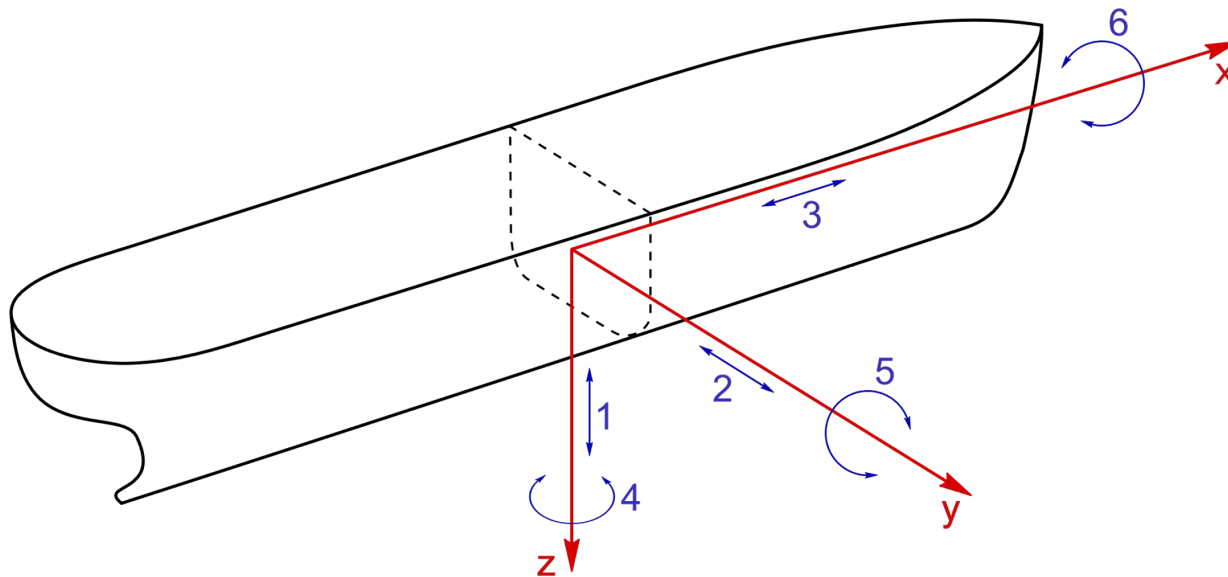


Modélisation du roulis d'un bateau et de systèmes de stabilisation



Introduction

Sommaire

1^{er} modèle : La liaison pivot

- Sans stabilisation
- Masses mobiles
- Cuve antiroulis

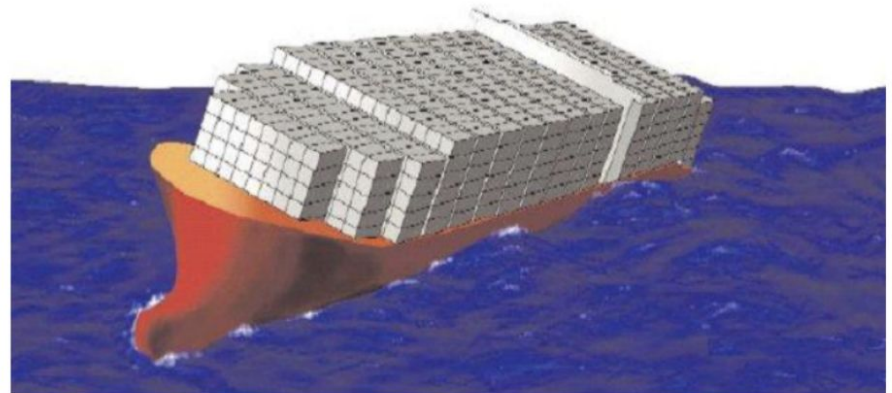
2^{ème} modèle : L'effort de l'eau

- Exemple
- Influence de la forme de la coque

Conclusion et annexes

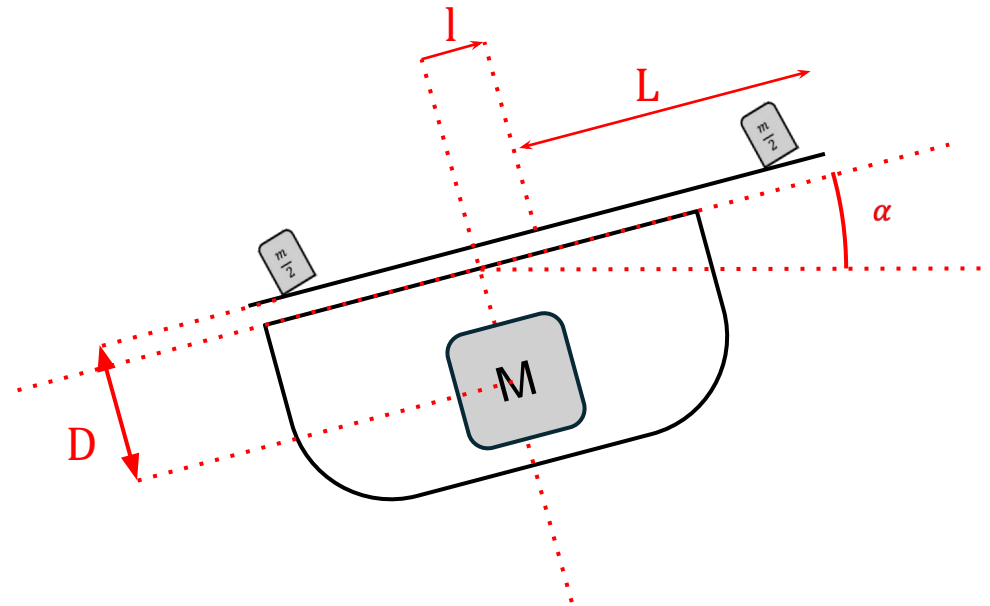
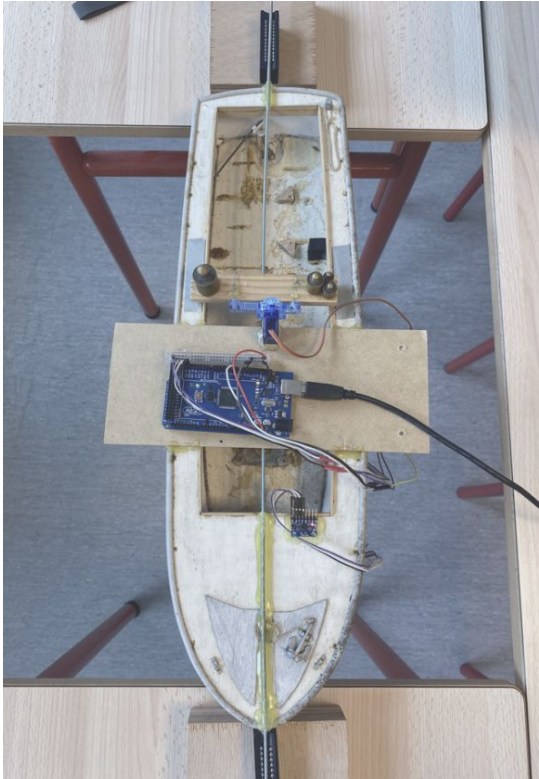
Problématiques :

- Comment limiter le roulis d'un bateau ?
- Quelle est l'efficacité des différents systèmes ?



Source : afcan.org

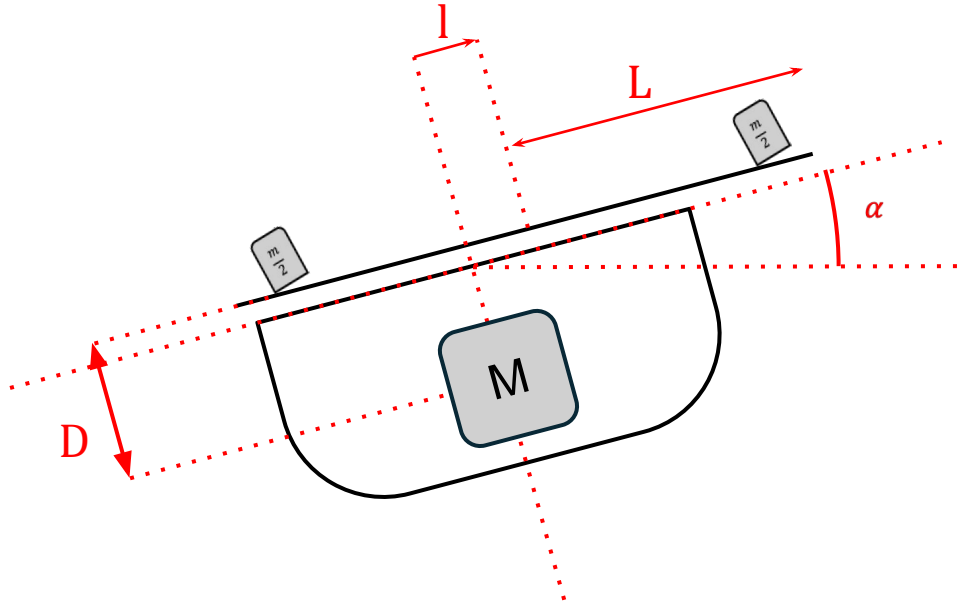
1^{er} modèle : le pendule



Matériel :

- Coque bateau modélisme
- Capteur angulaire (accéléromètre)
- Axe de rotation
- Arduino (traitement des données)
- Masses + moteur

1^{ère} expérience : pas de stabilisation



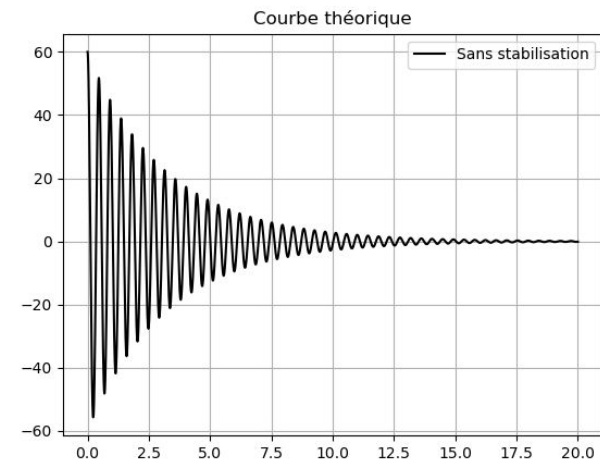
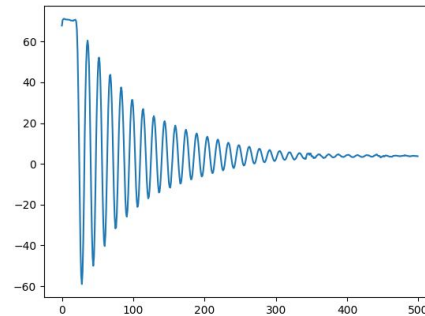
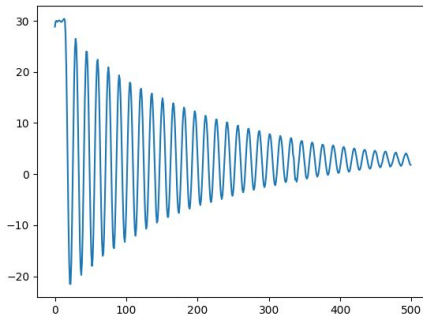
$$l=0$$

$$J\ddot{\alpha} = -f\dot{\alpha} + Mdg \sin \alpha$$

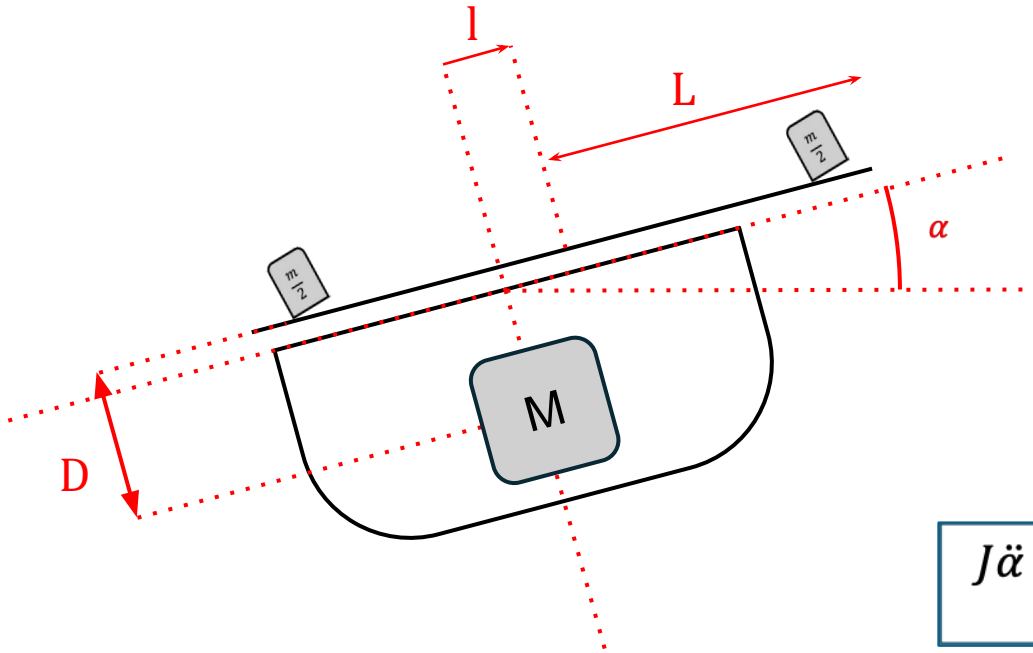
$$J = mL^2 + Md^2$$

Objectifs :

- Valider le modèle
- Déterminer le coefficient de frottement f



2^{ème} expérience : masses mobiles



$$J\ddot{\alpha} = -f\dot{\alpha} - ml g \cos \alpha + Mdg \sin \alpha$$
$$J = mL^2 + Md^2$$

Stratégie :

$$l = R\alpha$$

Moteur commandé par Arduino

Hypothèses :

- L'accélération des masses est négligeable seul leur position est considérée
- Moteur infiniment rapide

2^{ème} expérience : masses mobiles

Résultats :

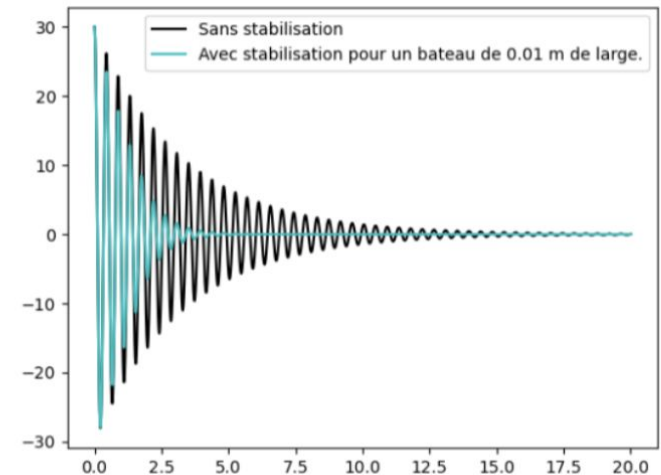
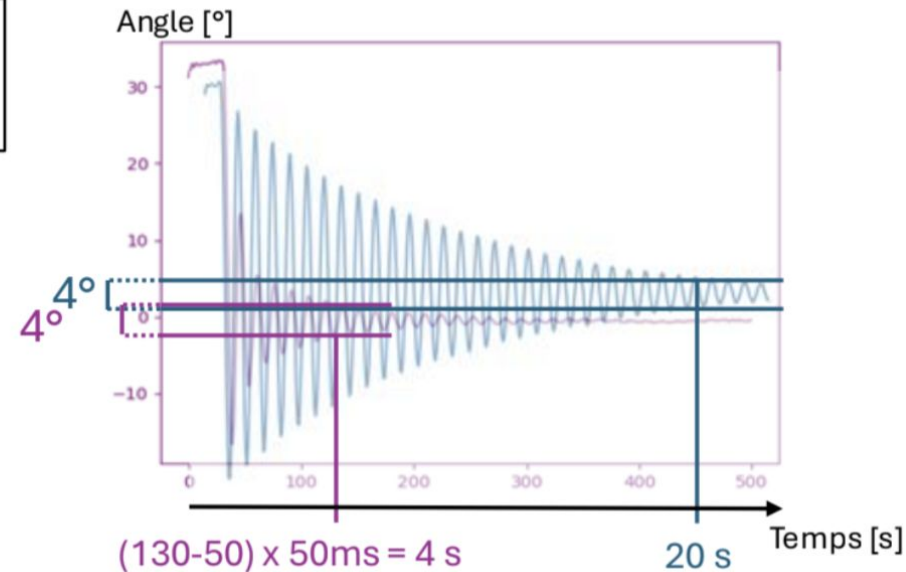
Angle de départ	Temps avec moteur	Temps sans moteur
30°	4 s	20 s
50°	6,5 s	20 s
70°	7 s	15 s

Pour 30°, la simulation donne un temps de stabilisation sans le système est de 19,2s et de 5,27s avec.

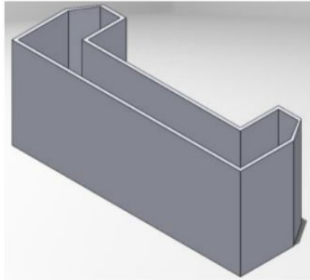
$K = 0,002 \text{ m/rad}$ (coefficient de stratégie)

Difficulté :

Moteur lent et déphasé d'une demi-période donc $-K$ pour corriger le décalage



3^{ème} expérience : cuve antiroulis



Cuve + eau

Arduino

Capteur

Capteur angulaire

Avantages :

Système passif (pas d'alimentation ni de panne)

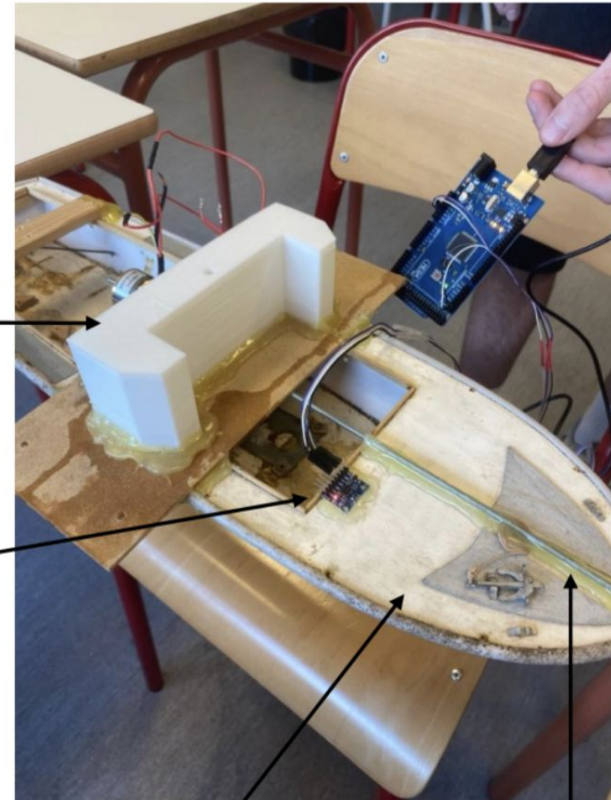
Peu coûteux

Pas d'entretien

Inconvénients :

Poids

Encombrement

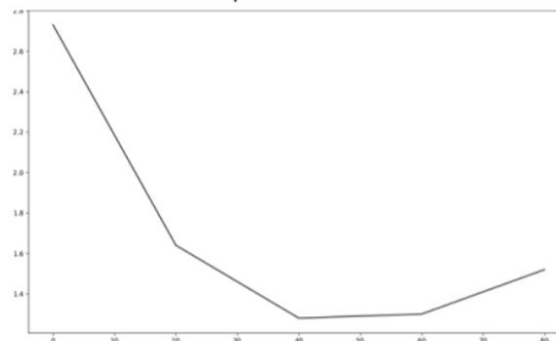
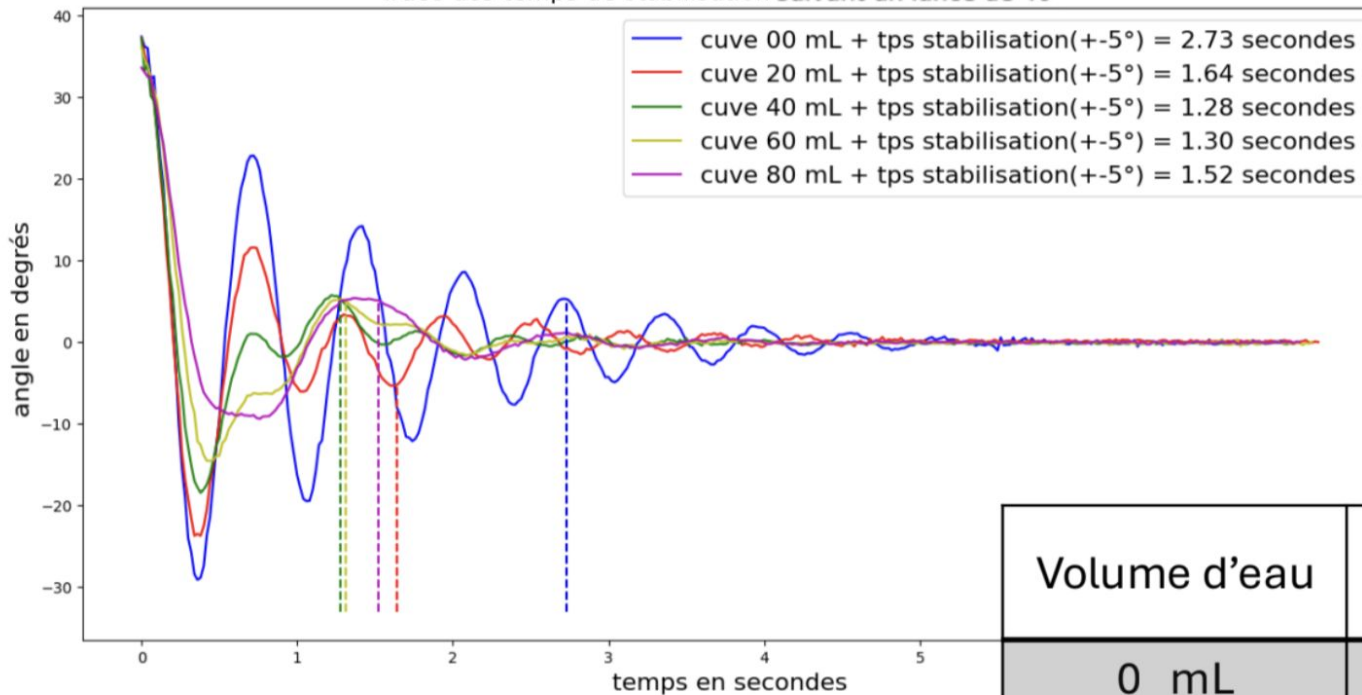


Coque bateau modélisme

Axe de rotation

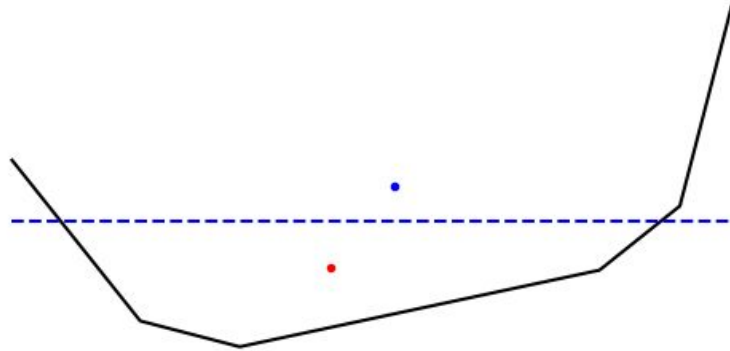
3^{ème} expérience : cuve antirollis

Tracé des temps de stabilisation suivant un lancé de 40°



Volume d'eau	Temps de stabilisation
0 mL	2,73 s
20 mL	1,64 s
40 mL	1,28 s
60 mL	1,30 s
80 mL	1,52 s

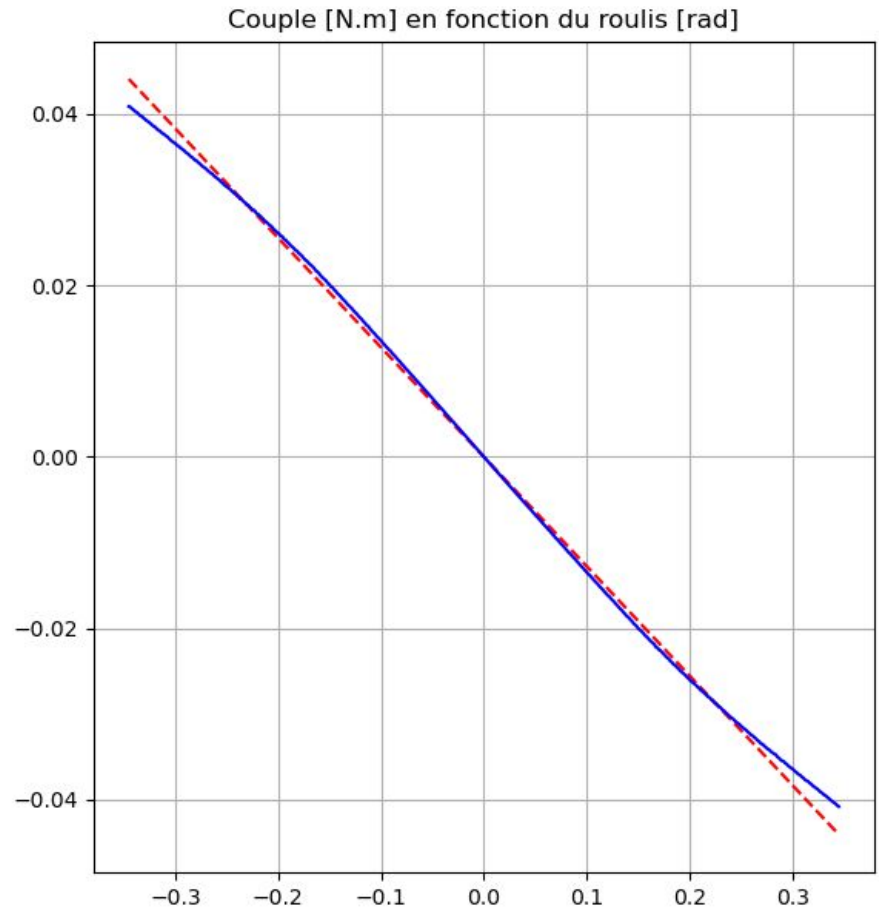
2^{ème} modèle : effort de l'eau sur la coque



- Centre de gravité (de rotation)
- - Centre de carène
- Ligne de flottaison

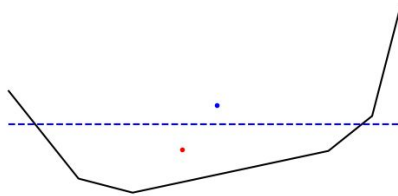
Hypothèses :

- Forme simple de translation
- Masse homogène

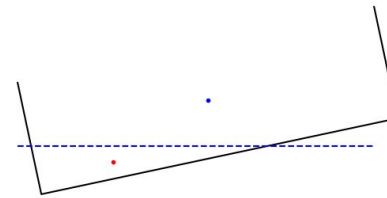
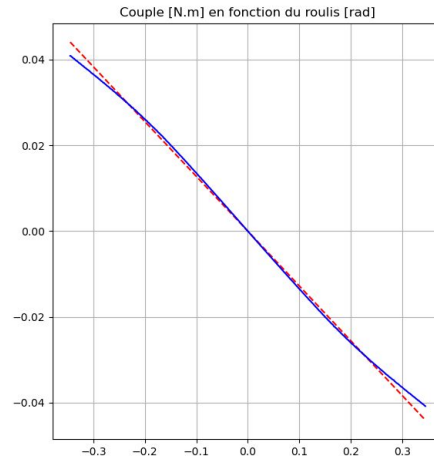


Approximation : $K = -0.1276$
N.m/rad

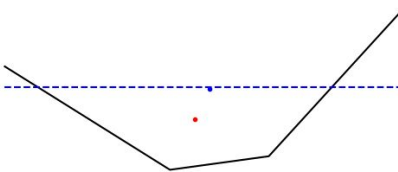
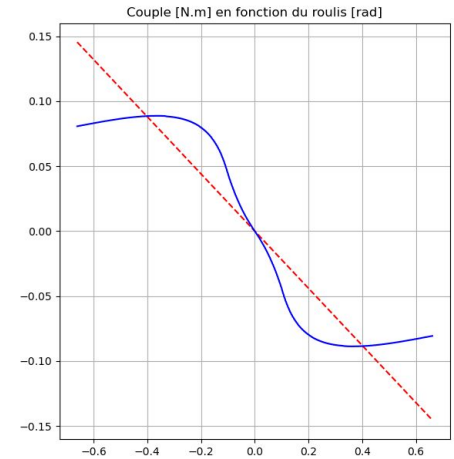
2^{ème} modèle : effort de l'eau sur la coque



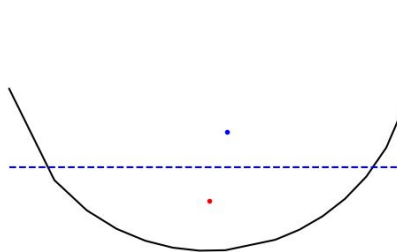
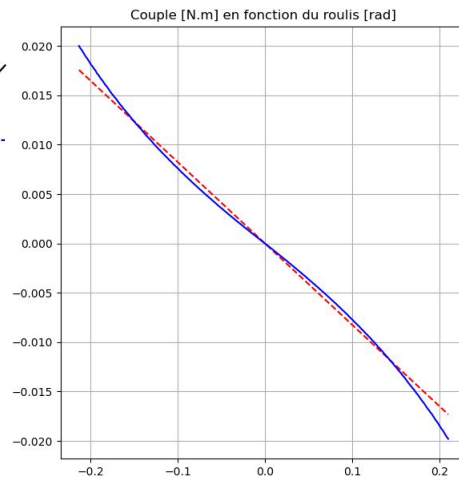
$$K = -0.1276 \text{ N.m/rad}$$



$$K = -0.2200 \text{ N.m/rad}$$



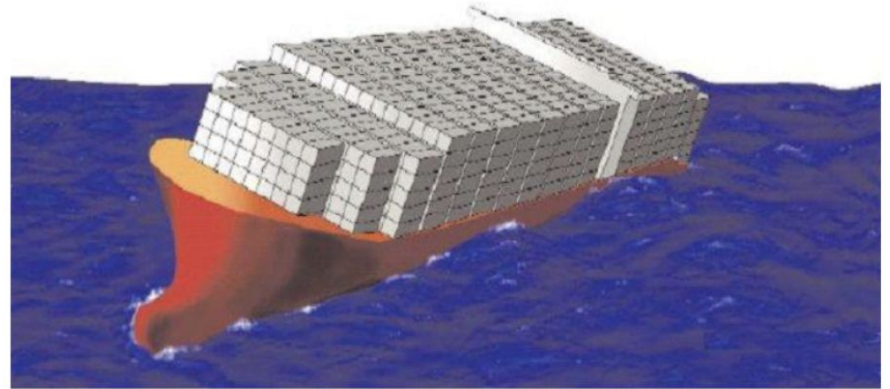
$$K = -0.0825 \text{ N.m/rad}$$



$$K = -0.0631 \text{ N.m/rad}$$



Conclusions



Source : afcan.org

- Chaque système divise par 2 voir 3 le temps de stabilisation
- La forme de la coque définit grandement le comportement et la stabilité du bateau
- Le choix du/des système de stabilisation est lié à la taille du navire
La forme de la coque est principalement liée aux conditions de navigation du bateau
Combiner plusieurs solution est le moyen de plus efficace et le plus fiable
- Le principal danger est le phénomène de résonance qu'il faut absolument éviter

Annexes

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from scipy.integrate import odeint
4
5 #Données du modèle
6 M=0.7 #700g :la masse du bateau vide
7 m=0.1 #100g :les deux masses de notre système
8 D=0.03 #3cm :distance entre le centre de gravité et l'axe de rotation
9 L=0.06 #6cm :distance entre les masses et l'axe de rotation
10 g=9.81 #[N.kg] accélération gravitationnelle
11 F=0.0006#[N.m.s]coefficient de frottement
12 K=0.002 #[m/rad] coefficient de stratégie
13 J=M*D**2+m*L**2 #moment d'inertie du bateau
14
15 #Conditions initiales
16 v0=0 #[rad/s]
17 p0=30*np.pi/180 #[rad]
18 CI = np.array( [v0, p0] )
19 Lbateau= 0.01 #10cm
20
21 #Définition de la translation des masses
22 def L(v):
23     if abs(K*v)<(Lbateau/2):
24         return K*v
25     else:
26         return (K*v>=0)*(Lbateau/2)
27
28 #Dérivée de la position et de la vitesse angulaire avec l'équation différentielle
29 def derivee(y,t):
30     p=y[1]
31     v=y[0]
32     return np.array([(F*v+m*g*L(v)*np.cos(p)+M*g*D*np.sin(p))/(-J) ,v])
33 deb, fin= 0, 20
34 T=np.linspace(deb,fin,10000)
35
36 #Solution avec le système actif
37 S_avec=odeint(derivee, CI, T)*180/np.pi
38 #Solution sans le système actif
39 K=0
40 S_sans=odeint(derivee, CI, T)*180/np.pi
41
42 #Détermination du temps de stabilisation
43 tsans=0
44 for i in range(len(S_sans[:,1])):
45     if S_sans[:, 1][i]<=-5*np.pi/180 or S_sans[:, 1][i]>5*np.pi/180:
46         tsans=T[i]
47 print('Le temps de stabilisation sans le système est de', tsans, 'secondes.')
48 print('')
49 tavec=0
50 for i in range(len(S_avec[:,1])):
51     if S_avec[:, 1][i]<=-5*np.pi/180 or S_avec[:, 1][i]>5*np.pi/180:
52         tavec=T[i]
53 print('Le temps de stabilisation avec le système est de', tavec, 'secondes.')
54 #Tracés des solutions
55 plt.plot(T,S_sans[:,1], 'k', label="Sans stabilisation")
56 plt.plot(T,S_avec[:,1], 'c', label="Avec stabilisation pour un bateau de "+str(Lbateau)+" m de large.")
57 plt.legend(loc='best')
58 plt.show()

```

```

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.widgets import Slider

#Valeurs numériques
g=9.81 #Accélération gravitationnelle
p=1000 #Masse volumique de l'eau (kg.m-3)
M=0.2 #Masse du bateau (kg)
L=0.1 #Longueur du bateau (m)

#Forme de coque (partie gauche)
Xg=np.array([-0.08, -0.06, -0.04])
Yg=np.array([0, -0.04, -0.05])

#Symétrie
Xbis=list(Xg)+list(-Xg[::-1])
Ybis=list(Yg)+list(Yg[::-1])

#Origine du repère placé au centre de gravité
def baryc(nnX,nnY):
    nX=list(np.copy(nnX))
    nY=list(np.copy(nnY))
    bx,by,bA=0,0,0
    nX.append(nX[0])
    nY.append(nY[0])
    for i in range(len(nX)-1):
        bx+=(nX[i]+nX[i+1])*(nX[i]*nY[i+1]-nX[i+1]*nY[i])
        by+=(nY[i]+nY[i+1])*(nX[i]*nY[i+1]-nX[i+1]*nY[i])
        bA+=(nX[i]*nY[i+1]-nX[i+1]*nY[i])/2
    return bx/6/bA,by/6/bA

bar=baryc(Xbis,Ybis)
Xbis=[k-bar[0] for k in Xbis]
Ybis=[k-bar[1] for k in Ybis]

#Rotation du bateau
def Rot(alp):
    R=np.array([[np.cos(alp), -np.sin(alp)],
                [np.sin(alp), np.cos(alp)]])
    X,Y=[],[]
    for i in range(len(Xbis)):
        Tem=R@np.array([Xbis[i],
                        Ybis[i]])
        X.append(Tem[0])
        Y.append(Tem[1])
    return X,Y

```

```

#Ligne de flottaison
pas=0.0001 #Montée de l'eau à chaque itération (m)
A=M/(p*L) #Aire de la section sous l'eau

def decoupe(Xd,Yd,pasd):
    nouvX=[]
    nouvY=[]
    for i in range(len(Yd)-1):
        c=Yd[i]
        nouvY.append(c)
        nouvX.append(Xd[i])
        if Yd[i+1]<Yd[i]:
            pasX=(Xd[i+1]-Xd[i])/(Yd[i]-Yd[i+1])*pasd
            while c-pasd>Yd[i+1]:
                c-=pasd
                nouvY.append(c)
                nouvX.append(nouvX[-1]+pasX)
        elif Yd[i+1]>Yd[i]:
            pasX=(Xd[i+1]-Xd[i])/(Yd[i+1]-Yd[i])*pasd
            while c+pasd<Yd[i+1]:
                c+=pasd
                nouvY.append(c)
                nouvX.append(nouvX[-1]+pasX)
        else:
            nouvY.append(c)
            nouvX.append(Xd[i])
            nouvY.append(Yd[-1])
            nouvX.append(Xd[-1])
    return nouvX,nouvY

def niveau(Xdo,Ydo,Ad,pasd):
    Xd,Yd=decoupe(Xdo,Ydo,pasd)
    At=0
    niv=min(Yd)
    iG=len(Xd)//2
    iD=iG+1
    plt.title('')
    chav=0
    while At<Ad:
        niv+=pasd
        if Yd[0]<niv or Yd[-1]<niv:
            plt.title('Chavirement !')
            chav=1
            break
        At+=(Xd[iD]-Xd[iG])*pasd
        At+=(1/2)*(Xd[iG]-Xd[iG-1])*pasd
        At+=(1/2)*(Xd[iD+1]-Xd[iD])*pasd
        iG-=1
        iD+=1
    return niv,chav

```

#Barycentre sous une ligne de flotaison

```
def barycentre(Xd,Yd,ni):
    nX,nY=[],[]
    for i in range(len(Xd)-1):
        if Yd[i]<=ni:
            nX.append(Xd[i])
            nY.append(Yd[i])
        elif Yd[i]>ni and Yd[i+1]<ni:
            nY.append(ni)
            nX.append((Yd[i]-ni)/(Yd[i]-Yd[i+1])*(Xd[i+1]-Xd[i])+Xd[i])
        if Yd[i]<ni and Yd[i+1]>ni:
            nY.append(ni)
            nX.append((ni-Yd[i])/(Yd[i+1]-Yd[i])*(Xd[i+1]-Xd[i])+Xd[i])
    if Yd[-1]<ni:
        nY.append(ni)
        nX.append((ni-Yd[-2])/(Yd[-1]-Yd[-2])*(Xd[-1]-Xd[-2])+Xd[-2])
    bx,by,bA=0,0,0
    nX.append(nX[0])
    nY.append(nY[0])
    for i in range(len(nX)-1):
        bx+=(nX[i]+nX[i+1])*(nX[i]*nY[i+1]-nX[i+1]*nY[i])
        by+=(nY[i]+nY[i+1])*(nX[i]*nY[i+1]-nX[i+1]*nY[i])
        bA+=(nX[i]*nY[i+1]-nX[i+1]*nY[i])/2
    return bx/6/bA,by/6/bA
```

#Calcul du moment de la poussée d'Archimède autour de l'axe de rotation [N.m]

```
def moment(Xd,Yd,nivd):
    return M*g*(barycentre(Xd,Yd,nivd)[0])
```

#Affichage

```
def update(val):
    an=angle.val
    X,Y=Rot(an)
    niv=niveau(X,Y,A,pas)[0]
    Xb,Yb=barycentre(X,Y,niv)
    mome=moment(X,Y,niv)
    bate.set_xdata(X)
    bate.set_ydata(Y)
    nive.set_xdata([X[0],X[-1]])
    nive.set_ydata([niv,niv])
    bary.set_data(barycentre(X,Y,niv))
    vert.set_xdata([an,an])
    mom.set_xdata([an])
    mom.set_ydata([mome])
    fig.suptitle('Couple : '+str(mome)+' N.m')
    fig.canvas.draw_idle()
```

```
fig,(ax1,ax2) = plt.subplots(1,2)
plt.subplots_adjust(bottom=0.25)
```

```
X0,Y0=Rot(0)
niv0=niveau(X0,Y0,A,pas)[0]
```

```
ax1.plot([0],[0],'.b')
bate, = ax1.plot(X0,Y0,'k')
nive, = ax1.plot([X0[0],X0[-1]],[niv0,niv0], '--b')
bary, = ax1.plot(barycentre(X0,Y0,niv0)[0],barycentre(X0,Y0,niv0)[1],'.r')
ax1.axis('equal')
ax1.axis('off')
```

```
Xt=[]
Yt=[]
for a in np.linspace(-np.pi/4,np.pi/4,1000):
    Xd,Yd=Rot(a)
    nivd=niveau(Xd,Yd,A,pas)[0]
    if niveau(Xd,Yd,A,pas)[1]==0:
        Yt.append(moment(Xd, Yd, nivd))
        Xt.append(a)
```

#Couple [N.m] en fonction du roulis [rad]

```
moyb=0
for i in range(int(len(Xt)/2)):
    moyb+=Yt[i]
moyb=moyb/int(len(Xt)/2)
pente=moyb/Xt[int(len(Xt)/4)]
print('pente :',pente)
```

```
ax2.plot([min(Xt),max(Xt)],[pente*min(Xt),pente*max(Xt)], '--r')
ax2.plot(Xt,Yt,'b')
vert, = ax2.plot([0,0],[max(Yt),min(Yt)], '-k')
ax2.grid()
ax2.set_title('')
mom,=ax2.plot([0],[0], 'or')
ax2.set_title('Couple [N.m] en fonction du roulis [rad]')
```

```
ax_slider = plt.axes([0.2, 0.1, 0.65, 0.03])
angle=Slider(ax_slider, 'Angle de roulis [rad]:', -np.pi/4, np.pi/4, valinit=0)
angle.on_changed(update)
plt.show()
```