \frac{1}{a^2}\left(a\left(\frac{2-k}{2(2+k)}\right)\right) + \frac{k\,a}{k+2}\left(a\left(\frac{2-k}{2(2+k)}\right)\right)$$

$$= \frac{k}{k+2}\left(a\left(\frac{2-\frac{a}{2}(2-k)}{2+k}\right)\right) - \frac{1}{k+2}\left(a\left(\frac{k+\frac{a}{2}(2-k)}{k+2}\right)\right) + a\left(\frac{2-k}{2(k+2)}\right)$$

$$= \frac{k}{k+2}\frac{a}{2} - \frac{2}{k+2}\frac{a}{2} + a\left(\frac{2+k}{2(k+2)}\right)$$

$$+ a\left(\frac{k-2}{2(k+2)}\right) + a\left(\frac{2-k}{2(k+2)}\right) = 0$$

Donc, les solutions trouvés sont bien valables.

Ainsi, les deux points L_4 et L_5 se trouvent aux sommets d'un triangle équilatéral dont la base est la distance $M_0 M = a$

Unicité des solutions

1. Equation du point L_1.

$\forall x \in \mathbb{R}_+^* \setminus \{1\}$, on pose $f_1(x) = (1+k)x - 1 + \frac{x}{(1-x)^2} - \frac{k}{x^2}$

On a alors :

$\forall x \in \mathbb{R}_+^* \setminus \{1\}$, $f_1'(x) = 1 + k + \frac{2}{(1-x)^3} + \frac{2k}{x^3}$

Soit $x \in \mathbb{R}_+^* \setminus \{1\}$

$$\frac{2}{(1-x)^3} + \frac{2k}{x^3} = \frac{2x^3 + 2k(1-x)^3}{x^3(1-x)^3}$$

$$= \frac{2x^3 + 2k(1 - 3x + 3x^2 - x^3)}{(x(1-x))^3}$$

$$= \frac{2x^3 + 2k - 6kx + 6kx^2 - 2kx^3}{(x(1-x))^3}$$

$$= \frac{2x^3(1-k) + 6kx^2 - 6kx + 2k}{(x(1-x))^3}$$

Dans les hypothèse de L_1, $k \in]0, 1[$ et L_1 se situe entre la Terre et le soleil. Donc, $x \in]0, 1[$.

Donc $2x^3(1-k) + 6kx^2 - 6kx + 2k > 0$

Et, $(x(1-x))^3 > 0$

Donc $\forall x \in \mathbb{R}_+^* \setminus \{1\}$, $f_1'(x) > 0$

On a alors :

x	0		1	
f_1'	$\|$	$+$	$\|$	$+$
$f_1(x)$	$-\infty$	$\nearrow$	$+\infty$	$\nearrow$

f_1 est strictement croissante et continue sur $]0, 1[$

Donc, f_1 s'annule une seul fois sur $]0, 1[$

2. Equation du point L_2.

$\forall x \in \mathbb{R}_+^* \setminus \{1\}$, on pose $f_2(x) = 1 + (1+k)x - \frac{1}{(1-x)^2} - \frac{k}{x^2}$

Donc $\forall x \in \mathbb{R}_+^* \setminus \{1\}$ $f_2'(x) = 1 + k + \frac{2}{(1-x)^3} + \frac{2k}{x^3}$.

Donc :

x	0		1	
f_2'	$\|$	$+$	$\|$	$+$
$f_2(x)$	$-\infty$	$\nearrow$	$+\infty$	$\nearrow$

f_2 est strictement croissante et continue sur $\mathbb{R}_+^* \setminus \{1\}$.

Donc f_2 s'annule qu'une seul fois sur $\mathbb{R}_+^* \setminus \{1\}$


```python
import matplotlib.pyplot as plt
import numpy as np
#
# L'objectif est ici de calculer les positions des points de Lagrange pour les système Terre / Soleil
#On cherche donc à résoudre un problème à trois corps dont un qui a une masse nulle. On veut déterminer les positions auxquelles il est stable

### I) Variables globales
masse_soleil = 1.989 * 10 ** 30 #en kg
masse_terre = 5.972 * 10 ** 24 #en kg

rayon_soleil = 696340000 #en mètres
rayon_terre = 6371000 #en mètres

G = 6.67408 * 10 ** (-11) #en m**3 kg ** (-1) s ** (-2)
distance_T_S_ua = 1 #en ua
distance_T_S_m = 150000000000 #en m

k = masse_terre / masse_soleil

### II) Calculs initiaux

#On calcule maintenant la vitesse angulaire w, d'après la troisième loi de Kepler:

w = (G * (masse_terre + masse_soleil) / (distance_T_S_ua ** 3)) ** (1 / 2)

### III) Premières équations: Position des deux premiers points

#Soit r la distance entre le Soleil et le sattelite
#D'après le PFD:  ∑Fext = ma
#Donc, le point étant soumis aux forces gravitationnelles du Soleil et de la Terre:
# - m * (v ** 2 / r) = G * m ((- masse_soleil / r ** 2) + (masse_terre / (distance_T_S - r))
#Le but ici est de résoudre cette équation en r
#On peut simplifier l'équation par m et poser :
#v = (w * r) ** 2, il suffit de remplacer w par la formule vue dans II
```

```
rapport_masse = masse_terre / masse_soleil
rapport_masse_2 = masse_terre / (masse_terre + masse_soleil)

def f(r):
   """On associe l'équation a une fonction"""
   return -1 + rapport_masse * (r ** 2 / (distance_T_S_ua - r) ** 2) + (r / distance_T_S_ua) ** 3

#La position du second point etant de l'autre côté de la Terre, on peut définir la fonction
suivante associée à l'équation du second point telle que pour tout x dans R g(x) = -f(x)

def g(r):
   """On associe l'équation a une fonction"""
   return -1 - rapport_masse * (r ** 2 / (distance_T_S_ua - r) ** 2) + (r / distance_T_S_ua) ** 3

#Pour trouver les racines, il faut utiliser la méthode de la dichotomie, on a donc besoin de
calculer les dérivées de f et g

def fprime(r):
    return - 1 + 2 * rapport_masse * (r * (distance_T_S_ua - r) ** 2 + (r ** 2) * r *
(distance_T_S_ua - r)) + r / (distance_T_S_ua ** 2)

def gprime(r):
    return -1 - 2 * rapport_masse * (r * (distance_T_S_ua - r) ** 2 + (r ** 2) * r *
(distance_T_S_ua - r)) + r / (distance_T_S_ua ** 2)
```

```
def dichotomie(f, a, b):
    """On va chercher à retrecir l'intervalle [a, b] en utilisant
f(c) avec c = abs(b - a) / 2"""
    c = (b + a) / 2
    while abs(abs(f(b)) - abs(f(a))) > 10 ** (-15):
        if f(b) * f(c) > 0:
            b = c
        else:
            a = c
        c = (b + a) / 2
    return c

#la solution est renvoyée en UA pour l'avoir en km, il suffit de
la multiplier par 1,495 978 700 millions
```

```
### IV) Equation pour déterminer le point L3

def L3(x):
    return (1 + k) * x + k - k / ((1 + x) ** 2) - 1 / x ** 2


### V) positions des points L4 et L5

#Les points L4 et L5 sont positionnés de tel sorte que       les
triangles Soleil-Terre-L4 et Soleil-Terre-L5 sont equilaterales.
#Ils sont donc tous les deux postionnés a une distance de 1ua de
la Terre
```

```
def graph():
    X_f = np.arange(0, 1, 10**(-5))
    X_g = np.arange(1, 2, 10**(-5))
    y_g = []
    y_f = []
    for i in range(len(X_f)):
        y_f.append(f(X_f[i]))
        y_g.append(g(X_g[i]))
    Y_g = np.array(y_g)
    Y_f = np.array(y_f)
    plt.axis([0.98, 1.02, -1., 1.5])
    plt.plot(X_f, Y_f)
    plt.plot(X_g, Y_g)
    plt.grid()
    plt.show()
```

Triangulation

Objectif : Déterminer la distance Terre - astéroïde à l'aide des points L_4 et L_5.

1ère étape : Déterminer la distance points de Lagrange - astéroïde.

On a le triangle quelconque suivant :

Avec : L_4 le point de la zone L_4
L_5 le point de la zone L_5
A la position de l'astéroïde
H le projeté de A sur (L_4L_5)
α, β, γ les angles correspondants
a la distance L_4L_5
b " " AL_4
c " " AL_5

On a S l'aire du triangle L_4AL_5

On a donc : $S = \frac{1}{2} L_4L_5 \times AH = \frac{a}{2} AH$

Or $\sin(\beta) = \frac{AH}{c}$

Donc $S = \frac{ac}{2} \sin(\beta)$

De même, $S = \frac{1}{2} ab \sin(\gamma) = \frac{1}{2} bc \sin(\alpha)$

Donc on a :

$ac \sin(\beta) = ab \sin(\gamma)$

D'où $\frac{c}{\sin(\gamma)} = \frac{b}{\sin(\beta)}$

Puis on a :

$ab \sin(\gamma) = bc \sin(\alpha)$

D'où $\frac{a}{\sin(\alpha)} = \frac{c}{\sin(\gamma)}$

Ainsi on a : $\frac{a}{\sin(\alpha)} = \frac{b}{\sin(\beta)} = \frac{c}{\sin(\gamma)}$

On en déduit donc :

$$b = a \times \frac{\sin(\beta)}{\sin(\alpha)} = a \times \frac{\sin(\beta)}{\sin(\pi - \gamma) - \beta)} = a \times \frac{\sin(\beta)}{\sin(\gamma + \beta)}$$

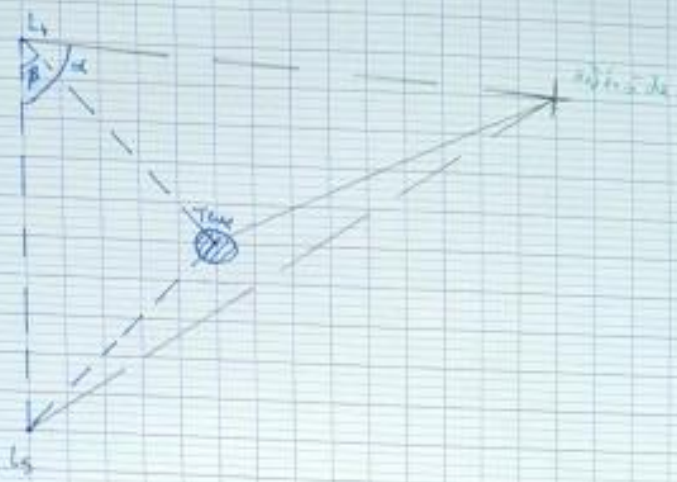
Donc,

$$b = a \times \frac{\sin(\beta)}{\sin(\gamma + \beta)}$$

et

$$c = a \times \frac{\sin(\gamma)}{\sin(\gamma + \beta)}$$

2ème étape: Déterminer la distance Terre - astéroïde



Démonstration M. Al Kashi:

Soit le Triangle ABC suivant:

$$a^2 = \|\vec{BC}\|^2 = \|\vec{AC} - \vec{AB}\|^2$$
$$= \|\vec{AC}\|^2 + \|\vec{AB}\|^2 - 2\,\vec{AB}\cdot\vec{AC}$$
$$\underline{a^2 = b^2 + c^2 - 2bc\cos(\alpha)}$$

On connaît l'angle $\alpha = \widehat{L_5 L_4 A}$ et $\beta = \widehat{L_5 L_4 T}$
On pose alors $\gamma = \alpha - \beta = \widehat{TL_4 A}$

Ainsi, d'après le Théorème de Al-Kashi:

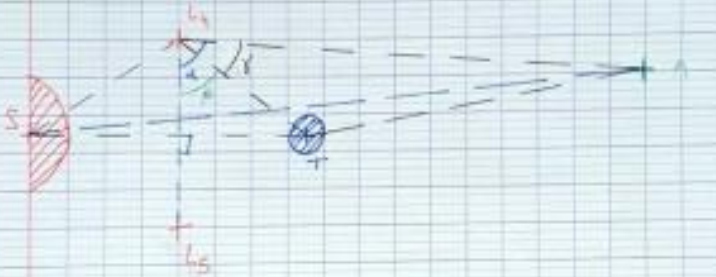
$$(TA)^2 = (L_4 T)^2 + (L_4 A)^2 - 2(L_4 T)(L_4 A)\cos(\gamma)$$

$$\text{Or, } L_4 A = (L_4 L_5) \times \frac{\sin(\widehat{L_4 L_5 A})}{\sin(\alpha + \widehat{L_4 L_5 A})}$$

Donc:

$$TA = \sqrt{(L_4 T)^2 + \left((L_4 L_5) \times \frac{\sin(\widehat{L_4 L_5 A})}{\sin(\alpha + \widehat{L_4 L_5 A})}\right)^2 - 2(L_4 T)(L_4 L_5)\frac{\sin(\widehat{L_4 L_5 A})\cos(\gamma)}{\sin(\alpha + \widehat{L_4 L_5 A})}}$$

3ème étape. Déterminer la distance Soleil astéroïde.



On connaît maintenant:
- la distance Terre - Astéroïde
- la distance Terre - Soleil
- la distance L_4 - Soleil
- la distance L_4 - Terre
- la distance L_4 - Astéroïde
- l'angle $\alpha = \widehat{L_5 L_4 A}$
- l'angle $\beta = \widehat{L_5 L_4 T}$
- l'angle $\gamma = \alpha - \beta = \widehat{TL_4 A}$

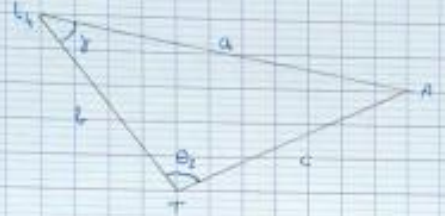
On cherche alors à déterminer l'angle $\theta = \widehat{STA}$ pour déterminer la distance S-A à l'aide du Théorème d'Al-Kashi.

On a $\theta = \widehat{STA} = \underbrace{\widehat{STL_4}}_{\theta_3} + \underbrace{\widehat{L_4TA}}_{\theta_2}$.

Le triangle (STL_4) est un triangle équilatéral.

Donc $\widehat{STL_4} = 60° = \frac{\pi}{3} = \theta_3$

Déterminons $\widehat{L_4TA}$:



D'après la Th. d'Al Kashi :

$$a^2 = b^2 + c^2 - 2bc\cos(\theta_2)$$

$$2bc\cos(\theta_2) = b^2 + c^2 - a^2$$

$$\cos(\theta_2) = \frac{b}{2c} + \frac{c}{2b} - \frac{a^2}{2bc}$$

$$\theta_2 = \text{Arccos}\left(\frac{1}{2}\left(\frac{b}{c} + \frac{c}{b} - \frac{a^2}{bc}\right)\right)$$

$$\theta_2 = \text{Arccos}\left(\frac{b^2 + c^2 - a^2}{2bc}\right)$$

Donc, $\theta = \theta_3 + \theta_2 = \text{Arccos}\left(\frac{b^2 + c^2 - a^2}{2bc}\right) + \frac{\pi}{3}$

Ainsi, d'après la Th. d'Al Kashi :

$$(SA)^2 = (ST)^2 + (TA)^2 - 2(ST)(TA)\cos(\theta)$$

$$(SA)^2 = (ST)^2 + \left(\sqrt{(L_4T)^2 + \left((L_4L_5)\frac{\sin(L_4\widehat{L_5}A)}{\sin(\alpha + L_4\widehat{L_5}A)}\right)^2 - 2(L_4T)(L_4L_5)\frac{\sin(L_4\widehat{L_5}A)\cos(\gamma)}{\sin(\alpha + L_4\widehat{L_5}A)}}\right)^2$$

$$- 2(ST)\left(\sqrt{(L_4T)^2 + \left((L_4L_5)\frac{\sin(L_4\widehat{L_5}A)}{\sin(\alpha + L_4\widehat{L_5}A)}\right)^2 - 2(L_4T)(L_4L_5)\frac{\sin(L_4\widehat{L_5}A)\cos(\gamma)}{\sin(\alpha + L_4\widehat{L_5}A)}}\right)$$

$$\times \cos\left(\text{Arccos}\left(\frac{(L_4T)^2 + (TA)^2 - (L_4A)^2}{2(L_4T)(TA)}\right) + \frac{\pi}{3}\right)$$

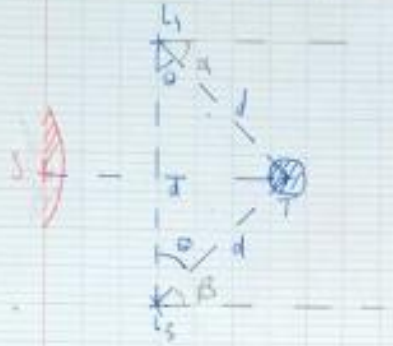
On remarque que : $(ST) = (SL_4) = (SL_5)$

$$(SA)^2 = (ST)^2 + \left[(ST)^2 + \left((L_4L_5)\frac{\sin(L_4\widehat{L_5}A)}{\sin(\alpha + L_4\widehat{L_5}A)}\right)^2 - 2(ST)(L_4L_5)\frac{\sin(L_4\widehat{L_5}A)\cos(\gamma)}{\sin(\alpha + L_4\widehat{L_5}A)}\right]$$

$$- 2(ST)\left[(ST)^2 + \left((L_4L_5)\frac{\sin(L_4\widehat{L_5}A)}{\sin(\alpha + L_4\widehat{L_5}A)}\right)^2 - 2(ST)(L_4L_5)\frac{\sin(L_4\widehat{L_5}A)\cos(\gamma)}{\sin(\alpha + L_4\widehat{L_5}A)}\right]^{1/2}$$

$$\times \cos\left[\text{Arccos}\left(\frac{(L_4T)^2 + (ST)^2 + \left((L_4L_5)\frac{\sin(L_4\widehat{L_5}A)}{\sin(\alpha + L_4\widehat{L_5}A)}\right)^2 - 2(ST)(L_4L_5)\frac{\sin(L_4\widehat{L_5}A)\cos(\gamma)}{\sin(\alpha + L_4\widehat{L_5}A)}}{2(ST)\sqrt{(ST)^2 + \left((L_4L_5)\frac{\sin(L_4\widehat{L_5}A)}{\sin(\alpha + L_4\widehat{L_5}A)}\right)^2 - 2(ST)(L_4L_5)\frac{\sin(L_4\widehat{L_5}A)\cos(\gamma)}{\sin(\alpha + L_4\widehat{L_5}A)}}}\right) + \frac{\pi}{3}\right]$$

Calcul préliminaire sur la géométrie du pt de Lagrange.



I. Trouver les angles α et β.

Remarque : Le triangle L_1TL_5 est isocèle en T
donc $\alpha = \beta$

Calculons θ :

Le triangle SL_1T est équilatéral.
Donc, l'angle $\widehat{SL_1T} = \frac{\pi}{3}$

Donc, $\theta = \frac{\widehat{SL_1T}}{2} = \frac{\pi}{6}$

D'où $\alpha = \beta = \frac{\pi}{2} - \frac{\pi}{6} = \frac{\pi}{3}$

II. Trouver la distance L_1L_5.

Trouvons x :

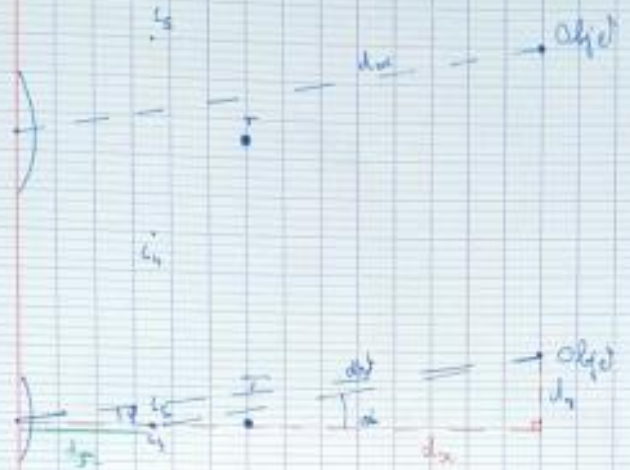
D'après le Th. de Pythagore :

$d^2 = x^2 + \frac{d^2}{4}$

$x = \sqrt{d^2 + \frac{d^2}{4}} = d\sqrt{1+\frac{1}{4}} = d\frac{\sqrt{5}}{2}$

Donc $L_1L_5 = 2x = \sqrt{5}\,d$

Compléments positions coord. cylindriques



$\sin(\alpha) = \frac{d_y}{d_{xc} - d_{SL}}$

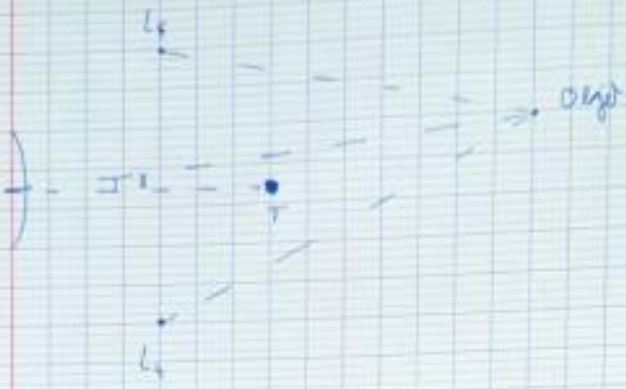
Donc $d_y = (d_{xc} - d_{SL}) \sin(\alpha)$

Donc, $d_{TL} = \sqrt{d_x^2 + d_y^2}$

Rem : $d_{SL} = \frac{\text{distance Terre - Hublot}}{2}$

$\tan(\varphi) = \frac{d_x}{d_{TL}}$

$\varphi = \arctan\left(\frac{d_x}{d_{TL}}\right)$



Dans le triangle $S\hat{T}\,Objet$

D'après la Th. d'Al-Kashi, on a :

$(S\text{-}Objet)^2 = (ST)^2 + (T\text{-}objet)^2 - 2(ST)(T\,objet)\cos(\alpha)$

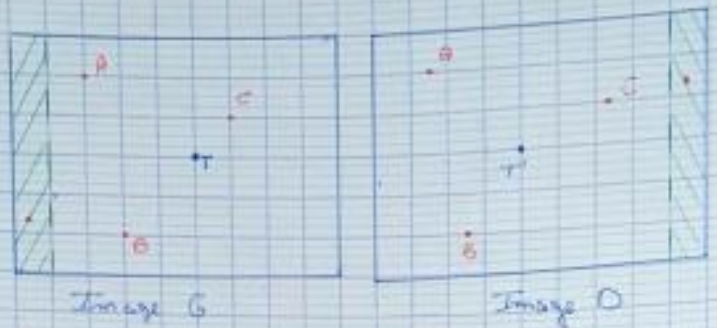
$\cos(\alpha) = \frac{(ST)^2 + (T\text{-}objet)^2 - (S\text{-}objet)^2}{2(ST)(T\text{-}objet)}$

De plus, d'après la loi des sinus :

$\frac{\sin(\alpha)}{(S\text{-}objet)} = \frac{\sin(\gamma)}{(T\,objet)}$

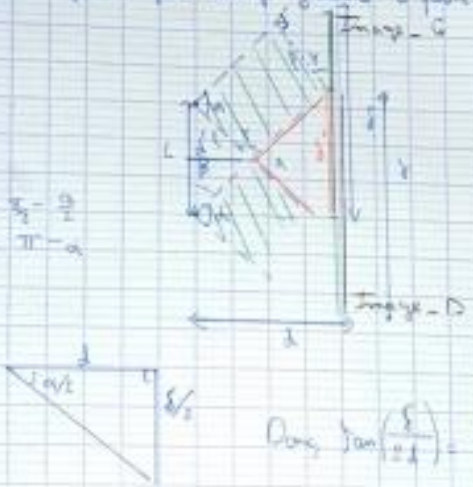
D'où $\sin(\gamma) = \sin(\alpha)\frac{(T\text{-}objet)}{(S\text{-}objet)}$

Méthode pour bonne cylindroïde



Objectif : Créer une liste de triplé de la forme :
[(Point_A_Image_G, Point_A_Image_D); ...] (*)

1ère étape : Supprimer les parties non communes aux images (partie gauche pour image gauche et partie droite pour image droite)

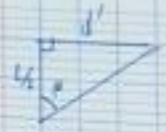


$\beta = \frac{\pi}{2} - \gamma$

$\alpha = \pi - \gamma$

Donc $\tan\left(\frac{\delta}{2d}\right) = \frac{\alpha}{2}$

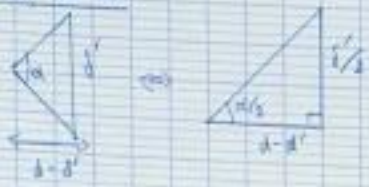
Donc $\delta = 2d \operatorname{Arctan}\left(\frac{\alpha}{2}\right)$



$\tan\left(\frac{L/2}{d'}\right) = \beta$

Donc $d' = \frac{L}{2} \operatorname{Arctan}(\beta)$

Calcul de δ' :



Donc $\tan\left(\frac{\delta'}{2(\delta-\delta')}\right) = \frac{\alpha}{2}$

Donc $\delta' = 2(d-d') \operatorname{Arctan}\left(\frac{\alpha}{2}\right)$

Ainsi, on peut en déduire la partie de l'image à retirer :

$$\begin{aligned} L = \delta - \delta' &= 2d \operatorname{Arctan}\left(\frac{\alpha}{2}\right) - 2(d-d') \operatorname{Arctan}\left(\frac{\alpha}{2}\right) \\ &= 2d' \operatorname{Arctan}\left(\frac{\alpha}{2}\right) \\ &= \frac{L}{2} \operatorname{Arctan}(\beta) \operatorname{Arctan}\left(\frac{\alpha}{2}\right) \end{aligned}$$ (*)

Grâce à la condition Pixel - Angle, on peut enlever la bonne partie de l'image.

2ème étape : Classer les points par rapport à leurs coordonnées "y".

On obtient alors 2 listes : L_{G_Y} et L_{D_Y} des classements des points des 2 images par rapport à "y".

(Vérifier que $L_{D\_Y} = L_{D\_Y}$, mais normalement c'est ok)

3ème étape: Classer les points par rapport à leur avance (en pixel) par rapport à la Terre.
On a alors deux LVE: LD_T et LG_T.

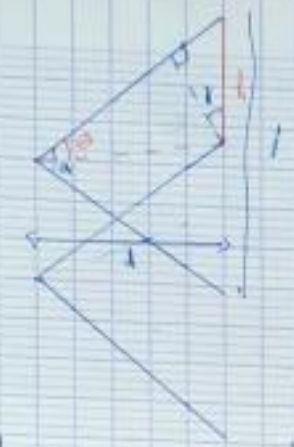
4ème étape: Créer la LVE: LP_ comme sur le schéma suivant:

- Prendre le classement de LD_Y (= LG_Y)
- Pour 2 étoiles ayant les même coordonnées en "y" faire:
  - classer par rapport à l'index dans LD_T & LG_T
  - Si leur index de la LVE est le même, faire:
    - classer par rapport à la coord. "x".

On a ainsi construit une liste, dont le classement des index est bijectif, comme définie en (*)

⚠ On néglige les éventuels étoiles qui pourraient être cachés derrière la Terre sur une image, et non pas sur l'autre image.
Par la suite, on supposera que ce problème n'existe pas.

(*)



δ	±
h	θ_x

$$\theta_x = \frac{h}{l}$$

Donc, le nombre de pixel à franchir est:

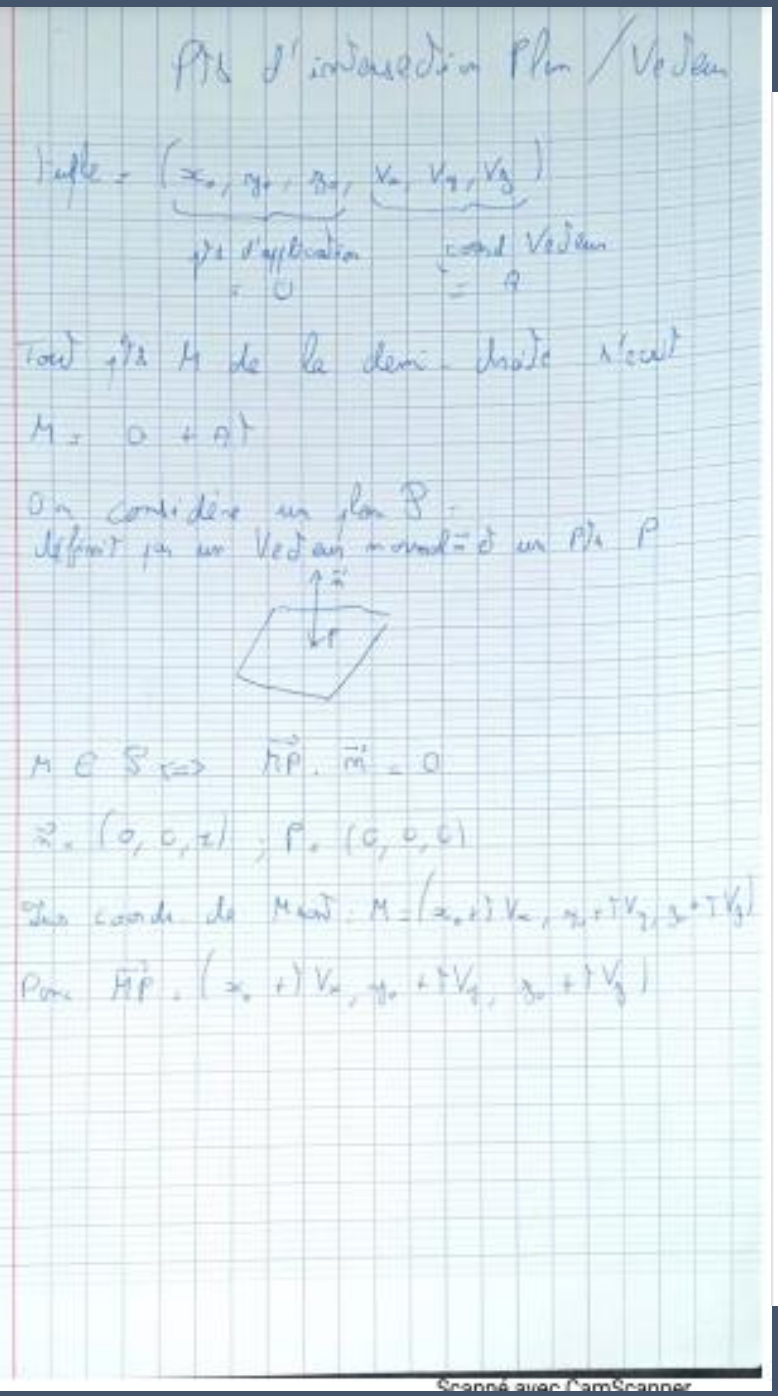
$$n = \text{abs}\_\text{pixel}\_\text{std} \times \theta_x$$

$$\theta_x = \frac{\frac{L}{2} A_{TAN}\left(\frac{\pi}{2} - \frac{\alpha}{2}\right) A_{TAN}\left(\frac{\alpha}{2}\right)}{2\, d\, A_{TAN}\left(\frac{\alpha}{2}\right)}$$

$$= \frac{L}{4d} A_{TAN}\left(\frac{\pi}{2} - \frac{\alpha}{2}\right)$$

$$\Rightarrow n \approx 0 \quad (\text{quand } d \to +\infty)$$

Conclusion: Il est difficile de déterminer précisément la partie des deux images communes.
Il est aussi impossible d'identifier une même étoile sur 2 images.
On suppose donc qu'il n'y a qu'une seul étoile




```python
import PIL
import numpy as np
import PIL.ImageOps
import matplotlib.pyplot as plt
import math
from PIL import Image

plt.close("all")

"""Le but est-ici de prendre en arguments 2 images prises au
points L4 et L5 et de déterminer la distance entre le soleil et
le points observé"""

## Variables (a eventuellement modifier)

test_image = "test_algo.jpg"
image_1 = "test_algo.jpg" #premiere serie de photo
image_2 = "test_algo.jpg" #premiere serie de photo
dt = 3600 # en s (temps entre 2 series de photos)
imagedt_1 = "test_algo.jpg" #deuxieme serie de photo
imagedt_2 = "test_algo.jpg" #deuxieme serie de photo
angle_L_T = math.pi / 3 #rad, angle entre le pts de Lagrange et
la terre
distance_T_S = 150000000 #km
distance_L4_L5 = math.sqrt(5) * distance_T_S
rayon_Terre = 6371 #km
n = [0, 0, 1] # vecteur normal au plan de la trajectoire de la
Terre
P = [0, 0, 0] # Origine, position du Soleil
a = 0 # origine des temps
```

```
## Transformation images

def image_grise(nomFichier):
    '''retourne l'image en niveaus de gris sous forme de
tableau'''
    Img = PIL.Image.open(nomFichier)
    Img = PIL.ImageOps.grayscale(Img)
    largeur,hauteur = Img.size
    imdata=Img.getdata()
    tab=np.array(imdata)
    return  np.reshape(tab,(hauteur,largeur))

def seuillage(im, seuil):
    l,h,t = im.shape
    for i in range(l):
        for j in range(h):
            if im[i,j, 0] > seuil:
                im[i, j]= 255
            else:
                im[i, j] = 0
    return im

seuillage(img2, 50)
seuillage(img1, 50)
seuillage(img, 50)
plt.imshow(img)
plt.title('Image seuillée')
plt.show()
```

```
## Ouverture d'images

def extraction_image(nomFichier):
    '''retourne l'image sous forme de tableau'''
    img = Image.open(nomFichier)
    return np.array(img)

def sauvegarde_image(img,nomFichier):
    ''' img : matrice image
    nomFichier : nom du fichier de sortie'''
    imgpil = Image.fromarray(img) # Transformation du tableau en
image PIL
    imgpil.save(nomFichier)

img2 = extraction_image(image_2)
img1 = extraction_image(image_1)
img = extraction_image(test_image) #test
img1 = img1[::10,::10,:]
img2 = img2[::10,::10,:]
img = img[::10,::10,:]
plt.imshow(img)
plt.show()
```

```python
def est_noir(pixel):
    a, b, c = pixel
    if a == 0 and b == 0 and c == 0:
        return True
    else:
        return False

def paquet(img, pix, L):
    """renvoie la liste des coordonnées des pixels voisins noirs
de pix"""
    x, y = pix
    l, h, t = img.shape
    if not est_noir(img[x, y]):
        L.append(pix)
        img[x, y] = 0
    else:
        return None
    liste_voisin = [(x + 1, y), (x - 1, y), (x, y + 1), (x, y -
1)]
    for i in range(len(liste_voisin)):
        x,y = liste_voisin[i]
        if 0 <= x < l and 0 <= y < h :
            paquet(img, liste_voisin[i], L)
```

```python
def paquet_all(img):
    """renvoie une liste de liste de paquet de chaque objet
celeste"""
    M = [] # liste final
    l, h, t = img.shape
    for i in range(l):
        for j in range(h):
            L = []
            paquet(img, (i, j), L)
            if L != [None] and L != []:
                M.append(L)
    return M

def Trouve_terre(L):
    """prend en argument la liste renvoyé par paquet_all et
renvoie la position de l'objet : Terre dans cette liste"""
    L_len = [] # liste verifiant pour tout i dans [|0, len(L)|]
L_len[i] = len(L[i])
    for i in range(len(L)):
        L_len.append(len(L[i]))
    maxi = max(L_len) # taille de la liste la plus grande (donc
de la liste Terre)
    for i in range(len(L_len)):
        if L_len[i] == maxi:
            return i
```

```python
def arrondi(x):
    """prend en argument un floatant et renvoie l'entier le plus proche"""
    if x - int(x) < 0.5:
        return int(x)
    else:
        return int(x) + 1

def moyenne_paquet(L):
    """prend en argument la liste renvoyé par paquet_all et renvoie une liste des points centraux de chaque paquet"""
    M = []
    for i in range(len(L)):
        sx = 0 # somme des coord en x
        sy = 0 # somme des coord en y
        for j in range(len(L[i])):
            x, y = L[i][j]
            sx, sy = sx + x, sy + y
        moy_paquet = (arrondi(sx / len(L[i])), arrondi(sy / len(L[i])))
        M.append(moy_paquet)
    return M
```

```
def pointe_image(img):
    """prend en argument l'iamge en noir et blanc et renvoie une
image blanche avec un pixel rouge sur la postion des etoiles et
un pixel bleu sur la position de la Terre"""
    L = paquet_all(img)
    k = Trouve_terre(L)
    L = moyenne_paquet(L)
    for i in range(len(L)):
        x, y = L[i]
        if i == k:
            img[x, y, 2] = 255
        else:
            img[x, y, 0] = 255
    return img

img2 = pointe_image(img2)
img1 = pointe_image(img1)
img = pointe_image(img)
plt.imshow(img)
plt.title('image pointé')
plt.show()
```

```
## Determiner distance Soleil-objet

# On suppose connaitre les angles alpha et beta formé par les
angles L5-L4-Objet et L4-L5-Objet

print("pour tester, choisissez des valeurs pour alpha (=a) et
beta (=b)")
print("attention a respecter a + b < pi")
a = input("a = ")
b = input("b = ")
a = float(a)
b = float(b)

def distance_L4_A(a, b):
    return distance_L4_L5 * math.sin(b) / math.sin(a + b)


def distance_L5_A(a, b):
    return distance_L4_L5 * math.sin(a) / math.sin(a + b)


def ang_T_L4_A(a):
    return a - (math.pi / 6)


def distance_T_A(a, b):
    gamma = ang_T_L4_A(a)
    return math.sqrt(distance_T_S ** 2 + (distance_L4_A(a, b)) **
2 - 2 * distance_T_S * distance_L4_A(a, b) * math.cos(gamma))
```

```
def ang_S_T_A(a, b):
    return math.acos((distance_T_S ** 2 + (distance_T_A(a, b)) **
2 - (distance_L4_A(a, b)) ** 2) / (2 * distance_T_S *
distance_T_A(a, b))) + math.pi / 3


def distance_A_S(a, b):
    return math.sqrt(distance_T_S ** 2 + (distance_T_A(a, b)) **
2 - 2 * distance_T_S * distance_T_A(a, b) * math.cos(ang_S_T_A(a,
b)))

print("l'objet étudié est a", int(distance_T_A(a,b)), "km de la
Terre et ", int(distance_A_S(a, b)), "km du soleil")

## Trouver asteroide (complet)

"""Non abouti car trop compliqué pour le moment."""

def coord_etoiles(img):
    """prend en argument une image pointé et renvoie les
coordonnées des points autre que la Terre puis les coordonnées de
la Terre"""
    L = []
    l, h, t = img.shape
    for i in range(l):
        for j in range(h):
            if not est_noir(img[i, j]) and img[i, j, 2] == 0:
                L.append((i, j))
            if not est_noir(img[i, j]) and img[i, j, 0] == 0:
                Terre = (i, j)
    return L, Terre
```

```
def tronque_gauche(img):
    """enleve la partie non commune de l'image de gauche"""
    l, h, t = img.shape
    alpha = angle_par_pixel * h
    beta = (math.pi - alpha) / 2
    h = distance_L4_L5 / 2 * math.atan(beta) * math.atan(alpha /
2)
    teta = 0

## Trouve asteroide (simplifié)

"""On suppose ici que les 2 photos sont composées uniquement de
la Terre et d'une seul étoile (qui est l'objet étudié)"""

def coord_etoile(img):
    """prend en argument une image pointé et renvoie les
coordonnées du point autre que la Terre puis les coordonnées de
la Terre"""
    L = []
    l, h, t = img.shape
    for i in range(l):
        for j in range(h):
            if not est_noir(img[i, j]) and img[i, j, 2] == 0:
                L.append((i, j))
            if not est_noir(img[i, j]) and img[i, j, 0] == 0:
                Terre = (i, j)
    return L, Terre
```

```
## Trouver l'angle correspondant a un pixel

def centre_image(img):
    """renvoie les coords du points centrale de l'image"""
    l, h, t = img.shape
    return (int(l / 2), int(h / 2))

def conv_pixel_angle_x(img):
    xc, yc = centre_image(img)
    L, T = coord_etoiles(img)
    xt, yt = T
    return (angle_L_T) / (abs(xc - xt))

angle_par_pixel = conv_pixel_angle_x(img)

## Trouver l'angle L5-L4-Objet (resp L5-L4-Objet)

# on suppose connaitre les coords du pixel de l'objet etudié sur
l'image qu'on notera pix

def angle_manquant(img):
    """ trouve l'angle manquant pour que l'objectif de l'appareil
photo puisse prendre une image a 180°"""
    a = angle_par_pixel
    l, h, t = img.shape
    return math.pi / 2 - (h / 2 * a)

def Trouve_angle(pix, img):
    """donne les coordonnée (en angle) du pixel etudié"""
    xc, yc = centre_image(img)
    (xo, yo) = pix
    l, h, t = img.shape
    a = angle_par_pixel
    am = angle_manquant(img)
    return ((l - xo) * a + am, (yc - yo) * a)
```

```
def angle_L_objet(img1, img2):
    """renvoie l'angle L4-L5-objet, Terre-L5-objet, L5-L4-objet,
Terre-L4-objet"""
    #angle etoile img1
    L, _ = coord_etoile(img1)
    L = L[0]
    [a, b] = L
    pix_img1 = (a, b)
    angle_x_img1, angle_y_img1 = Trouve_angle(pix_img1, img1)
    #angle etoile img2
    L, _ = coord_etoile(img1)
    L = L[0]
    [a, b] = L
    pix_img2 = (a, b)
    angle_x_img2, angle_y_img2 = Trouve_angle(pix_img2, img2)
    return angle_x_img1, angle_y_img1, angle_x_img2, angle_y_img2

## Trouver distance Soleil-objet

def distance_soleil_objet(img1, img2):
    """prend en argument 2 images pris au point L4 et L5 et
renvoie la distance Soleil-objet"""
    angle_x_img1, angle_y_img1, angle_x_img2, angle_y_img2 =
angle_L_objet(img1, img2)
    dx = distance_A_S(angle_x_img1, angle_x_img2)
    dsl = distance_T_S / 2
    angle_y_moyen = (angle_y_img1 + angle_y_img2) / 2
    dy = (dx - dsl) * math.sin(angle_y_moyen)
    dtot = math.sqrt(dx ** 2 + dy ** 2)
    return dtot
```

```
## Trouver vitesse radiale, vitesses angulaires (sur "phi" et
"teta")

def vitesse_radial(img1, img2, imgdt1, imgdt2):
    """prend en argument 2 couples d'images en t et t + dt et
renvoie la vitesse radial de l'objet"""
    dtot = distance_soleil_objet(img1, img2)
    dtot_dt = distance_soleil_objet(imgdt1, imgdt2)
    v_radial = (dtot_dt - dtot) / dt
    return v_radial

def vitesse_angulaire_phi(img1, img2, imgdt1, imgdt2):
    """prend en argument 2 couples d'images en t et t + dt et
renvoie la vitesse angulaire sur phi de l'objet"""
    angle_x_img1, angle_y_img1, angle_x_img2, angle_y_img2 =
angle_L_objet(img1, img2)
    dx = distance_A_S(angle_x_img1, angle_x_img2)
    dsl = distance_T_S / 2
    angle_y_moyen = (angle_y_img1 + angle_y_img2) / 2
    dy = (dx - dsl) * math.sin(angle_y_moyen)
    dtot = math.sqrt(dx ** 2 + dy ** 2)
    phi = math.acos(dx / dtot) #determination de phi sur la
premiere serie d'image
    angle_x_img1dt, angle_y_img1dt, angle_x_img2dt,
angle_y_img2dt = angle_L_objet(imgdt1, imgdt2)
    dxdt = distance_A_S(angle_x_img1dt, angle_x_img2dt)
    angle_y_moyendt = (angle_y_img1dt + angle_y_img2dt) / 2
    dydt = (dxdt - dsl) * math.sin(angle_y_moyendt)
    dtotdt = math.sqrt(dxdt ** 2 + dydt ** 2)
    phidt = math.acos(dxdt / dtotdt) #determinationde phi sur la
deuxieme serie d'image
    v_angulaire_phi = (phidt - phi) / dt
    return v_angulaire_phi
```

```
def vitesse_angulaire_teta(img1, img2, imgdt1, imgdt2):
    """prend en argument 2 couples d'images en t et t + dt et
renvoie la vitesse angulaire sur teta de l'objet"""
    angle_x_img1, angle_y_img1, angle_x_img2, angle_y_img2 =
angle_L_objet(img1, img2)
    T0 = distance_T_A(angle_x_img1, angle_x_img2)
    S0 = distance_A_S(angle_x_img1, angle_x_img2)
    alpha = math.acos((distance_T_S ** 2 + T0 ** 2 - S0 ** 2) /
(2 * distance_T_S * T0))
    teta = math.asin((math.sin(alpha) * T0) / S0) #determination
de teta sur la premiere serie d'image
    angle_x_img1dt, angle_y_img1dt, angle_x_img2dt,
angle_y_img2dt = angle_L_objet(imgdt1, imgdt2)
    T0dt = distance_T_A(angle_x_img1dt, angle_x_img2dt)
    S0dt = distance_A_S(angle_x_img1dt, angle_x_img2dt)
    alphadt = math.acos((distance_T_S ** 2 + T0dt ** 2 - S0dt **
2) / (2 * distance_T_S * T0dt))
    tetadt = math.asin((math.sin(alphadt) * T0dt) / S0dt) + ((2 *
math.pi * dt) / (365.25 * 24 * 60 * 60)) #determination de teta
sur la deuxieme serie d'image en prenant compte du deplacement de
la Terre par rapport au soleil
    v_angulaire_teta = (tetadt - teta) / dt
    return v_angulaire_teta
```

```
def vitesse_coord_carthesienne(img1, img2, imgdt1, imgdt2):
    """prend en argument 2 couples d'images en t et t + dt et
renvoie une liste de 3 elements correspondents au coordonnées
(e_x, e_y, e_z) du vecteur vitesse en coordonnées
carthesiennes"""
    V_spherique = vitesse_coord_cylindrique(img1, img2, imgdt1,
imgdt2)
    V_carthesienne = [V_spherique[0] * math.sin(V_spherique[1]) *
math.cos(V_spherique[2]), V_spherique[0] *
math.sin(V_spherique[1]) * math.sin(V_spherique[2]),
V_spherique[0] * math.cos(V_spherique[1])]
    return V_carthesienne
```

```
def vitesse_coord_cylindrique(img1, img2, imgdt1, imgdt2):
    """prend en argument 2 couples d'images en t et t + dt et
renvoie une liste de 3 elements correspondents au coordonnées
(e_r, e_teta, e_phi) du vecteur vitesse en coordonnées
spheriques"""
    v_angulaire_teta = vitesse_angulaire_teta(img1, img2, imgdt1,
imgdt2)
    v_angulaire_phi = vitesse_angulaire_phi(img1, img2, imgdt1,
imgdt2)
    v_radial = vitesse_radial(img1, img2, imgdt1, imgdt2)
    angle_x_img1dt, angle_y_img1dt, angle_x_img2dt,
angle_y_img2dt = angle_L_objet(imgdt1, imgdt2)
    TOdt = distance_T_A(angle_x_img1dt, angle_x_img2dt)
    SOdt = distance_A_S(angle_x_img1dt, angle_x_img2dt)
    alphadt = math.acos((distance_T_S ** 2 + TOdt ** 2 - SOdt **
2) / (2 * distance_T_S * TOdt))
    tetadt = math.asin((math.sin(alphadt) * TOdt) / SOdt) + ((2 *
math.pi * dt) / (365.25 * 24 * 60 * 60))
    return [v_radial, SOdt * v_angulaire_teta, SOdt *
math.sin(tetadt) * v_angulaire_phi]
```

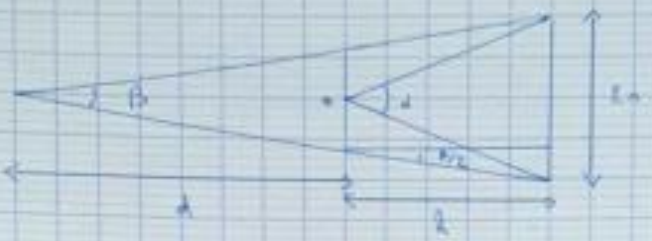
```
## Trouver le point en coordonnées carthesiennes de l'objet

def coords_pts_spherique(img1, img2, imgdt1, imgdt2):
    """Determine les coordonnées spherique de l'objet"""
    angle_x_img1dt, angle_y_img1dt, angle_x_img2dt,
angle_y_img2dt = angle_L_objet(imgdt1, imgdt2)
    TOdt = distance_T_A(angle_x_img1dt, angle_x_img2dt)
    SOdt = distance_A_S(angle_x_img1dt, angle_x_img2dt)
    alphadt = math.acos((distance_T_S ** 2 + TOdt ** 2 - SOdt **
2) / (2 * distance_T_S * TOdt))
    tetadt = math.asin((math.sin(alphadt) * TOdt) / SOdt) + ((2 *
math.pi * dt) / (365.25 * 24 * 60 * 60))
    dxdt = distance_A_S(angle_x_img1dt, angle_x_img2dt)
    angle_y_moyendt = (angle_y_img1dt + angle_y_img2dt) / 2
    dydt = (dxdt - dsl) * math.sin(angle_y_moyendt)
    dtotdt = math.sqrt(dxdt ** 2 + dydt ** 2)
    phidt = math.acos(dxdt / dtotdt)
    pos_spherique = [SOdt, tetadt, phidt]
    return pos_spherique
```

```
def coords_pts_carthesienne(img1, img2, imgdt1, imgdt2):
    """Determine les coordonnées carthesienne de l'objet"""
    pos_spherique = coords_pts_spherique(img1, img2, imgdt1,
imgdt2)
    pos_carthesienne = [pos_spherique[0] *
math.sin(pos_spherique[1]) * math.cos(pos_spherique[2]),
pos_spherique[0] * math.sin(pos_spherique[1]) *
math.sin(pos_spherique[2]), pos_spherique[0] *
math.cos(pos_spherique[1])]
    return pos_carthesienne

### Determinons si le vecteur vitesse est orienté vers la Terre

def position_M(pos_carthesienne, V_carthesienne, t):
    """Renvoie la position d'un objet suivant la droite
(pos_carthesienne, V_carthesienne) au temps t"""
    pos = pos_carthesienne
    for i in range (len(V_carthesienne)):
        pos[i] = pos[i] + t * V_carthesienne
    return pos

def produit_scalaire(vect1, vect2):
    """Renvoie le produit scalaire des vecteurs vect1 et vect2"""
    res = 0
    for i in range(len(vect1)):
        res = res + vect1[i] * vect2[i]
    return res

def vecteur_MP(M, P):
    """Prend en argument 2 liste definissant la position des
points M et P et renvoie le vecteur Vect(MP)"""
    vect_MP = [P[0] - M[0], P[1] - M[1], P[2] - M[2]]
    return vect_MP
```

```
def equation_intersection(pos_carthesienne, V_carthesienne, n,
t):
    """renvoie Vect(MP).Vect(n)"""
    return
produit_scalaire(vecteur_MP(position_M(pos_carthesienne,
V_carthesienne, t), P), n)

def Trouve_t(pos_carthesienne, V_carthesienne, b):
    """Trouve le temps t de l'intersection entre l'objet et le
plan constituant la trajectoire de la Terre en resolvant par
dichotomie equation_intersection = 0"""
    prec = 3600   # 1h
    if (equation_intersection(pos_carthesienne, V_carthesienne,
a) < 0 and equation_intersection(pos_carthesienne,
V_carthesienne, b) > 0) or
(equation_intersection(pos_carthesienne, V_carthesienne, a) > 0
and equation_intersection(pos_carthesienne, V_carthesienne, b) <
0):
        while abs(b - a) < prec:
            m = 0.5 * (a + b)
            if equation_intersection(pos_carthesienne,
V_carthesienne, m) < 0:
                a = m
            else:
                b = m
    else:
        print("Pas de solution")
    sol = (a + b) / 2
    a = 0   # Pour eviter les conflits si on execute plusieurs
fois la fonction
    return sol
```

```
def point_intersection(pos_carthesienne, V_carthesienne, b):
    """renvoie le points d'intersection entre le plan et la
trajectoire de l'objet"""
    t = Trouve_t(pos_carthesienne, V_carthesienne, b)
    return position_M(pos_carthesienne, V_carthesienne, t)

def collision(pos_carthesienne, V_carthesienne, b, marge_erreur):
    """Renvoie True si l'objet rentrera en collision avec la
Terre et False sinon, marge erreur etant une liste de taille 3
avec la marge d'erreur sur x, y et z"""
    pos = point_intersection(pos_carthesienne, V_carthesienne, b)
    if (abs(pos[0] - distance_T_S) < marge_erreur[0]) and
(abs(pos[1] - distance_T_S) < marge_erreur[1]) and (abs(pos[2] -
rayon_Terre) < marge_erreur[2]):
        return True
    else:
        return False
```

Calcul de marge



α = alpha pixel = angle par pixel

$\tan\left(\frac{\alpha}{2}\right) = \frac{n}{h} \Rightarrow \alpha = 2 \arctan\left(\frac{n}{h}\right)$

De même, $\beta = 2 \arctan\left(\frac{n}{h+d}\right)$

$\tan\left(\frac{\beta}{2}\right) = \frac{n}{kl} \Rightarrow n = kl \tan\left(\frac{\beta}{2}\right)$

$2n = 2d \tan\left(\frac{\beta}{2}\right) + 2k \tan\left(\frac{\beta}{2}\right) \times h$

$= 2\tan\left(\frac{\beta}{2}\right)(d+h)$

$n = \tan\left(\frac{\beta}{2}\right)(d+h) \quad$ or $\quad h = \frac{n}{\tan\left(\frac{\beta}{2}\right)}$

Donc $2\tan\left(\frac{\beta}{2}\right)\left(d + \frac{n}{\tan(\alpha/2)}\right) = n$

$\Leftrightarrow 2\tan\left(\frac{\beta}{2}\right) d + 2n \frac{\tan\left(\frac{\beta}{2}\right)}{\tan(\alpha/2)} = n$

$\Leftrightarrow 2\tan\left(\frac{\beta}{2}\right) d = n\left(1 - 2\frac{\tan\left(\frac{\beta}{2}\right)}{\tan(\alpha/2)}\right)$

$\Leftrightarrow n = \frac{2d \tan\left(\frac{\beta}{2}\right)}{1 - 2\frac{\tan\left(\frac{\beta}{2}\right)}{\tan\left(\frac{\alpha}{2}\right)}}$

```
## Variables

p_asteroide = 3780 #kg/m³
angle_photo = 1.12 #angle de mon telephone

## Calcul rayon objet celeste

def image_grise(nomFichier):
    '''
    retourne l'image en niveaus de gris sous forme de tableau
    '''
    Img = PIL.Image.open(nomFichier)
    Img = PIL.ImageOps.grayscale(Img)
    largeur,hauteur = Img.size
    imdata=Img.getdata()
    tab=np.array(imdata)
    return  np.reshape(tab,(hauteur,largeur))

def extraction_image(nomFichier):
    '''
    retourne l'image sous forme de tableau
    '''
    img = Image.open(nomFichier)
    return  np.array(img)

def sauvegarde_image(img,nomFichier):
    '''
    img : matrice image
    nomFichier : nom du fichier de sortie
    '''
    imgpil = Image.fromarray(img) # Transformation du tableau en
image PIL
    imgpil.save(nomFichier)
```

```
#img = extraction_image('tchouri.jpg') #test
#plt.imshow(img)
#plt.show()
img1 = extraction_image(img1)
img2 = extraction_image(img2)
img1 = img1[::10,::10,:]
img2 = img2[::10,::10,:]
#plt.imshow(img2)
#plt.show()

def seuillage(im, seuil):
    l, h, t = im.shape
    for i in range(l):
        for j in range(h):
            if im[i,j, 0] > seuil:
                im[i, j]= 0
            else:
                im[i, j] = 255
    return im

#seuillage(img, 50)
seuillage(img1, 50)
seuillage(img2, 50)
#plt.imshow(img)
#plt.title('Image seuillée')
#plt.show()
plt.imshow(img1)
plt.show()
```

```
def extremite(img):
    """renvoie les coordonnées du point le plus haut, du point le
plus bas, et des extremités"""
    coordpoints = []
    l, h, t = img.shape
    i = 0
    j = 0
    while len(coordpoints) < 1: #pts hauts
        while coordpoints == []:
            if j == h:
                i = i +1
                j = 0
            if img[i, j, 0] == 255:
                coordpoints.append((i, j))
            j = j + 1
    i = l - 1
    j = h - 1
    while len(coordpoints) < 2: #pts bas
        while len(coordpoints) == 1:
            if j == 0:
                i = i - 1
                j = h - 1
            if img[i, j, 0] == 255:
                coordpoints.append((i, j))
            j = j - 1
    return coordpoints

def trouve_Rayon(coordpoints):
    """determine la taille vertical de l'objet en pixel"""
    mini_x, mini_y = coordpoints[0]
    maxi_x, maxi_y = coordpoints[-1]
    return (maxi_x - mini_x) / 2
```

```
def angle_par_pixel(img):
    """Détermine le nombre de radian par pixel"""
    l, h, t = img.shape
    return angle_photo / h

def masse(img1, img2, d):
    """Détermine la masse de l'objet en suposant qu'il est
spherique"""
    r_metre = trouve_Rayon_reel(img1, img2, d)
    return 4 / 3 * math.pi * r_metre ** 3

def trouve_Rayon_reel(img1, img2, d):
    """Détermine le rayon de l'objet observé"""
    alpha = trouve_Rayon(extremite(img1)) * angle_par_pixel(img1)
    beta = trouve_Rayon(extremite(img2)) * angle_par_pixel(img2)
    return ((2 * d * math.tan(beta / 2)) / (1 - 2 *
((math.tan(beta / 2)) / math.tan(alpha / 2)))) / 2


taille_reel = 9 / 2 #taille de l'objet testé en cm
taille_mesuree = trouve_Rayon_reel(img1, img2, 20) #en cm

print("L'objet étudié mesure", taille_reel, "cm, et l'algorithme
mesure", taille_mesuree, "cm. Soit une precision de :",
taille_reel / taille_mesuree)
```