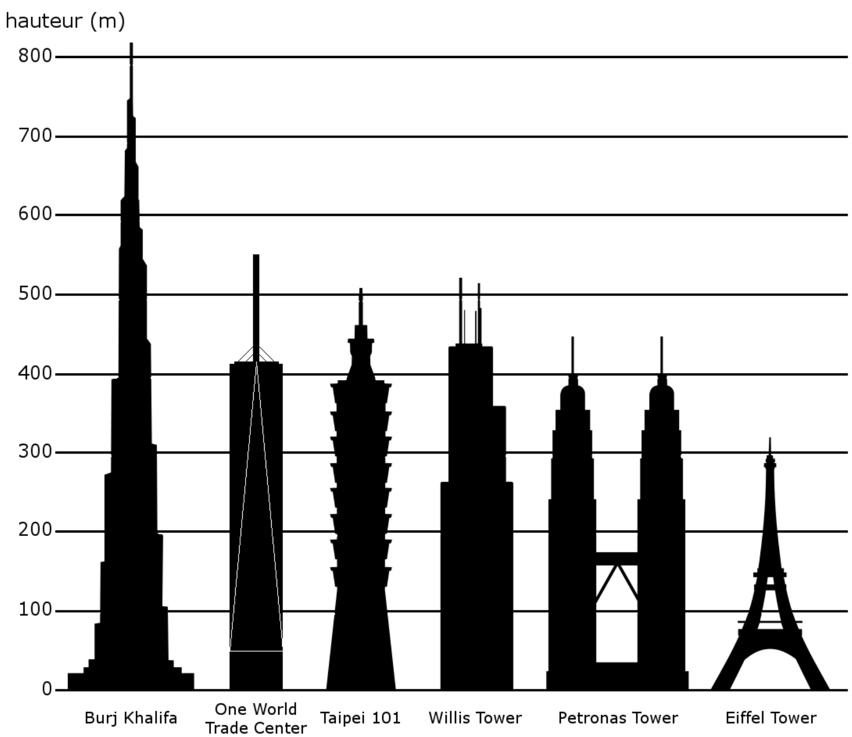
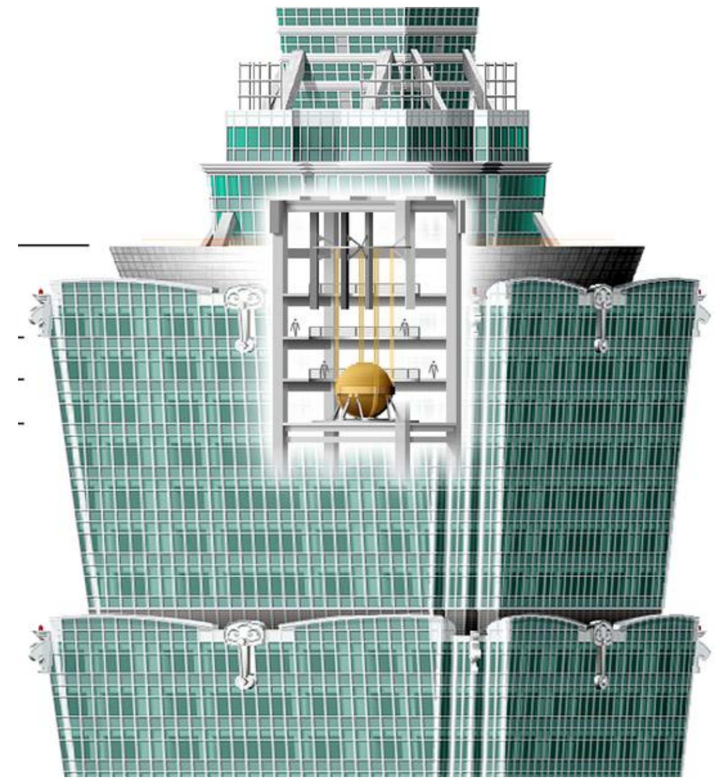
TIPE:
Réduction des oscillations de bâtiments par la mise en place d'amortisseur à masse accordée

ROUGER Nicodème
N° candidat: 30065

Amortisseurs à masse accordée:





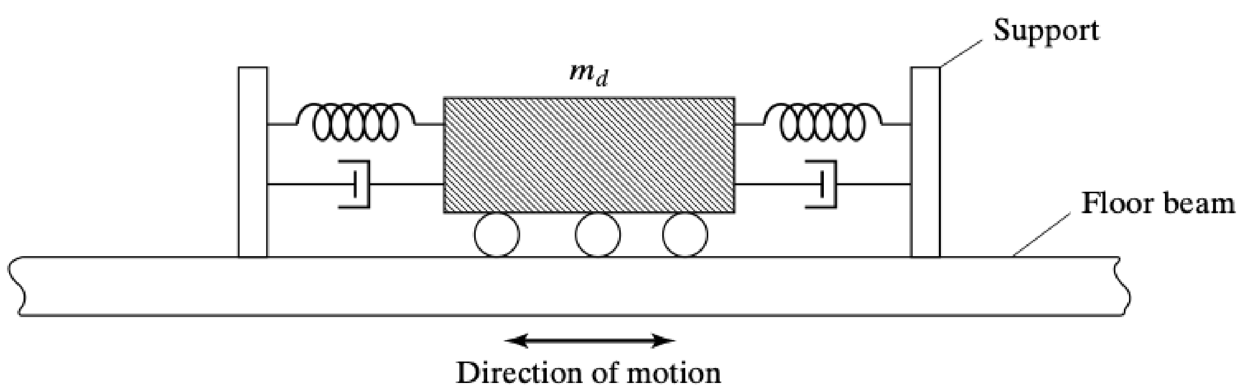


FIGURE 4.2: Schematic diagram of a translational tuned mass damper.

Problématique:

Quels sont les effets des amortisseurs à masse accordée? Comment prévoir leur action? Comment optimiser son effet avant de le mettre en œuvre pour garantir la résistance aux aléas naturels?

Plan:

1. Familiarisation avec le système:

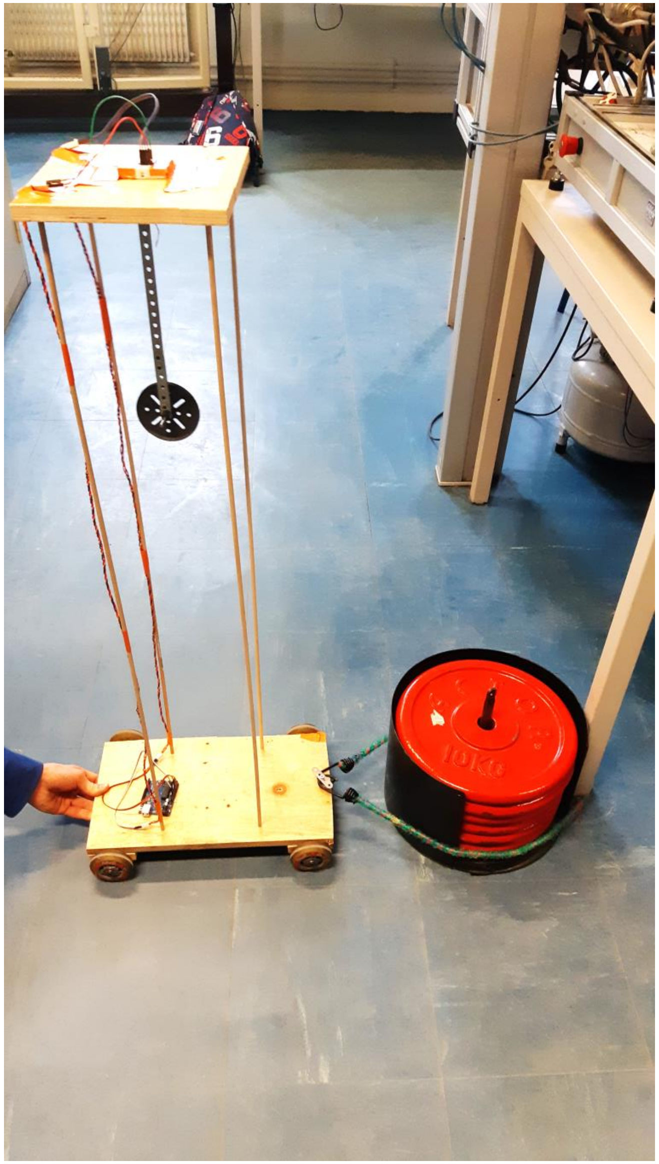
- Modélisation simple
- Conception d'une maquette

2. Mise en place d'un modèle:

- Choix d'un modèle approprié
- Mise en équation puis résolution numérique

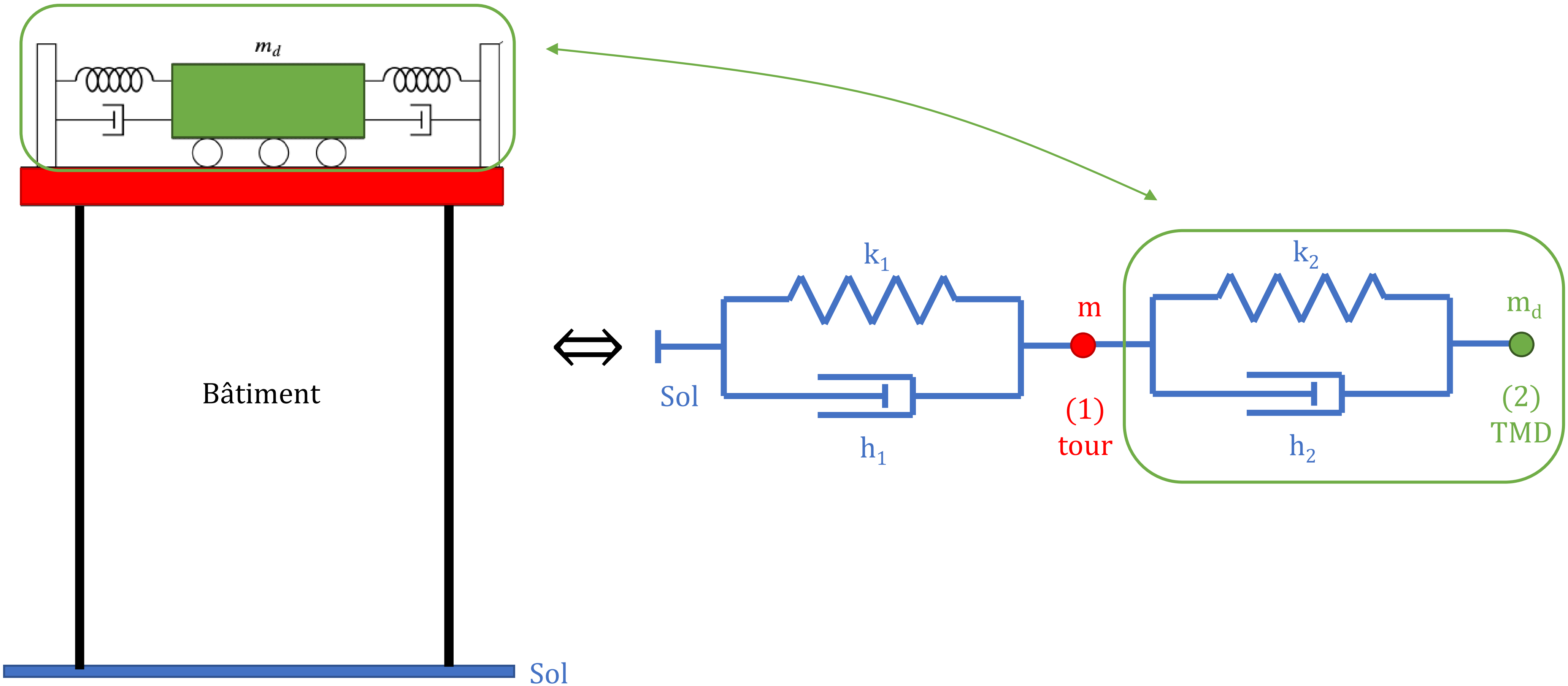
3. Optimisation du système avant sa réalisation

- Etude informatique
- Mise en œuvre sur la maquette



Première approche: étude d’une modélisation à une dimension

Sans pendule: seulement amortisseurs et ressorts



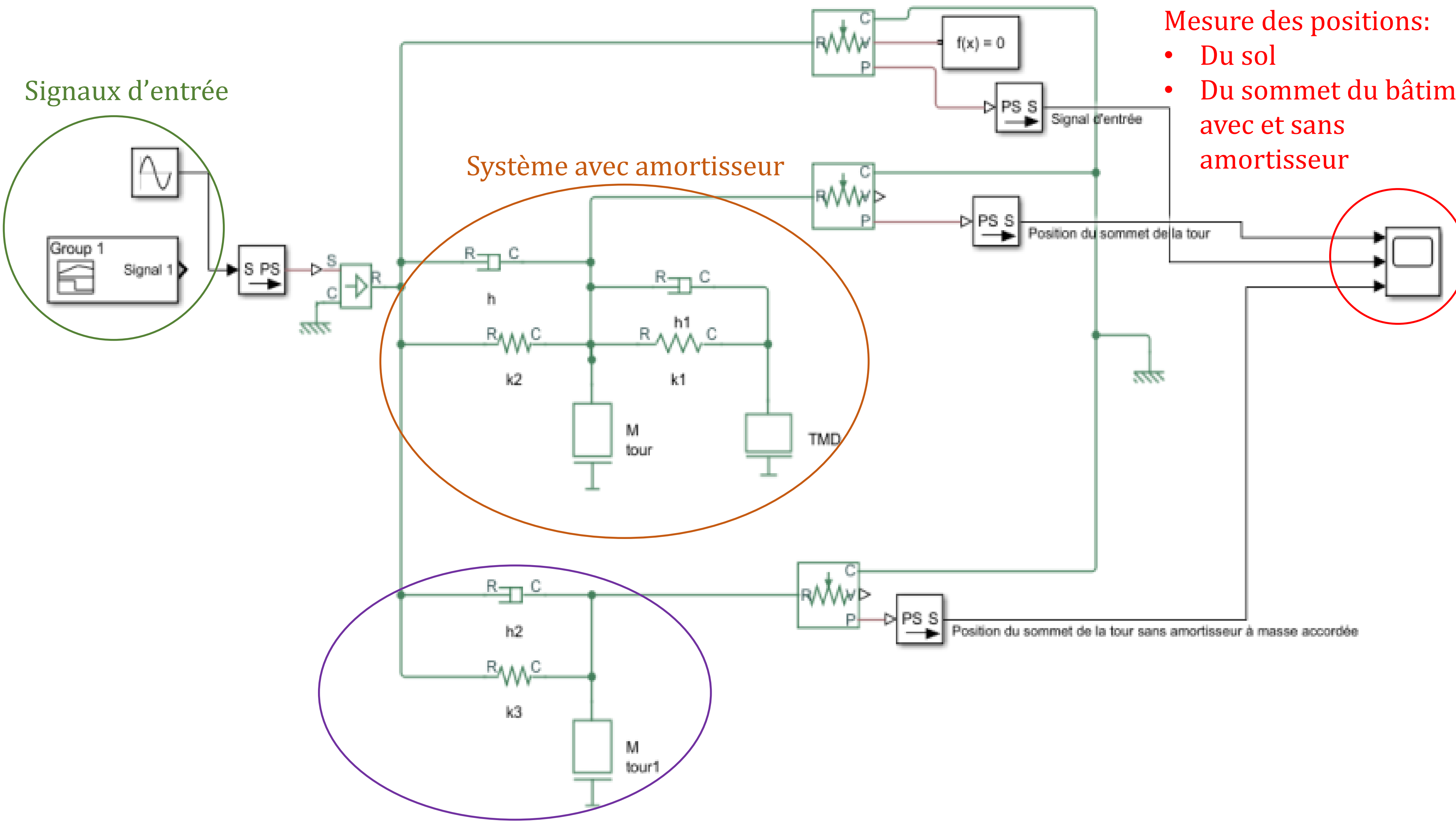
Simulation Matlab:

Signaux d'entrée

Système avec amortisseur

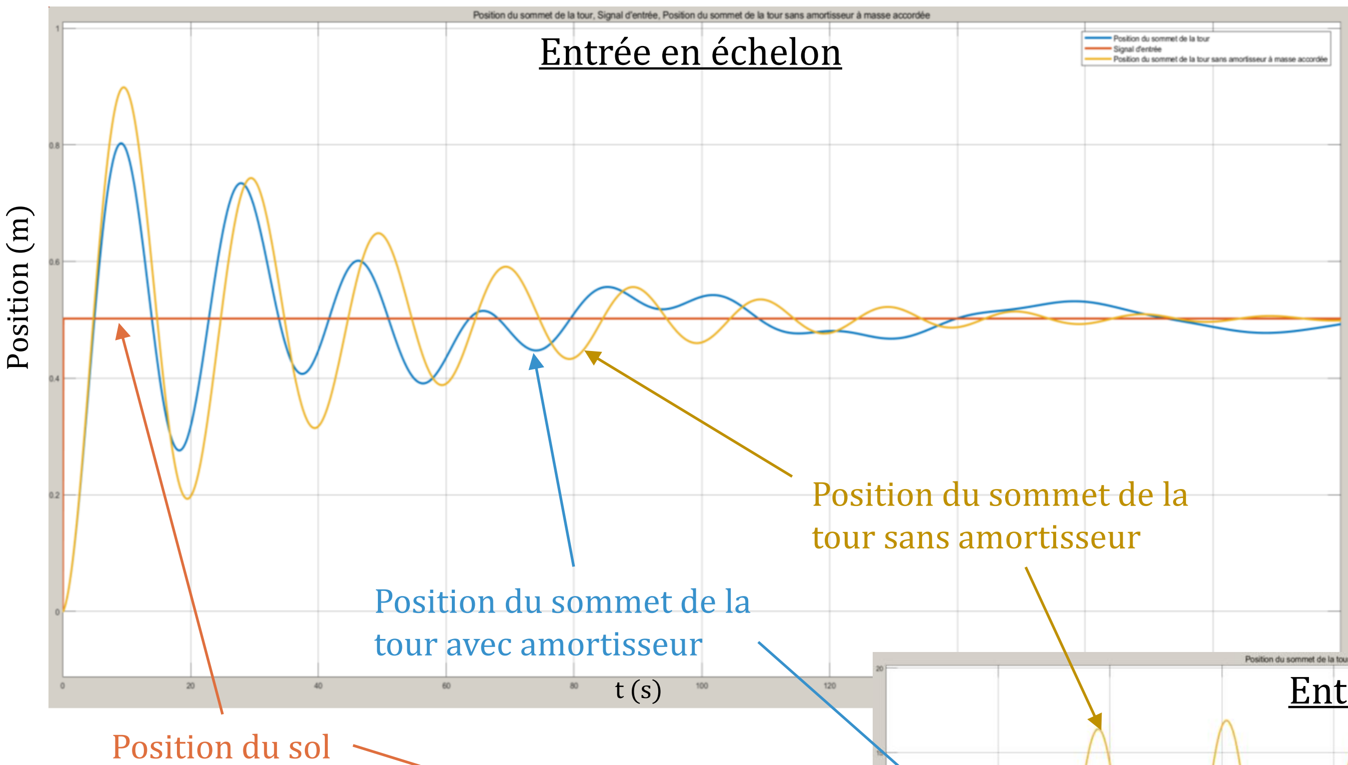
Mesure des positions:

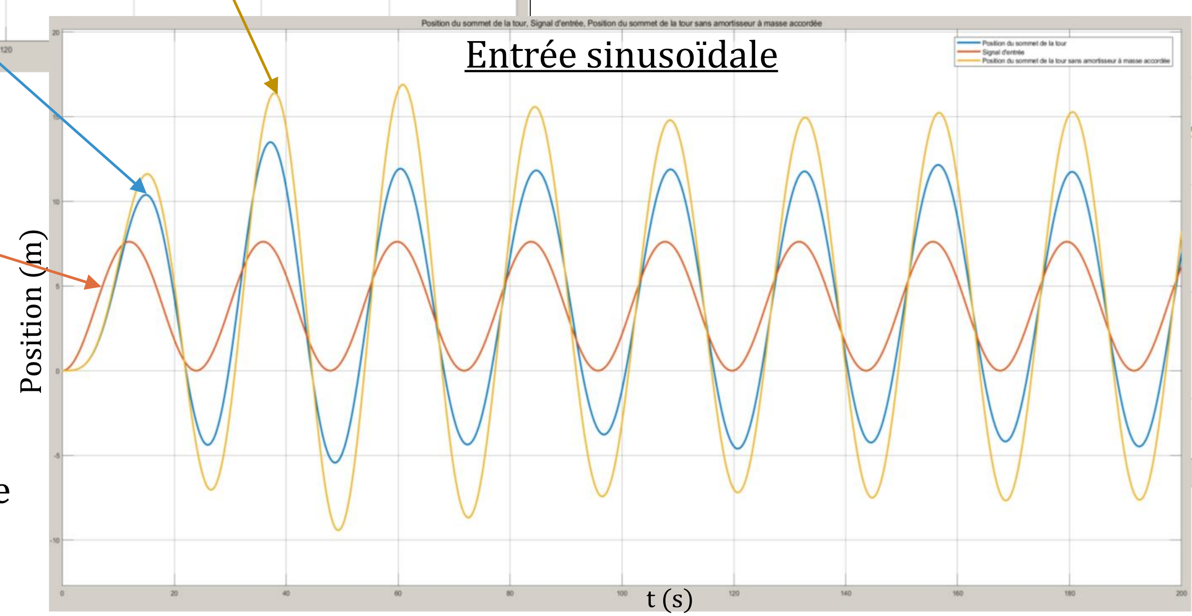
- Du sol
- Du sommet du bâtiment avec et sans amortisseur



Système sans amortisseur

Simulation Matlab:

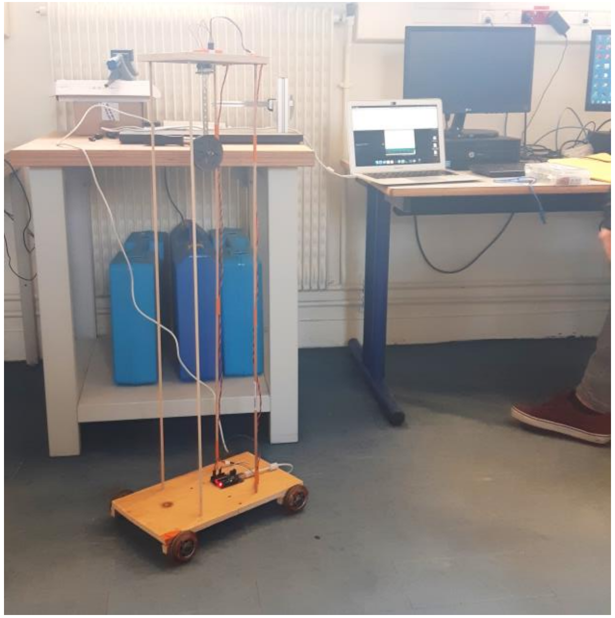




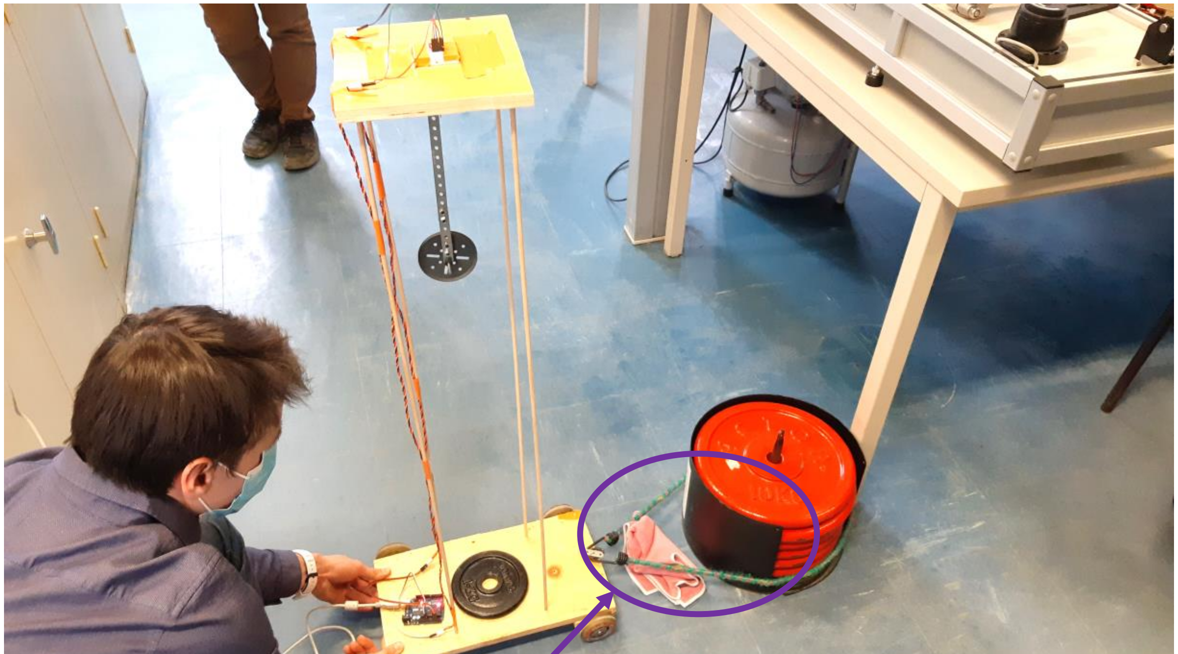
Effets visibles de l'amortisseur à masse accordée

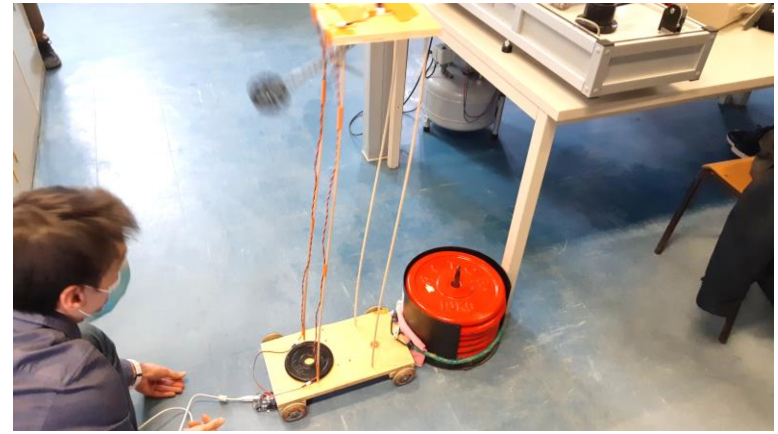
Construction de la maquette:

Sur le modèle du TMD de la tour Taipei 101



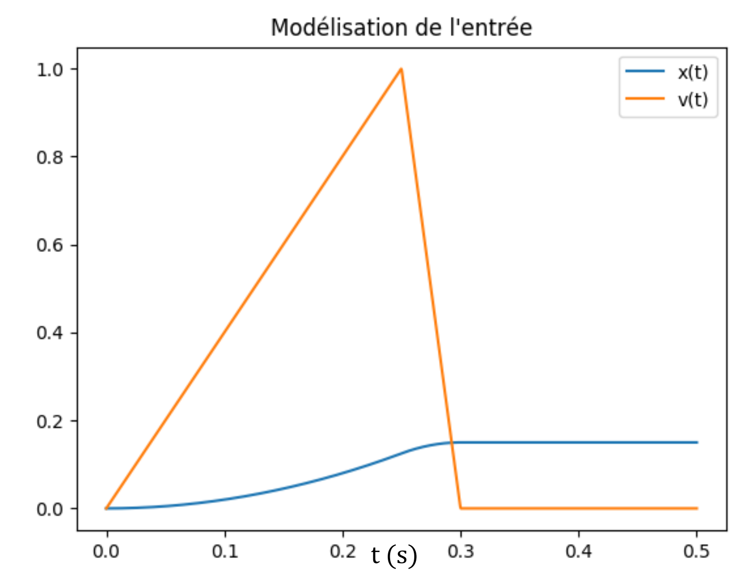
Perturbation extérieure en choc:





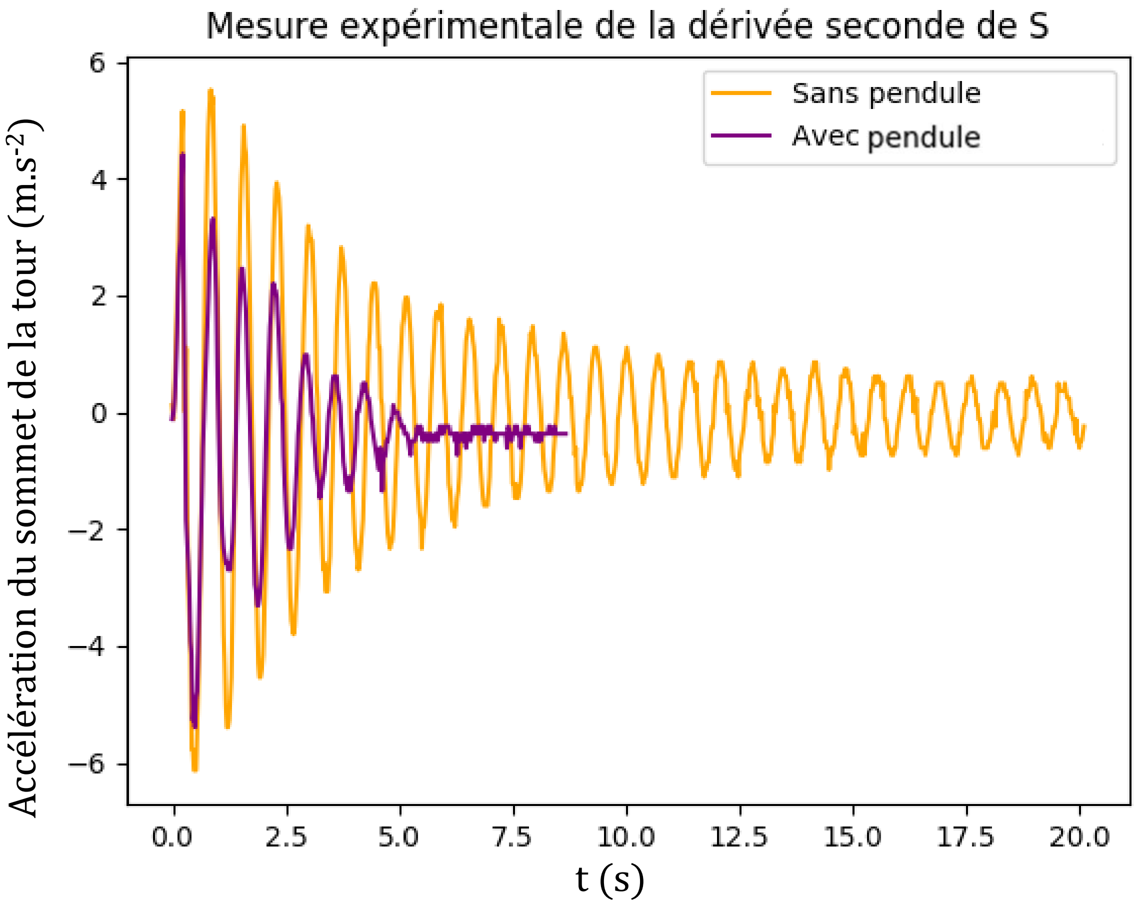
Ressort de traction pour
simuler une perturbation

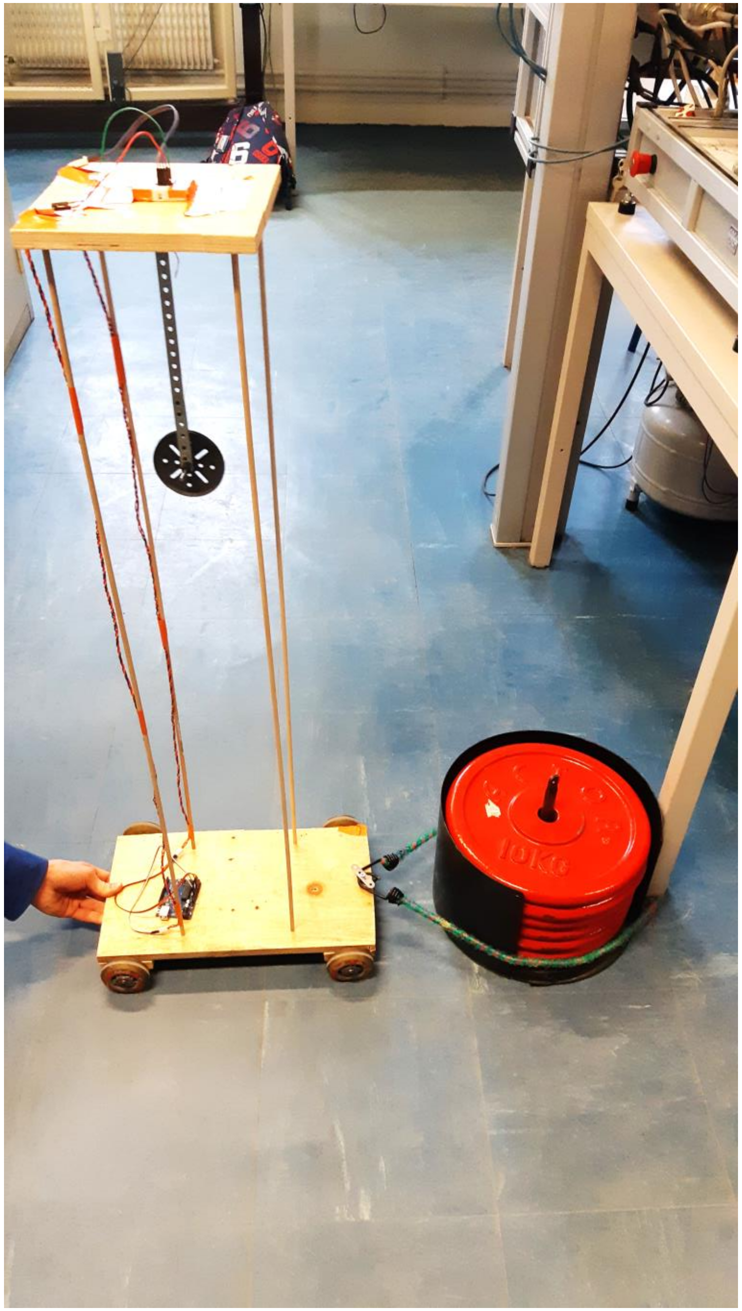
Entrée en triangle de vitesse



Mesures et modélisation en annexe D; script Python en annexe I

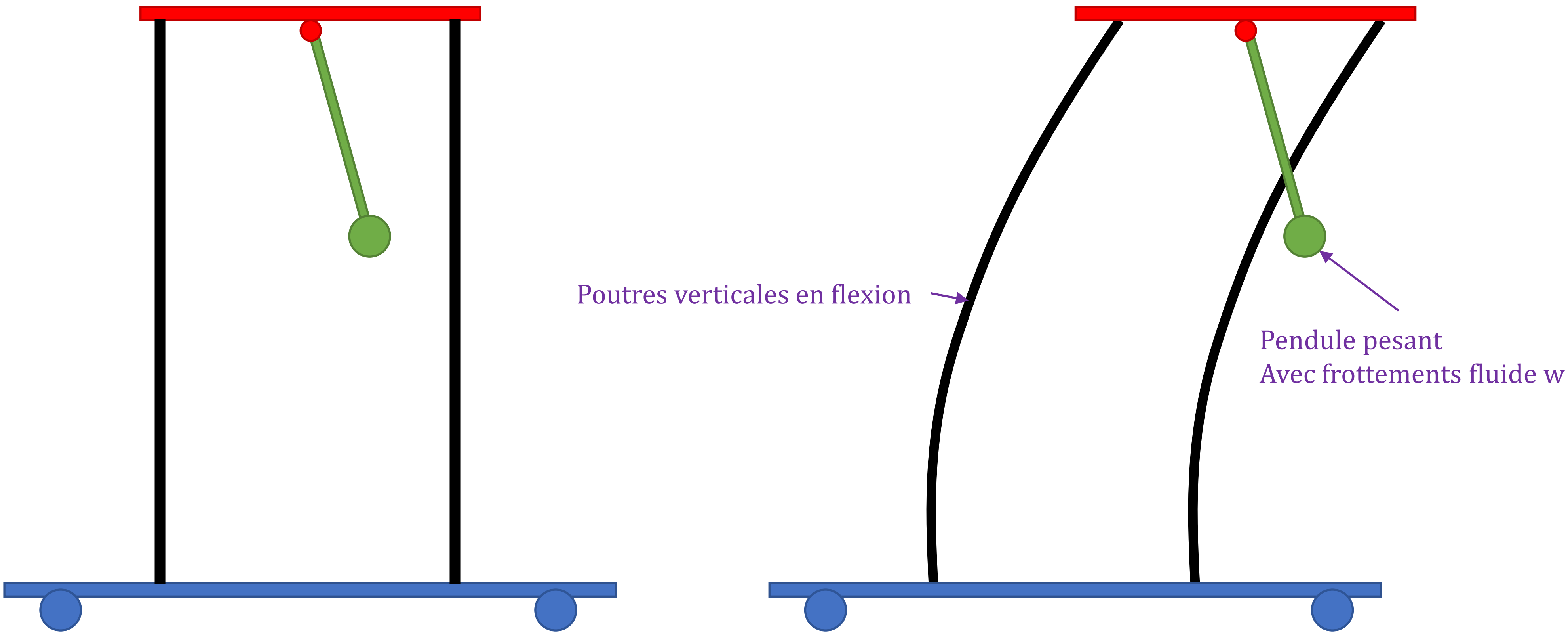
Comparaison avec et sans pendule:





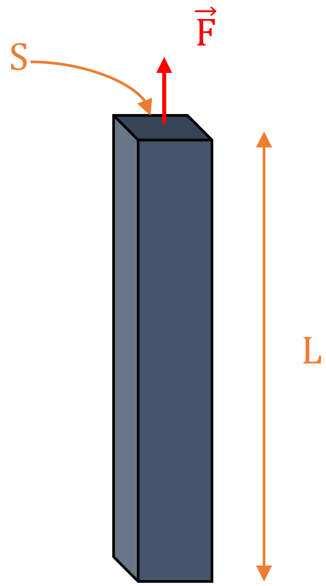
Script Python en annexe N

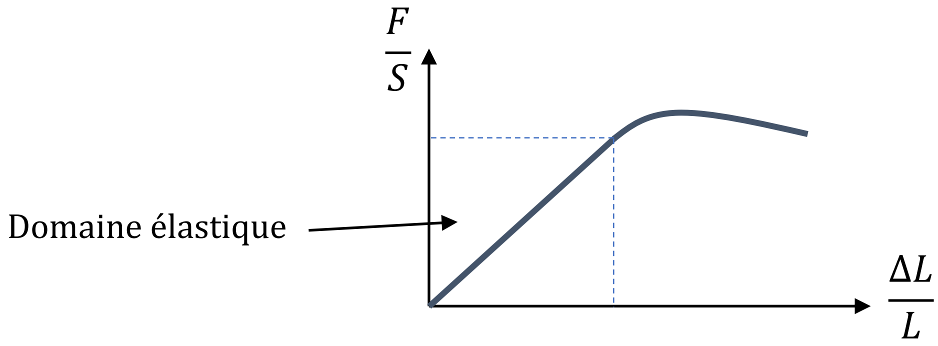
Modélisation de la maquette:



Modélisation d’une poutre verticale:

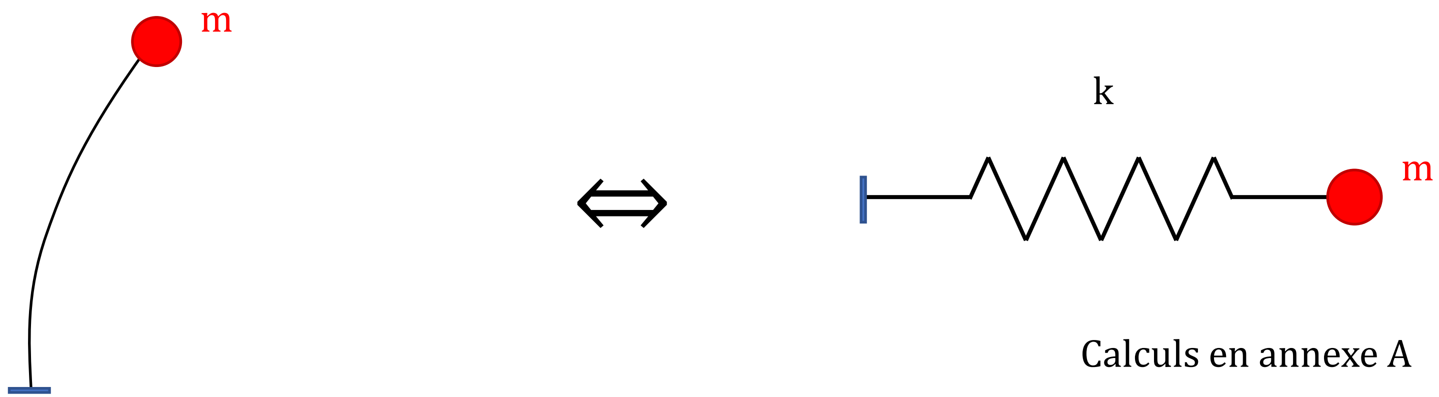
Définition du module d’Young E dans le domaine d’élasticité





$$\frac{\Delta L}{L} = \frac{F}{ES}$$

Application à une poutre verticale encastrée en flexion

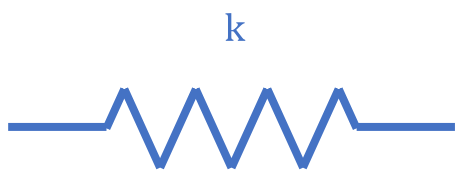


Calculs en annexe A

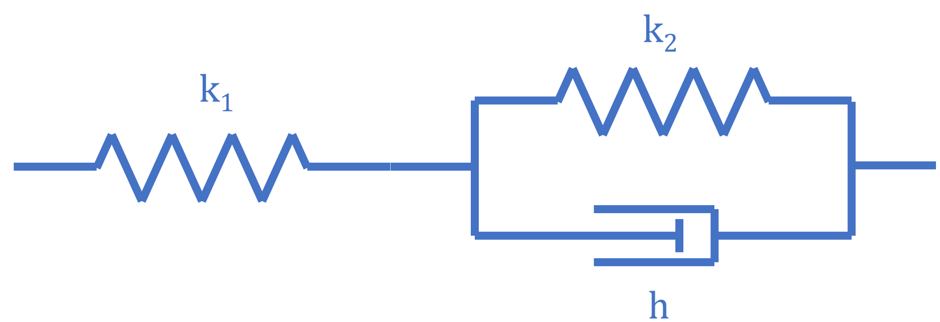
Modélisation d'une poutre verticale:

Différents modèles possibles

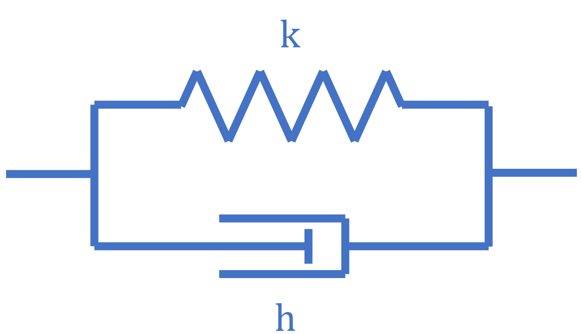
- Utilisation du module d'Young



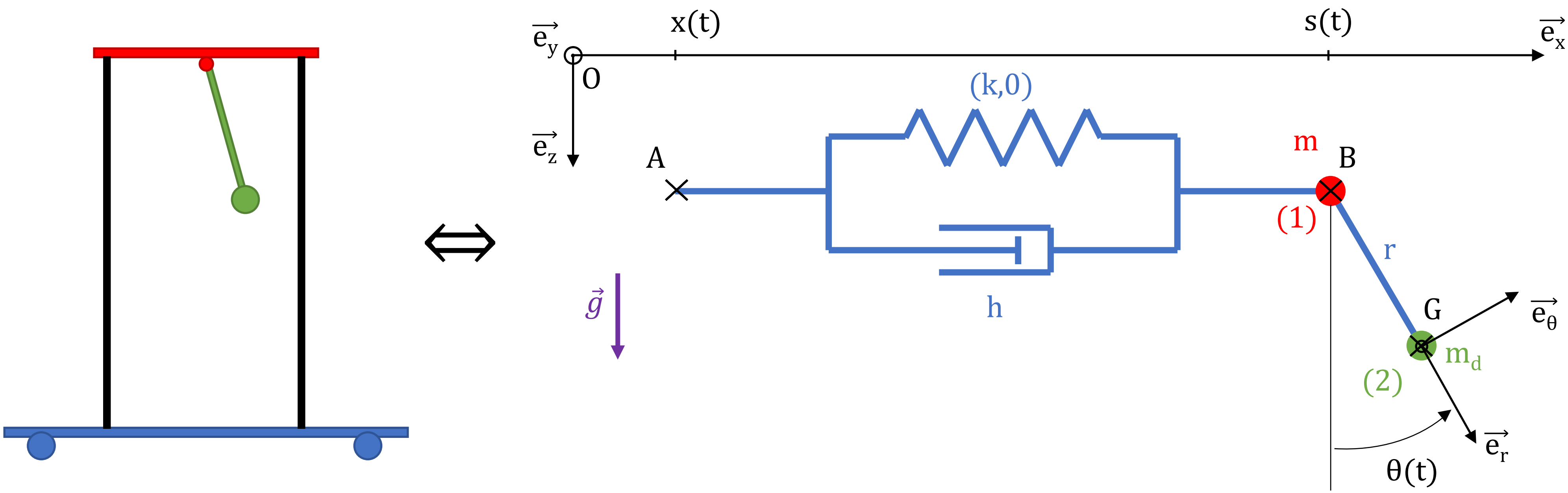
- Poynting-Thomson



- Kelvin-Voigt



Modèle retenu et mise en équation du système:



$$\boxed{\begin{pmatrix} m_d r & m_d \cos\theta \\ m_d r \cos\theta & m + m_d \end{pmatrix} \begin{pmatrix} \ddot{\theta} \\ \ddot{s} \end{pmatrix} = \begin{pmatrix} -m_d g \sin\theta - w\dot{\theta} \\ h(\dot{x} - \dot{s}) + k(x - s) - w\dot{\theta}\cos\theta + m_d r \dot{\theta}^2 \sin\theta \end{pmatrix}}$$

$$Système\ inversible \Leftrightarrow m_d r(m + m_d) - m_d^2 r \cos^2\theta \neq 0$$

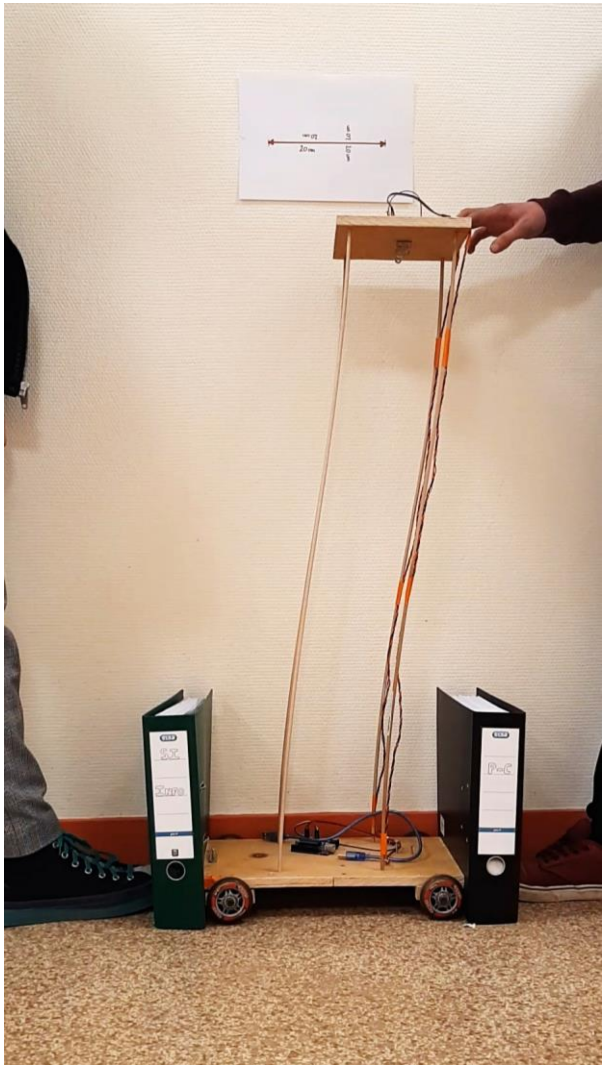
$$\Leftrightarrow \cos^2\theta \neq \frac{m_d r(m + m_d)}{m_d^2 r}$$

$$\Leftrightarrow \cos^2\theta \neq \frac{m + m_d}{m}$$

Calculs en annexe B

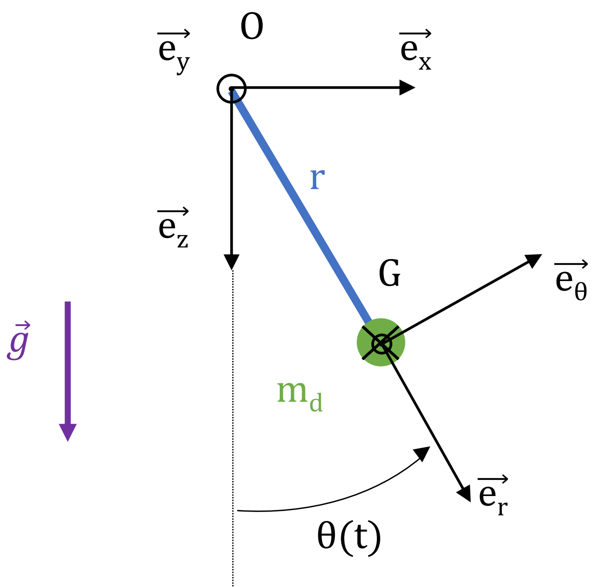
Détermination des paramètres de la maquette par pointage informatique:

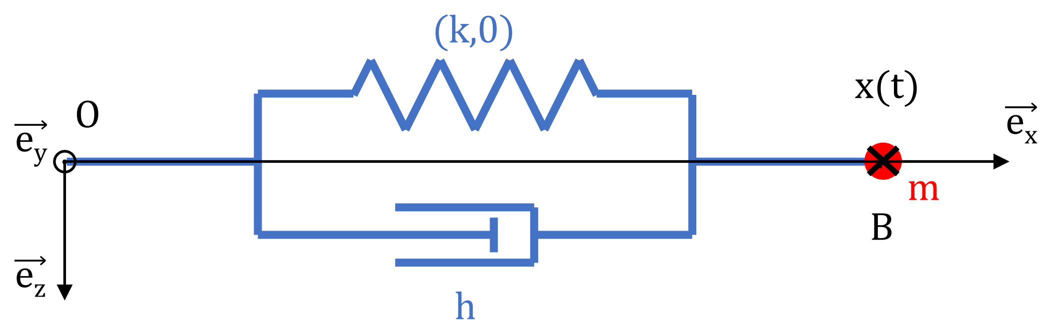
Raideur k du ressort
et
coefficient de frottement h des poutres verticales



Coefficient de frottement w du pendule pesant

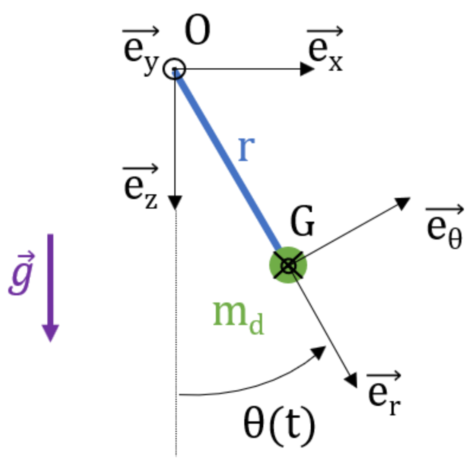
Détermination des paramètres de la maquette par pointage informatique:



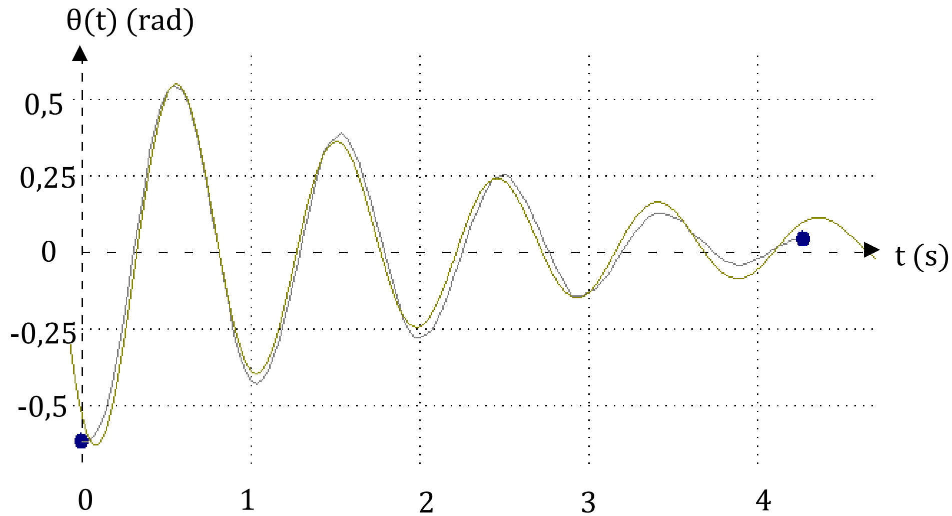


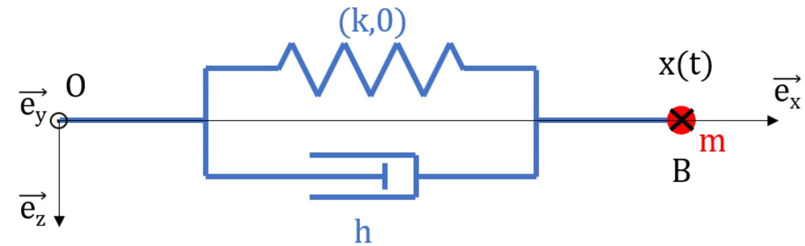
$$(E) : \ddot{f} + 2\xi\omega_0\dot{f} + \omega_0^2 f = 0$$
$$\Leftrightarrow f(t) = y_m \cos(\omega t + \varphi)e^{-\xi\omega_0 t} \ avec \ \omega = \omega_0\sqrt{1-\xi^2}$$

Détermination des paramètres de la maquette par pointage informatique:



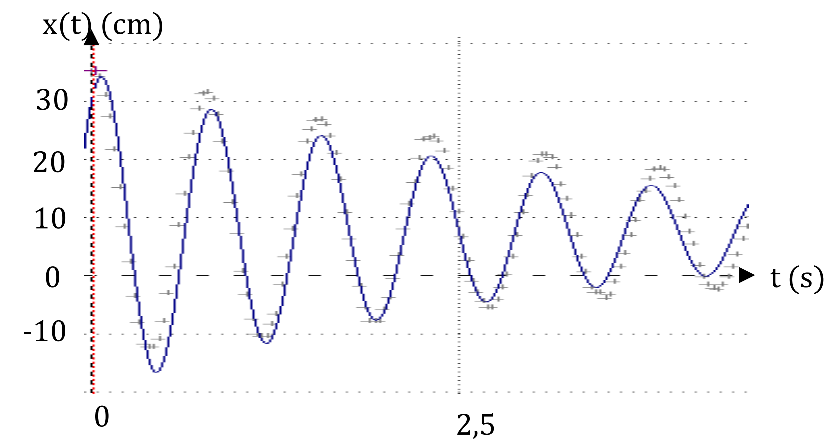
$$w = \frac{2\xi\omega_0 J}{r} = 2\xi\omega_0 m_d r$$





$$h = 2m\xi\omega_0$$

$$k = \frac{m\omega^2}{\left(\frac{h}{4\omega m} \pm \sqrt{1 + \left(\frac{h}{4\omega m}\right)^2}\ \right)^2}$$



Application numérique:

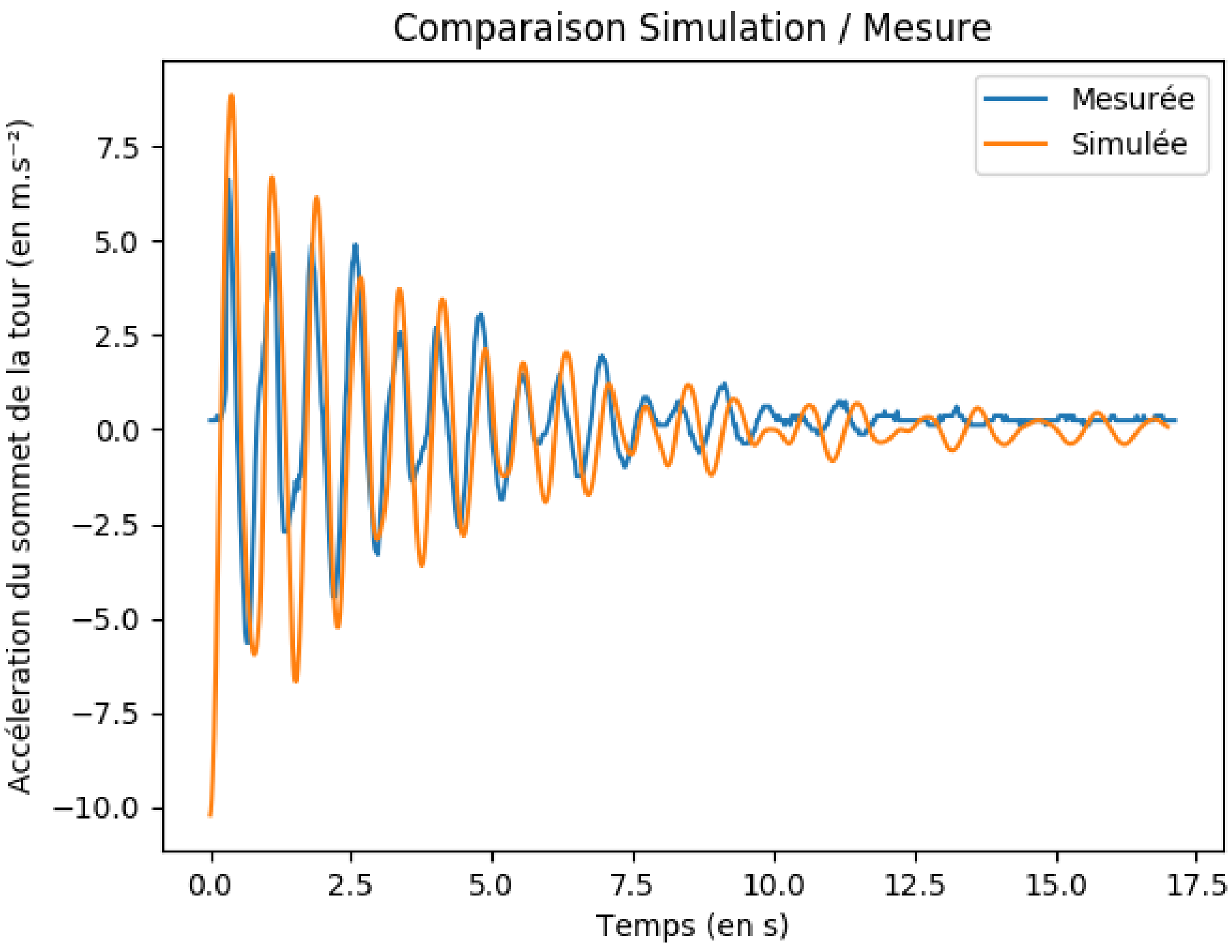
$$\boxed{w = 1{,}4.10^{-3}\ \mathrm{N.s}}$$

Calculs en annexe C

Applications numériques:

$$\boxed{\begin{array}{l} k = 39\ \mathrm{N.m} \\ h = 0{,}37\ \mathrm{N.s.m^{-1}} \end{array}}$$

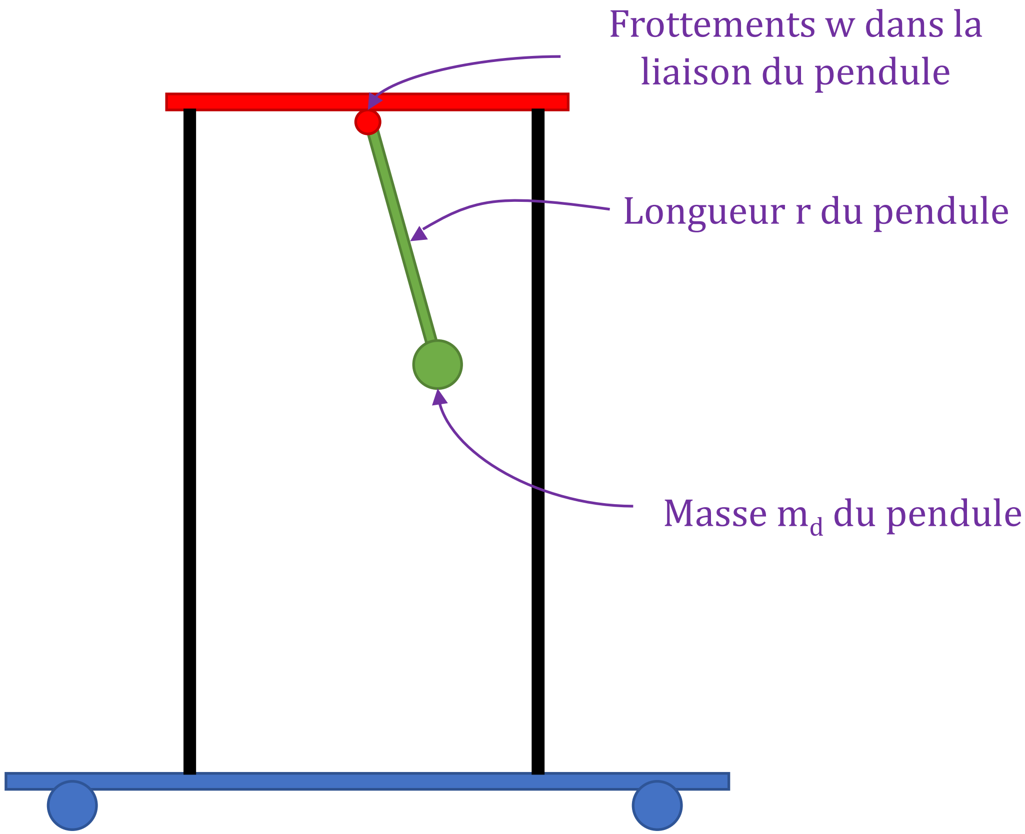
Comparaison de la modélisation python avec les mesures sur la maquette:



Script Python en annexes H et N

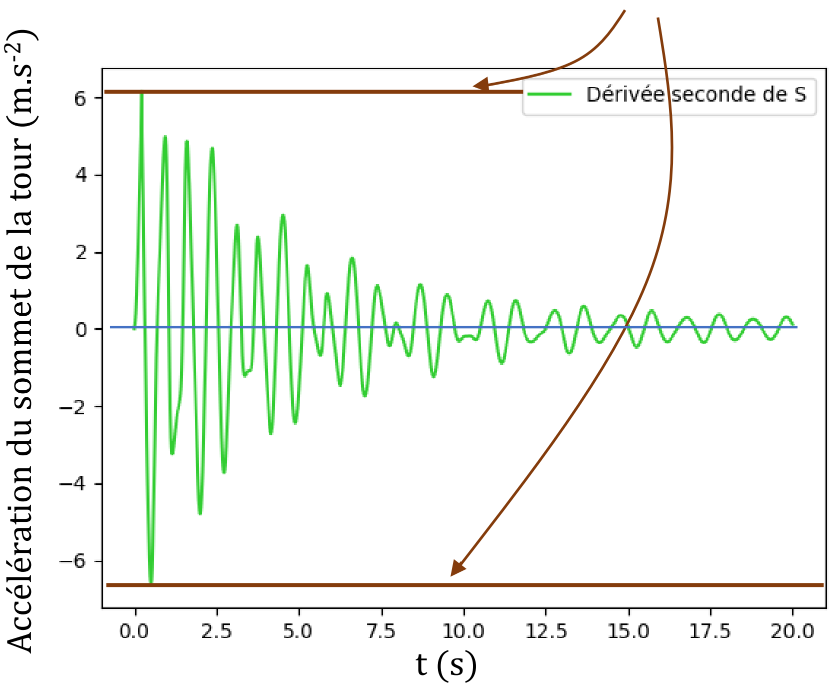
Optimisation des paramètres avec python:

Paramètres modifiables



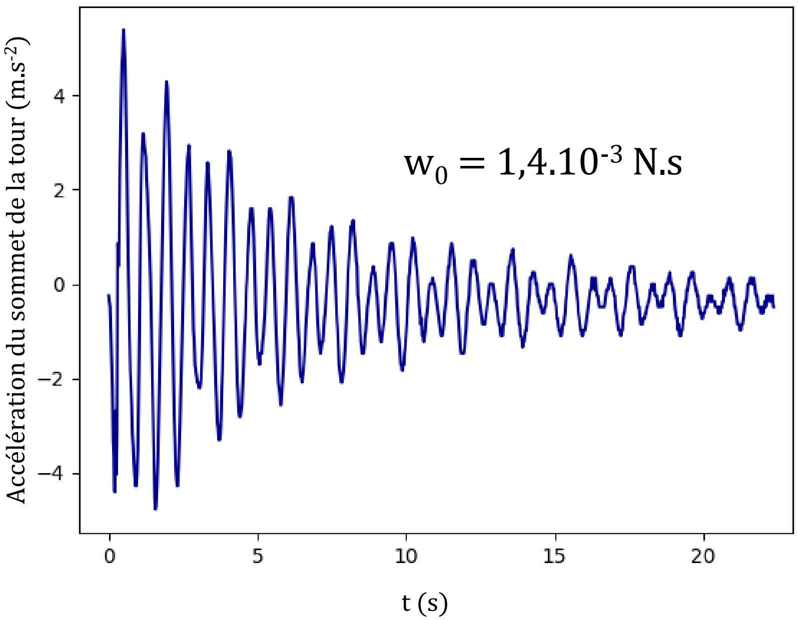
Critères d'optimisation:

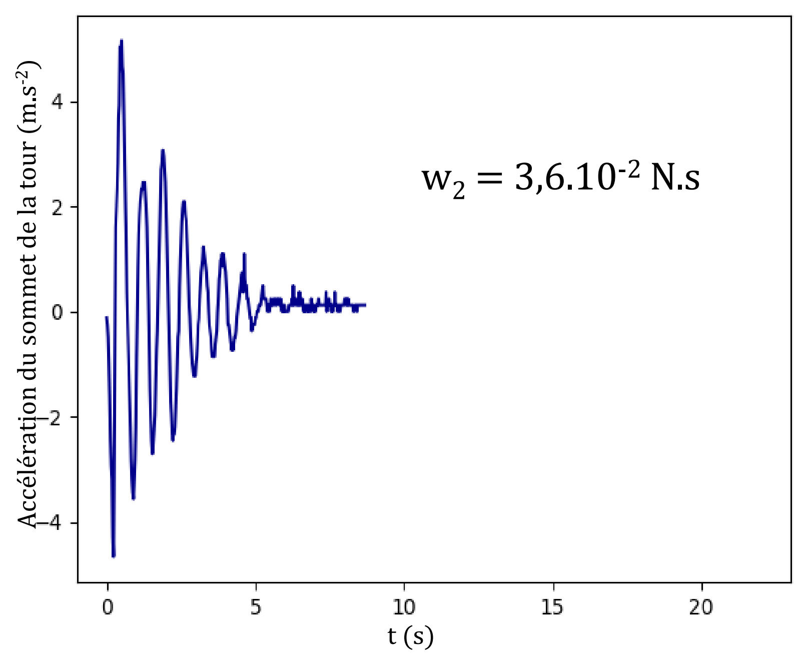
- Accélération maximale ressentie au sommet de la tour



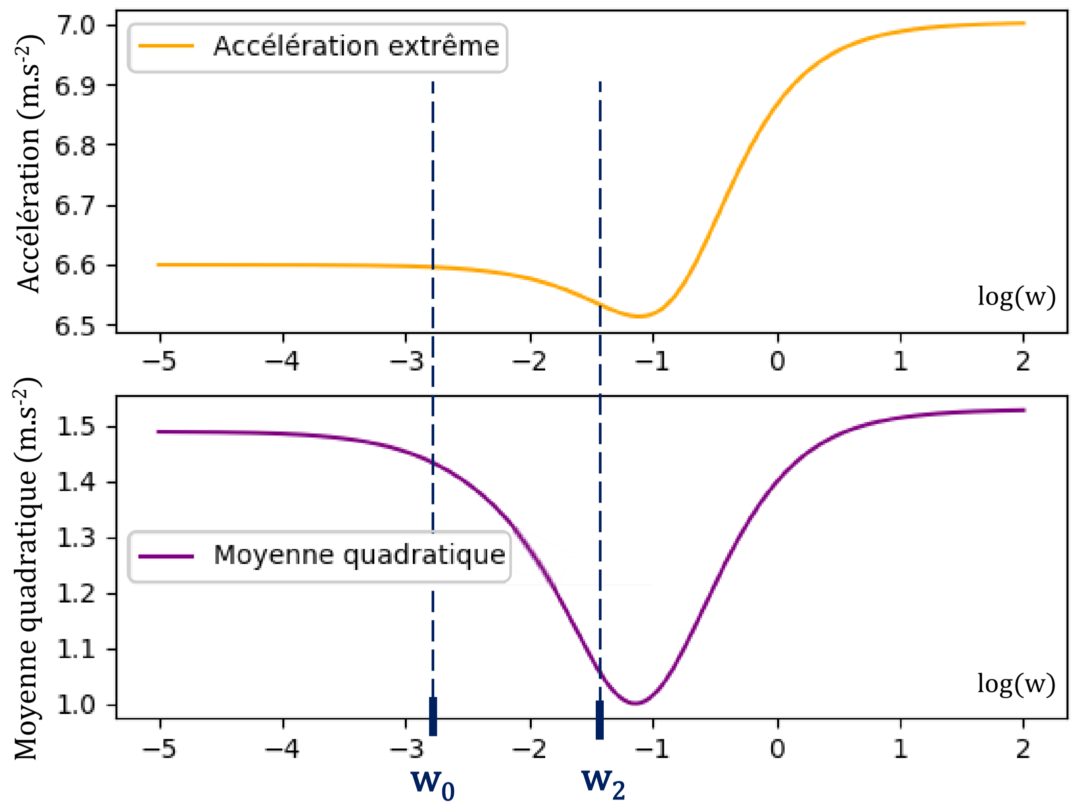
- Moyenne quadratique de l'accélération: surface entre la courbe et la valeur moyenne. Caractérise la rapidité du retour à la position d'équilibre

Optimisation expérimentale puis informatique à masse et longueur fixées:





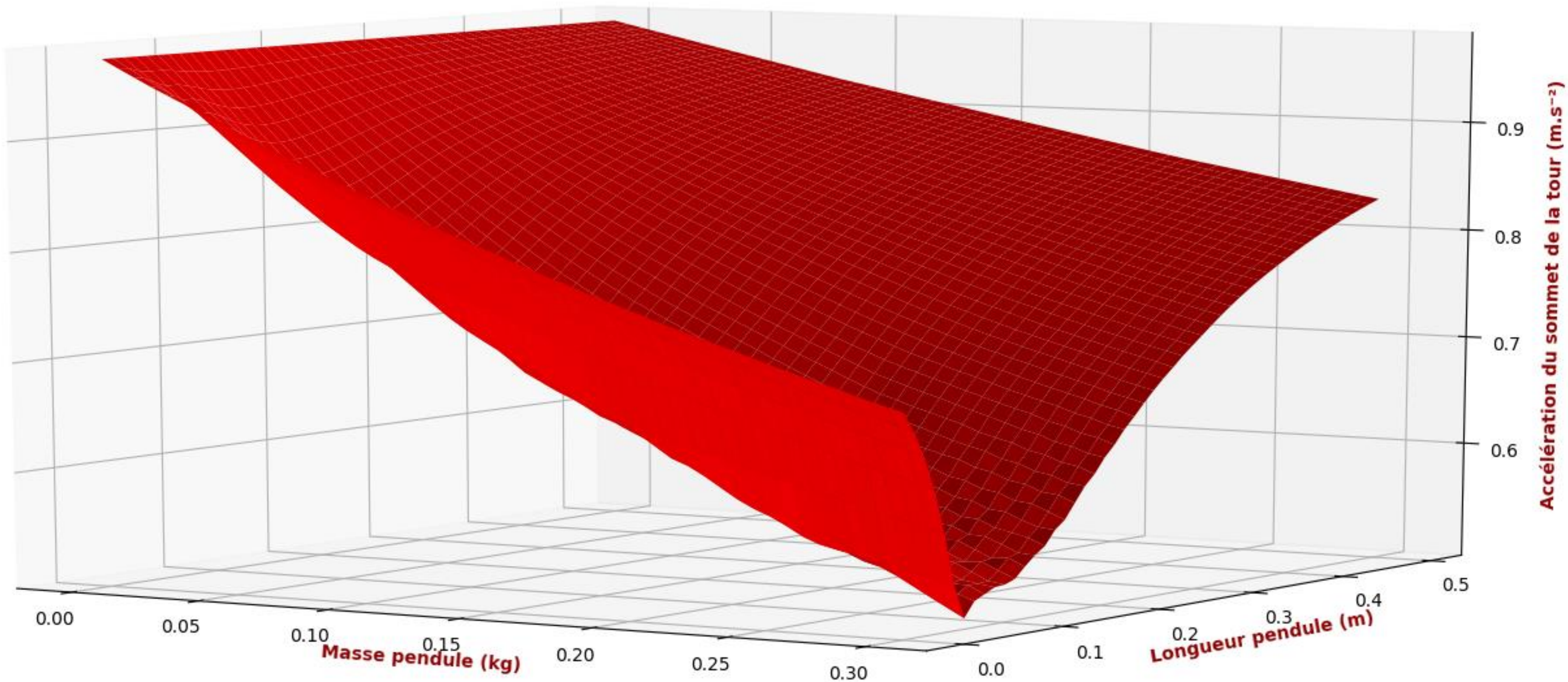
- Frottement w_2 proche du frottement optimal
- Accélérations maximales similaires pour w_0 et w_2



Script Python en annexe I et N

Optimisation des paramètres avec python:

Courbe de l'accélération maximale optimisée en fonction de la masse et de la longueur du pendule

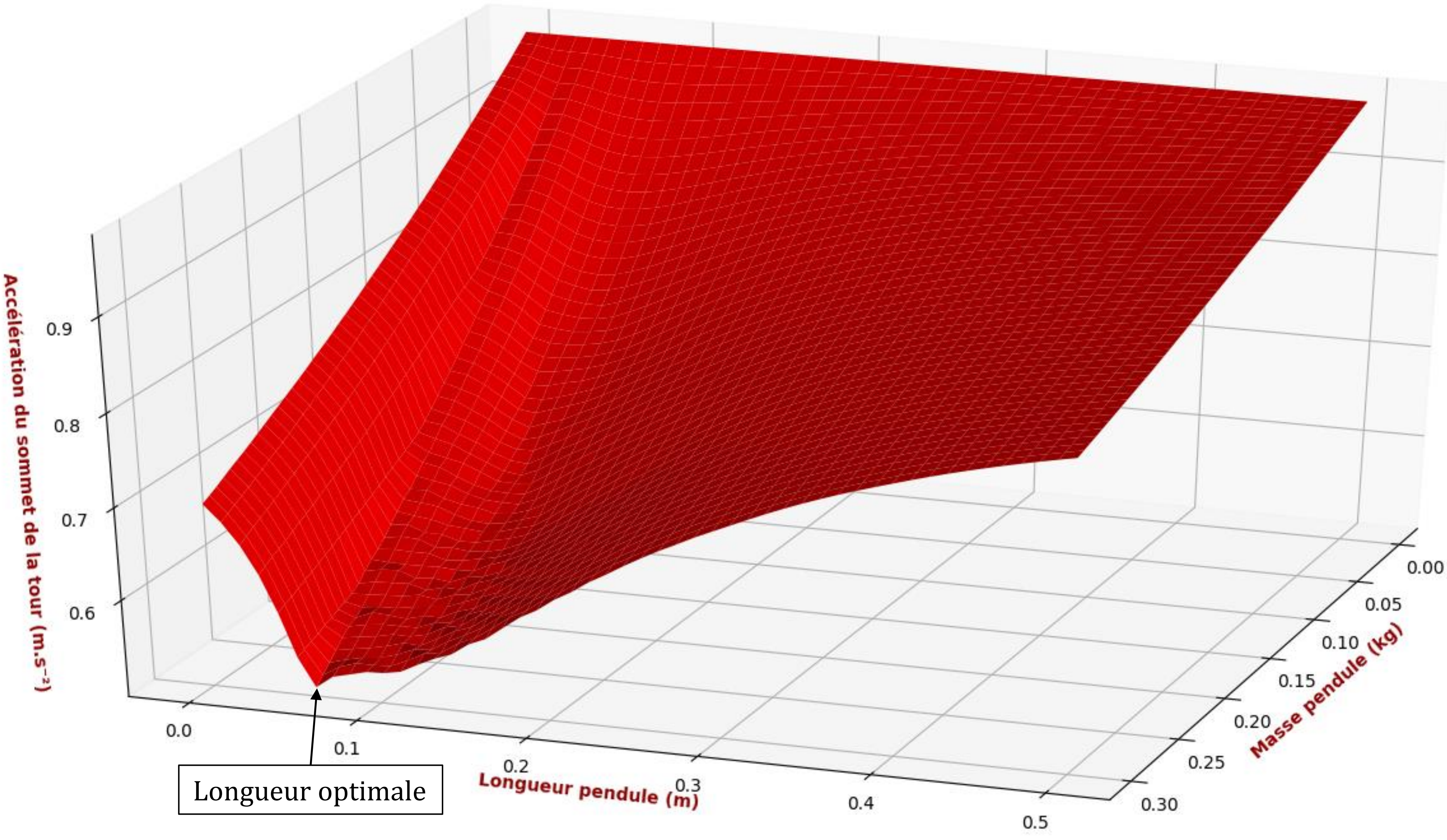


Pas de masse optimale

Script Python en annexe J

Optimisation des paramètres avec python:

Courbe de l'accélération maximale optimisée en fonction de la masse et de la longueur du pendule



Script Python en annexe J

Conclusion:

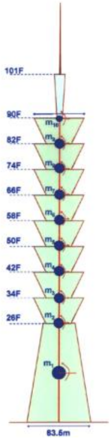
- Efficacité vérifiée

- Modélisation complexe mais possible et proche de la réalité du laboratoire

- Optimisations:
  - Indispensables avant une mise en œuvre à grande échelle
  - Nécessité d’un asservissement

- Un modèle qui ne peut se transposer à plus grande échelle car maquette trop différente d’un gratte ciel réel

- Modèles plus complexes à mettre en œuvre:
  - Travail sur les structures complexes
  - Modélisation plus fine des matériaux



Annexe : Plan

Études mécaniques et résultats informatiques:

Annexe A: Modélisation d'une poutre verticale

Annexe B: Mise en équation du système

Annexe C: Étude permettant la détermination des constantes

Annexe D: Signal d'entrée en triangle de vitesse

Annexe E: Étude et optimisation harmonique

Annexe F: Optimisation à masse fixée

Annexe G: Comparaison Euler / Scipy

Scripts Python:

Annexe H: Outils modélisation

Annexe I: Mise en œuvre des optimisations

Annexe J: Plan d'optimisation

Annexe K: Comparaison Euler / Scipy

Annexe L: Étude harmonique

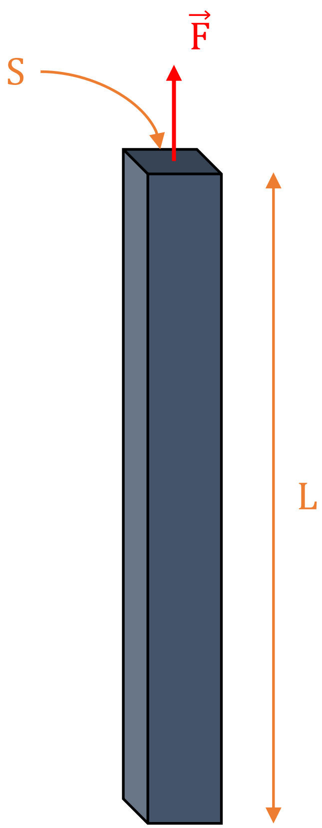
Annexe M: Étude à masse fixée

Annexe N: Exploitation des fichiers Arduino

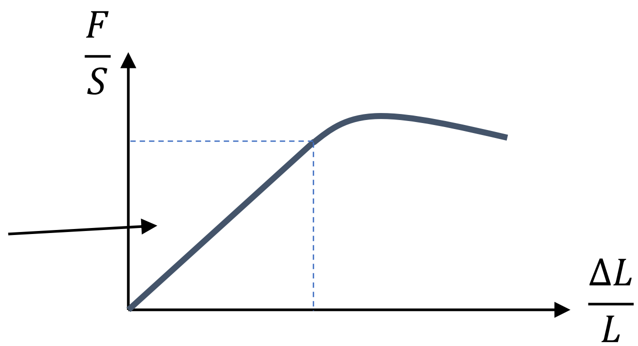
Annexe A

Modélisation d'une poutre verticale:

Définition du module d'Young E dans le domaine d'élasticité



Domaine élastique

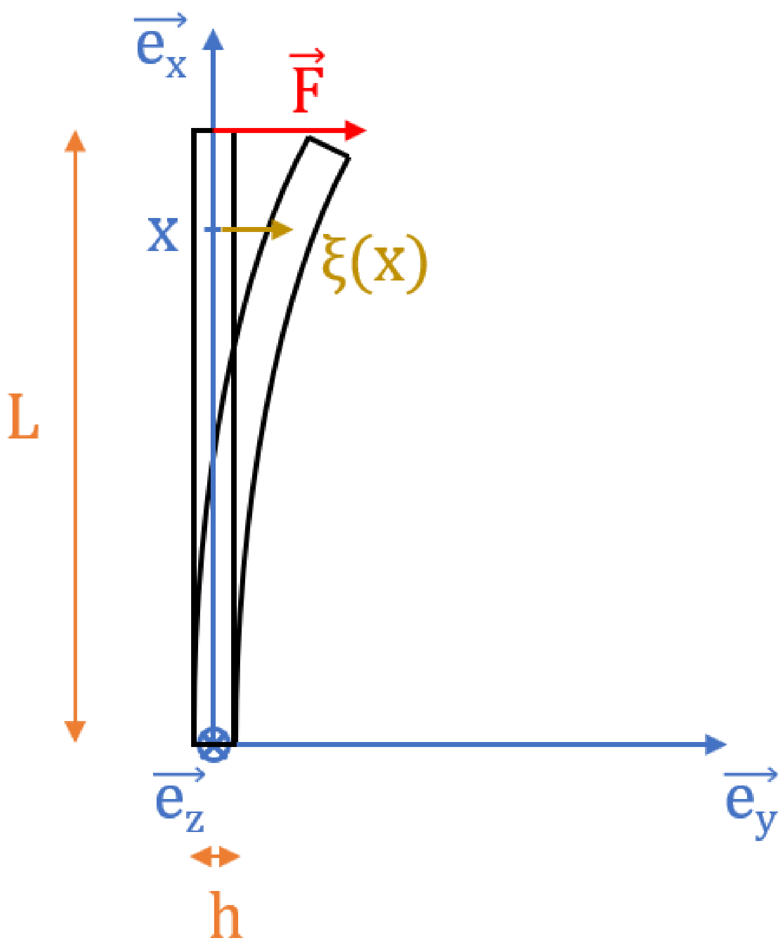


$$\frac{\Delta L}{L} = \frac{F}{ES}$$

Annexe A

Modélisation d'une poutre verticale:

Déformation d'une poutre encastrée soumise à une force perpendiculaire sur l'extrémité libre:



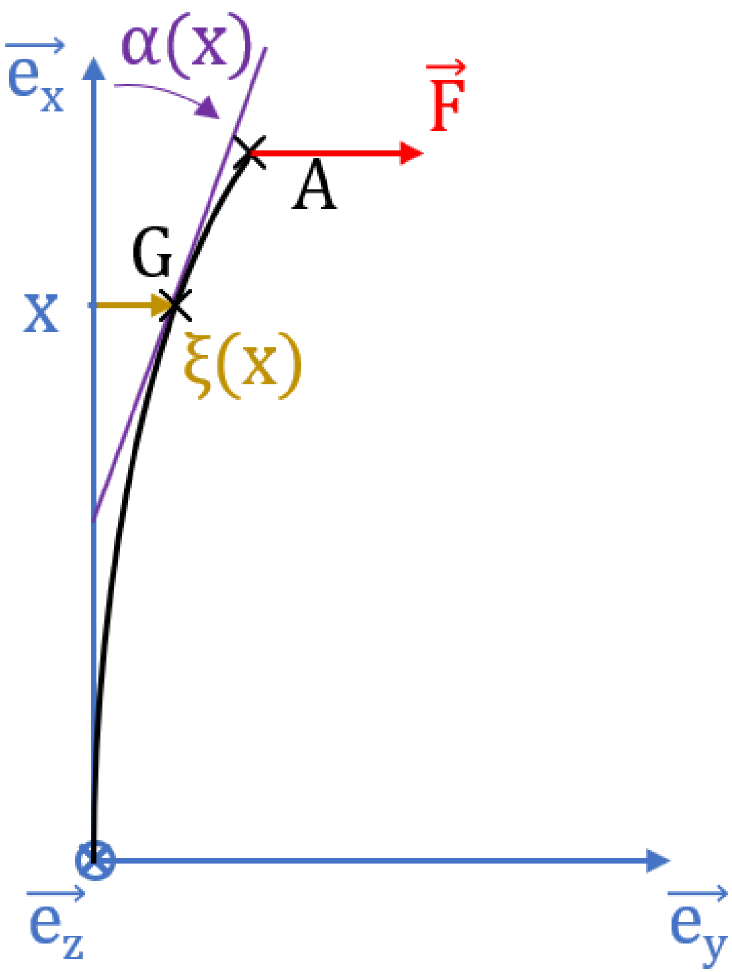
Poutre de section rectangulaire:

- Longueur L
- Largeur b
- Epaisseur h

Annexe A

Modélisation d'une poutre verticale:

$$\alpha(x) = \frac{dy}{dx} = \frac{d\xi}{dx}$$



Moment quadratique d'une section par rapport à Gz:

$$I_{gz} = \frac{bh^3}{12}$$

Moment fléchissant:

$$M(x) = EI_{gz}\frac{\mathrm{d}\alpha}{\mathrm{d}x}(x) = EI_{gz}\frac{\mathrm{d}^2\xi}{\mathrm{d}x^2}(x)$$

Moment de la force $\vec{F}$:

$$\overrightarrow{M_{G,\vec{F}}} = \overrightarrow{GA} \wedge \vec{F} = F(L-x)\cdot\vec{e_z}$$

$$\acute{A}\ l'\acute{e}quilibre:\ M(x) = \overrightarrow{M_{G,\vec{F}}}\cdot\vec{e_z} \Rightarrow EI_{gz}\frac{\mathrm{d}^2\xi}{\mathrm{d}x^2} = F(L-x)$$

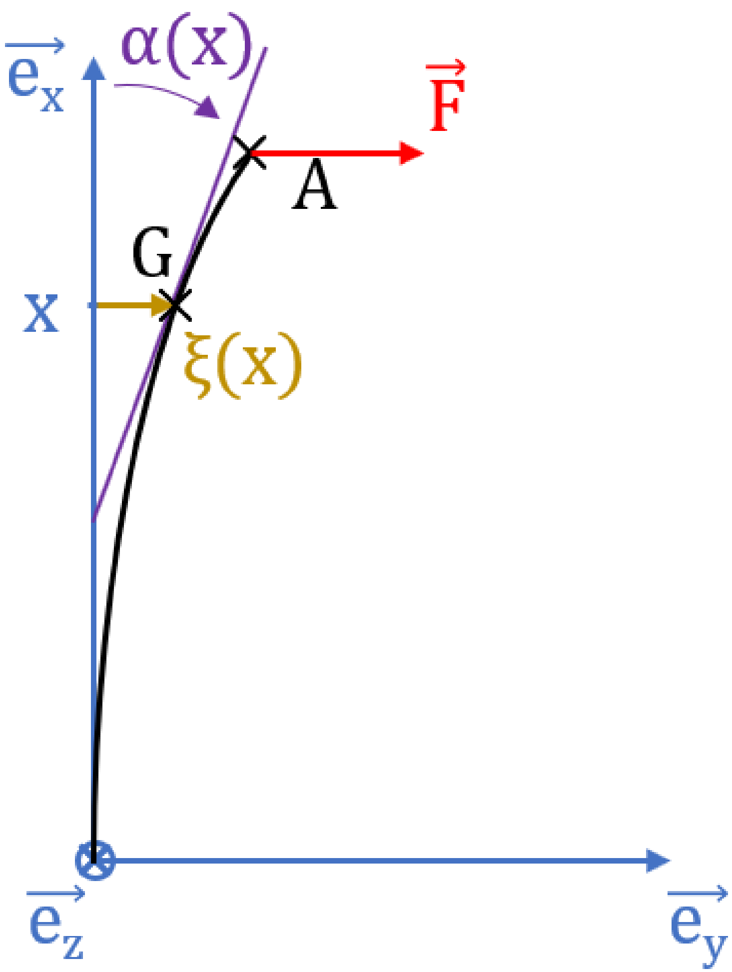
$$d'o\grave{u}:\ \xi(x) = \frac{12F}{Ebh^3}(L\frac{x^2}{2} - \frac{x^3}{6}) + Ax + B$$

$$\begin{cases}\xi(0) = 0\\ \dot{\xi}(0) = 0\end{cases} \Rightarrow A = B = 0$$

$$Enfin:\ \xi(x) = \frac{12F}{Ebh^3}(L\frac{x^2}{2} - \frac{x^3}{6})$$

Annexe A

Modélisation d'une poutre verticale:



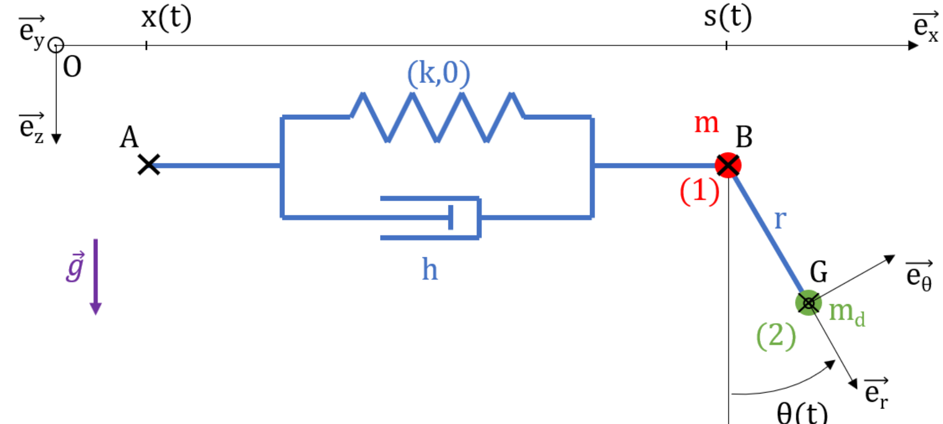
$$\Delta = \xi(L) = \frac{4FL^3}{Ebh^3}$$

$$F = k\Delta \ avec \ k = \frac{Eb}{4}\frac{h}{L}^3$$

D'où l'équivalence avec un ressort

Annexe B

Mise en équation du système:



$$\Re_0 = (0, \overrightarrow{e_x}, \overrightarrow{e_y}, \overrightarrow{e_z})\ \textit{Repère galiléen}$$
$$\overrightarrow{OA} = x(t) \cdot \overrightarrow{e_x}$$
$$\overrightarrow{OB} = s(t) \cdot \overrightarrow{e_x}$$
$$\overrightarrow{AG} = r \cdot \overrightarrow{e_r}$$
$$\theta = (\overrightarrow{e_z}, \overrightarrow{e_r})$$

1- On isole (2)

BAME à (2):

- Poids: $_G\left\{ \begin{matrix} m_d g \cdot \overrightarrow{e_z} \\ \overrightarrow{0} \end{matrix} \right\}$
- 1→2: $_B\left\{ \begin{matrix} -T_0 \cdot \overrightarrow{e_r} \\ \overrightarrow{0} \end{matrix} \right\}$
- Frottements: $_G\left\{ \begin{matrix} -w\dot{\theta} \cdot \overrightarrow{e_\theta} \\ \overrightarrow{0} \end{matrix} \right\}$

Torseur dynamique de (2) au point B:

$$_B\left\{ \begin{matrix} m_d \cdot \overrightarrow{a_{G\in 2/0}} \\ \overrightarrow{\delta_{A\in 2/0}} \end{matrix} \right\} = {}_B\left\{ \begin{matrix} m_d(\ddot{s} \cdot \overrightarrow{e_x} + r\ddot{\theta} \cdot \overrightarrow{e_\theta} - m_d r\dot{\theta}^2 \cdot \overrightarrow{e_r}) \\ J\ddot{\theta} \cdot \overrightarrow{e_y} \end{matrix} \right\} \quad \text{avec} \quad J = \int r^2 dm = m_d r^2$$

Théorème de la résultante dynamique appliqué à (2):

$$m_d(\ddot{s} \cdot \overrightarrow{e_x} + r\ddot{\theta} \cdot \overrightarrow{e_\theta} - r\dot{\theta}^2 \cdot \overrightarrow{e_r}) = m_d g \cdot \overrightarrow{e_z} - T_0 \cdot \overrightarrow{e_r} - w\dot{\theta} \cdot \overrightarrow{e_\theta}$$

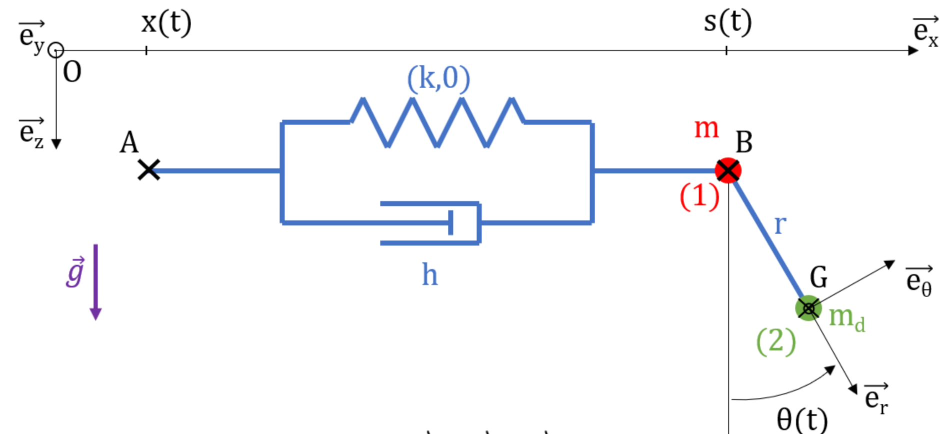
Puis par projection selon $\overrightarrow{e_x}$ et $\overrightarrow{e_z}$:

$$\boxed{\left\{ \begin{array}{l} m_d(\ddot{s} + r\ddot{\theta}\cos\theta - r\dot{\theta}^2 \sin\theta) = -T_0 \sin\theta - w\dot{\theta}\cos\theta \\ -m_d(r\ddot{\theta}\sin\theta + r\dot{\theta}^2\cos\theta) = m_d g - T_0\cos\theta + w\dot{\theta}\sin\theta \end{array} \right.}$$

Annexe B

Mise en équation du système:

2- On isole (1)



$\Re_0 = (0, \overrightarrow{e_x}, \overrightarrow{e_y}, \overrightarrow{e_z})$ *Repère galiléen*

$\overrightarrow{OA} = x(t) \cdot \overrightarrow{e_x}$

$\overrightarrow{OB} = s(t) \cdot \overrightarrow{e_x}$

$\overrightarrow{AG} = r \cdot \overrightarrow{e_r}$

$\theta = (\overrightarrow{e_z}, \overrightarrow{e_r})$

BAME à (1):

- Poids: $${}_B\left\{ \begin{array}{c} mg \cdot \overrightarrow{e_z} \\ \overrightarrow{0} \end{array} \right\}$$
- 2→1: $${}_B\left\{ \begin{array}{c} T_0 \cdot \overrightarrow{e_r} \\ \overrightarrow{0} \end{array} \right\}$$
- Frottements: $${}_B\left\{ \begin{array}{c} -h(\dot{s} - \dot{x}) \cdot \overrightarrow{e_x} \\ \overrightarrow{0} \end{array} \right\}$$
- Ressort: $${}_B\left\{ \begin{array}{c} -k(s - x) \cdot \overrightarrow{e_x} \\ \overrightarrow{0} \end{array} \right\}$$

Torseur dynamique de (1) au point B: $${}_B\left\{ \begin{array}{c} m\ddot{s} \cdot \overrightarrow{e_x} \\ \overrightarrow{0} \end{array} \right\}$$

Théorème de la résultante dynamique appliqué à (1) et projeté selon $\vec{e_x}$:

$$T_0 \sin\theta = m\ddot{s} + h(\dot{s} - \dot{x}) + k(s - x)$$

Annexe B

Mise en équation du système:

$$\begin{cases} m_d(\ddot{s} + r\ddot{\theta}\cos\theta - r\dot{\theta}^2\sin\theta) = -T_0\sin\theta - w\dot{\theta}\cos\theta & (1) \\ m_d g = -m_d(r\ddot{\theta}\sin\theta + r\dot{\theta}^2\cos\theta) + T_0\cos\theta - w\dot{\theta}\sin\theta & (2) \\ m\ddot{s} = T_0\sin\theta + h(\dot{x} - \dot{s}) + k(x - s) & (3) \end{cases}$$

$puis\ (1) * \cos\theta + (2) * \sin\theta\ donne:$
$m_d g\sin\theta = m_d r(\ddot{\theta}\sin^2\theta + \dot{\theta}^2\cos\theta\sin\theta) - T_0\cos\theta\sin\theta + w\dot{\theta}\sin^2\theta + m_d(\ddot{s}\cos\theta +$
$r(\ddot{\theta}\cos^2\theta - \dot{\theta}^2\cos\theta\sin\theta)) + T_0\cos\theta\sin\theta + w\dot{\theta}\cos^2\theta$
$\Rightarrow m_d g\sin\theta = m_d r\ddot{\theta} + w\dot{\theta} + m_d\ddot{s}\cos\theta$

$enfin\ (1)\ dans\ (3)\ donne:$
$m\ddot{s} = -w\dot{\theta}\cos\theta - m_d(\ddot{s} + r(\ddot{\theta}\cos\theta - \dot{\theta}^2\sin\theta)) + h(\dot{x} - \dot{s}) + k(x - s)$
$\Rightarrow (m + m_d)\ddot{s} + m_d r\ddot{\theta}\cos\theta = h(\dot{x} - \dot{s}) + k(x - s) - w\dot{\theta}\cos\theta + m_d r\dot{\theta}^2\sin\theta$

$$\begin{cases} (m + m_d)\ddot{s} + m_d r\ddot{\theta}\cos\theta = h(\dot{x} - \dot{s}) + k(x - s) - w\dot{\theta}\cos\theta + m_d r\dot{\theta}^2\sin\theta \\ m_d g\sin\theta = m_d r\ddot{\theta} + w\dot{\theta} + m_d\ddot{s}\cos\theta \end{cases}$$

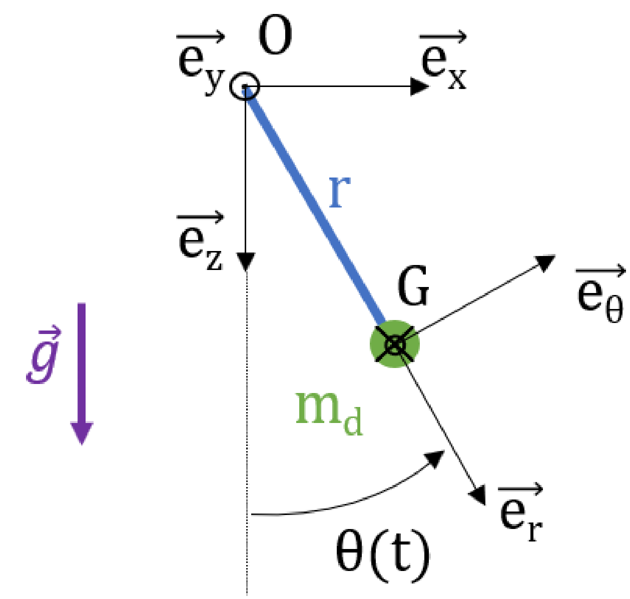
$$\Rightarrow \begin{cases} (m + m_d)\ddot{s} + m_d r\ddot{\theta}\cos\theta = h(\dot{x} - \dot{s}) + k(x - s) - w\dot{\theta}\cos\theta + m_d r\dot{\theta}^2\sin\theta \\ m_d r\ddot{\theta} + m_d\ddot{s}\cos\theta = -m_d g\sin\theta - w\dot{\theta} \end{cases}$$

Conclusion:

$$\boxed{\begin{pmatrix} m_d r & m_d\cos\theta \\ m_d r\cos\theta & m + m_d \end{pmatrix} \begin{pmatrix} \ddot{\theta} \\ \ddot{s} \end{pmatrix} = \begin{pmatrix} -m_d g\sin\theta - w\dot{\theta} \\ h(\dot{x} - \dot{s}) + k(x - s) - w\dot{\theta}\cos\theta + m_d r\dot{\theta}^2\sin\theta \end{pmatrix}}$$

Annexe C

Étude permettant la détermination des constantes:



On isole m_d:

BAME à m_d:

- Poids: $${}_G\left\{ \begin{array}{c} m_d g \cdot \vec{e_z} \\ \vec{0} \end{array} \right\}$$
- Frottement: $${}_G\left\{ \begin{array}{c} -w\dot{\theta} \cdot \vec{e_\theta} \\ \vec{0} \end{array} \right\}$$
- Réaction du support: $${}_O\left\{ \begin{array}{c} -T_0 \cdot \vec{e_r} \\ \vec{0} \end{array} \right\}$$

Torseur dynamique de m_d au point O:

$${}_O\left\{ \begin{array}{c} m_d r\ddot{\theta} \cdot \vec{e_\theta} - m_d r\dot{\theta}^2 \cdot \vec{e_r} \\ J\ddot{\theta} \cdot \vec{e_y} \end{array} \right\} \quad \text{avec} \quad J = \int r^2 dm = m_d r^2$$

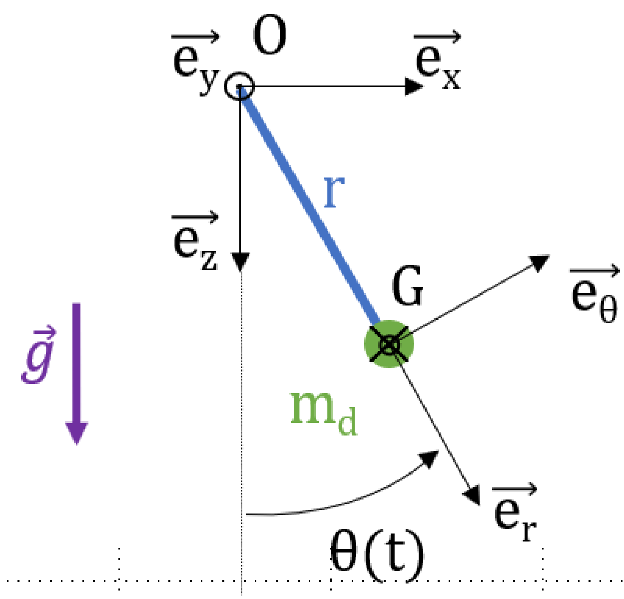
Théorème du moment dynamique appliqué à m_d en O et projeté selon $\vec{e_y}$:

$$J\ddot{\theta} = (\overrightarrow{OG} \wedge m_d g \cdot \vec{e_z} + \overrightarrow{OG} \wedge -w\dot{\theta} \cdot \vec{e_\theta}) \cdot \vec{e_y}$$
$$\Leftrightarrow J\ddot{\theta} = -rm_d g \sin\theta - rw\dot{\theta}$$
$$\Leftrightarrow \ddot{\theta} + \frac{wr}{J}\dot{\theta} + \frac{m_d r g}{J}\sin\theta = 0$$
$$\Leftrightarrow (E)\ avec\ \theta << 1 \quad \text{et} \quad \boxed{\omega_0 = \sqrt{\frac{m_d r g}{J}} \quad \xi = \frac{w}{2}\sqrt{\frac{r}{m_d g J}}}$$

$$\boxed{w = \frac{2\xi\omega_0 J}{r} = 2\xi\omega_0 m_d r}$$

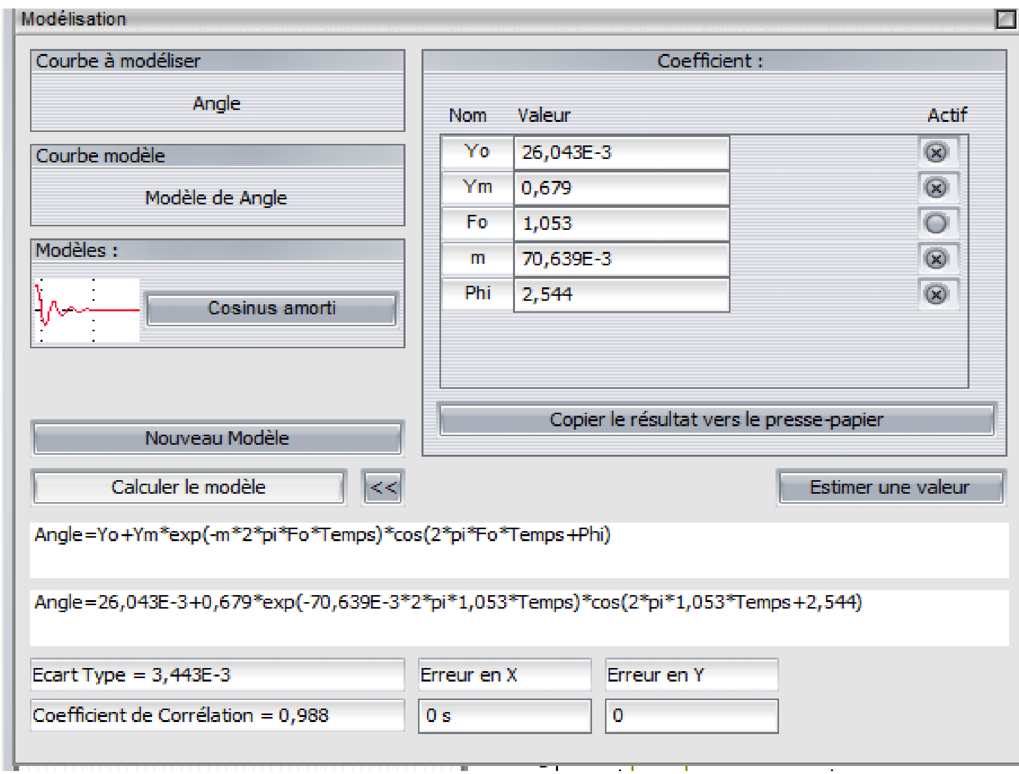
Annexe C

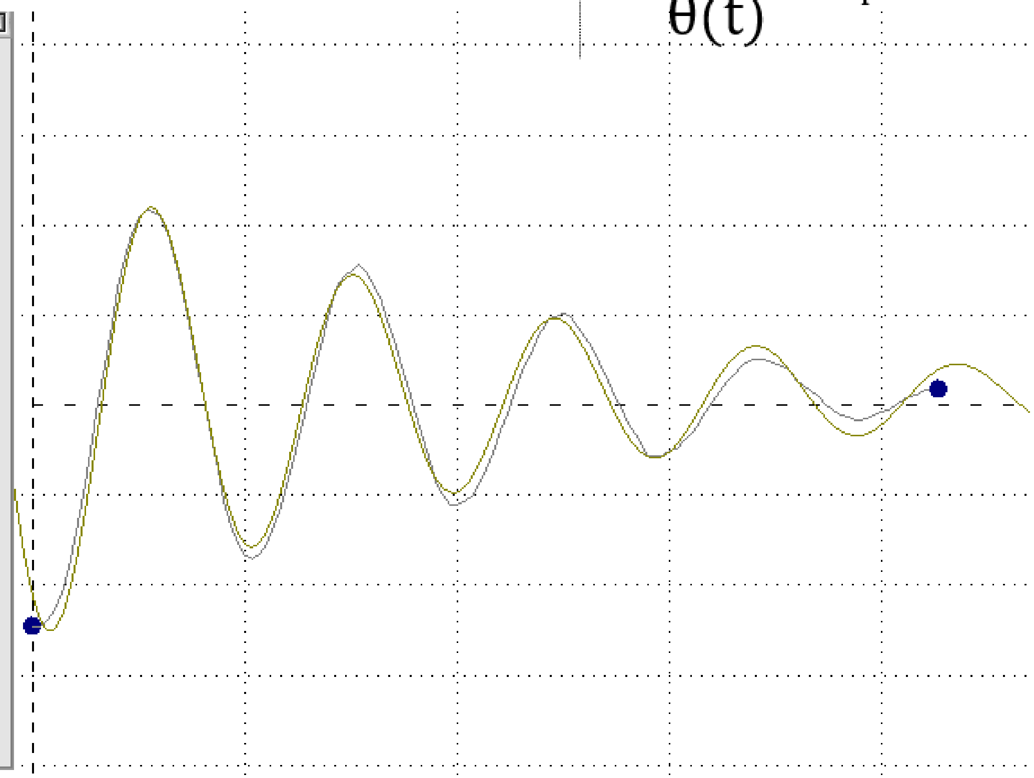
Étude permettant la détermination des constantes:



Application numérique:

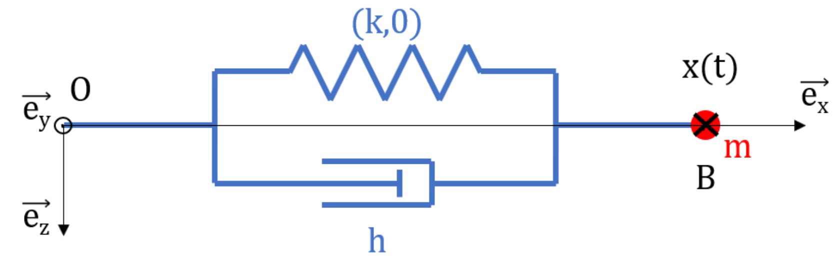
$$\boxed{w = 1{,}4.10^{-3}\ \text{N.s}}$$





Annexe C

Étude permettant la détermination des constantes:



$$(E) : \ddot{f} + 2\xi\omega_0\dot{f} + \omega_0^2 f = 0$$
$$\Leftrightarrow f(t) = y_m\cos(\omega t + \varphi)e^{-\xi\omega_0 t} \ avec \ \omega = \omega_0\sqrt{1-\xi^2}$$

On isole m:

$$E_c = \frac{1}{2}m\dot{x}^2$$
$$P_{ressort} = -kx\dot{x}$$
$$P_{frottement} = -h\dot{x}^2$$

On applique le théorème de l'énergie cinétique à m:

$$\frac{\mathrm{d}E_c}{\mathrm{d}t} = \sum P_{ext}$$
$$\Leftrightarrow m\ddot{x}\dot{x} = -kx\dot{x} - h\dot{x}^2$$
$$\Leftrightarrow m\ddot{x} = -kx - h\dot{x}$$
$$\Leftrightarrow (E)$$

avec $\boxed{\omega_0 = \sqrt{\frac{k}{m}} \quad \xi = \frac{h}{2\sqrt{km}}}$

Puis par identification des paramètres:

$$\begin{cases} k = m\omega_0^2 = \frac{h^2}{4m\xi^2} \\ \omega_0 = \frac{\omega}{\sqrt{1-\xi}} \\ \xi = \frac{h}{2m\omega_0} \end{cases}$$

$$\Rightarrow \xi = \frac{h\sqrt{1-\xi}}{2m\omega} \Rightarrow \xi^2 + (\frac{h}{2m\omega})^2(\xi - 1) = 0$$
$$\Rightarrow \xi^2 = (\frac{h}{2m\omega})^2(\frac{h}{4\omega m} \pm \sqrt{1 + (\frac{f}{4\omega m})^2}\ )$$
$$\Rightarrow k = \frac{m\omega^2}{(\frac{h}{4\omega m} \pm \sqrt{1 + (\frac{f}{4\omega m})^2}\ )^2}$$

$$\boxed{k = \frac{m\omega^2}{(\frac{h}{4\omega m} \pm \sqrt{1 + (\frac{h}{4\omega m})^2}\ )^2}}$$

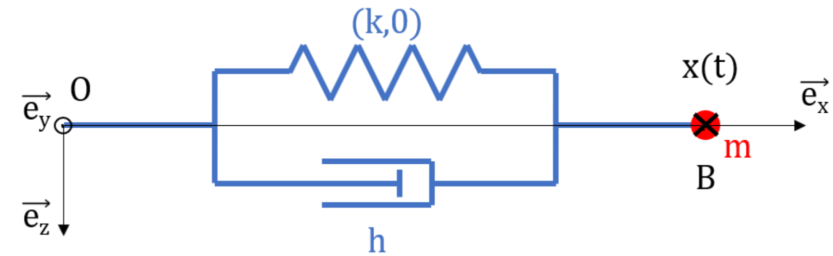
$$\boxed{h = 2m\xi\omega_0}$$

Annexe C

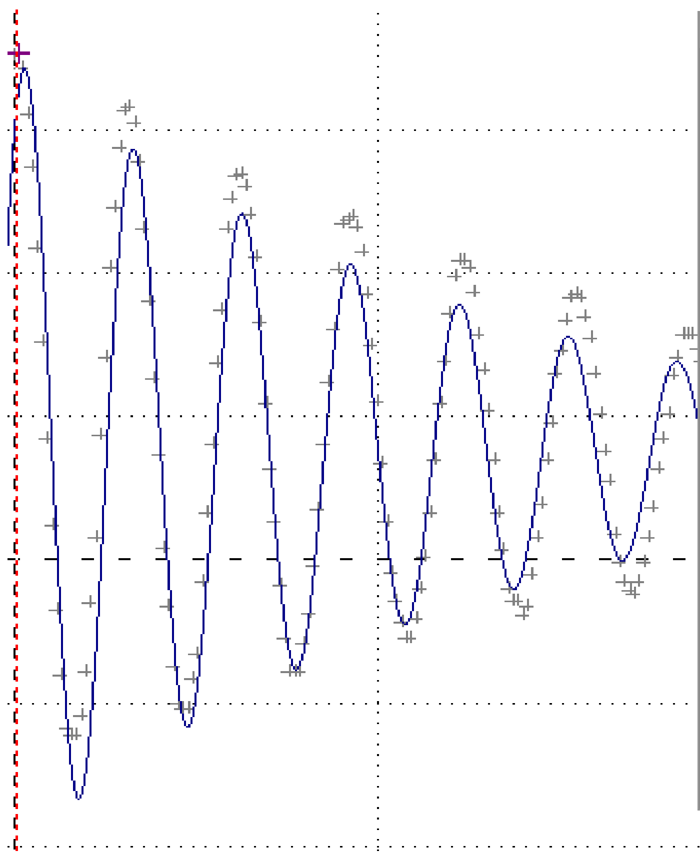
Étude permettant la détermination des constantes:

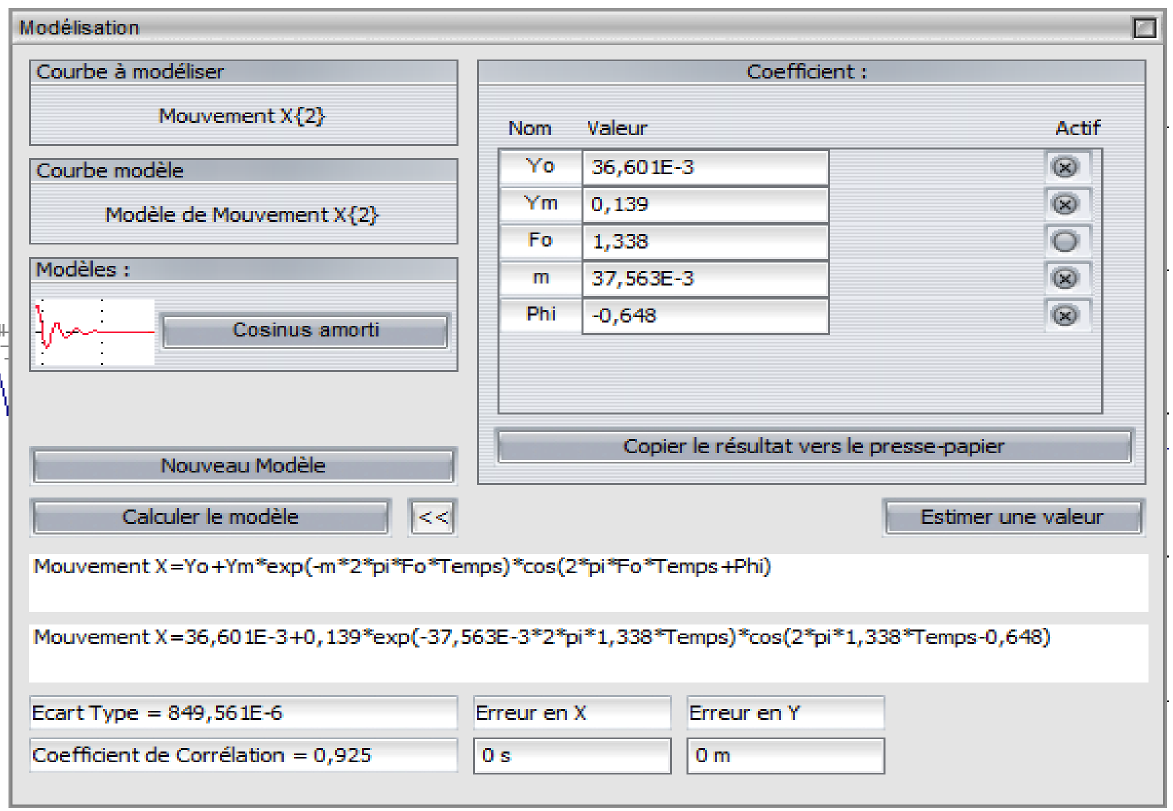
Applications numériques:

$$k = 39 \text{ N.m}$$
$$h = 0{,}37 \text{ N.s.m}^{-1}$$



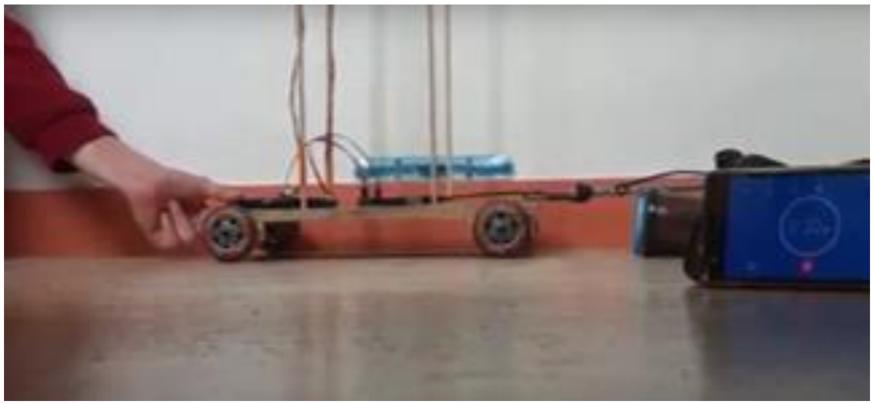
$$(E) : \ddot{f} + 2\xi\omega_0\dot{f} + \omega_0^2 f = 0$$
$$\Leftrightarrow f(t) = y_m \cos(\omega t + \varphi)e^{-\xi\omega_0 t} \; avec \; \omega = \omega_0\sqrt{1-\xi^2}$$

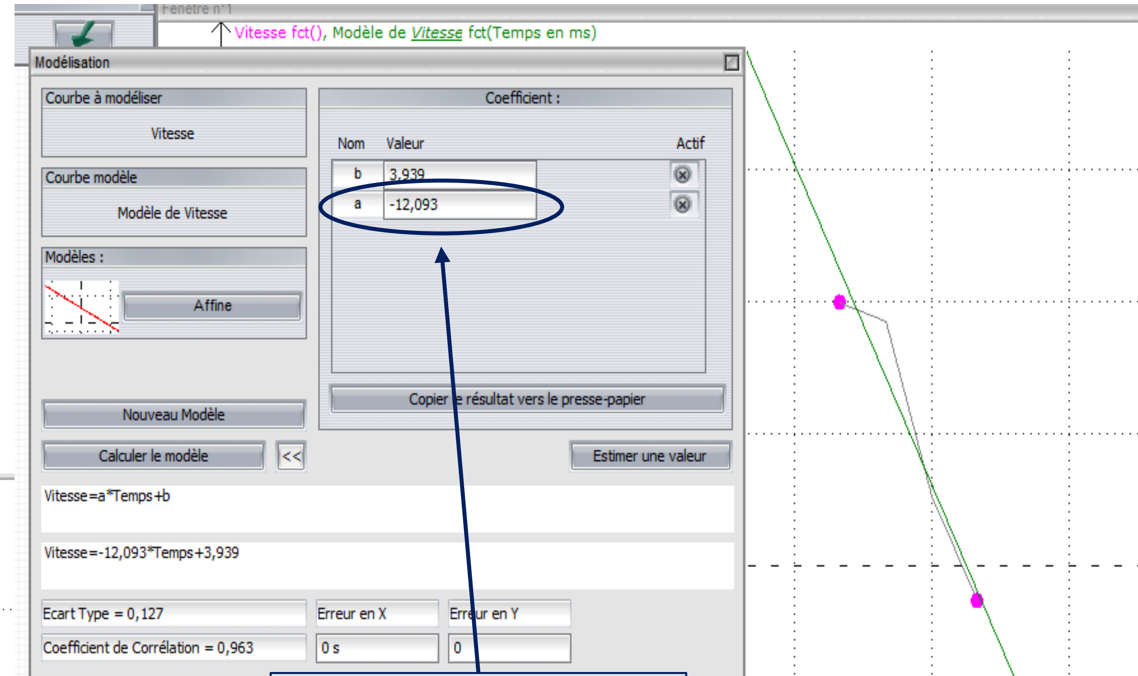


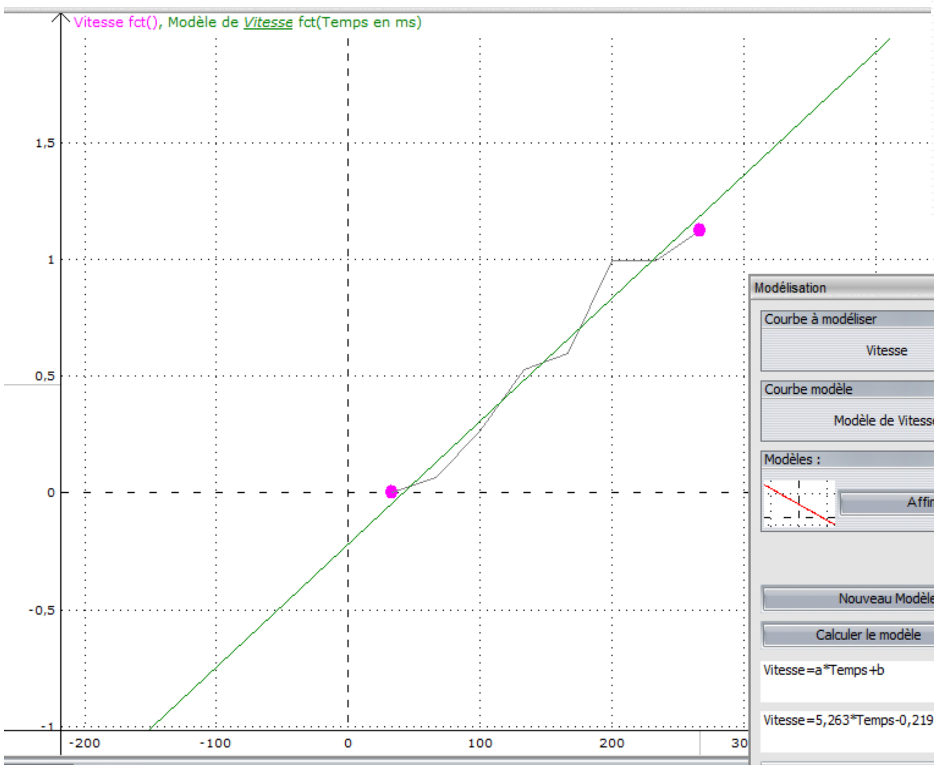


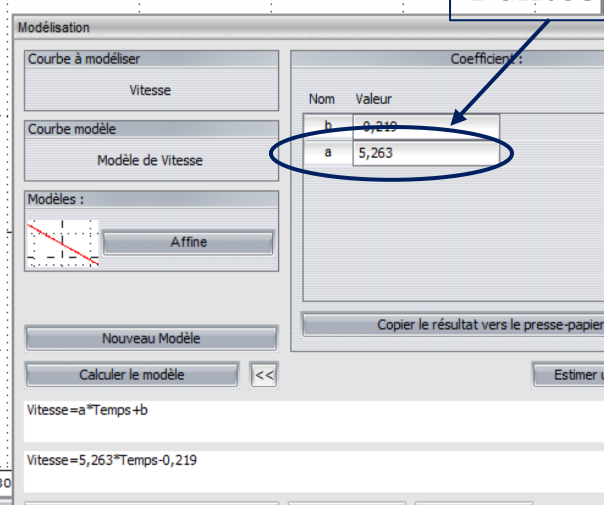
Annexe D

Signal d'entrée en triangle de vitesse:

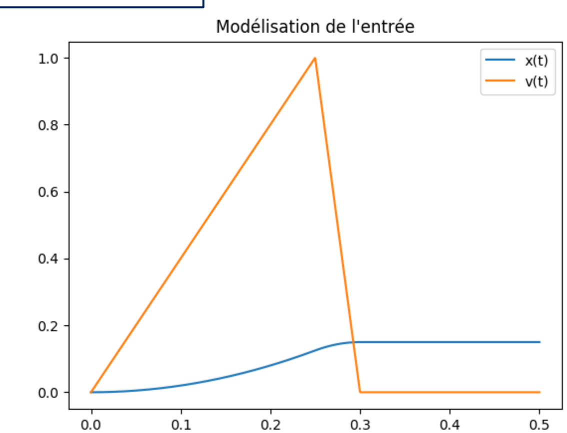








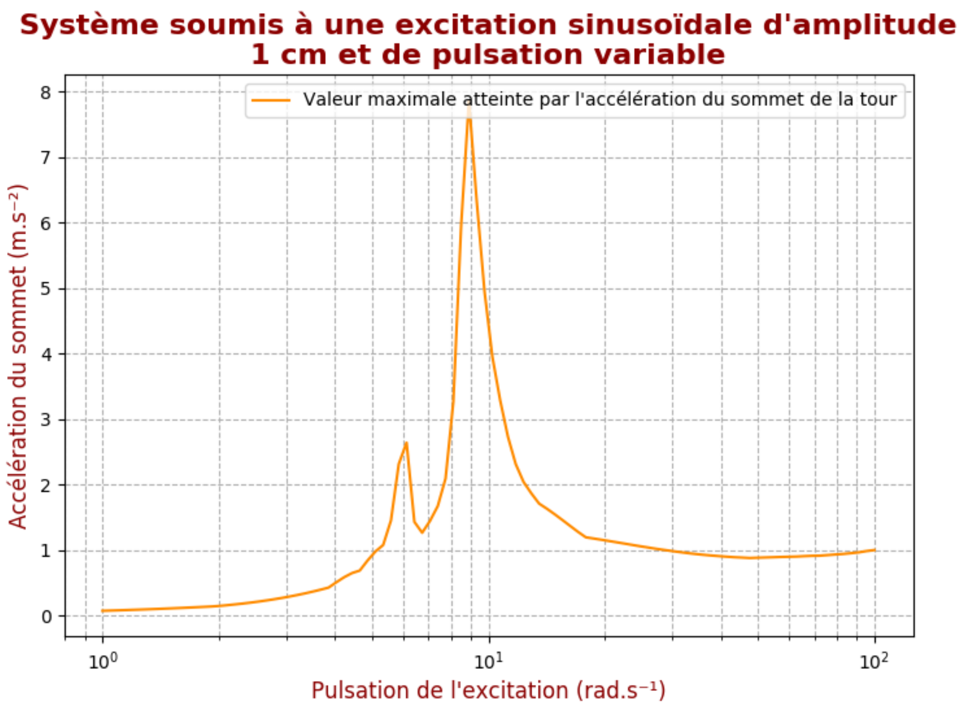
Pentes de la vitesse

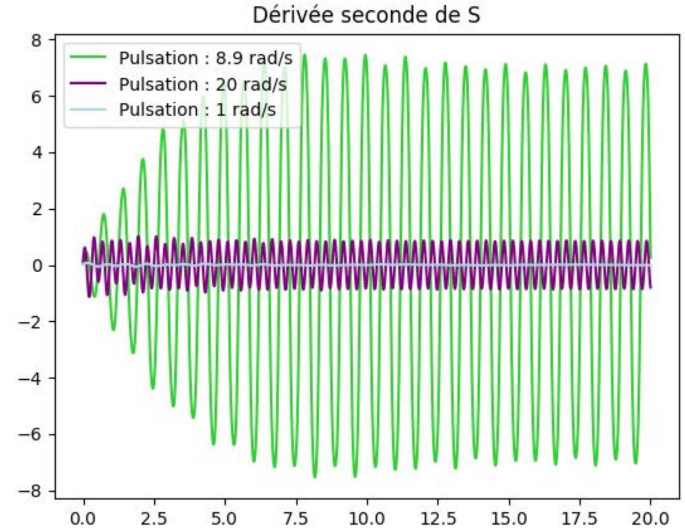


Script Python en annexe I

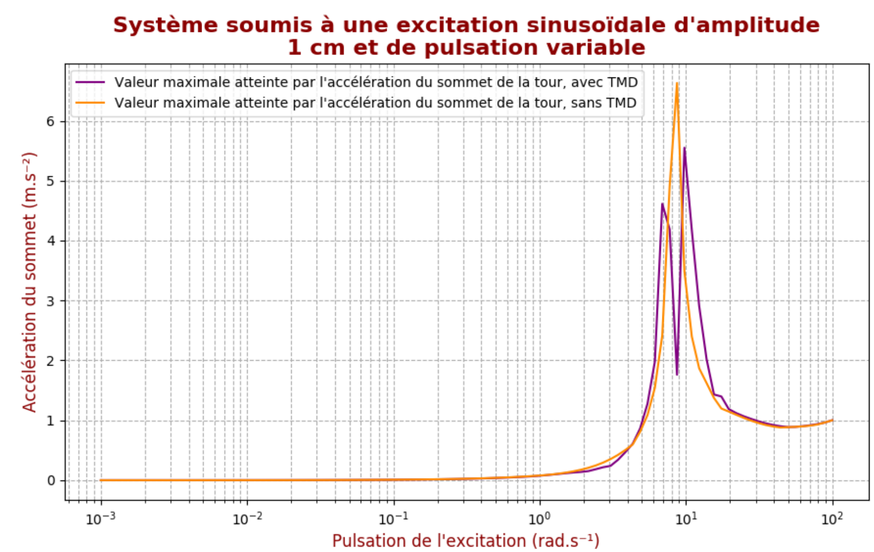
Annexe E

Étude et optimisation harmonique:





t (s)

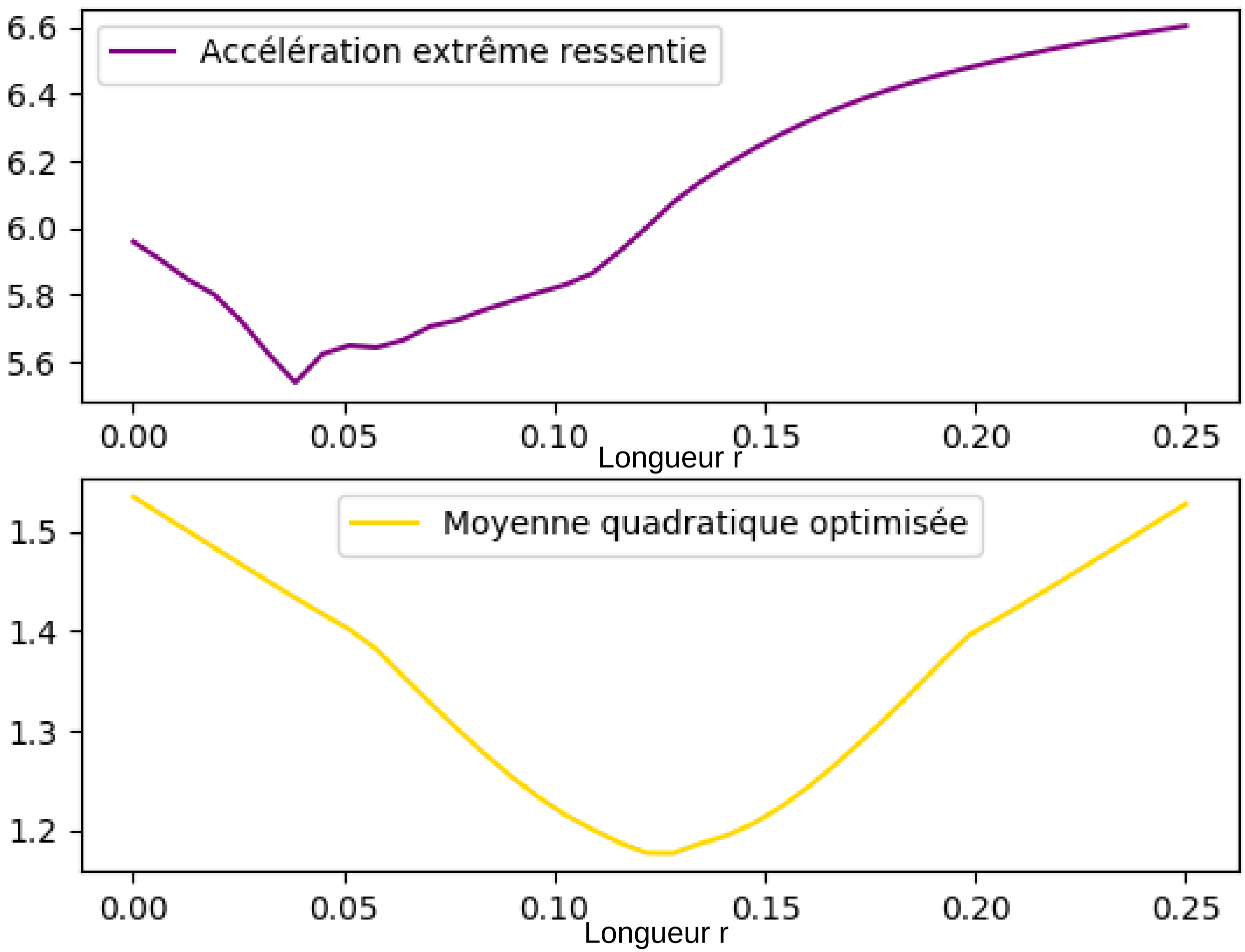


Script Python en annexe L

Avec des paramètres optimisés pour la pulsation de résonance

Annexe F

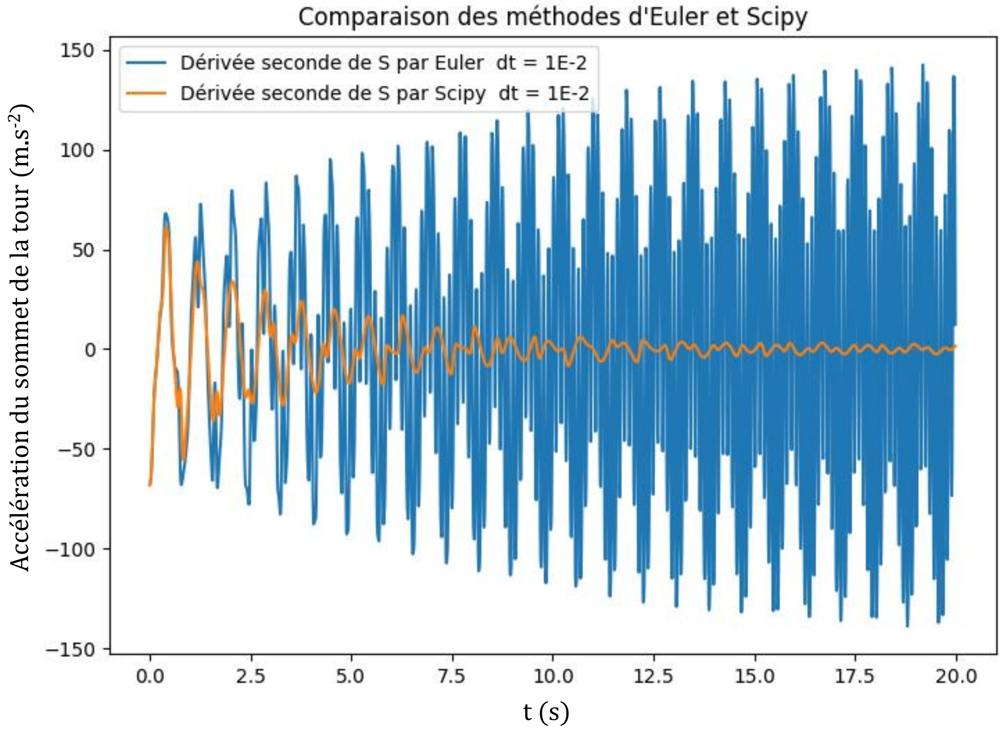
Optimisation à masse fixée:

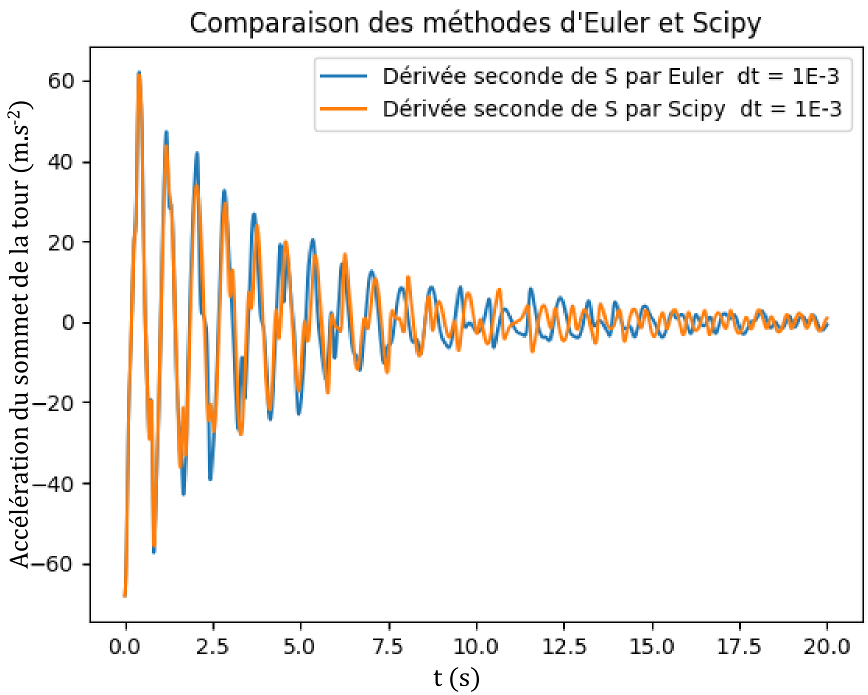


Script Python en annexe M

Annexe G

Comparaison Euler / Scipy:





Script Python en annexe K

Annexe H

Outils modélisation

```python
"""Programme définissant les fonctions utilisées pour la modélisation"""
import numpy as np
import scipy.integrate as integr
import matplotlib.pyplot as plt
"""Constantes de travail"""
m=0.580     # Masse Tour
h=0.372     # Frottements tiges
g=9.81      # Champ de pesenteur terrestre
k=39.46     # Constante de rappel du ressort équivalent
"""Valeurs de la maquette"""
w=1.4E-3         # Frottement pendule
L=.21            # Longueur TMD
md=.085          # Masse TMD
x=lambda t:0     # Peturbation
xp=lambda t:0    # Dérivée de la perturbation
###############
# Utilitaires #
###############
def systeme_spp_opp(Y,t):
    """Renvoie les dérivées secondes à l'instant t"""
    # Système matriciel : A X = B
    # Avec X le vecteur dérivée seconde de théta / dérivée seconde de s
    o,op,s,sp=Y
    #Matrice A
    Aa=md*L
    Ab=np.cos(o)*md
    Ac=md*L*np.cos(o)
    Ad=m+md
    detA=Aa*Ad-Ab*Ac
    assert detA!=0 , "Determinant de A nul !"
    detAi=1/detA #L'inverse de detA qui nous sera utile

    #Matrice B
    Ba=-md*g*np.sin(o)-w*op
    Bb=k*(x(t)-s)+h*(xp(t)-sp)+md*L*np.sin(o)*op**2-w*op*np.cos(o)

    X=[]
    X.append(detAi*(Ad*Ba-Ab*Bb))
    X.append(detAi*(-Ac*Ba+Aa*Bb))
    return(X)

def fCI(o=0,op=0,s=0,sp=0):
    """Renvoie les conditions initiales"""
    return(o,op,s,sp)
```

```python
#################
# Méthode Scipy #
#################
def f(Y,t):
    X=systeme_spp_opp(Y,t)
    return(np.array([Y[1],X[0],Y[3],X[1]]))

def Methode_Scipy(CI,duree,precision):
    T = np.arange(0, duree, precision)
    Y = integr.odeint(f, np.array(CI), T)
    # On veut renvoyer aussi les courbes des dérivées secondes
    OPP=[]
    SPP=[]
    for i in range(len(Y[:])):
        X=systeme_spp_opp(Y[i],T[i])
        OPP.append(X[0])
        SPP.append(X[1])
    V=[Y[:,0],Y[:,1],OPP,Y[:,2],Y[:,3],SPP]
    return(T,V)

##############
# Affichages #
##############

def modification_para(xx=lambda t:0,xpp=lambda t:0,ww=7.57E-4,mdd=.058,LL=.24):
    """ Modifie les données de l'experience"""
    global x,xp,w,md,L
    x,xp,w,md,L=xx,xpp,ww,mdd,LL

def aff(couleur,temps=10,extr=0):
    """Affiche la modélisation de la dérivée seconde de S"""
    T,V=Methode_Scipy(fCI(),temps,1e-3)

    choix=[5]
    textelegende=["Theta","Dérivée de Theta","Dérivée seconde de Theta",
                  "S","Dérivée de S","Dérivée seconde de S"]
    for c in choix:
        plt.plot(T,V[c],label=textelegende[c],color=couleur)

        if extr:
            plt.plot([0,temps],[max(V[c]),max(V[c])],color=couleur)
            plt.plot([0,temps],[min(V[c]),min(V[c])],color=couleur)
    plt.legend()
```

Annexe I

Mise en œuvre des optimisations

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

from OutilsModelisation import *
from AxeArduino import *
#######################################################

def MoyenneQuadratique(Y):
    tot=0
    for i in Y:
        tot+=i
    moy=tot/len(Y)
    totq=0
    for i in Y: totq+=(i-moy)**2
    return np.sqrt(totq/len(Y))
##########################
# Définition des entrées #
##########################
vm=1.1
tm=.233
dt=0
def xx(t):
    if t<= tm:
        return(t**2*vm/(2*tm))
    elif t<= tm+dt:
        return(vm/dt*(t*(tm+dt-t/2)-tm/2*(tm+dt)))
    else:
        return xx(tm+dt)
def xpp(t):
    if t<= tm:
        return(vm/tm*t)
    elif t<= tm+dt:
        return(vm/dt*(tm+dt-t))
    else:
        return(0)

def EchelonPosition(couleur,ww=1.4E-3,mdd=.085,LL=0.21):
    modification_para(xx=xx,xpp=xpp,ww=ww,mdd=mdd,LL=LL)
    aff(couleur)

plt.title("Dérivée seconde de S")
EchelonPosition("orange") # Avec les valeurs de la maquette
EchelonPosition("purple",ww=36e-3) # Frottement w2
plt.show()
```

```
#################
# Optimisations #
#################
def VariationW(mdd=0.085,LL=0.21):
    """ Optimisation à md et L fixés """
    varW=np.linspace(-5,2,200)
    deltaMax=[]
    deltaMax2=[]
    for j in range(len(varW)):
        modification_para(xx,xpp,10**varW[j],mdd,LL)
        Y=Methode_Scipy(fCI(),20,1e-3)[1][5]
        deltaMax.append(max(max(Y),abs(min(Y))))
        deltaMax2.append(MoyenneQuadratique(Y))

    plt.subplot(211)
    plt.plot(varW,deltaMax,color="orange",label="Accélération extrême")
    plt.legend()

    plt.subplot(212)
    plt.plot(varW,deltaMax2,color="purple",label="Moyenne quadratique")

    plt.legend()
    plt.show()
    # On affiche les log des deux frottements optimaux trouvés :
    print(varW[deltaMax.index(min(deltaMax))])
    print(varW[deltaMax2.index(min(deltaMax2))])

#######################################################
def VariationL(mdd=0.087,ww=1.4E-3):
    """ Optimisation à md et w fixés """
    varL=np.linspace(1E-4,.4,200)
    deltaMax=[]
    deltaMax2=[]
    for j in range(len(varL)):
        modification_para(xx,xpp,ww,mdd,varL[j])
        Y=Methode_Scipy(fCI(),20,1e-3)[1][5]
        deltaMax.append(max(max(Y),abs(min(Y))))
        deltaMax2.append(MoyenneQuadratique(Y))

    # On affiche les log des deux frottements optimaux trouvés :
    print(varL[deltaMax.index(min(deltaMax))])
    print(varL[deltaMax2.index(min(deltaMax2))])
    plt.subplot(211)
    plt.plot(varL,deltaMax)
    plt.subplot(212)
    plt.plot(varL,deltaMax2)
    plt.show()
```

Annexe J

Plan d'optimisation

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

from mpl_toolkits.mplot3d import Axes3D
from OutilsModelisation import *

##########
# Entrée #
##########

vm=.1
tm=.5
dt=.05
def xx(t):
    if t<= tm:
        return(t**2*vm/(2*tm))
    elif t<= tm+dt:
        return(vm/dt*(t*(tm+dt-t/2)-tm/2*(tm+dt)))
    else:
        return xx(tm+dt)
def xpp(t):
    if t<= tm:
        return(vm/tm*t)
    elif t<= tm+dt:
        return(vm/dt*(tm+dt-t))
    else:
        return(0)
"""
xx=lambda t : 0.01*np.sin(8.2*t)
xpp=lambda t : 8.2*0.01*np.cos(8.2*t)
"""
```

```
#########################################
# Fonction d'optimisation du frottement #
#########################################

def MoyenneQuadratique(Y):
    """Renvoie la moyenne quadratique de la simulation"""
    tot=0
    for i in Y:
        tot+=i
    moy=tot/len(Y)

    totq=0
    for i in Y:
        totq+=(i-moy)**2
    return np.sqrt(totq/len(Y))

def Optimisation(m_,l_,wmin,wmax,nbW,dureeScipy,precisionScipy,methode):
    """Fonction d'optimisation à m et L fixés"""
    essais = np.linspace(wmin,wmax,nbW)
    listeAcc = []
    for w_ in essais:
        modification_para(xx=xx,xpp=xpp,ww=10**w_,mdd=m_,LL=l_)
        sim = Methode_Scipy(fCI(),dureeScipy,precisionScipy)[1][5]

        if methode:
            ma=abs(max(sim))
            mi=abs(min(sim))
            listeAcc.append(max(ma,mi))
        else : listeAcc.append(MoyenneQuadratique(sim))

    accMinimale = min( listeAcc )
    p=listeAcc.index(accMinimale)
    woptimal= essais[p]

    return (accMinimale,woptimal)
```

Annexe J

Plan d'optimisation

```python
############################
# Surface d'optimisation #
############################

""" Constantes """
nb  = 200  # Nombre de points pour la courbe
m_m  = .085 # Masse maximale du pendule
l_m = .25  # Longueur maximale du pendule
""""""

# Plan M,L :

Axes = Axes3D(plt.figure())

def Plan(Mm,Lm,largeurGrille):
    M_ = np.linspace(1E-4, Mm, largeurGrille)
    L_ = np.linspace(1E-4, Lm, largeurGrille)
    return(M_,L_)
MM,LL=Plan(m_m,l_m,nb)
L,M = np.meshgrid(LL,MM)

# Cotes :

Z = np.zeros([nb,nb],float)
W = np.zeros([nb,nb],float)

""" Constantes """

Cwmin            = -5   # Log du frottement minimal
Cwmax            = 1    # Log du frottement maximal
CnbW             = 100  # Nombre de valeur du frottement essayées
CdureeScipy      = 2    # Durée de la simulation
CprecisionScipy  = 1e-2 # Précision de la simulation

methode          = True # True pour extreme, False pour quadratique
""""""
```

```python
def Cotes():
    c=nb**2
    for i in range(nb):
        for j in range(nb):
            print(c)
            c-=1

            Lz,Lw=OptimisationW(MM[i],LL[j],Cwmin,Cwmax,
                                          CnbW,CdureeScipy,CprecisionScipy,methode)
            Z[i,j]=Lz
            W[i,j]=Lw

# Affichage 3D :

Cotes()

Axes.plot_surface(M,L,Z)
Axes.set_xlabel("Masse pendule (kg)",fontdict={'color':'darkred',
                    'weight': 'bold',
                      'size': 10})
Axes.set_ylabel("Longueur pendule (m)",fontdict={'color':'darkred',
                    'weight': 'bold',
                      'size': 10})
Axes.set_zlabel("Accélération du sommet de la tour (m.s\u207B\u00B2)",
                 fontdict={'color':'darkred',
                    'weight': 'bold',
                      'size': 10})
plt.show()
```

Annexe K

Comparaison Euler / Scipy

```python
"""Modélisation : Comparaison Euler/Scipy"""
import numpy as np
import scipy.integrate as integr
import matplotlib.pyplot as plt
""" Constantes """
g=9.81     # Champ de pesenteur terrestre
m=.58      # Masse Tour
k=39.46    # Constante de rappel du ressort équivalent au bâtiment
h=.372     # Coefficient du frottement dans les tiges
md=.085    # Masse TMD
L=.21      # Longueur pendule (TMD)
w=7E-4     # Frottement dans la liaison pendule

##############
# Utilitaires
##############
def systeme_spp_opp(Y):
    """Renvoie les dérivées secondes de theta et s à l'instant t"""
    # Système matriciel : A X = B
    # Avec X le vecteur dérivée seconde de théta / dérivée seconde de s
    o,op,s,sp=Y

    #Matrice A
    Aa=md*L
    Ab=np.cos(o)*md
    Ac=md*L*np.cos(o)
    Ad=m+md
    detA=Aa*Ad-Ab*Ac

    assert detA!=0 , "Determinant de A nul !"
    detAi=1/detA #L'inverse de detA qui nous sera utile

    #Matrice B
    Ba=-md*g*np.sin(o)-w*op
    Bb=-k*s-h*sp+md*L*np.sin(o)*op**2-w*op*np.cos(o)

    X=[]
    X.append(detAi*(Ad*Ba-Ab*Bb))
    X.append(detAi*(-Ac*Ba+Aa*Bb))
    return(X)

def fCI(o=0,op=0,s=0,sp=0):
    """Renvoie les conditions initiales"""
    return(o,op,s,sp)
```

```python
################
# Méthode Scipy
################

def f(Y,t):
    X=systeme_spp_opp(Y)
    return(np.array([Y[1],X[0],Y[3],X[1]]))

def Methode_Scipy(CI,duree,dt):
    T = np.arange(0, duree, 10**-dt)
    Y = integr.odeint(f, np.array(CI), T)

    # On veut renvoyer aussi les courbes des dérivées secondes
    OPP=[]
    SPP=[]
    for i in range(len(Y[:])):
        X=systeme_spp_opp(Y[i])
        OPP.append(X[0])
        SPP.append(X[1])

    V=[Y[:,0],Y[:,1],OPP,Y[:,2],Y[:,3],SPP]
    return(T,V,dt," par Scipy ")

################
# Méthode Euler
################

def Methode_Euler(CI,duree,dt):
    l=[0,1,3,4]#Les grandeurs auxquelles on applique Euler

    T = np.arange(0, duree, 10**-dt)
    V=[[CI[0]],[CI[1]],[],[CI[2]],[CI[3]],[]]#Conditions initiales
    for t in T:
        Y=[V[0][-1],V[1][-1],V[3][-1],V[4][-1]]
        X=systeme_spp_opp(Y)#Le vecteur dérivée seconde

        V[2].append(X[0])#opp
        V[5].append(X[1])#spp
        #Mise en place Euler
        for i in l:
            V[i].append(V[i][-1]+V[i+1][-1]*10**-dt)
    for i in l:#Les grandeurs non dérivées seconde ont une valeur en trop
        V[i].pop()
    return(T,V,dt," par Euler ")
```

Annexe K

Comparaison Euler / Scipy

```
#############
# Affichage #
#############

def affichage_courbes(donnees,choix):
    """Choix des courbes"""
    T,V,precision,nom_methode=donnees


    textelegende=["Theta","Dérivée de Theta",
                  "Dérivée seconde de Theta","S","Dérivée de S","Dérivée seconde de S"]
    for c in choix:
        plt.plot(T, V[c],label=textelegende[c]+nom_methode+" dt = 1E-"+str(precision))
    plt.legend()

def comparaison(comp):
    # comp contient le code de la méthode : 0 pour Euler, 1 pour Scipy
    # et la précision pour chaque métode

    for c in comp:
        if c[0]==0:
            affichage_courbes(Methode_Euler(fCI(s=1),20,c[1]),[5])
        else:
            affichage_courbes(Methode_Scipy(fCI(s=1),20,c[1]),[5])
    plt.title("Comparaison des méthodes d'Euler et Scipy")
    plt.show()

comparaison([(1,4),(1,3)])
```

Annexe L

Étude harmonique

```python
"""Étude harmonique"""

import numpy as np
import scipy.integrate as integr
import matplotlib.pyplot as plt

m=0.580 #Masse Tour
h=0.372 #Frottement tiges
g=9.81 #Champ de pesenteur terrestre
k=40 #Constante de rappel du ressort équivalent

w=1e8#1.4E-3 #Frottement pendule
L=0.21 #Longueur TMD
md=1e-4#0.085 #Masse TMD
x= lambda t: 0  #Excitation
xp= lambda t: 0 #Dérivée de l'exciation

##########################################################
# Fonctions similaires à celles présentes dans les outils #
##########################################################

def systeme_spp_opp(s,sp,o,op,t):
    Aa=md*L
    Ab=np.cos(o)*md
    Ac=md*L*np.cos(o)
    Ad=m+md
    detA=Aa*Ad-Ab*Ac
    detAi=1/detA #L'inverse de detA qui nous sera utile
    #Matrice B
    Ba=-md*g*np.sin(o)-w*op
    Bb=k*(x(t)-s)+h*(xp(t)-sp)+md*L*np.sin(o)*op**2-w*op*np.cos(o)
    X=[]
    X.append(detAi*(Ad*Ba-Ab*Bb))
    X.append(detAi*(-Ac*Ba+Aa*Bb))
    return(X)
def fCI(o=0,op=0,s=0,sp=0):
    return(o,op,s,sp)

################

def f(Y,t):
    X=systeme_spp_opp(Y[2],Y[3],Y[0],Y[1],t)
    return(np.array([Y[1],X[0],Y[3],X[1]]))
```

Annexe L

Étude harmonique

```
def MethodeScipy(CI,duree):
    T = np.arange(0, duree, 10**-2.5)
    Y = integr.odeint(f, np.array(CI), T)
    #Acceleration:
    opp=[]
    spp=[]
    for i in range(len(Y[:,0])):
        X=systeme_spp_opp(Y[i,2],Y[i,3],Y[i,0],Y[i,1],T[i])
        opp.append(X[0])
        spp.append(X[1])
    ma=max(spp)
    mi=abs(min(spp))
    return(max(ma,mi))
#############
# Resonance #
#############
def resonance():
    global x,xp
    essais= 10**np.linspace(0,2,100)
    reponses = []
    for j in range(len(essais)) :
        print(100-j)
        i=essais[j]
        x= lambda t: np.sin(i *t)*0.01
        xp= lambda t: i* np.cos(i *t)*0.01
        a=MethodeScipy(fCI(),10)
        reponses.append(a)
    plt.plot(essais,reponses,label="Valeur maximale atteinte par l'accélération du sommet de la tour" ,color='darkorange')
    plt.legend()
    plt.grid(True,which="both",linestyle='--')
    plt.xscale('log')
    plt.xlabel("Pulsation de l'excitation (rad.s\u207B\u00B9)",fontdict={'color':'darkred',
              'weight': 'normal',
                'size': 12})
    plt.ylabel("Accélération du sommet (m.s\u207B\u00B2)",fontdict={'color':'darkred',
              'weight': 'normal',
                'size': 12})
    plt.title("Système soumis à une excitation sinusoïdale d'amplitude\n1 cm et de pulsation variable",
          fontdict={'color':'darkred',
              'weight': 'bold',
                'size': 16})
    plt.show()
    print(max(reponses),essais[list(reponses).index(max(reponses))])
resonance()
```

Annexe M

Étude à masse fixée

```
import numpy as np
import matplotlib.pyplot as plt
from OutilsModelisation import *

##########
# Entrée #
##########
vm=1.1
tm=.233
dt=0
def xx(t):
    if t<= tm:
        return(t**2*vm/(2*tm))
    elif t<= tm+dt:
        return(vm/dt*(t*(tm+dt-t/2)-tm/2*(tm+dt)))
    else:
        return xx(tm+dt)
def xpp(t):
    if t<= tm:
        return(vm/tm*t)
    elif t<= tm+dt:
        return(vm/dt*(tm+dt-t))
    else:
        return(0)

#xx=lambda t : 0.01*np.sin(8.9*t)
#xpp=lambda t : 8.9*0.01*np.cos(8.9*t)
```

Annexe M

Étude à masse fixée

```
##################################
# Fonction d'optimisation de w #
##################################
def MoyenneQuadratique(Y):
    """Renvoie la moyenne quadratique de la simulation"""
    tot=0
    for i in Y:
        tot+=i
    moy=tot/len(Y)

    totq=0
    for i in Y:
        totq+=(i-moy)**2
    return np.sqrt(totq/len(Y))

def Optimisation(l_,wmin,wmax,nbW,dureeScipy,precisionScipy):

    essais = np.linspace(wmin,wmax,nbW)
    listeAccExtreme1 = []
    listeAccExtreme2 = []
    for w_ in essais:
        modification_para(xx=xx,xpp=xpp,ww=10**w_,mdd=m_,LL=l_)
        sim = Methode_Scipy(fCI(),dureeScipy,precisionScipy)[1][5]

        ma=abs(max(sim))
        mi=abs(min(sim))
        listeAccExtreme1.append(max(ma,mi))
        listeAccExtreme2.append(MoyenneQuadratique(sim))

    accExtremeMinimale1 = min( listeAccExtreme1 )
    p1=listeAccExtreme1.index(accExtremeMinimale1)
    woptimal1= essais[p1]

    accExtremeMinimale2 = min( listeAccExtreme2 )
    p2=listeAccExtreme2.index(accExtremeMinimale2)
    woptimal2= essais[p2]

    return ((accExtremeMinimale1,accExtremeMinimale2),(woptimal1,woptimal2))
```

```
#########################
# Courbe d'optimisation #
#########################

""" Constantes """
nb  = 200  # Nombre de points pour la courbe
m_  = .085 # Masse fixée
l_m = .25  # Longueur maximale du pendule

""""""""

# Valeurs de L :
L = np.linspace(1E-4,l_m,nb)

# Listes :
Z1 = np.zeros(nb,float)
W1 = np.zeros(nb,float)

Z2 = np.zeros(nb,float)
W2 = np.zeros(nb,float)
""" Constantes """
Cwmin            = -5   # Log du frottement minimal
Cwmax            = 1    # Log du frottement maximal
CnbW             = 100  # Nombre de valeur du frottement essayées
CdureeScipy      = 20   # Durée de la simulation
CprecisionScipy  = 1e-3 # Précision de la simulation
""""""""
def GenerationListes():
    for i in range(nb):
        Lz,Lw=Optimisation(L[i],Cwmin,Cwmax,CnbW,CdureeScipy,CprecisionScipy)
        Z1[i]=Lz[0]
        Z2[i]=Lz[1]
        W1[i]=Lw[0]
        W2[i]=Lw[1]

# Affichage
GenerationListes()

plt.subplot(211)
plt.plot(L,Z1)

plt.subplot(212)
plt.plot(L,Z2)
plt.show()
```

Annexe N

Exploitation des fichiers Arduino

```
import matplotlib.pyplot as plt

def Arduino1axe(Serie):
    L=[]
    doc="/Users/maxencerichard/Documents/PSI*/TIPE/CapturesCoolTerm/Capture"+str(Serie)
    with open(doc,'r') as file:
        lignes=file.readlines()
    for i in range(3,len(lignes)-2):
        #La dernière donnée est souvent intraitable car tronquée lors de son écriture
        #Les trois premières lignes sont des lignes d'informations sur la série de données
        L.append(float((int(lignes[i][:-1])-338)*9.81/80))
    absc=[i/40 for i in range(len(L))]
    return(absc,L)

choix=["3","4","5","6"]
for c in choix:
    X,Y=Arduino1axe("N0"+c+".txt")
    plt.plot(X,Y)
plt.show()
```

Sources:

Diapositive 1:

Par Armand du Plessis — Travail personnel, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=10591308

Diapositive 2:

Image tirée de : Connor, Jerome J., Structural Motion Engineering, chapitre 4 : Intro to structural motion control
Par Freshgod — Travail personnel and parts from Comparaison gratte-ciels from Freshgod, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=26037710
Par Someformofhuman — Travail personnel, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=3799263

Diapositive 4:

Image tirée de : Connor, Jerome J., Structural Motion Engineering, chapitre 4 : Intro to structural motion control

Diapositive 7:

Par Someformofhuman — Travail personnel, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=3799263

Diapositive 22:

Par Someformofhuman — Travail personnel, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=3799263
Image tirée de : Ioannis Kourakis : Thèse : Structural Systems and Tuned Mass Dampers of Super-Tall Buildings : Case Study of Taipei 101