Simulations sur Python séance 2/2

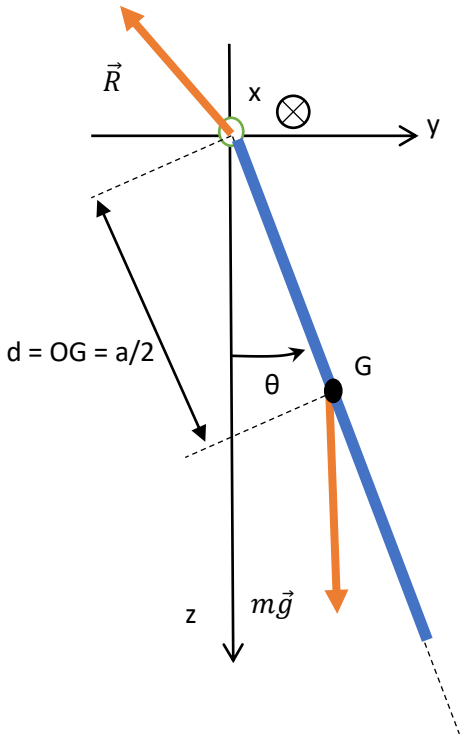
Cette seconde séance a pour objectif de prolonger les études précédentes, en développant les méthodes nécessaires à l'intégration d'équations différentielles du second ordre.

Elle sera l'occasion de soulignez les effets observables dans le cas de non linéarité d'un système décrit par une équation différentielle du second ordre.

La situation d'application sera celle d'un pendule pesant.

Des indications sont fournies pour guider la rédaction des programmes répondant aux questions traitées. Des fichiers de correction sont disponibles sur le réseau dans le dossier PCSI.

1. Pendule pesant, étude du mouvement par simulation.



Une barre homogène de longueur a et de masse m est suspendue par une liaison pivot idéale placée à son extrémité, à un support fixe.

On repère la position de la barre par l'angle θ qu'elle fait avec la verticale descendante.

On rappelle l'expression de son moment d'inertie J par rapport à son axe de rotation, dans ces conditions :

$$J = ma^2/3.$$

On note g = 9,8 m.s^{-2} l'accélération de la pesanteur.

On suppose a = 50 cm et m = 0,20 kg ; soit J = 1,67.10^{-2} kg.m^2

L'écriture du Théorème du Moment Cinétique amène en projection sur l'axe (Ox) du mouvement :

$$\ddot{\theta} + \frac{mga}{2J}.sin\theta = 0$$

On se propose de construire une simulation de ce mouvement, en employant la méthode d'Euler. Celle-ci n'étant applicable a priori que pour des équations différentielles du premier ordre, on développe l'équation différentielle précédente en un système différentiel :

$$\dot{\omega} + \frac{mga}{2J}.sin\theta = 0 \; ; \omega = \dot{\theta}$$

1. Exprimer ces équations différentielles en notation de Leibnitz (on fait apparaître explicitement la dérivée temporelle comme d/dt) et en déduire les relations différentielles :
   dθ = ω.dt et dω = - K.sinθ.dt où K = mga/(2J).

2. On discrétise ces relations en identifiant la quantité différentielle dt à une petite durée Δt. Dans ces conditions, la quantité différentielle dθ correspondra à θ(t+Δt) – θ(t) et la quantité différentielle dω correspondra à ω(t+Δt) – ω(t).
   θ(t+Δt) – θ(t) = ω(t).Δt et ω(t+Δt) – ω(t) = - K.sin(θ(t)).Δt

   En déduire une relation itérative permettant le calcul des valeurs prises par θ et ω aux différents instants, connaissant les conditions initiales du mouvement.

3. Traduire cette démarche en un programme sur Python permettant le tracé par simulation de l'évolution temporelle θ = θ(t) ainsi que l'évolution temporelle de la vitesse angulaire ω = ω(t), en choisissant un jeu de conditions initiales adéquates, ainsi qu'une durée de simulation pertinente.

4. Tester différentes conditions initiales, et les interpréter physiquement :
   vitesse angulaire initiale nulle : $\dot{\theta} = 0$; valeurs initiales pour θ : 10° ; 25° ; 45° ; 90° ; 175°.

   Commenter les résultats observés. En quoi constate-t-on un effet non linéaire ? Pour quelles conditions est-il observable ?

5. La trajectoire de phase est définie comme le graphe ω = $\dot{\theta}$ en fonction de θ. La tracer pour quelques conditions initiales. Quelle est son allure pour θ faible ? Expliquer en vous appuyant sur l'équation horaire θ(t) obtenue pour θ restant suffisamment faible.

Indications :

Il est nécessaire de déclarer les valeurs initiales θ_o sur θ et ω_o sur ω.

Le calcul itératif pourra être conduit à l'aide d'une boucle « while » (tant que…). Sa syntaxe se résume à :

```
while (condition) :
    Réaliser une succession d'opérations
```

Les ordres inclus dans la boucle doivent bien sûr contenir un évaluateur dont la valeur évoluera jusqu'à satisfaire la condition d'arrêt.

On créera des listes de valeurs pour θ et ω, ne contenant initialement que les valeur initiales θ_o et ω_o de la position angulaire θ et de la vitesse angulaire ω.

La fonction « append » permet d'adjoindre un nouvel élément à une liste : liste.append(élément)

Tous les paramètres doivent avoir une valeur numérique. K = 3g/(2a) = 29,4 s^{-2}.

Pour évaluer un ordre de grandeur de la durée de simulation t_{max}, on peut linéariser l'équation du mouvement à θ faible :

$$\ddot{\theta} + K.\theta = 0$$

On déduit alors une solution sinusoïdale de période :

$$T_o = 2\pi.\sqrt{\frac{2a}{3g}} = 1{,}2\ s$$

Une durée t_{max} de quelques périodes sera convenable. Un pas de calcul Δt de l'ordre du millième de T donnera une précision satisfaisante : Δt = 0,001 s.

Il peut être pertinent, mais pas nécessaire, de créer une fonction « simulation » contenant la procédure exposée, dépendant des paramètres inventoriés précédemment, ce qui permettra d'exécuter des tracés pour diverses conditions.

Solution rédigée pour débugage éventuel :

```
Deltat = 0.001
theta0 = 0
omega0 = 3
t=0

theta = theta0
omega = omega0
tmax = 5

angle = [theta0]
vitang = [omega0]
temps = [0]

while t < tmax:
   t = t + deltat
   theta = theta + omega*deltat
   omega = omega – 29.4*sin(theta)*deltat
   temps.append(t)
   angle.append(theta)
   vitang.append(omega)

plot(temps, angle)
show()
```

2. **Pendule pesant, prise en compte des frottements.**
   Le modèle théorique va faire intervenir un couple de frottement fluide s'opposant au mouvement, dont le moment s'écrit : - h. $\dot{\theta}\ \overrightarrow{e_x}$, l'unitaire $\overrightarrow{e_x}$ étant porté par l'axe (Ox) autour duquel a lieu la rotation.

   Modifier les relations précédentes (et le programme en conséquence) de façon à prendre en compte un frottement linéaire de moment : - h. $\dot{\theta}$.

   Tester numériquement pour diverses valeurs de h : h = 0,9 Nms ou h = 0,18 Nms ou h = 0,018 Nms ; valeurs renvoyant aux trois types classiques de réponse d'un système dynamique stable : régimes apériodique, critique et pseudo-périodique.

   Procéder au tracé des courbes θ(t), ω(t) et ω en fonction de θ pour ces trois cas de figure.

3. Effets non linéaires : le non isochronisme des oscillations.

3.1 Comparaison des mouvements selon leur amplitude.

On reprend l'équation du mouvement du pendule pesant, en l'absence de frottements :

$$\ddot{\theta} + \frac{mga}{2J}.sin\theta = 0$$

$$soit\ avec\ J\ =\ ma^2/3 :\quad \ddot{\theta} + \frac{3g}{2a}.sin\theta = 0$$

Et l'on souhaite tracer sur un même graphe l'évolution θ(t) pour des conditions initiales mettant en jeu une vitesse angulaire initiale nulle et une valeur d'angle initial 0,1 rad ; 0,3 rad ; 1 rad ; 1,5 rad ; 3 rad et 3,13 rad, ce qui permettra de comparer visuellement leurs périodes respectives.

Nous allons employer la commande odeint, disponible dans la bibliothèque scipy.integrate pour conduire cette résolution.

La syntaxe de cette commande est relativement simple pour le cas d'une équation différentielle du premier ordre portant sur une fonction y(t), de forme : dy/dt = F(y, t) où F(y, t) est une fonction de y(t) et de t exprimant la relation entre la dérivée dy/dt et la fonction y(t) ainsi que la variable t.

Définir d'abord la fonction F :

```
def F(y,t) ;
        dy_dt = …
        return dy_dt
```

Déterminer un intervalle de temps t, avec un pas d'incrémentation pertinent commandé par le nombre n d'intervalles, en créant un tableau de valeurs pour t : t = np.linspace($t_{initial}$, t_{final}, n)

Appeler la commande odeint selon la syntaxe : Y = odeint(F, y0, t)

F fait référence à la fonction définie plus haut, t est la liste des instants pour lesquels le calcul de y(t) est demandé et Y sera la liste des valeurs solution. La valeur initiale de y(t) est y0.

L'équation du mouvement étant cependant d'ordre 2, elle va se scinder en un système de deux équations différentielles d'ordre 1, auquel il va falloir adapter la syntaxe précédente.

$$\dot{\omega} + \frac{3g}{2a}.sin\theta = 0\ \ ;\ \ \omega\ =\ \dot{\theta}$$

La résolution vise donc à déterminer deux fonctions du temps θ(t) et ω(t). La grandeur y à intégrer par la commande odeint va être remplacée par une grandeur vectorielle sys, à deux coordonnées : sys = [θ, ω].

La fonction F qui est définie dans cette syntaxe doit associer à chaque coordonnée l'expression de sa dérivée temporelle, issue des équations différentielles précédentes.

Les conditions initiales sont fournies par un tableau (array) contenant les deux valeurs initiales de θ et ω.

Utiliser cette procédure pour tracer sur un même graphe θ(t) pour les différentes conditions initiales (employer une boucle for). Comparer les périodes observées et l'allure des évolutions temporelles.

En cas de difficulté, on peut se reporter au programme ci-dessous, qui réalise les opérations souhaitées :

```
from scipy import *
from pylab import *
from scipy.integrate import odeint

#courbe d'évolution temporelle"
clf()
m = 0.2
g = 9.8
a = 0.50          # entrée des paramètres physiques
def deriv(sys,t):
   x=sys[0]      # x (notant θ) première coordonnée des vecteurs solutions [θ, ω]
   v=sys[1]      # v (notant ω) seconde coordonnée des vecteurs solutions [θ, ω]
   vitesse=v     # vitesse sera l'expression appelée pour la dérivée de la première coordonnée
   accel=-(3*g/2/a)*sin(x) # accel sera l'expression appelée pour la dérivée de la seconde coordonnée
   return[vitesse,accel]
t=linspace(0,5,100)     # tableau des valeurs de temps pour lequel les solutions [θ, ω] seront calculées
liste=[0.1, 0.3, 1, 1.5, 3, 3.13] #liste des valeurs de position angulaire initiale, parcourue dans la boucle for
for i in liste :
   CI=array([i,0])
   sols=odeint(deriv,CI,t)
   x=sols[:,0]  # Listes des valeurs de θ, donc des premières coordonnées des solutions [θ, ω]
   v=sols[:,1] # Listes des valeurs de ω, donc des secondes coordonnées des solutions [θ, ω]
   plot(t,x)
title('x(t)')
show()
```

En modifiant le programme, réaliser le tracé du portrait de phase ω = f(θ). Commenter l'allure des trajectoires de phase obtenues pour les différentes conditions initiales.

3.2 Calcul de la période selon l'amplitude du mouvement.

Les simulations précédentes font apparaître que la période est modifiée avec l'amplitude du mouvement. On parle de non isochronisme des oscillations, par opposition au cas de l'oscillateur harmonique pour lequel l'isochronisme des oscillations signifie que la période est indépendante des conditions initiales déterminant l'amplitude du mouvement.

Le calcul de la période du pendule pesant va être obtenu par intégration numérique.

3.21 Mise en équation.

On s'appuie sur l'équation de conservation de l'énergie : $E_c + E_p = E_m = Cste$.

En exploitant la conservation de l'énergie mécanique pour le pendule lâché avec l'amplitude θ_0 sans vitesse initiale :

$$\frac{1}{2}J\dot{\theta}^2 - mg(a/2)cos\theta = -mg(a/2)cos\theta_0 \text{ où J } = \text{ ma}^2/3$$

On en déduit, pour une phase du mouvement où $\dot{\theta}$ est négatif :

$$\frac{d\theta}{dt} = -\sqrt{\frac{mga}{J}}\sqrt{cos\theta - cos\theta_0}$$

$$soit \quad \frac{d\theta}{dt} = -\sqrt{\frac{3g}{a}}\sqrt{cos\theta - cos\theta_0}$$

Lorsqu'on lâche le pendule sans vitesse à la position angulaire initiale θ_o, il va en effet revenir vers sa position d'équilibre θ = 0, passer en cette position et poursuivre son mouvement d'oscillation jusqu'à la position $-\theta_o$ puis poursuivre le mouvement en sens inverse, à $\dot{\theta}$ est positif jusqu'à atteindre la position initiale θ_o puis réitérer périodiquement ces évolutions. Le mouvement de θ_o à θ = 0 représente un quart de la période T du pendule.

On intègre ensuite sur un quart de période après avoir séparé les variables :

$$\int_0^{T/4} dt = -\sqrt{\frac{a}{3g}} \int_{\theta_0}^{0} \frac{d\theta}{\sqrt{cos\theta - cos\theta_0}}$$

Ce qui donne finalement

$$T = 4\sqrt{\frac{a}{3g}} \int_0^{\theta_0} \frac{d\theta}{\sqrt{cos\theta - cos\theta_0}}$$

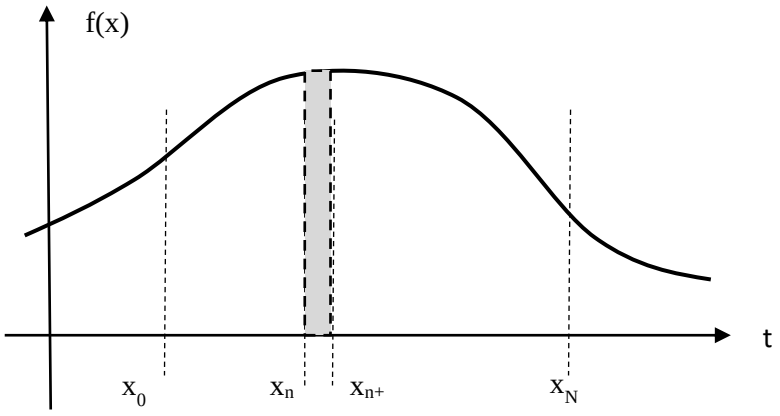
Le calcul de l'intégrale sera conduit numériquement.

3.22 Calcul par la méthode des rectangles ou des trapèzes.

Méthode des rectangles :

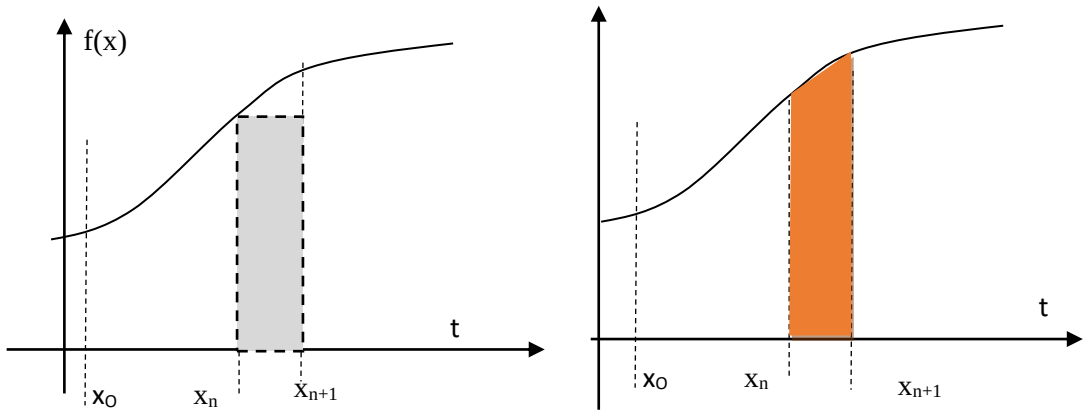
Elle consiste à remplacer l'intégration par une sommation discrète de termes élémentaires $f(x_n).\Delta x$ où $f(x_n)$ est la valeur calculée pour la valeur x_n de la fonction f(x) et Δx est le pas d'intégration. Chaque terme représente donc la surface d'un rectangle de largeur Δx et de hauteur $f(x_n)$. La sommation est conduite sur les N termes correspondant à l'intervalle d'intégration $[x_o ; x_N]$.

$$\int_{x_0}^{x_N} f(x)dx \simeq \sum_{n=0}^{n=N-1} f(x_n).\Delta x$$



Méthode des trapèzes :

La convergence du calcul numérique vers la valeur exacte de l'intégrale dépend évidemment du pas d'intégration.



Mais la convergence peut aussi être améliorée en sommant les aires délimitées en tenant compte des valeurs de la fonction f(x) pour $x = x_n$ et $x = x_{n+1}$.

Chaque aire correspondant à une surface trapézoïdale amène un terme d'expression :

$$\frac{f(x_n) + f(x_{n+1})}{2}.\Delta x$$

Soit un calcul numérique de l'intégrale selon :

$$\int_{x_0}^{x_N} f(x)dx \simeq \sum_{n=0}^{n=N-1} \frac{f(x_n) + f(x_{n+1})}{2}.\Delta x$$

En utilisant l'une des deux méthodes précédentes, déterminer la période d'oscillation du pendule pesant pour des conditions initiales amenant une vitesse angulaire nulle et une position d'angle θ = 3 rad.

3.3 Non isochronisme : évolution de la période avec l'amplitude du mouvement.

On souhaite visualiser le non isochronisme par le tracé d'un graphe $T = T(\theta_o)$.

Intégrer le calcul précédent dans une boucle explorant diverses valeurs de position angulaire initiale selon un pas suffisamment précis pour obtenir une courbe continue $T = T(\theta_o)$ l'amplitude variant de 1° à 170°. Procéder au tracé.

La méthode numérique employée reste approximative. Un calcul numérique plus exacte peut être obtenu Au moyen de la commande **quad**, présente dans la bibliothèque **scipy.integrate**.

La syntaxe de cette commande est évidente : quad(fonction, borne inférieure, borne supérieure). Il faut bien sûr avoir préalablement défini la fonction que l'on souhaite intégrer.
Attention, la commande quad renvoie un doublet de valeurs T,err où T est la période (résultat de l'intégration) et err la majoration de l'erreur commise pour le calcul de l'intégrale.

4. Incertitude sur la mesure de la période.

On se place dans le cas d'oscillations de faible amplitude, dont on a montré qu'elles sont isochrones. La période attendue est alors la période propre dont l'expression théorique est :

$$T_o = 2\pi\sqrt{\frac{2a}{3g}}$$

On souhaite confronter la mesure effectuée expérimentalement à la valeur établie à partir de la théorie.
On relève : T_{omes} = 1,17 s avec une incertitude-type u(T_{omes}) = 0,02 s.

La valeur de l'accélération de la pesanteur est connue avec une incertitude type de 0.01 m.s^{-2}
La mesure de la longueur a de la barre est a = 0,500 m avec une incertitude type de 1 mm.
Déterminer la valeur moyenne et l'écart-type sur la période en utilisant la méthode de Monte-Carlo. Pour les deux grandeurs, g et a, la distribution est supposée suivre une loi uniforme.

A l'aide d'une boucle for, faire un grand nombre de tirages aléatoires pour les valeurs de a, de g (sur la bibliothèque numpy : np.random.uniform(valeur, incertitude)) et les enregistrer sur une liste (par la commande append.) ainsi que les valeurs consécutives de T_o.

Tracer les histogrammes des grandeurs a, g et T_o.
(sur la bibliothèque matplotlib.pyplot : plt.hist(liste des tirages, bins = 100) bins étant le nombre d'intervalles).

Déterminer la valeur moyenne et l'écart-type sur T_o : np.mean( ) et np.std( ).

On rappelle que l'écart normalisé E_N est le rapport de la valeur absolue de l'écart entre la valeur mesurée X et la valeur de référence $X_{réf}$, à la racine de la somme des incertitudes-types aux carrées de chacune des valeurs u(X) et u($X_{réf}$).

$$E_N = \frac{|X - X_{réf}|}{\sqrt{\mathrm{u(X)}^2 + \mathrm{u}(X_{réf})^2}} = \frac{\sqrt{(X - X_{réf})^2}}{\sqrt{\mathrm{u(X)}^2 + \mathrm{u}(X_{réf})^2}}$$

La valeur de référence est la valeur de la grandeur X telle qu'elle est attendue (valeur tabulée, résultat théorique…). Si cette valeur est très précisément connue, son incertitude type u($X_{réf}$) sera négligeable.
Les conclusions pour l'écart normalisé sont définies par le critère suivant :
$|E_N| < 2$: Aptitude satisfaisante
$|E_N| > 2$: Aptitude non satisfaisante

Quelle conclusion doit être retenue quant à la mesure effectuée ?