# Recherches de zéro d'une fonction

## Répulsion magnétique entre deux aimants sur un plan incliné

On considère deux aimants en répulsion sur un plan incliné d'angle alpha. On a posé pour cela une cale de hauteur h sous le bord d'une plaque de longueur L et on néglige les actions mécaniques de frottements grâce à une surface en Téflon par exemple.

La loi de répulsion des deux aimants (Nord-Nord ou Sud-Sud) est donnée par : x étant l'abscisse le long du plan incliné.

$$F(x) = k\left(\frac{x_0}{x}\right)^n$$

**Introduire dans le code python les données de modélisation suivantes :**

```
##DONNEES PHYSIQUES
g = 9.81      #ms-2
m = 0.189     #kg     masse de l'aimant
x0 = 1        #m      [distance de référence]        # 6cm < x < 12cm
L = 1.2       #m      (longueur plan incliné)
h = 0.07      #m      1cm < h < 12cm lors de l'expérience. On pourra changer sa valeur.
alpha = asin(h/L) #angle inclinaison

#Paramètres de la loi F(x)=k(x0/x)**n
k = 2.e-6     #N
n = 4         #exposant de la loi de puissance.
```

Vous introduirez de même tous les modules nécessaires.

# 1 Fonctions de base pour la recherche des zéros

La recherche numérique de zéro nécessite à minima d'accéder aux valeurs de la fonction à étudier elle même, voire pour la méthode de Newton d'accéder de surcroît aux valeurs de sa dérivée. D'un point de vue physique la ou les position(s) d'équilibre s'obtiennent comme étant des zéros de la dérivée de l'énergie potentielle.

**Équilibre :** $\frac{dE_p}{dx}(x_{eq}) = 0$ **Stabilité :** $\frac{d^2E_p}{dx^2}(x_{eq}) > 0$

## 1a Fonctions abscisses et ordonnées

On reprend la démarche générale du TD précédent sur les fonctions.

- Écrire une fonction **mesAbscisses**(a, b, N=10) qui renvoie une liste d'abscisses de taille N.
- Écrire une fonction **mesOrdonnees**(fonction, listeAbscisse) qui renvoie la liste de taille N des ordonnées associées aux abscisses et évaluées pour la fonction passée en argument.

Ces listes seront nécessaires pour le tracé de notre fonction.

## 1b Fonction Energie potentielle et ses dérivées

- Déterminer la fonction énergie potentielle du système et en déduire que l'équation de recherche de zéro (c-à-d le PFD) sera :

- Calculer ensuite sa dérivée.

$$mg\frac{h}{L} - k\left(\frac{x_0}{x}\right)^n = 0$$

## 1c l'Equation dont on veut trouver le zéro et sa dérivée

On ne sait pas à l'avance à quelle abscisse il nous faudra connaître la valeur de la fonction ou de sa dérivée :

- Écrire une fonction **equationZero**(x, h=0.07) qui renvoie l'expression de l'équation en fonction de l'abscisse x passée en argument. h est laissé en paramètre car on peut vouloir le changer.
- Écrire une fonction **equationDerive**(x, h=0.07) qui revoie la valeur de la dérivée de l'équation.

# 2 Recherches de zéro

Cette partie se subdivise en deux approches fondamentalement distinctes : L**ire le cours associé.**
Rq : dans cette partie on sait qu'il n'y a qu'un seul zéro pour la méthode de Newton.

## 2a Recherche de Zéro par la méthode de Newton

La **méthode de Newton** consiste en partant d'une abscisse arbitraire, à extrapoler la tangente en ce point pour trouver l'intersection avec l'axe des abscisses en espérant se rapprocher du zéro rapidement. On ré-itère cette opération jusqu'à atteindre une convergence suffisante !
La méthode de Newton converge très rapidement mais impose de connaitre la dérivée de la fonction et des critères beaucoup plus contraignant sur sa dérivabilité et sa convexité.
**Exemple :** les calculatrices obtiennent la racine carrée très rapidement par cette méthode, car la fonction racine carrée est parfaitement connue, dérivable et à dérivées successives continues.

i - Écrire une fonction **newton**(xStart, fonction, derive, epsilon=1.e-16) qui prend en argument, l'abscisse de départ, la fonction et sa dérivée (objets fonction en python) et un critère de converge. Cette fonction renverra la valeur Xeq du zéro.

ii - Ajouter un compteur qui comptabilise le nombre « nb » d'itérations nécessaires à la convergence. La fonction renverra donc un tuple (Xeq, nb).

## 2b Recherche de Zéro par la méthode de Dichotomie

La **méthode par dichotomie** consiste à prendre deux points de départ xMin et xMax de signes opposés : on coupe alors l'intervalle en deux et on regarde le signe. On réduit ainsi la taille de l'intervalle de moitié en ne gardant que la moitié où le signe change : **Diviser pour régner !!!**
On ré-itère cette opération jusqu'à atteindre une convergence suffisante !
La méthode par dichotomie converge certes moins rapidement mais s'applique à une bien plus grande variété de problème : il suffit que la fonction soit monotone sur le domaine.
**Exemple :** la recherche d'informations dans une base de données triées en faite par dichotomie.

i - Écrire une fonction **dichotomie**(xMin, xMax, fonction, epsilon=1.e-16): qui prend en arguments les deux positions de gauche et droite, l'objet fonction et le critère de convergence. Cette fonction renverra la position Xeq obtenue. On fera l'hypothèse que la fonction est croissante.

ii - Optimiser votre fonction en déterminant dès le départ si votre fonction est croissante ou décroissante et en adaptant la recherche dichotomique.

iii - Ajouter de même un compteur. La fonction renverra donc un tuple (Xeq, nb)

## 3 Tracé de la courbe & Comparaison des méthodes

On se donne pour le tracé l'intervalle : 3cm < x < 12cm. Soient xMin = 0.03m et xMax = 0.12m. On prendra donc ces 2 bornes pour la méthode par dichotomie et la borne xMin pour démarrer la méthode de Newton.

## 3a Comparaison des deux recherches de zéro.

Lancer les deux méthodes de recherche et comparer les résultats obtenus. Comment explique-t-on ici qu'une méthode l'emporte largement sur l'autre au regard des critères donnés dans le cours et résumés en partie 2.

## 3b Tracé et affichage des résultats

On donne le programme suivant, que vous pouvez copier à la fin de votre fichier python.
[Il faudra sans doute toutefois modifier les guillemets dans le code]

i - Commentez le code partout où cela est indiqué. Et expliquez comment varie Xeq pour différentes valeurs de h.

On rappel que la fonction python **lambda** inspirée des langages fonctionnels (λ-calcul) permet de définir une fonction au sein d'un élément de syntaxe tel qu'un argument de fonction :
**lambda x : x**2** # est interprétée en python comme un objet fonction -> la fonction carrée.
Il n'y a pas besoin de la stocker dans une variable pour le passer en argument.

ii - Expliquer précisément à quoi sert l'utilisation de la syntaxe lambda dans les fonctions qui suivent.

```
xMin =  3e-2        #C1 :
xMax = 12e-2
h=0.06              #paramètre à modifier.

x=mesAbscisses(xMin, xMax, 100)  #C2 :
y=mesOrdonnees(lambda v:equationZero(v, h), x ) #C3 :

from matplotlib import pyplot as plt  #C4 :

plt.plot([xMin, xMax], [0,0])     #C5 :
plt.plot(x, y, 'b');              #C6 :
plt.show()                        #C7 :

#C8 :
print( 'Recherche de zéro par la méthode de Newton : ' )
monZeroNewton = newton(xMin, lambda v:equationZero(v,h), lambda v:equationDerive(v,h))


#C9 :
print(  'Recherche de zéro par la méthode de Dichotomie : ')
monZeroDicho = dichotomie(xMin, xMax, lambda v:equationZero(v,h))

print(  'Newton       : x =',  monZeroNewton[0],  'en',  monZeroNewton[1],  'étapes.' )
print(  'Dichotomie : x =',  monZeroDicho[0],    'en',  monZeroDicho[1],    'étapes.' )
```