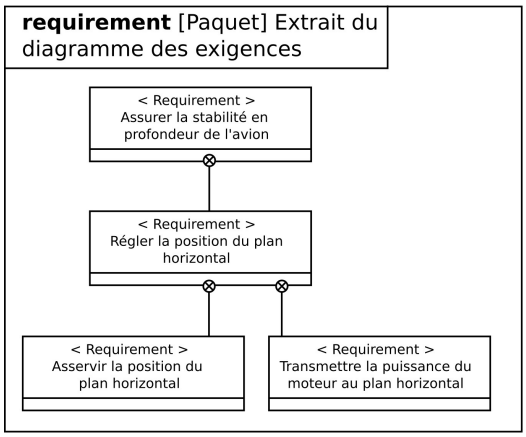
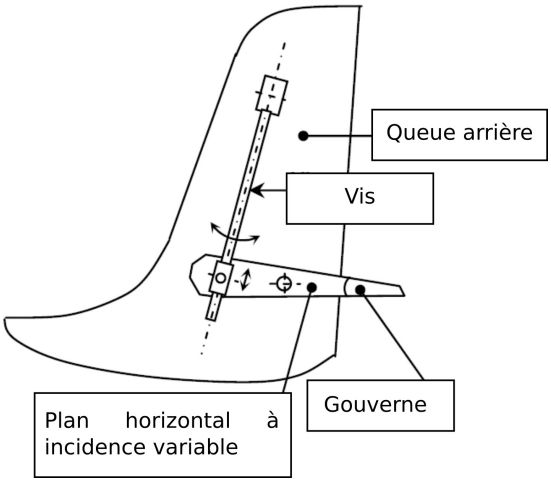
Entrée [1]:
```
## Bibliotèques nécessaires

import matplotlib.pyplot as plt
import numpy as np
```

Loi entrée-sortie du robot MAXPID

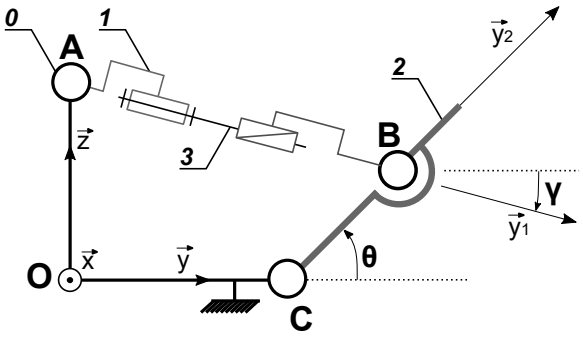
L'empennage d'un avion peut être assimilé à une mini-aile située à l'extrémité du fuselage de l'avion. Il subit donc, comme la voilure principale, une force de portance et une force de traînée appliquées en son centre de poussée. Un empennage est composé de deux parties :

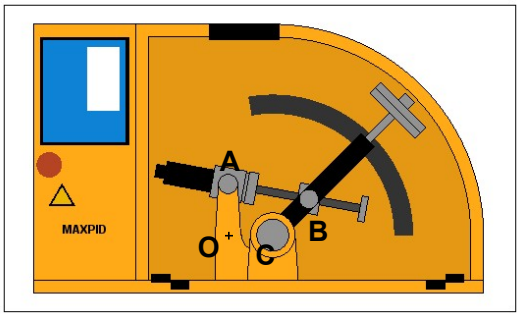
- la première, plan horizontal et gouverne, assure la stabilité en profondeur de l'avion,
- la deuxième, queue arrière, assure la stabilité en direction. Nous nous intéressons dans ce sujet au système qui permet d'asservir en position l'ensemble plan horizontal et gouverne. Les fonctions de ce système sont décrites ci-dessous :





On s'intéresse ici à l'exigence "Régler la position du plan horizontal". Pour ce faire, il faut alors connaitre la loi entrée-sortie : $y = f(\theta)$ où y représente la longueur $\overrightarrow{AB}$ et θ l'angle d'orientation de la gouverne (voir schéma cinématique ci-dessous). f sera une fonction à déterminer.





Pour des soucis de simplicité, la manipulation n'est pas exactement dans la même configuration que l'empennage. Le cahier des charges imposerait alors que, sur la manipulation étudiée au lycée, ***l'angle*** θ ***varie de*** $\theta_{\min}$ ***=10° à*** $\theta_{\max}$ ***=80°.***

Le paramétrage est le suivant :

- $\overrightarrow{OA} = L_1 \overrightarrow{z}$ avec $L_1 = 90$ mm
- $\overrightarrow{OC} = L_2 \overrightarrow{y}$ avec $L_2 = 65$ mm
- $\overrightarrow{AB} = y\overrightarrow{y_1}$
- $\overrightarrow{CB} = L_3\overrightarrow{y_2}$ avec $L_3 = 85$ mm
- $(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z})$ est la base associée au solide 0, $(\overrightarrow{x_1}, \overrightarrow{y_1}, \overrightarrow{z_1})$ est la base associée au solide 1 et

$(\overrightarrow{x_2}, \overrightarrow{y_2}, \overrightarrow{z_2})$ est la base associée au solide 2 avec $\overrightarrow{x} = \overrightarrow{x_1} = \overrightarrow{x_2}$; $\theta = (\overrightarrow{y}, \overrightarrow{y_2})$ et $\gamma = (\overrightarrow{y}, \overrightarrow{y_1})$

Mise en place de la loi entrée-sortie analytique et confrontation expérimentale

Question 1. A partir d'une fermeture géométrique, déterminer une relation entre y et θ et des données constantes connues.

Question 2. Modifier cette équation et la mettre sous la forme $y = f(\theta)$. Serait-il possible d'exprimer simplement θ en fonction de y ?

Nous allons, dans un premier temps, représenter la courbe associée à la fonction f à partir du code ci-dessous.

NOTA 1 - `theta` est ici un tableau `numpy` qui n'a pas exactement les mêmes propriétés qu'une liste "classique". Il est par exemple possible d'écrire directement `y = 5*theta + 2` ce qui calculera directement chaque terme `y[i]` en fonction de `theta[i]` et renverra donc un tableau numpy `y` de même longueur que `theta` . Le même type de syntaxe avec une liste "classique" mène à une erreur et imposerait de mettre en place une boucle `for` .

NOTA 2 - on utilisera les fonctions `np.sqrt` , `np.cos` ou encore `np.sin` pour les fonctions mathématiques usuelles.

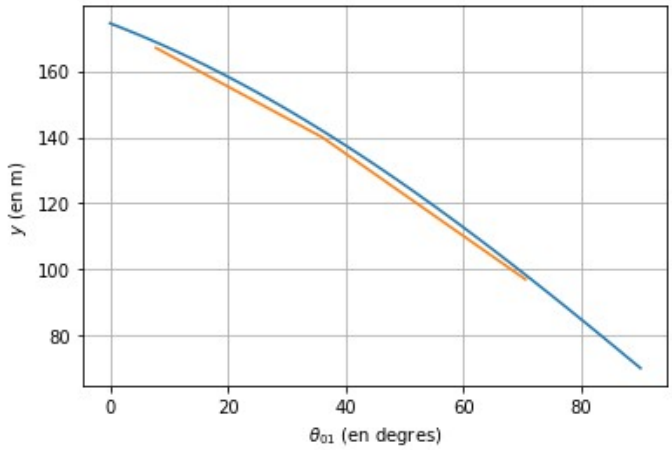
Question 3. Compléter la ligne `y = ...` . Exécuter ensuite pour afficher le graphique.

!!! MANIPULATION !!! Sur la maquette, relever les longueurs L_1, L_2 et L_3. Relever également, pour un angle θ donné, la longueur y associée (vous ferez trois ou quatre relevés). Compléter et modifier alors en conséquence les valeur numériques du code ci-dessous.

!!! CONFRONTATION !!! Discutez de la qualité du modèle établi vis-à-vis des mesures effectuées.

Entrée [2]:
```
## Dimensions du mécanisme en mm (y sera aussi en mm)
L1 = 80.
L2 = 70.
L3 = 85.


## Résolution numérique
theta = np.linspace(0.,np.pi/2,100)
y =  np.sqrt(  (L2 + L3*np.cos(theta))**2 + (L1 - L3*np.sin(theta))**2  )

theta = 180*theta/(np.pi) ## pour affichage en degrés

## Tracé de beta en fonction theta01
plt.plot(theta,y)

theta_exp = [7.8,36.,70.5] ## valeurs numériques des angles relevés
y_exp = [167.,140.,97.] ## valeurs numériques des y relevés
plt.plot(theta_exp,y_exp)

## Options du tracé
plt.grid(True)
plt.xlabel('$\\theta_{01}$ (en degres)')
plt.ylabel('$y$ (en m)')
```

Out[2]:
```
Text(0, 0.5, '$y$ (en m)')
```



Entrée [3]:
```
## pour q4
theta = np.linspace(10.*(np.pi/180.),(80/90)*np.pi/2,2)
y =  np.sqrt(  (L2 + L3*np.cos(theta))**2 + (L1 - L3*np.sin(theta))**2  )

theta = 180*theta/(np.pi) ## pour affichage en degrés

print(y[0],y[1])
```

```
166.98082837432273 84.84119207832032
```

Question 4. Relever, à partir de la courbe tracée, la course nécessaire de la vis qui correspond alors à $y_{\max} - y_{\min}$ où $y_{\max}$ et $y_{\min}$ sont respectivement le maximum et le minimum de y lorsque $\theta \in [\theta_{\min}, \theta_{\max}]$.

Modèle inverse et résolution numérique

On veut savoir quelle sera la précision de positionnement de l'empennage sachant que le positionnement avec le système écrou et la commande associée se fait à $\Delta y = 0.4$ mm près. Cela signifie donc que, pour une position de référence $y_{\text{ref}} = 140$ mm par exemple, y sera compris dans l'intervalle $[y_+ ; y_-]$ où $y_+ = y_{\text{ref}} + \frac{\Delta y}{2}$ et $y_+ = y_{\text{ref}} - \frac{\Delta y}{2}$ et donc que θ sera compris dans l'intervalle $[\theta_+ ; \theta_-]$ où θ_+ est l'angle θ obtenu lorsque $y = y_-$ et θ_- est l'angle θ obtenu lorsque $y = y_+$ (car courbe décroissante).

Il faut donc résoudre, pour une longueur Y donnée, une équation du type $g(\theta) = 0$.

Question 5. Ecrire cette équation $g(\theta) = 0$.

Résolution avec la méthode de dichotomie

Question 6. Compléter le code ci-dessous afin de définir la fonction g.

Entrée [4]:
```
Y = 140.
def g(t): ## t représente la variable theta en radians !
    return (L2 + L3*np.cos(t))**2 + (L1 - L3*np.sin(t))**2 - Y**2
```

Question 7. Compléter le code ci-dessous afin de définir la fonction `dicho(a,b,g,eps)` qui résout l'équation $g(\theta)$ et qui prend en argument :

- `a` et `b` les bornes "à gauche" et "à droite" de l'intervalle de recherche de la solution ;
- `g` la fonction associée et
- `eps` l'erreur admise sur la solution.

Entrée [5]:
```
## Résolution par méthode de dichotomie
def dicho(a,b,g,eps):
    while abs(b-a)>eps:
        m = (a+b)/2
        if g(m)*g(a)<0:
            b = m
        else:
            a = m
    return m

print((180/np.pi)*dicho(0.,np.pi/2,g,0.000001),'degrés')
```

```
37.77558803558349 degrés
```

Un problème ici est que la fonction g est à re-définir pour chaque valeur de Y donnée. On propose donc de modifier le code précédent afin de créer une fonction `resol_dicho(a,b,Y,eps)` où cette fois-ci la fonction g, alors renommée gY sera re-définie avant la résolution.

Question 8. Compléter le code ci-dessous afin de prendre en compte cette modification.

Entrée [6]:
```
def resol_dicho(a,b,Y,eps):
    def gY(t):
        return (L2 + L3*np.cos(t))**2 + (L1 - L3*np.sin(t))**2 - Y**2

    return(dicho(a,b,gY,eps))

#exemple pour yref = 180. mm
print((180/np.pi)*resol_dicho(0.,np.pi/2,140.,0.000001),'degrés')
```

```
37.77558803558349 degrés
```

Question 9. Déterminer les instructions nécessaires pour déterminer l'intervalle $[\theta_+; \theta_-]$ et en déduire la précision sur l'angle θ.

Entrée [7]:
```
Y = 140 #mm
dy = 0.4 #mm
Ym = Y - dy
Yp = Y + dy

Tp = (180/np.pi)*resol_dicho(0.,np.pi/2,Ym,0.000001)
Tm= (180/np.pi)*resol_dicho(0.,np.pi/2,Yp,0.000001)

dt = Tp - Tm
print('intervalle = [',Tm,',',Tp,'] en degrés')
print('précision =',dt,'degrés')
```

```
intervalle = [ 37.41930484771729 , 38.13075542449951 ] en degrés
précision = 0.7114505767822195 degrés
```

Résolution avec la méthode de Newton

On cherche toujours à résoudre l'équation $g(\theta) = 0$ mais cette fois-ci avec la méthode de Newton. La fonction à résoudre reste toujours la même que celle de la question 6.

Question 10. Calculer $\frac{\mathrm{d}g}{\mathrm{d}\theta}(\theta)$.

Question 11. Compléter le code ci-dessous afin de définir la fonction `gprim(t)` qui renvoie la dérivée de g pour un angle `t` correspondant à θ.

Entrée [8]:
```
def gprim(t): ## t représente la variable theta en radians !
    return -2*L3*np.sin(t)*( L2 + L3*np.cos(t) ) - 2*L3*np.cos(t)*( L1 - L3*np.sin(t) )
```

Question 12. Compléter le code ci-dessous afin de définir la fonction `newton(t0,g,eps)` qui résout l'équation $g(\theta)$ et qui prend en argument :

- `t0` la valeur initiale de recherche associée à la méthode de Newton ;
- `g` la fonction associée et
- `eps` l'erreur admise sur la solution.

Entrée [9]:
```
def newton(t0,g,eps):
    t = t0 - g(t0)/gprim(t0)
    while abs(g(t))>eps:
        t = t - g(t)/gprim(t)
    return t
```

Question 13. Compléter le code ci-dessous en vous inspirant de la question 8, afin de définir la fonction `resol_newton(t0,Y,eps)` qui résout l'équation $g(\theta)$ et qui prend en argument :

- `t0` la valeur initiale de recherche associée à la méthode de Newton ;
- `Y` la valeur associée de Y à la fonction g et
- `eps` l'erreur admise sur la solution.

Entrée [10]:
```
def resol_newton(t0,Y,eps):
    def gY(t):
        return (L2 + L3*np.cos(t))**2 + (L2 - L3*np.sin(t))**2 - Y**2

    return(newton(t0,gY,eps))

#exemple pour yref = 180. mm
print((180/np.pi)*resol_newton(np.pi/4,140.,0.000001),'degrés')
```

```
36.198691785424764 degrés
```

Question 14. Déterminer les instructions nécessaires, en utilisant la fonction `resol_newton` , pour déterminer l'intervalle $[\theta_+ ; \theta_-]$ et en déduire la précision sur l'angle θ.

Entrée [11]:
```
Y = 140 #mm
dy = 0.4 #mm
Ym = Y - dy
Yp = Y + dy

Tp = (180/np.pi)*resol_newton(np.pi/4,Ym,0.000001)
Tm= (180/np.pi)*resol_newton(np.pi/4,Yp,0.000001)

dt = Tp - Tm
print('intervalle = [',Tm,',',Tp,'] en degrés')
print('précision =',dt,'degrés')
```

```
intervalle = [ 35.81208176558001 , 36.58379699343627 ] en degrés
précision = 0.7717152278562622 degrés
```

Entrée [ ]: