TP 9

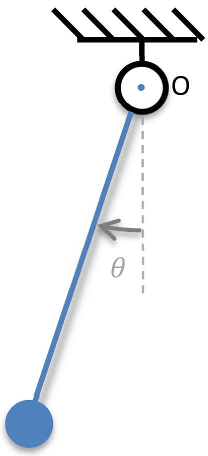
Résolution numérique d'une équation du second ordre non linéaire, le cas du pendule.

Objectifs:

1) Résoudre une équation différentiel non linéaire du second ordre
2) Comparaison de la solution du problème non linéaire avec la solution du problème linéarisé

1 *Mise en situation*

Même s'il peut sembler basique, le problème du pendule permet d'appréhender la différence qui existe entre équations différentielles linéaires et non linéaires, et donc de détecter les limites des approximations linéaires. En effet, traditionnellement, on linéarise l'équation pour faciliter sa résolution. D'autre part, d'un point de vue technique, il existe de nombreux systèmes munis de masses pendues (grues, hélicoptères transportant une charge, systèmes anti-séisme…) dont l'étude du comportement dynamique permet de dimensionner les pièces mécaniques et de concevoir les calculateurs de tels systèmes. Par exemple ceux permettant de limiter les oscillations.



Modèle : Soit un pendule de masse $500\ g$, dont la tige a une longueur $l = 10\ cm$ et une masse négligeable. On suppose également les frottements négligeables (air et liaison en O).

Les équations de la dynamique mènent à l'équation différentielle suivante :

$$l\,\ddot{\theta}(t) + g\,sin(\theta(t)) = 0,$$

où $g = 9{,}81\ m.s^{-2}$ est l'accélération de la pesanteur.

On prendra dans la suite du sujet la condition initiale $\dot{\theta}(0) = 0\ rad.s^{-1}$. On notera θ_0 la valeur initiale de θ.

On notera $\omega(t)$ la vitesse de rotation ($\omega(t) = \dot{\theta}(t)$).

2 *Travail demandé*

Travail 1. Ecrire la modélisation proposée, en faisant l'approximation que θ est petit, sous la forme d'un problème de Cauchy. On notera $f_1(t, \theta(t))$ la fonction associée.

Travail 2. Ecrire la fonction python notée `f1(t,theta)` ou et renvoyant $f_1(t, \theta(t))$.

On effectuera les simulations de façon à observer environ 3 oscillations du pendule. On rappelle que la pulsation des oscillations est d'environ $\omega_O = \sqrt{g/l}$.

Travail 3. Après avoir donné les relations de récurrence associées, écrire la fonction python `Euler2(f,h,theta0,w0,tf)` permettant de calculer l'évolution approchée de $\theta(t)$ en utilisant le schéma d'Euler explicite (en version non vectorielle). `f` représente la fonction du problème de Cauchy, h le pas de temps, `theta0` et `w0` les position et vitesse initiales, `tf` la durée de la simulation. La fonction devra renvoyer les trois listes des valeurs du temps, de $\theta(t)$ et de $\omega(t)$.

Travail 4. En exploitant la fonction `Euler2` , tracer l'évolution de $\theta(t)$ en prenant $\theta(0) = \pi/6$, des pas de temps de $0{,}005\ s$, de $0{,}001\ s$ puis de $0{,}0001\ s$. Commenter les résultats obtenus.

Travail 5. Que se passe-t-il pour un pas de temps de $0{,}05\ s$.

Travail 6. Reprendre les questions 1 et 2 sans faire d'approximation sur θ. On notera $f_2(t, \theta(t))$ la fonction du problème de Cauchy associé. Comparer les courbes obtenues pour les versions linéaires et non-linéaires en prenant un pas de temps de $0{,}0001\ s$, et en prenant $\theta(0) = \pi/15$ puis $\theta(0) = \pi/1.1$.

3 *Pour aller plus loin*

On suppose maintenant qu'un couple de frottement sec constant noté $C_r = 0{,}05\ Nm$ s'exerce au niveau de la liaison avec le support en O. L'équation devient alors :

$$m\, l^2\, \ddot{\theta}(t) + m\, g\, l\, sin\big(\theta(t)\big) + a\dot{\theta}(t) = 0.$$

Avec $a = 2\ N.m.s.rad^{-1}$

Travail 7. Tracer l'évolution approchée de $\theta(t)$ en faisant l'approximation que θ est petit, et en prenant $\theta(0) = \pi/2$.

Travail 8. Reprendre le travail précédent, mais en utilisant Euler implicite, en version linéarisée.

Annexe NUMPY

`import numpy as np` → charge le module numpy sous le nom **np**

Construction de tableaux (de type ndarray)

`np.zeros(n)` → crée un vecteur dont les n composantes sont nulles

`np.zeros((n,m))` → crée une matrice $n\times m$, dont les éléments sont nuls

`np.eye(n)` → crée la matrice identité d'ordre n

`np.linspace(a,b,n)` → crée un vecteur de n valeurs régulièrement espacées de a à b

`np.arange(a,b,dx)` → crée un vecteur de valeurs de a incluse à b exclue avec un pas dx

`M.shape` → tuple donnant les dimensions de M

`M.size` → le nombre d'éléments de M

`M.ndim` → le nombre de dimensions de M

`M.sum()` → somme de tous les éléments de M

`M.min()` → plus petit élément de M

`M.max()` → plus grand élément de M

argument **axis** optionnel : 0 → lignes, 1 → colonnes :

`M.sum(0)` → somme des lignes

`M.min(0)` → plus petits éléments, sur chaque colonne

`M.max(1)` → plus grands éléments, sur chaque ligne

`import numpy.linalg as la`

`la.det(M)` → déterminant de la matrice carrée M

`la.inv(M)` → inverse de M

`la.eigvals(M)` → valeurs propres de M

`la.matrix_rank(M)` → rang de M

`la.matrix_power(M,n)` → M^n (n entier)

`la.solve(A,B)` → renvoie X tel que $A\,X = B$

`import scipy.integrate as spi`

`spi.odeint(F,Y0,LT)` → renvoie une solution numérique du problème de Cauchy $\mathbf{Y}'(t) = \mathbf{F}(\mathbf{Y}(t),t)$, où $\mathbf{Y}$ est un vecteur d'ordre n, avec la condition initiale $\mathbf{Y}(t_0)$ = `Y0`, pour les valeurs de t dans la liste `LT` de longueur k commençant par t_0, sous forme d'une matrice $n\times k$

`spi.quad(f,a,b)` → renvoie une évaluation numérique de l'intégrale : $\int_a^b f(t)\,dt$

Conversion ndarray <-> liste

`V = np.array([1,2,3])` → V : vecteur $(1 \ 2 \ 3)$

`L = V.tolist()` → L : liste [1, 2, 3]

`M = np.array([[1,2],[3,4]])` → M : matrice $\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$

`L = M.tolist()` → L : liste [[1, 2], [3,4]]

Extraction d'une partie de matrice

`M[i], M[i,:]` → ligne de M d'index i

`M[:,j]` → colonne de M d'index j

`M[i:i+h,j:j+l]` → sous-matrice $h\times l$

Copier un tableau avec la méthode copy :

`M2 = M1.copy()`

`M1+M2, M1*M2, M**2` → opérations « terme-à-terme »

`c*M` → multiplication de la matrice M par le scalaire c

`M+c` → matrice obtenue en ajoutant le scalaire c à chaque terme de M

`V1.dot(V2)`, `np.dot(V1,V2)` → renvoie le produit scalaire de deux vecteurs

`M.dot(V)`, `np.dot(M,V)` → renvoie le produit d'une matrice par un vecteur

`M1.dot(M2)`, `np.dot(M1,M2)` → renvoie le produit de deux matrices

`M.transpose()`, `np.transpose(M)` → renvoie une copie de M transposée (ne modifie pas M)

`M.trace()`, `np.trace(M)` → renvoie la trace de M

Fonctions mathématiques usuelles

`np.exp, np.sin, np.cos, np.sqrt` *etc.*

→ fonctions qui s'appliquent sur des réels ou des complexes, mais aussi sur des vecteurs et des matrices (s'appliquent à chaque terme), qui sont optimisées en durée de calcul.

Rappel : ce mémento est fourni à titre indicatif. Il ne faut le considérer ni comme exhaustif, ni comme exclusif, ni comme un minimum à connaître absolument (l'examinateur n'attend pas du candidat qu'il connaisse parfaitement toutes ces fonctions et ces commandes).