T.P. résolution numérique d'équations différentielles avec Python

1 Méthode d'Euler

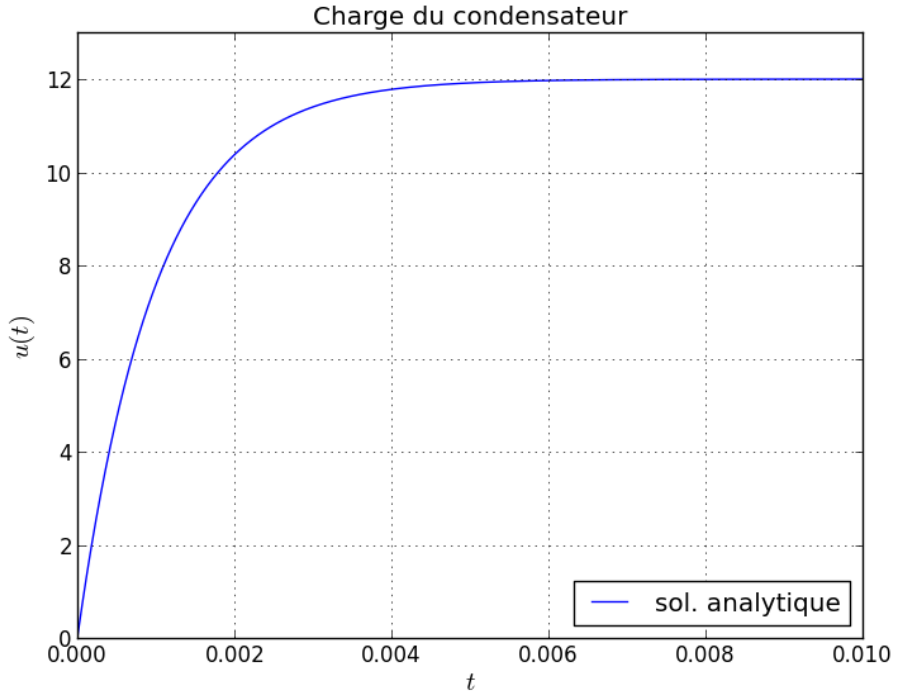
1.1 Résolution numérique d'une équation différentielle du premier ordre par la méthode d'Euler.

On souhaite résoudre l'équation différentielle linéaire modélisant la charge d'un condensateur :

$$\frac{du(t)}{dt} + \frac{1}{\tau}u(t) = \frac{E}{\tau}$$

Avec la constante de temps $\tau = RC$.

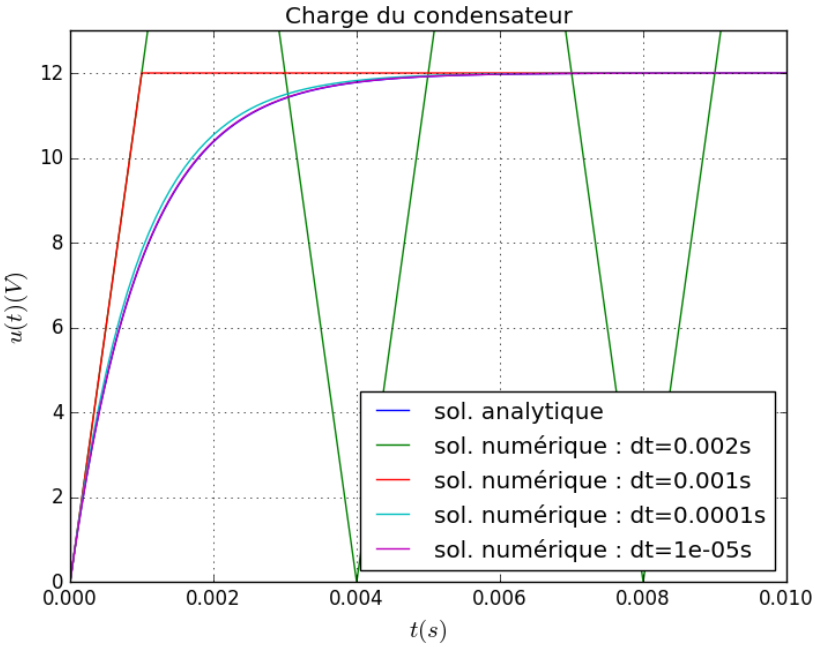
1. Donner la solution de cette équation en prenant E = 12 V, R = 50 Ω et C = 20 µF, sachant que le condensateur est initialement déchargé.
2. Tracer la solution analytique sur une durée de 10τ pour obtenir le graphe suivant :



Remarque : l'argument **`loc = 4`** dans la fonction **`legend()`** permet de placer la légende en bas à droite du graphe.

3. Programmer la fonction **`du_dt`** correspondant à l'équation différentielle à résoudre.
4. Programmer la fonction appelée **`euler`** permettant la résolution numérique de cette équation différentielle. Pour rappel, la fonction **`euler`** va prendre en arguments :
   - une fonction f
   - les bornes a et b de l'intervalle d'étude (a et b sont donc des temps)
   - le pas h.
   - la condition initiale y_0
5. Résoudre numériquement cette équation pour h=0.00001, a=0 s, b=10 τ s et $u(0)$= 0V. Tracer cette solution et la comparer à la solution analytique en zoomant sur les courbes afin de voir les différences.
6. Ensuite, résoudre numériquement cette équation pour des pas de 0.002, 0.001, 0.0001 et 0.00001. On le fera à l'aide d'une boucle : **`for dt in [0.002,0.001,0.001,0.00001]:`**

7. Tracer les quatre solutions précédentes sur le même graphe que la solution analytique et zoomer sur la courbe pour voir les différences entre la solution exacte et les solutions numériques.
8. Conclure quant à l'effet du pas de calcul.



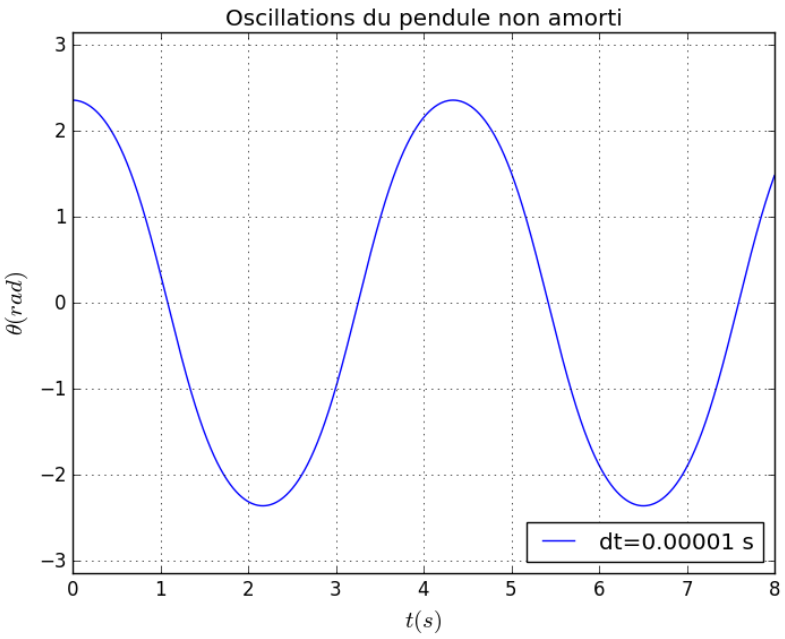
1.2 Résolution numérique d'une équation différentielle du deuxième ordre par la méthode d'Euler

1.2.1 Exemple du pendule simple non amorti

L'équation différentielle correspondant à l'oscillation non amortie d'un pendule simple est :

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

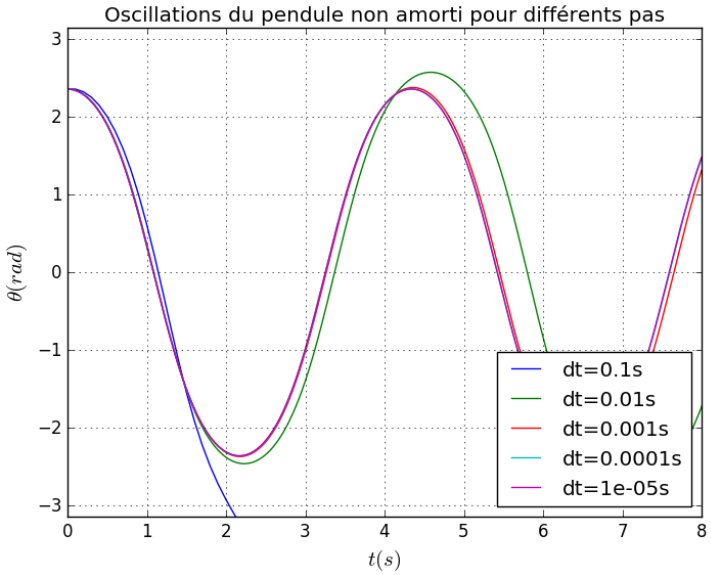
avec $\omega_0^2 = \frac{g}{L}$ (L : longueur du pendule et g constante de gravitation terrestre), l'angle θ étant défini entre la verticale et la direction du pendule.



a) Résoudre numériquement l'équation différentielle pour un pas de 0.00001 s pour t allant de 0 à 8 s avec les conditions initiales $\theta(0) = \frac{3\pi}{4}$ et $\dot{\theta}(0) = 0$. On prendra $g = 9.8$ m.s^{-2} et $L = 2.0$ m.
On définira pour cela la fonction **`pendule`** (avec les fonctions de numpy) ainsi que la fonction **`euler`** qui va prendre en entrée :

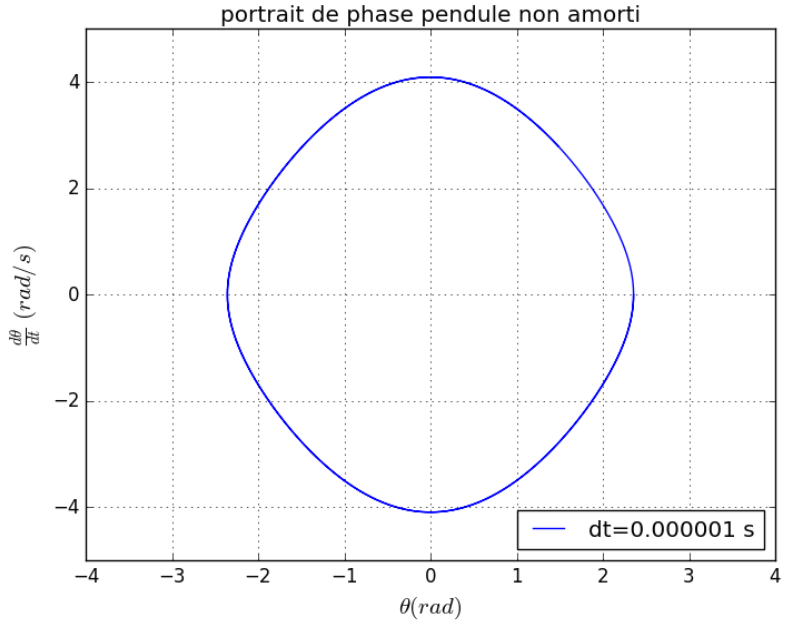
- une fonction f
- les bornes a et b de l'intervalle d'étude (a et b sont donc des temps)
- le pas h.
- la condition initiale y_0
- la condition initiale y_0'

b) Faire de même pour des pas de 0.1, 0.01, 0.001, 0.0001 et 0.00001 s. Superposez les courbes sur le même graphe.



c) Tracer le portrait de phase $\frac{d\theta}{dt} = f(\theta)$ pour un unique pas de 0.00001 s et avec les mêmes conditions initiales que précédemment.

Remarques :
On obtient la fraction $\frac{a}{b}$ avec **`\frac{a}{b}`**.
On obtient des lettres grecques avec **`r"$\lettre_grecque$"`** (le **`r`** signifie que la chaîne est du type raw et non une chaîne Python classique, ce qui permet d'écrire des équations *TeX*).



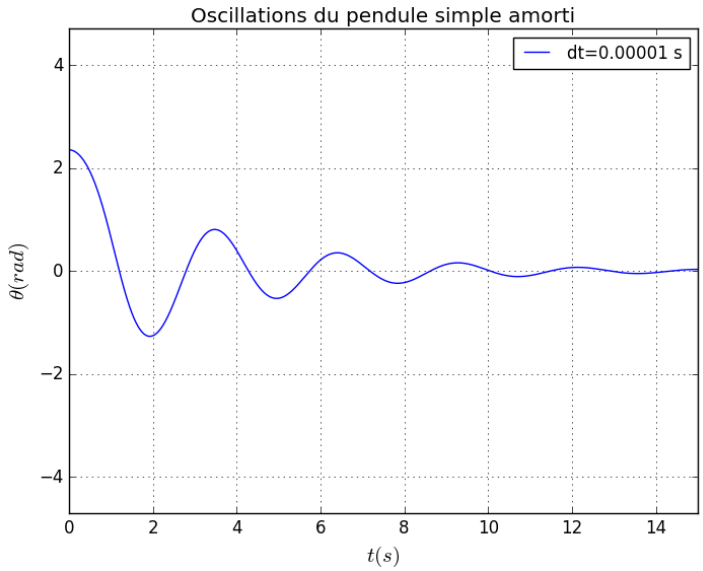
1.2.2 Exemple du pendule simple amorti

a) Reprendre l'exemple précédent en considérant un frottement visqueux conduisant à l'équation :

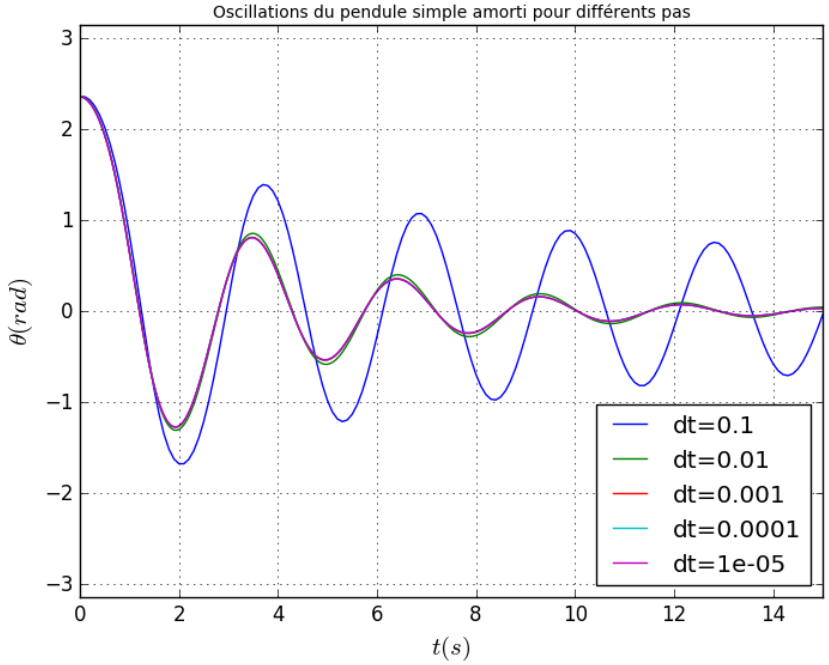
$$\ddot{\theta} + \frac{\omega_0}{Q}\dot{\theta} + \omega_0^2 \sin\theta = 0$$

Tracer la nouvelle solution pour un pas=0.00001 sur un intervalle de temps de 15 s. On prendra comme facteur de qualité $Q = 4$ et les conditions initiales : $\theta_0 = \frac{3\pi}{4}$ rad ; $\dot{\theta}_0 = 0\, rad/s$

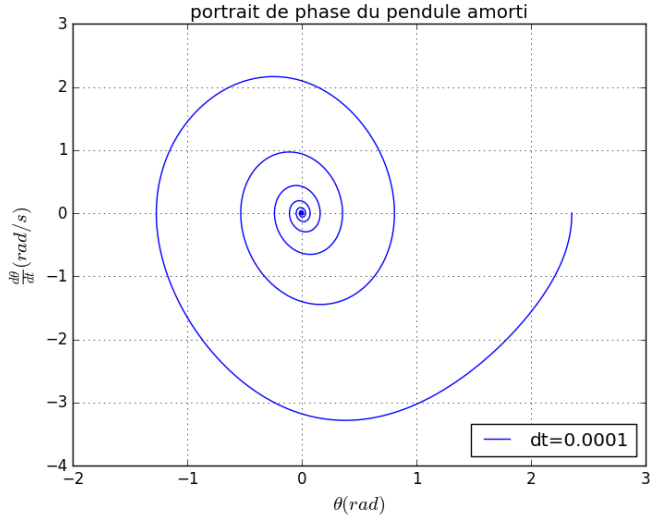
Remarque : la racine carrée de numpy est la fonction **`np.sqrt()`**.



b) Tracer sur la même figure les différentes solutions pour des pas différents : 0.1 s, 0.01 s, 0.001 s, 0.0001 s, 0.00001 s



c) On tracera également le portrait de phase pendant une durée plus longue (30 s, par exemple) et pour un pas de 0.0001 s (mêmes conditions initiales).



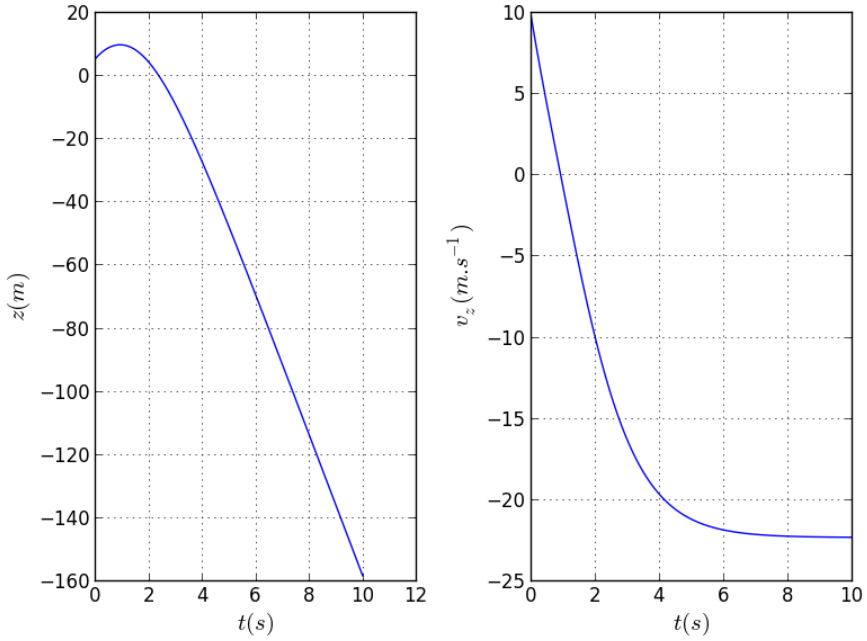
1.2.3 Exemple 2

Déterminer la trajectoire et la vitesse d'un système vérifiant l'équation :

$$\ddot{z} + 0{,}02 \times \dot{z} \times |\dot{z}| = -10$$

Avec $z(0) = 5.0$ m et $\dot{z}(0) = 10.0$ m.s^{-1}. On considère ici un mouvement à une dimension selon l'axe (Oz) vertical vers le haut avec un frottement quadratique.
Tracer le graphe pour $t \in [0{,}10]$ s et $dt = 0.01$ s.
On observe bien l'existence d'une vitesse limite.

Remarque : la valeur absolue de numpy s'écrit **`np.abs()`**.



La commande **`plt.subplot()`** permet de tracer plusieurs figures dans une même fenêtre. Les arguments de cette commande sont le nombre de lignes de graphes, le nombre de colonnes de graphes et le numéro de la figure.
La commande **`plt.tight_layout()`** permet d'espacer davantage les figures.

2 Utilisation de la bibliothèque scipy pour résoudre des équations différentielles

2.1 Généralités

Pour gagner en fiabilité et en rapidité, il est possible d'utiliser des fonctions fournies par les bibliothèques de Python. On peut par exemple se servir de la bibliothèque scipy qui est très fournie.

On trouve en ligne un tutoriel officiel :

http://docs.scipy.org/doc/scipy/reference

La bibliothèque **`scipy.integrate`** contient la fonction **`odeint`** (intégration des équations différentielles ordinaires), qui résout numériquement des équations différentielles ordinaires, c'est-à-dire sans dérivées partielles. On charge la fonction par :

```
from scipy.integrate import odeint
```

Une utilisation basique sera de la forme : **`odeint(`**f, y_0, t**`)`** pour résoudre l'équation $y'(t) = f(y(t), t)$ sur un intervalle $[a, b]$.

Remarque importante : Lorsque l'on définira la fonction f utilisée en paramètre dans **`odeint`**, il faudra qu'elle admette comme argument d'entrée la variable t (même si elle n'apparaît pas dans l'équation). **t doit figurer en tant que deuxième argument de f, et non en premier**.

t est défini dans le programme comme un tableau de valeur défini sur l'intervalle $[a, b]$. Ce tableau est de type **`array`** et il est défini par la fonction **`arange`** ou la fonction **`linspace`** de la bibliothèque **`numpy`**.

```
>>> t=np.arange(0,10*to,10*to/1000) #t est un tableau de 1000 valeurs réparties sur un intervalle de 10*to
>>> t=np.linspace(0,10*to,1000) #t est un tableau de 1000 valeurs réparties sur un intervalle de 10*to
```

La solution renvoyée par **`odeint`** est un tableau contenant une estimation de la solution aux différents temps.

2.2 Travail demandé

a) Calculer numériquement la solution de l'équation différentielle linéaire de la partie 1.1 avec la fonction *odeint*, en utilisant les mêmes valeurs numériques. Le tableau de temps à utiliser comprendra 1000 points. Tracer le résultat ainsi que la solution exacte. Zoomer jusqu'à observer une différence entre les deux courbes. Conclusion ?

b) Réaliser le même travail avec l'équation différentielle de la partie 1.2.1 en utilisant la méthode de vectorialisation vue en cours pour laquelle on résout l'équation : $[y'(t), y''(t)] = F([y(t), y'(t)])$ ou $\frac{d\,X}{dt} = F(X)$ avec $X = [y(t), y'(t)]$

Si on veut récupérer la solution $y(t)$, il faut récupérer uniquement la première colonne du tableau solution. Pour cela, on utilisera la commande mon_tableau[:,0] qui retourne les éléments de la première colonne contenus dans mon_tableau.

```
>>> tableau = np.array([[2,1],[3,4],[5,6]])
>>> print(tableau[:,0])
 [2 3 5]
```

Si on veut récupérer la solution $y'(t)$, il faut récupérer uniquement la deuxième colonne du tableau solution. Pour cela, on utilisera la commande mon_tableau [:, 1] qui retourne les éléments de la seconde colonne contenus dans mon_tableau.

Remarque : comme on utilise la bibliothèque numpy pour créer le tableau de temps, ce dernier n'est pas de type « liste » mais de type « array ». Il en va de même pour la solution obtenue par odeint.

c) Réaliser le même travail avec l'équation différentielle de la partie 1.2.2. Tracer le portrait de phase de la solution.

d) Réaliser le même travail avec l'équation différentielle de la partie 1.2.3. Tracer deux graphes cote à cote représentant l'évolution en fonction du temps de $z(t)$ et $\dot{z}(t)$.