**Objectifs** ***en physique :*** *comparer graphiquement les différents régimes d'amortissement des oscillations libres amorties ; mettre en évidence l'influence de la valeur du facteur de qualité et des conditions initiales.*
***en informatique :*** *utiliser* ***Spyder****, définir une fonction dans un script, tracer le graphe d'une fonction, superposer plusieurs graphes, et accessoirement sauvegarder le résultat final sous forme d'un fichier image ; utiliser des instructions conditionnelles (if, else/elif).*

**NB** ***Compte-rendu :*** *Exercice* ***F3****, script (fichier *.PY) à déposer sur Éclat :*
► *Espace des classes > Classe ATS > Dossiers partagés > TP info-physique >* ***séance N°1***

## PRÉREQUIS

Vous êtes probablement capables à ce stade de :

- Réaliser des calculs simples dans la console de Spyder (*Scientific Python Development EnviRonment*).
- Écrire un script dans l'éditeur de Spyder et l'exécuter.
- Importer les modules `numpy` et `matplotlib.pyplot` (voir le rappel ci-dessous dans "Modules").

### Modules (ou bibliothèques)

Le module NumPy (abréviation de *Numerical Python*) propose un ensemble de fonctions mathématiques ainsi que la prise en charge des tableaux et des matrices. Le module Matplotlib (et son interface pyplot) permet de représenter des données sous forme de graphiques. Pour les utiliser, vous devez les importer au préalable en plaçant les lignes suivantes dans votre script, en général au début) :

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

### Écriture en Python dans Spyder, rappel de l'essentiel

écriture et opérations simples

| vraie vie | = | $a+b$ | $a-b$ | $a\times b$ | $a\div b$ | $a^b$ | $2{,}5$ | $\pi$ | $a\times 10^n$ | $\sqrt{x}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Python | = | `a+b` | `a-b` | `a*b` | `a/b` | `a**b` | `2.5` | `np.pi` | `aEn` | `np.sqrt(x)` |

fonctions de base

| vraie vie | $e^x$ | $\ln x$ | $\cos x$ | $\sin x$ |
|---|---|---|---|---|
| Python | `np.exp(x)` | `np.log(x)` | `np.cos(x)` | `np.sin(x)` |

opérateurs de comparaison

| = | ≠ | < | ⩽ | > | ⩾ |
|---|---|---|---|---|---|
| == | != | < | <= | > | >= |

## 1. FONCTIONS

### 1.1. Listes à distribution régulière

a) <u>généralités</u>

En général, une fonction sera une liste de valeurs. Par exemple, si on veut manipuler la fonction cosinus sur l'intervalle $[0, 2\pi]$, pour faire un graphique ou pour simuler un phénomène, on manipulera un objet du type `y=cos(liste)`, où `liste` désigne ce qu'on appelle une liste à distribution régulière.

*Notez qu'en général, les données réelles ne sont effectivement qu'une liste de nombres (par exemple la température mesurée toutes les secondes par une sonde, envoyée dans LatisPro ou tout autre logiciel d'acquisition et de traitements de données).*

*Remarque : la notion de liste dans Python est beaucoup plus large, et ne se borne pas aux listes à distribution régulière. On peut même par exemple créer des listes contenant des valeurs de types différents (par exemple entier et chaîne de caractères).*
*Les listes ordonnées ne contenant que des nombres (flottants, entiers, complexes) sont appelées <u>tableaux</u>. Les tableaux sont accessibles à partir du module Numpy.*

b) <u>syntaxe</u>

Voici deux façons de créer des listes dont les éléments sont régulièrement espacés :

- `np.arange(début,fin,pas)` permet de créer une liste dont les éléments commencent à la valeur *début*, ne dépassent pas la valeur limite *fin* et sont espacés de la valeur du *pas*.
- `np.linspace(début,fin,nb)` permet de créer une liste comportant *nb* éléments et allant de la valeur *début* à la valeur *fin* (*linspace* pour `lin`*early* `spaced` *vector* – vecteur à composantes linéairement espacées). Cette seconde méthode garantit que les bornes sont incluses.

Exemple, dans la console :

```
In [ ]:  import numpy as np      (pour pouvoir utiliser les instructions np.* contenues dans ce module)

In [ ]:  np.arange(0,1,0.1)
Out[ ]:  array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])

In [ ]:  np.linspace(0,1,11)
Out[ ]:  array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

c) la fonction **`len()`**

La fonction `len()` vous permet de connaître la longueur d'une liste, c'est-à-dire le nombre d'éléments que contient la liste.

► Essayez-la !

d) les fonctions **`range()`** et **`list()`**

L'instruction `list(range(début, fin, pas))` génère une liste d'entiers dont les éléments commencent à la valeur *début*, ne dépassent pas la valeur limite *fin* et sont espacés de la valeur du *pas*.

NB : `début` est facultatif (valeur par défaut `0`), `pas` également (valeur par défaut `1`).

► Faites un essai !

e) la fonction **`append()`**

La fonction `append()` permet d'ajouter un élément à la fin d'une liste.

► Testez-la !

***Exercice L1 (solution disponible)***

1) Dans la console, créer et afficher une liste nommée `K` allant de $0$ à $2\pi$ par pas de $1{,}5$.
2) De même, créer et afficher une liste nommée `L` comportant $5$ valeurs régulièrement espacées entre $0$ et $2\pi$.
3) Vérifier à l'aide d'une fonction que la liste `L` comporte bien $5$ éléments.

***Exercice L2 (solution disponible)***

Considérons un mouvement oscillatoire non amorti $X\cos(\omega_0 t)$ avec $X = 10{,}0$ cm et une période propre $T_0 = 1{,}00$ s.

On déduira $\omega_0$ de $T_0$.

Calculer dans la console $X\cos(\omega_0 t)$ pour $t = 0{,}5$ s.

Calculer ensuite $X\cos(\omega_0 t)$ pour 20 valeurs de $t$ comprises entre $0$ et $2$ s.

## 1.2. Fonctions à 1 variable

Lorsqu'une tâche doit être réalisée plusieurs fois par un programme avec seulement des paramètres différents, on peut l'isoler au sein d'une fonction. Cette fonction peut alors avantageusement être définie dans un script.

a) syntaxe

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nom_fonction(liste de paramètres –variables):
      bloc d'instructions     #se terminant généralement par return quelque_chose
```

Vous pouvez choisir n'importe quel nom pour la fonction que vous créez, à l'exception des mots-clés réservés du langage, et à la condition de n'utiliser aucun caractère spécial ou accentué (le tiret du bas « _ » est permis). Comme c'est le cas pour les noms de variables, on utilise par convention des minuscules, notamment au début du nom.

Remarquez que le bloc d'instructions est indenté (décalé, en retrait, pour en faciliter la lisibilité).

b) exemple

Placez-vous dans la fenêtre des scripts et ouvrez un nouveau fichier (icone du menu, ou Ctrl-N).
Soit la fonction $f(x)=2x+3$ . Voici comment la définir dans un script :

```
def f(x):
    return 2*x+3
```

Remarque : `f` est le nom de la fonction, `x` est la variable, qui recevra sa valeur lorsque la fonction sera appelée. `return` renvoie la valeur `2*x+3`.

On peut maintenant exécuter le programme en cliquant sur ▶.
La console affiche :

```
In []:  runfile('chemin/fonction sanstitre1.py ', wdir=' chemin ')
```

La fonction est alors définie, et peut être utilisée pour calculer des valeurs dans la console :

```
In []:  f(10)
Out[]:  23
```

Elle peut également s'appliquer en une seule opération à une liste de valeurs (série d'abscisses) pour obtenir une autre liste de valeurs (série d'ordonnées).

```
In []:  f(np.arange(0,10,3))
Out[]:  array([ 3,  9, 15, 21])

In []:  f(np.linspace(0,10,5))
Out[]:  array([ 3.,  8., 13., 18., 23.])
```

***Exercice F1 (solution disponible)***

1) Définir dans un script la fonction correspondant à un mouvement oscillatoire en régime critique :
$x(t)=x_{eq}+(At+B)\,\mathrm{e}^{-\omega_0 t}$ , avec $A=v_0+\omega_0\left(x_0-x_{eq}\right)$, $B=x_0-x_{eq}$ , $x_{eq}$ désignant la position d'équilibre, $x_0$ la position initiale et $v_0$ la vitesse initiale. Les valeurs des différentes constantes devront être indiquées au début du script. On déduira $\omega_0$ de $T_0$, elle-même indiquée parmi ces constantes. On prendra dans un premier temps les valeurs suivantes : $T_0=0{,}8$ s ; $x_{eq}=-0{,}5$ m ; $x_0=1$ m ; $v_0=0$ .
On utilisera des lignes de commentaires pour rendre le script plus lisible.
Enregistrer cette fonction.
2) L'appliquer à différentes valeurs de $t$.
3) Utiliser cette fonction pour déterminer 10 valeurs de la position à des dates régulièrement espacées entre $t=0$ et $t=1$ s.

### 1.3. Les instructions conditionnelles

Nous parlons ici des instructions `if… else…/elif….`
Revoyons ces notions sur un exemple : selon le temps qu'il fait, comment occuper son après-midi ? Pour décrire la météo, nous introduirons tout d'abord la variable `ciel` prenant les valeurs `dégagé`, `couvert` ou `pluvieux`.
Étudiez et testez les scripts proposés. Ce sera également l'occasion d'introduire la fonction `input()`.

a) structure à deux branches

```
ciel = input('Quel est l\'état du ciel, dégagé, couvert ou pluvieux ? : ')
if ciel == 'dégagé':
   activité = 'baignade'
else:
   activité = 'promenade'
print(activité)
```

b) structure à trois branches (ou plus)

```
ciel = input('Quel est l\'état du ciel, dégagé, couvert ou pluvieux ? : ')
if ciel == 'dégagé':
   activité = 'baignade'
elif ciel == 'pluvieux':
   activité = 'cinéma'
else:
   activité = 'promenade'
print(activité)
```

c) structures imbriquées

Affinons le raisonnement en ajoutant la variable `température` prenant les valeurs `chaud` ou `froid`.

```
ciel = input('Quel est l\'état du ciel, dégagé, couvert ou pluvieux ? : ')
température = input('Fait-il chaud, ou froid ? : ')
if ciel == 'dégagé':
   if température == 'chaud':
      activité = 'baignade'
   else:
      activité = 'promenade'
elif ciel == 'pluvieux':
   activité = 'cinéma'
else:
   activité = 'promenade'
print(activité)
```

Variante :

```
ciel = input('Quel est l\'état du ciel, dégagé, couvert ou pluvieux ? : ')
if ciel == 'dégagé':
    température = input('Fait-il chaud, ou froid ? : ')
    if température == 'chaud':
        activité = 'baignade'
    else:
        activité = 'promenade'
elif ciel == 'pluvieux':
activité = 'cinéma'
else:
activité = 'promenade'
print(activité)
```

d) utilisation d'opérateurs logiques

Les opérateurs logiques sont la conjonction ("et" → `and`), la disjonction ("ou" → `or`) et la négation ("non" → `not`).
Ici, nous pouvons avantageusement utiliser `and`.

```
ciel = input('Quel est l\'état du ciel, dégagé, couvert ou pluvieux ? : ')
température = input('Fait-il chaud, ou froid ? : ')
if ciel == 'dégagé' and température == 'chaud':
     activité = 'baignade'
elif ciel == 'pluvieux':
   activité = 'cinéma'
else:
   activité = 'promenade'
print(activité)
```

e) utilisation de la fonction `float()`

La fonction `float()` est utilisée pour convertir un nombre ou une chaîne de caractères représentant un nombre en un nombre à virgule flottante (float).
Exemple

```
In []:  float(np.pi)
Out[]:  3.141592653589793
```

***Exercice F2 (solution disponible)***

Dans l'idée de remplacer, dans le script précédent,
`température = input('Fait-il chaud, ou froid ? : ')`
par une instruction demandant la température qu'il fait, et une ligne de code en déduisant s'il fait chaud ou froid, écrivez un script où :
1) Vous demandez la température qu'il fait, notée `T` (utilisation de `input` et `float`).
2) Vous comparez cette température à un minimum que vous aurez défini (utilisation de `if…else`).
3) Vous en déduisez la valeur à affecter à `température` (`'chaud'` ou `'froid'`).
4) Vous affichez le résultat sous la forme « `il fait chaud` » ou « `il fait froid` » (utilisation de `print` avec plusieurs arguments séparés par des virgules).

### 1.4. Application aux oscillations libres amorties

Cela va être l'occasion d'utiliser à nouveau des instructions conditionnelles, puisque nous allons reprendre le problème des oscillations libres, mais cette fois-ci de manière plus générale, en envisageant tous les cas d'amortissement.

a) résultats de physique → *exercice possible à la maison*

Considérons sur l'axe $Ox$ le mouvement oscillatoire amorti d'un point autour de sa position d'équilibre $x_{eq}$.
La période propre $T_0$ est supposée connue. À $t = 0$, ce point passe par $x_0$ avec la vitesse $v_0$. Nous envisageons différentes valeurs du facteur de qualité. On peut montrer que :

• pour $Q > \frac{1}{2}$ : $x(t) = x_{eq} + \mathrm{e}^{-\frac{\omega_0}{2Q}t}\left(A\cos\left(\omega'_0 t\right) + B\sin\left(\omega'_0 t\right)\right)$ avec $\omega'_0 = \omega_0\sqrt{1-\frac{1}{4Q^2}}$

$A = x_0 - x_{eq}$ $\quad B = \frac{v_0 + \frac{\omega_0}{2Q}\left(x_0 - x_{eq}\right)}{\omega'_0}$

• pour $Q < \frac{1}{2}$ : $x(t) = x_{eq} + A\mathrm{e}^{-\delta_1 t} + B\mathrm{e}^{-\delta_2 t}$ avec $\delta_1 = \omega_0\left(\frac{1}{2Q} - \sqrt{\frac{1}{4Q^2}-1}\right)$ $\quad \delta_2 = \omega_0\left(\frac{1}{2Q} + \sqrt{\frac{1}{4Q^2}-1}\right)$

$A = \frac{\delta_2\left(x_0 - x_{eq}\right) + v_0}{\delta_2 - \delta_1}$ $\quad B = -\frac{\delta_1\left(x_0 - x_{eq}\right) + v_0}{\delta_2 - \delta_1}$

• pour $Q = \frac{1}{2}$ : $x(t) = x_{eq} + (At + B)\mathrm{e}^{-\omega_0 t}$ avec $A = v_0 + \omega_0\left(x_0 - x_{eq}\right)$ $\quad B = x_0 - x_{eq}$

b) définition de la fonction

Selon les valeurs de $Q$, la fonction donc prendra différentes formes.

***Exercice F3 (à rendre)***

Définir une fonction permettant d'obtenir l'équation horaire *x(t)* quel que soit le régime d'amortissement, en utilisant des *instructions conditionnelles*. Le script de la fonction contiendra la position d'équilibre, la période propre ainsi que les position et vitesse initiales. Il demandera à l'utilisateur d'entrer la valeur du facteur de qualité en utilisant la fonction `input` vue précédemment.
Enregistrer le script correspondant ('*mouvement oscillatoire - tout régime.py*', par exemple).
► C'est ce script que vous me transmettrez.

Pour la suite, le but sera d'obtenir le graphe de *x(t)*. Nous devons tout d'abord rappeler comment tracer un graphe avec Python.

## 2. GRAPHE D'UNE FONCTION À 1 VARIABLE

### 2.1. Les instructions `plot()` et `show()`

Pour tracer des courbes, Python n'est pas suffisant et nous avons besoin du module Matplotlib.
Importation (rappel) :

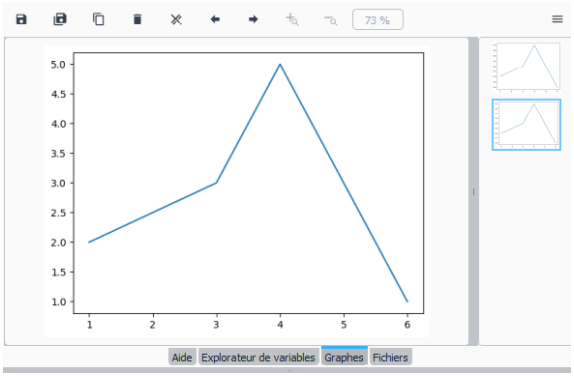
```
import matplotlib.pyplot as plt
```

L'instruction `plot()` permet de tracer une courbe qui relie des points dont les abscisses et ordonnées sont fournies en arguments : `plt.plot(abscisses,ordonnées)`
L'instruction `show()` permet d'afficher cette courbe à l'écran.

a) exemple

Dans le script qui suit , nous utilisons une liste de 4 abscisses et une liste de 4 ordonnées :

```
import matplotlib.pyplot as plt
plt.plot([1, 3, 4, 6], [2, 3, 5, 1])
plt.show()  # affiche la figure à l'écran
```




Visualisation du graphe ?
Les graphes sont affichés dans le volet [Graphes], cherchez bien…

b) application à notre problème d'oscillations

***Exercice G1 (solution disponible)***

> Reprenons la fonction définie et enregistrée dans l'exercice **F1**. Pour ce problème, c'est donc le temps qui sera en abscisse. Complétez le script définissant la fonction, et ajoutez les lignes permettant de :
> - définir une liste `t` correspondant aux abscisses (nous choisirons un intervalle de temps allant de $0$ à $6$ s, avec 15 points).
> - tracer le graphe de la fonction `f(t)`.

Comment ça, c'est pas terrible ? Trouvez une solution.
Pour info : *to plot points on a graph* se traduit par *reporter des points sur un graphique*. Donc en fait, Python pose les points et les relie par des segments…

### 2.2. Mise en forme sommaire

a) définition du domaine d'affichage des axes : **`xlim()`**, **`ylim()`**

```
plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)
```

Pour que la courbe remplisse tout l'espace horizontal du graphe, par exemple, on fait coïncider `xmin` et `xmax` avec les bornes de la liste d'abscisses.

b) l'instruction **`axis('equal')`**

L'instruction `plt.axis('equal')` permet d'avoir la même échelle sur l'axe des abscisses et sur l'axe des ordonnées afin d'avoir une base orthonormée. Ceci n'a d'intérêt que si les dimensions des deux grandeurs sont les mêmes.

c) ajout d'un titre et du nom des axes : **`title()`**, **`xlabel()`** et **`ylabel()`**

```
plt.title('titre du graphe')
plt.xlabel('nom des abscisses')
plt.ylabel('nom des ordonnées')
```

d) ajout d'une grille : **`grid()`**

Pour ajouter une grille au graphique, il suffit d'utiliser :

```
plt.grid()
```

e) couleur et épaisseur de la courbe, style de ligne

```
plt.plot(t, x(t), color='color①', linestyle='line_style②', linewidth=1)
```

① Nom en anglais ou code-couleur, ou encore code RGB. Les codes-couleur sont les suivants :

| couleur | bleu | vert | rouge | cyan | magenta | jaune | noir | blanc |
|---|---|---|---|---|---|---|---|---|
| code | b | g | r | c | m | y | k | w |

② Si on veut des pointillés ou des tirets, on remplace `line_style` par « `:` », « `--` » ou « `-.` ».

Pour modifier l'épaisseur des courbes, on change la valeur de l'argument `linewidth` : la valeur par défaut est `1`, toute valeur est utilisable, y compris des valeurs décimales inférieures à 1.

f) tracé des points uniquement

```
plt.plot(t, x(t), 'o')
```

g) ajout d'une légende : **`label / legend()`**

```
plt.plot(t, x(t), label='étiquette')
plt.legend()
```

Exemple d'amélioration du script précédent :

```
#graphe
plt.plot(x, y1, color='b', label='cos(x)')
plt.plot(x, y2, color='r', label='sin(x)')
plt.legend()
plt.show()
```

### 2.3. Superpositions de graphes

Pour afficher plusieurs courbes sur un même graphe, on peut procéder de la façon suivante (exemple avec les fonctions prédéfinies *cosinus* et *sinus* :

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

x = np.linspace(0, 2*np.pi, 100)
y1 = np.cos(x)
y2 = np.sin(x)
#graphe
plt.plot(x, y1)
plt.plot(x, y2)
plt.show()
```

### 2.4. Sauvegarde du graphique

Pour enregistrer un graphique au format PNG, il suffit, dans la fenêtre "Graphes", de cliquer sur 💾 ou de taper Ctrl-S.

***Exercice G2 (solution disponible)***

Reprendre le script de l'exercice **G1** :
- en limitant l'affichage des abscisses à l'intervalle *[0 ; 1,5]*,
- en ajoutant un titre et en nommant les axes,
- en colorant la courbe en rouge, et en la faisant apparaître deux fois plus épaisse,
- en ajoutant une légende.
- en ajoutant une grille.

## 3. EXPLOITATION PHYSIQUE

Enfin ! Le but était de comparer graphiquement les différents régimes d'amortissement des oscillations libres amorties et de mettre en évidence l'influence de la valeur du facteur de qualité et des conditions initiales.

### 3.1. Influence de $Q$

Nous allons ici tracer sur un même graphe les différentes réponses d'un système suivant différentes valeurs du facteur de qualité : $Q=0,25$ ; $Q=0,5$ ; $Q=1$ ; $Q=5$.

Principe : nous allons exécuter 4 fois le script contenant la fonction ('*mouvement oscillatoire - tout régime.py*') depuis un deuxième script chargé des graphes, que nous nommerons par exemple '*mouvement oscillatoire - tracé 4 régimes.py*'. Nous utiliserons pour cela l'instruction `exec` suivant ce modèle :

```
exec(open('mouvement oscillatoire - tout régime.py').read())
```

Une fois le script lancé, l'utilisateur entre $Q$ et le graphe correspondant est mis en mémoire, avec une couleur donnée, et une légende correspondant à la valeur de $Q$. NB : pour écrire un titre ou une légende contenant une variable, voici comment procéder :

```
label=f'Q={Q}'
```

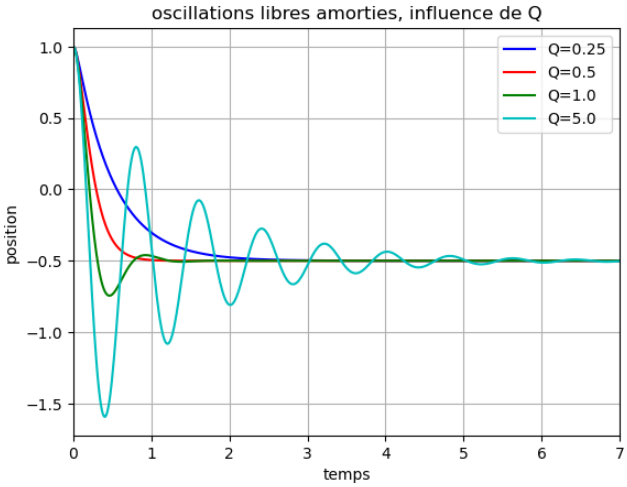
Le `f` indique que la légende contiendra une variable, et la variable, ici `Q`, est délimitée par des accolades.

Une fois le script exécuté 4 fois, avec les 4 valeurs de $Q$ souhaitées, on peut afficher les 4 graphes sur une même figure, avec `plt.show()`.
Sauvegarder le graphe.

***Exercice G3 (solution disponible)***

| Faites-le ! |
|---|

Résultat :




### 3.2. Influence des autres facteurs

Nous pouvons de même observer l'influence de la période propre, ainsi que des conditions initiales, sur le graphe de $x(t)$.