TD

Révisions - Méthodes numériques

Lycée Thiers - Physique-Chimie - MPI/MPI*- 2024-2025

Table des matières

1 Dérivation numérique d'un signal bruité

On cherche à calculer des fonctions dérivées et à tracer ces fonctions. Pour cela, on utilise les bibliothèques `numpy` et `matplotlib`.

1. Écrire trois fonctions `derivee_plus(y,x)`, `derivee_moins(y,x)` et `derivee_moy(y,x)` avec entrée `y` et `x` deux listes représentant $y = f(x)$ avec f une fonction quelconque et qui retournent une liste correspondant à la dérivée numérique de f. Pour estimer le nombre dérivée d'indice i, la première fonction fera la moyenne entre i et $i+1$, la seconde entre $i-1$ et i et la troisième entre $i-1$ et $i+1$.

Pour tester les fonctions de dérivation, on utilise la fonction $f : x \mapsto \sin(kx)$ avec $k = 2\pi$.

2. Tracer sous python la dérivée analytique attendue de cette fonction entre $x = -2$ et $x = 2$ ainsi que les trois courbes correspondantes aux différentes façon d'évaluer la dérivée.

Les signaux réels sont toujours bruités. Cela signifie que pour chaque point de mesure, une erreur aléatoire est présente. Pour simuler ce phénomène, on utilise la bibliothèque `random`.

3. Copier les lignes de codes suivantes dans un script puis l'exécuter pour différentes valeurs de `a` :

```
x = np.linspace(-2, 2, 300)
a = 0.1
y_bruit = np.sin(k*x) + a * np.random.rand(len(x))

plt.plot(x,y_bruit, 'r', label='signal bruité')
plt.show()
```

4. Calculer et tracer la dérivée numérique de `y_bruit` à l'aide de la fonction `derivee_moy` définie dans la partie précédente.
5. Pour amoindrir le problème des fluctuations aléatoires dues au bruit, écrire une fonction `lissage(y,k)` qui prend en entrée une liste `y` et un entier `k` et qui renvoie en sortie une liste dont l'élément i correspond à la moyenne glissante de y entre les éléments $i-k$ et $i+k$. Visualiser sur un graphique l'effet du lissage pour $k = 3$ et $k = 5$.
6. Calculer et tracer la dérivée numérique de `lissage(y_bruit,k)` pour $k = 3$ et $k = 5$ à l'aide de la fonction `derivee_moy` définie dans la partie précédente. Refaire le tracé pour plusieurs valeurs de l'amplitude du bruit `a`.

2 Résolution d'équation différentielle du premier ordre : la charge du condensateur

Pour présenter l'algorithme de la méthode d'Euler, nous utilisons un problème que l'on sait résoudre exactement.

Présentation du problème : On étudie théoriquement le montage électrique de la figure 1. On précise que la tension $E_0 = 5\,\mathrm{V}$ est une fonction constante, l'interrupteur initialement ouvert est fermé au temps $t = 0$ et le condensateur est initialement déchargé.

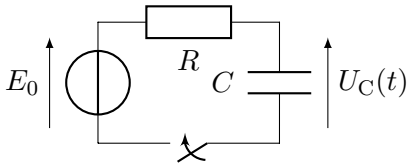


Fig. 1 – Schéma électrique de la charge du condensateur.

L'équation différentielle linéaire du premier ordre à coefficients constants de la charge d'un condensateur est

$$\frac{\mathrm{d}U_C}{\mathrm{d}t}(t) + \frac{1}{\tau}U_C(t) = \frac{E_0}{\tau}\ . \tag{2.1}$$

où $\tau = RC = 5\,\mathrm{s}$ est le temps caractéristique de la charge et de la décharge du condensateur.

En tenant compte des conditions initiales, la solution exacte de cette équation est

$$U_C(t) = E_0(1 - \exp(-t/\tau))\ . \tag{2.2}$$

1. Écrire une fonction python `u_prime(u,t)` qui prend en entrée deux nombres `u` et `t` et qui renvoie la valeur de $\frac{\mathrm{d}U_C}{\mathrm{d}t}(t)$ pour $u = U_C$, t et τ fixés selon l'équation différentielle (2.1). Le paramètre τ doit être représenté par une variable globale.
2. Écrire une fonction python `euler_explicite(f,y0,t0,tmax,n)` qui prend en entrée une fonction `f`, trois flottants `y0`, `t0`, `tmax` et un entier `n`. Cette fonction doit renvoyer deux listes dont la première représente la liste des y représentant la résolution numérique utilisant l'algorithme de la méthode d'Euler explicite avec $y'(t) = f(y(t), t)$ et $y(t_0) = y_0$ tandis que la seconde représente la liste des t correspondants. Les temps sont compris entre `t0` et `tmax` et comprend `n` points.
3. Pour $n = 50$, résoudre numériquement l'équation différentielle (2.1) de la charge du condensateur pour des temps compris entre $t = 0$ et $t_{\max} = 10\tau$ et tracer la solution obtenue.
4. Tracer sur un même graphique la solution exacte de l'équation différentielle (2.2) ainsi que différentes résolutions numériques de l'équation pour des valeurs différentes de `n`. Comparer le pas de résolution h au temps caractéristique de l'équation différentielle τ et en déduire comment choisir de façon raisonnable le pas h (et donc le nombre n).
5. Résoudre l'équation différentielle à l'aide de la fonction `odeint` de la bibliothèque `scipy.optimize` et comparer le résultat avec une résolution à l'aide de la méthode d'Euler explicite.
6. Vérifier que la résolution est cohérente pour la décharge du condensateur, soit avec les conditions initiales $U_c(0) = 5\,\mathrm{V}$ et $E_0 = 0\,\mathrm{V}$.

3 Résolution d'équation différentielle du second ordre : le pendule simple

Présentation du problème : L'équation différentielle vérifiée par l'angle $\theta(t)$ dans le cas d'oscillations où les frottements sont négligés est

$$\ddot{\theta}(t) + \omega_0^2 \sin\theta(t) = 0\ , \tag{3.1}$$

avec $\omega_0 = \sqrt{\frac{g}{\ell}}$.

Cette équation n'a pas de solution analytique simple accessible à notre niveau.

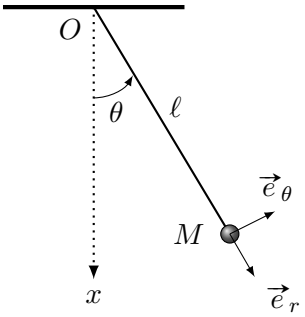


Fig. 2 – Le pendule simple

Pour tout cet exercice, on prend $\omega_0 = 1\,\text{rad/s}$.

3.1 Mise en forme du problème et première résolution

1. En réalisant le changement de variable adéquat, montrer que l'équation différentielle (3.1) d'ordre 2 peut se ramener à un système de deux équations différentielles d'ordre 1. Préciser les variables.
2. Écrire une fonction python `pendule(y,t)` qui prend en entrée une liste `y` de deux flottants représentant θ et $\dot{\theta}$ et un flottant `t` et qui renvoie les valeurs de $\ddot{\theta}(t)$ et $\dot{\theta}(t)$ **sous forme de liste** pour le système différentiel défini à la question précédente.
3. La fonction `odeint` de la bibliothèque `scipy.optimize` permet de résoudre numériquement des équations différentielles. Après avoir défini la fonction `pendule`, la pulsation propre `omega0` ainsi que les conditions initiales `theta0` et `thetapoint0`, entrez les commandes suivantes :

```
 omega0 = 1 # rad/s
theta0 = pi/5
thetapoint0 = 0

y0 = [theta0,thetapoint0]
t0 = 0
tmax = 10 * 2 * pi / omega0

t= np.linspace(t0, tmax, 1000)

sol_ode = odeint(pendule, y0, t)
```

   Le tableau `sol_ode` contient dans sa première colonne les valeurs de $\theta(t)$ et dans sa seconde colonne les valeurs de $\dot{\theta}(t) = \omega(t)$.
4. Tracer $\theta(t)$ pour les conditions initiales $\dot{\theta}_0 = 0\,\text{rad/s}$ et $\theta_0 = k\pi/5$ avec $k = 1, 2$ et 3.

3.2 Prise en compte des frottements

5. On prend maintenant en compte les frottements. L'équation différentielle du pendule devient

$$\ddot{\theta}(t) + \frac{\omega_0}{Q}\dot{\theta}(t) + \omega_0^2 \sin\theta(t) = 0\,.$$

   Modifier les fonctions précédentes pour prendre en compte le terme d'amortissement. Tracer les solutions pour les mêmes conditions initiales que précédemment pour $Q = 5$.

4 Exercices supplémentaires

4.1 Chute libre avec frottements fluides quadratiques

On étudie le mouvement d'une particule de masse m lancée avec un certain vecteur vitesse initiale $\vec{v_0}$ soumise à une force de frottements fluides.

1. Réaliser l'étude complète du problème afin d'obtenir l'équation vectorielle du mouvement.

   On suppose que $\vec{F}_\text{f} = -\kappa v \vec{v}$.

1. Écrire l'équation vectorielle du mouvemment.
2. En introduisant des variables adimensionnées, montrer qu' « aux temps longs », la vitesse de la particule est constante. On précisera le sens de l'expression « aux temps longs ».
3. Résoudre numériquement l'équation différentielle vérifiée par la vitesse en utilisant la méthode d'Euler. Discuter les résultats obtenus.

4.2 Épaisseur d'une aile d'avion

On considère un profil d'aile d'avion dont la demi-épaisseur $e(x)$ est approchée par l'équation

$$e(x) = 0.15\left(3.7\sqrt{x} - 3.4x - 0.3x^4\right)$$

avec $x \in [0, 1]$.

On cherche à trouver la valeur de x pour lequel cette demi-épaisseur est maximale. Autrement dit, on cherche à résoudre l'équation

$$f(x) = \frac{\mathrm{d}e}{\mathrm{d}x}(x) = 0 = 0.15\left(\frac{3.7}{2\sqrt{x}} - 3.4 - 1.2x^3\right) .$$

Cette équation non linéaire n'a pas de solution analytique. L'utilisation d'une méthode numérique pour la résoudre est indispensable.

1. Tracer f entre 0 et 1 et vérifier qu'elle vérifie bien les critères pour pouvoir appliquer la fonction une méthode de résolution numérique par dichotomie.
2. Écrire une fonction python `dicho(f,a,b,eps)` qui prend en entrée une fonction `f`, deux flottants `a` et `b` ainsi qu'une précision flottante `eps` et qui renvoie une valeur c comprise entre a et b telle que $|f(c)| < \varepsilon$ en utilisant la résolution par dichotomie. La fonction f appelée doit être strictement monotone entre a et b et s'annuler une fois dans l'intervalle. On ne cherchera pas à tester ces conditions dans le programme.
3. Appliquer la fonction `dicho` à la fonction f.
4. Comparer la valeur de la solution donnée par la fonction `bisect` de la bibliothèque `scipy.optimize` avec la valeur trouvée précédemment.

4.3 Périodes d'oscillation du pendule

Le mouvement pendulaire est régi par l'équation différentielle non linéaire

$$\ddot{\theta} + \omega_0^2 \sin\theta = 0 .$$

On peut montrer que la période d'une oscillation du pendule, dont les conditions initiales sont $\theta(0) = \theta_0$ et $\dot{\theta}(0) = 0$, est donnée par la relation

$$T = \frac{\sqrt{2}T_0}{\pi}\int_0^{\theta_0} \frac{\mathrm{d}\theta}{\sqrt{\cos\theta - \cos\theta_0}} ,$$

avec T_0 la période des petites oscillations du pendule.

1. Calculer avec python T/T_0 par la méthode des rectangles.
2. Calculer ce même rapport à l'aide de la fonction `qquad` de la bibliothèque `scipy.integrate`.
3. Comparer les résultats et montrer graphiquement que, pour les angles suffisamment faibles, le rapport T/T_0 est compatible avec la formule de Borda

$$\frac{T}{T_0} = 1 + \frac{1}{16}\theta_0^2 + \frac{11}{3072}\theta_0^4 .$$

La période d'oscillation du pendule dépend des conditions initiales, on dit que les oscillations ne sont pas isochrones.

4.4 Détermination du courant et de la tension d'un dipôle non linéaire

Une diode est un dipôle passif, non symétrique (elle a un sens dans un montage) et non linéaire (sa caractéristique statique n'est pas une droite).

L'équation de la caractéristique pour un courant positif est dans les limites de fonctionnement usuel est

$$i(u) = I_0 \left(\exp\left(\frac{u}{u_0}\right) - 1 \right),$$

où la tension u_0 et I_0 dépendent de la diode et de la température. On prendra $u_0 = 26\,\text{mV}$ et $I_0 = 25\,\text{nA}$.

Cette diode est placée en série avec un générateur de tension réel de tension constante $E_0 = 5\,\text{V}$ et une résistance de protection de $100\,\Omega$. Le point de fonctionnement du circuit correspond au couple courant-tension effectif dans le circuit. D'un point de vue pratique, c'est l'intersection de la courbe $u(i)$ avec la courbe $u = E_0 - R_{\text{eq}} i$ imposée par le générateur et les résistances.

1. En prenant en compte la résistance interne du générateur de $50\,\Omega$, quelle est la résistance totale du circuit ?
2. Réaliser un schéma de ce circuit équivalent.
3. Tracer les courbes tension/courant permettant de définir le point de fonctionnement du système.
4. À l'aide d'un code python permettant la résolution numérique d'équation basé sur la méthode dichotomique, en déduire le point de fonctionnement du circuit.

4.5 L'équation de l'oscillateur harmonique (extrait de l'épreuve informatique PT 2019)

On étudie l'équation différentielle de la forme :

$$\frac{\mathrm{d}^2 h}{\mathrm{d}t^2}(t) = -h(t)\ . \tag{4.1}$$

C'est une équation différentielle linéaire d'ordre deux classique appelée équation harmonique. Nous l'utiliserons pour tester la pertinence de certaines méthodes numériques.

1. Méthode d'Euler.
   i. Rappeler, en quelques lignes concises, le principe de la méthode d'Euler pour résoudre une équation différentielle.
   ii. Mettre l'équation différentielle précédente sous la forme d'un système différentiel linéaire du premier ordre sous forme matricielle en caractérisant la matrice A en exploitant le vecteur $X(t)$ tel que :

$$\frac{\mathrm{d}X(t)}{\mathrm{d}t} = AX(t) \qquad \text{avec} \qquad X(t) = \begin{pmatrix} h(t) \\ h'(t) \end{pmatrix}\ .$$

   iii. À partir de cette forme matricielle, introduire le pas de discrétisation temporel τ et mettre en œuvre la méthode d'Euler pour obtenir une équation aux différences finies correspondant au système initial. Vous pourrez noter de façon réduite $X(t_n) = X[n]$.
2. Méthode d'Euler à droite.
   Pour obtenir $X(t+\tau)$, nous pouvons procéder de la façon suivante :

$$X(t_n + \tau) = X(t_n) + \int_{t_n}^{t_n+\tau} \frac{\mathrm{d}X}{\mathrm{d}t}(t)\ \mathrm{d}t \approx X(t_n) + \tau \frac{\mathrm{d}X}{\mathrm{d}t}(t_n + \tau)\ .$$

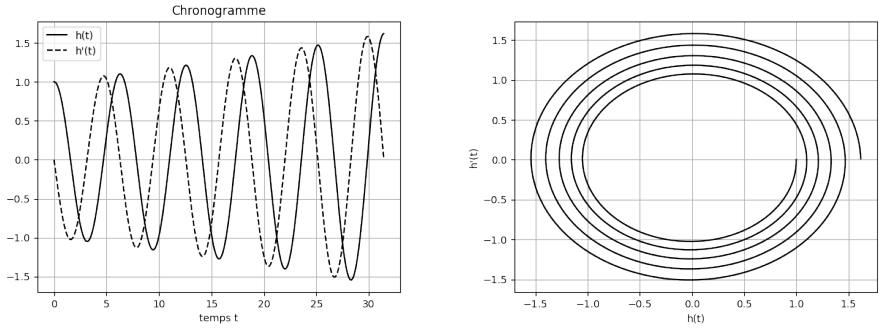
   C'est la méthode dite d'Euler *à droite*, ou *implicite*, la version classique étant dite *à gauche*, ou *explicite*.
   i. Exprimer l'équation liant dans ce cas $X[n+1]$ et $X[n]$.
   ii. Le code ci-après correspond à l'implémentation de la méthode d'Euler à gauche. Le modifier pour implémenter la méthode d'Euler à droite. Pour inverser une matrice, on peut utiliser la fonction `linalg.inv` de la bibliothèque `numpy`.

```
# Constantes paramétriques
npoints = 2**10 # index maximal des tableaux
tporte = 5*2*np.pi # durée d'affichage
X0 = [1.,0] # conditions initiales
```

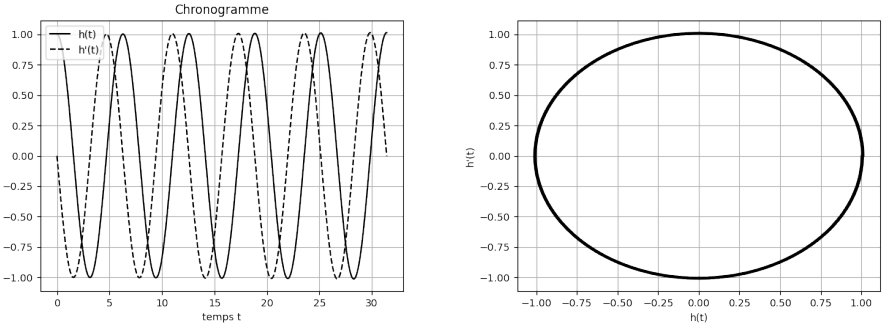
```python
# Initialisation des tableaux
t = np.linspace(0,tporte,npoints+1) # temps
tau = t[1] - t[0]
X = np.zeros((2,npoints+1),dtype=float) # h et h'

# Algorithme d'Euler
X[:,0] = X0 # initialisation
M = np.array([[1,tau], [-tau,1]]) # matrice de l'équation
for i in range(npoints): # calcul itératif
  X[:,i+1] = M.dot(X[:,i]) # A.dot(B) retourne la multiplication matricielle
  # de A et B
```

iii. La méthode d'Euler à droite est-elle plus pertinente que celle à gauche ?

3. Représentations graphiques.
   i. À partir du code donné à la question précédente, mettre en place les instructions correspondant aux figures ci-après. Vous veillerez à la présence des légendes des courbes, des axes et des titres.



   ii. Commentez les courbes obtenues et corrélez leurs allures aux propriétés de la méthode d'Euler et à l'équation (4.1).
   iii. Sans modifier le principe de l'algorithme utilisé, nous pouvons néanmoins obtenir les courbes ci-après. Quelle est, selon vous, la modification apportée au code précédent ? Et quel en est l'inconvénient ?



4. Méthode de Runge-Kutta d'ordre 2 (*aucune connaissance préalable n'est nécessaire*).
   $X(t+\tau)$ s'obtient à partir de $X(t)$ avec la classique relation intégrale :

$$X(t_n+\tau) = X(t_n) + \int_{t_n}^{t_n+\tau} \frac{\mathrm{d}X}{\mathrm{d}t}(t)\ \mathrm{d}t\ .$$

   Nous pouvons obtenir une version approchée de cette relation en utilisant l'approximation :

$$X(t_n+\tau) = X(t_n) + \frac{\tau}{2}\left(\frac{\mathrm{d}X}{\mathrm{d}t}(t_n+\tau) + \frac{\mathrm{d}X}{\mathrm{d}t}(t_n)\right)\ .$$

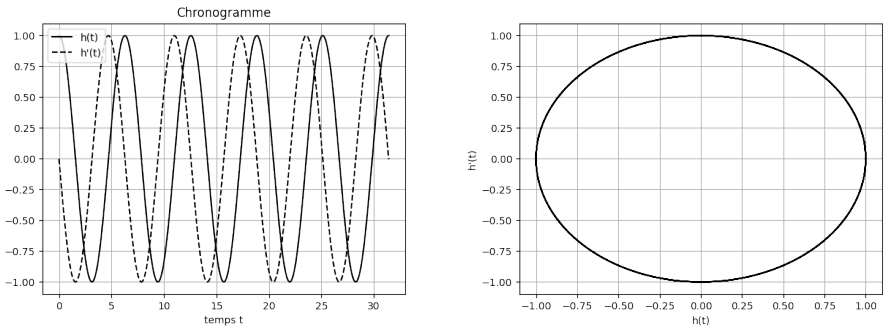
   i. Comment qualifier la méthode employée pour approximer l'intégrale ? Qualitativement, pourquoi est-elle préférable aux méthodes précédentes ?
   ii. L'analyse est faite sur la durée $T = N\tau$.
   La valeur de la dérivée de X au temps $t_n + \tau$ demande la mise en œuvre d'une méthode de calcul implicite, nous l'éviterons en estimant la valeur de X au temps $t_n + \tau$ par la méthode d'Euler, c'est le principe de la méthode de Runge-Kutta. L'algorithme de calcul est alors le suivant :

$$K_1 = \frac{\mathrm{d}X}{\mathrm{d}t}(t_n) = AX[n] \qquad K_2 = A(X[n] + \tau K_1)$$

$$X[n+1] = X[n] + \frac{\tau}{2}(K_1 + K_2)$$

Reprenez le code écrit pour la méthode d'Euler et adaptez le pour mettre en œuvre la méthode de Runge-Kutta.

iii. La simulation numérique dans des conditions identiques à la méthode d'Euler nous fournit les graphes ci-dessous. Ils semblent correspondre aux attendus usuels. Peut-on conclure à l'adéquation de la méthode de Runge-Kutta d'ordre deux ? Argumentez votre réponse.

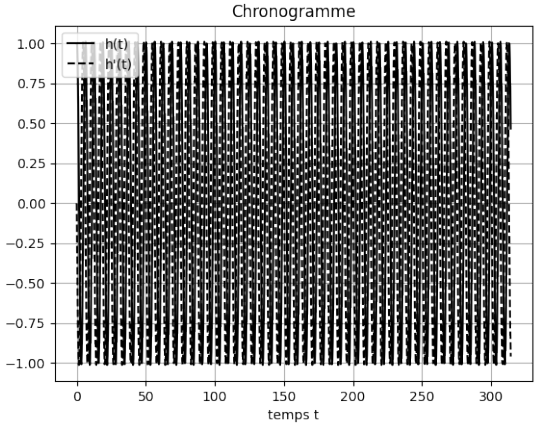


5. Méthode « stable ».
Nous implémentons le code ci-après :

```
# Constantes paramétriques
npoints = 2 * 2**10 # index maximal des tableaux
tporte = 50 * 2*np.pi # durée d'affichage
X0 = [1.,0] # conditions initiales
# Initialisation des tableaux
t = np.linspace(0,tporte,npoints+1) # temps
tau = t[1] - t[0]
X = np.zeros((2,npoints+1),dtype=float) # h et h'
# Algorithme XXX
X[:,0] = X0 # initialisation
B = np.array([[0,1], [-1,-tau]]) # matrice de couplage
M = np.identity(2) + tau * B # matrice d'évolution
for i in range(npoints): # calcul itératif
  X[:,i+1] = M.dot(X[:,i]) # A.dot(B) retourne la multiplication matricielle
  #de A et B
```

i. À partir du code déployé, donnez les équations itératives permettant la mise en œuvre de la simulation.

ii. Le code nous permet d'obtenir les courbes de la figures ci-dessous. Cette méthode semble-t-elle plus stable ?

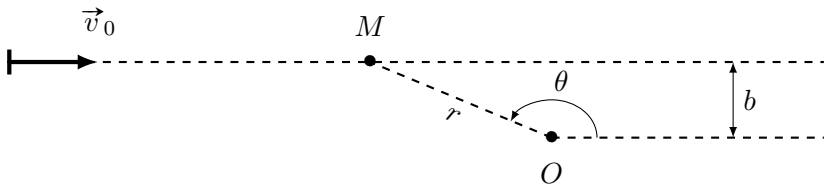


4.6 Trajectoire dans un champ de force centrale conservatif

4.6.1 Position du problème

Cette activité propose de simuler le mouvement d'une goutte d'oxygène liquide en caléfaction dans le champ magnétique produit par un petit aimant permanent de forme cylindrique. Elle s'appuie sur le travail de thèse réalisé par Keyvan Piroird (*Dynamiques spéciales de gouttes non-mouillantes.*, École Polytechnique X, 2011), et plus particulièrement sur les éléments développés dans le chapitre 2.

La goutte d'oxygène liquide étudiée peut glisser sans frottement sur une plaque horizontal en verre, confondue avec le plan Oxy. Sous cette plaque, à proximité du point O, se trouve un petit aimant cylindrique. Initialement, la goutte se trouve à grande distance de l'aimant ; on lui communique une vitesse initiale $\vec{v_0}$, avec un paramètre d'impact b.



Dans le plan Oxy, la force magnétique exercée par l'aimant sur la goutte est une force centrale de centre O, qui dérive de l'énergie potentielle

$$\mathcal{E}_\mathrm{p}(r) = -\frac{e_0 V}{q + \left(\dfrac{r}{r_0}\right)^6} \tag{1}$$

où V désigne le volume de la goutte et e_0, q et r_0 sont des paramètres obtenus par ajustement numérique.

▷ Recopier le code suivant donnant les valeurs numériques des constantes.

```
## Caractéristiques de la goutte étudiée

rho = 1141          # masse volumique de l'oxygène liquide (en kg/m^3)
a = .5e-3           # rayon de la goutte (en m)
V = 4/3*np.pi*a**3 # volume de la goutte (en m^3)
m = rho*V           # masse de la goutte (en kg)

## Paramètres figurant dans l'énergie potentielle magnétique

e0 = 9.409          # en J/m^3
q = 0.084           # sans dimension
r0 = 8.45e-3        # en m

## Paramètres à faire varier

v0 = 0.25          # en m/s
b = 5e-3           # en m
```

4.6.2 Graphe de l'énergie potentielle effective

On a classiquement

$$\mathcal{E}_{\mathrm{p,eff}}(r) = \frac{L^2}{2mr^2} + \mathcal{E}_\mathrm{p}(r) \tag{2}$$

où $L = -mbv_0$ désigne le moment cinétique de la goutte par rapport à l'axe Oz.

▷ Définir la fonction `Epeff(r)` renvoyant la valeur de l'énergie potentielle effective en fonction de r.

▷ Tracer cette énergie potentielle effective pour r entre 1 mm et 25 mm. Tracer successivement cette énergie pour plusieurs valeurs de b variant entre 1 et 15 mm.

4.6.3 Obtention des trajectoires

Équations du mouvement : La conservation de l'énergie mécanique et du moment cinétique de la goutte conduisent aux équations suivantes :

$$\begin{cases} \ddot{r}(t) = -\dfrac{1}{m}\dfrac{\mathrm{d}\mathcal{E}_{\mathrm{p,eff}}}{\mathrm{d}t}(r(t)) \ ; \\ \dot{\theta}(t) = \dfrac{L}{mr^2(t)} \ . \end{cases}$$

▷ Définir la fonction dérivée de l'énergie potentielle en recopiant le code suivant.

```
# Définition de la dérivée de l'énergie potentielle effective

def dEp_eff(r):
    """ Dérivée de l'énergie potentielle effective (l'expression
    analytique a été déterminée algébriquement)"""
    L0 = - m * b * v0
    return -L0**2/(m*r**3)+e0*V/(q+(r/r0)**6)**2*6*r**5/r0**6
```

En vue de l'intégration numérique, on va vectoriser le système différentiel précédent. Le vecteur inconnu, noté X par la suite, est défini par $X(t) = (r(t), \dot{r}(t), \theta(t))$ et les équations du mouvement se traduisent par le sytème différentiel d'ordre 1 suivant :

$$\frac{\mathrm{d}X}{\mathrm{d}t}(t) = \begin{pmatrix} \dot{r}(t) \\ -\dfrac{1}{m}\dfrac{\mathrm{d}\mathcal{E}_{\mathrm{p,eff}}}{\mathrm{d}t}(r(t)) \\ \dfrac{L}{mr^2(t)} \end{pmatrix} .$$

▷ Définir une fonction `eqn_mvt(X,t)` qui renvoie un vecteur à 3 dimensions décrit par l'équation ci-dessus.

Il ne reste plus qu'à définir les conditions initiales vérifiées par r, $\dot{r}$ et θ. Ici, comme on ne peut pas faire tendre $r(t=0)$ vers l'infini, on fixe arbitrairement $r(t=0) = 100 r_0$ (les graphes d'énergie potentielle effective tracés précédemment permettent de s'assurer que l'énergie potentielle effective est alors très faible). On vérifie par ailleurs que

$$\theta(t=0) = \pi - \arcsin\frac{b}{r(t=0)} \qquad \text{et} \qquad \dot{r}(t=0) = v_0 \cos\theta(t=0) \ .$$

▷ Définir une fonction `CI(b,v0)` qui renvoie les conditions initiales du vecteur X en utilisant les relations précédentes.

Résolution des équations du mouvement et représentation des trajectoires : Pour des raisons d'efficacité, on opte ici pour l'utilisation de la fonction préprogrammée `odeint` disponible dans le module `scipy.integrate`. Une fois les positions successives de la goutte déterminées numériquement en coordonnées polaires, on repasse en coordonnées cartésiennes pour faciliter la représentation de la trajectoire.

▷ Entrer le vecteur temps ci-dessous pour la résolution de l'équation différentielle.

```
# Définition du vecteur contenant les valeurs de t pour lesquelles on va estimer
# numériquement la position de la goutte

t = np.linspace(0,10,10000)
```

Exemple 1 : Cas où $v_0 = 25\,\mathrm{cm/s}$ - Influence de b sur la trajectoire :

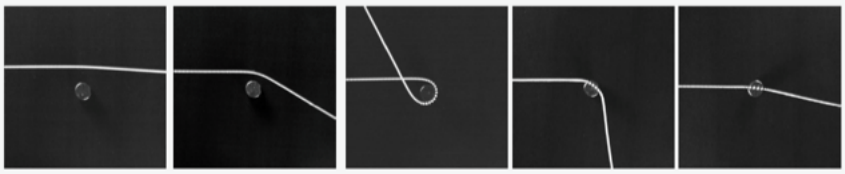


FIGURE 2.2 – Chronophotographies de trajectoires de gouttes d'oxygène à proximité d'un aimant, observées par le dessus. L'aimant (au centre de l'image) a un rayon de 5 mm et se trouve 2 mm sous la vitre. La vitesse initiale est la même sur toutes les images et vaut $V_{in} = 25$ cm/s. a) $b = 15$ mm, $\alpha = 3°$. b) $b = 11$ mm, $\alpha = 30°$. c) $b = 9$ mm, $\alpha = 245°$. d) $b = 5$ mm, $\alpha = 85°$. e) $b = 1$ mm, $\alpha = 15°$.

Fig. 3 – Image extraite de la thèse de Keyvan Piroird

▷ Recopier le code ci-dessous pour résoudre l'équation différentielle.

```
sol = odeint(eqn_mvt,CI(b,v0),t)
```

`sol` est une liste de triplets représentant r, $\dot{r}$ et θ pour chaque instant.

▷ Tracer la trajectoire correspondante en transformant les coordonnées polaires en coordonnées cartésiennes. On utilisera la commande suivante pour représenter l'aimant :

```
plt.plot([0],[0],'ko')
```

▷ Comparer avec les expériences de la figure 3. Tester avec $b = 8.6\,\text{mm}$, $8.8\,\text{mm}$ et $9\,\text{mm}$.

Exemple 2 : Cas où $b = 9.4\,\text{cm}$ - Influence de v_0 sur la trajectoire :

▷ Fixer b et tracer les trajectoires pour des vitesses initiales variant entre 15 et $25\,\text{cm/s}$.

▷ Le travail expérimental de Keyvan Piroird indique que la déflexion est maximale pour environ $22\,\text{cm/s}$, le constatez vous numériquement ?