TD numérique pour la physique n°2 – vectorisation des ED

Description du TD

A) Étude d'un système d'ED couplées d'ordre 1 et critique de la méthode d'Euler ;

B) Résolution numérique d'une ED d'ordre 2 bien connue.

Théorie

On rappelle que toute méthode de résolution numérique d'équations différentielles, comme Euler et RK4, se base sur la définition de la fonction fondamentale $\dot{y}=F(y,t)$.

Il semble donc que seule la résolution des ED d'ordre 1 est possible, ce qui pose problème car la majorité des ED rencontrées en physique sont d'ordre 2.

Q1. Pourquoi est-ce le cas ?

En réalité, la méthode est très générale, car y peut être un **vecteur** dans l'application de la méthode, vecteur de dimension quelconque.

Nous verrons qu'une ED d'ordre 2 peut se ramener à un système d'équations couplées d'ordre 1, donc à une ED d'ordre 1 portant sur un vecteur 2D.

Partie A

On va travailler sur l'évolution d'un écosystème très simplifié à deux acteurs : les proies (population x) et les prédateurs (population y : voir TD 14, exercice 7).

On utilise le modèle de Volterra-Lotka, où les 4 constantes sont strictement positives.

$$\begin{cases} \dfrac{dx}{dt}=x(\alpha-\beta y) \\ \dfrac{dy}{dt}=y(\delta x-\gamma) \end{cases}$$

Le vecteur décrivant les fonctions inconnues est donc $Y=\begin{pmatrix} x \\ y \end{pmatrix}$ et sa dérivée est $\dot{Y}=\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}$.

La fonction F est alors définie par $F:\begin{pmatrix} x \\ y \end{pmatrix},t\rightarrow\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}=\begin{pmatrix} x(\alpha-\beta y) \\ y(\delta x-\gamma) \end{pmatrix}$.

On voit que, pour l'implémenter, il faut :

- récupérer les deux coordonnées du vecteur Y
- construire et retourner un nouveau vecteur à partir de ses coordonnées calculées : $\dot{Y}$

Voir ces deux points techniques dans les Exigibles numériques.

Q2. Créer un fichier `TD 2 volterra.py`, et reprendre les `import` faits au début du TD précédent (n°1).

Définir la fonction `Fvolterra` en utilisant les valeurs $\left(\alpha=\frac{2}{3};\beta=\frac{4}{3};\gamma=\delta=1\right)$ – inutile de définir des variables pour ces coefficients, ils ne seront présents que dans `Fvolterra`…

Créer une liste (variable globale) de `N=500` dates de 0 à `tMax=30`, `N` et `tMax` étant également

définies comme des variables globales (donc facilement modifiables).

Q3. Test rapide : pour voir rapidement le type de ce que renvoie la résolution numérique par `odeint`, appeler `odeint` dans la console, avec la valeur initiale (2.0 , 0.6), et les dates [0,1,2].

Q4. En déduire le codage de la fonction `resoutVolterra`, qui admet comme arguments x0, y0 et qui renvoie les deux listes `[x]` et `[y]` dans un tuple.

Cette fonction appelle bien sûr `odeint`.

Q5. Afficher les courbes d'évolution des proies et des prédateurs (utiliser des légendes pour les courbes).

Comprendre la logique des courbes obtenues : quelle population croît ou décroît et pourquoi ? Plus ou moins vite ?

On veut maintenant voir comment Euler s'en sort…

Q6. Définir et coder (cf TD n°1) une fonction `euler2D`, qui admet exactement les mêmes arguments que les arguments obligatoires de `odeint` et fait la même chose.

Q7. Modifier la fonction `resoutVolterra`, pour qu'elle appelle `euler2D` au lieu de `odeint`, puis (après la question 8 éventuellement) sa déclaration pour que la méthode de résolution soit celle souhaitée.

Par exemple, si l'on veut utiliser la résolution par Euler, l'appel de `resoutVolterra` devra être : `xList,yList=volterra(x0,y0,euler2D)`

Vous pouvez, si vous le souhaitez, utiliser un '=' pour l'argument dans la déclaration de fonction pour que, par défaut, ce soit `odeint` qui est appelée (cf aide du TD n°1).

Q8. On a vu au TD n°1 que RK4 était plutôt fiable : on peut faire confiance à l'allure des courbes obtenues.

Que peut-on dire des erreurs commises lors de la résolution par la méthode d'Euler (N est relativement grand ici) ? Est-ce grave ? Comparer le comportement avec celui de la méthode d'Euler dans le TD n°1.

Partie B

On travaille maintenant sur une ED d'ordre 2 que l'on sait résoudre : les oscillations linéaires, amorties ou non.

Pour résoudre numériquement les équations d'ordre 2, on définit le vecteur $Y=\begin{pmatrix} x \\ \dot{x} \end{pmatrix}$.

Q9. Définir théoriquement son vecteur dérivé $\dot{Y}$ et la fonction `Foscill`, associée à l'équation différentielle linéaire d'ordre 2 homogène : $\ddot{x}+2\lambda\dot{x}+\omega_0^2 x=0$.

On n'utilise pas Q ici, car on veut pouvoir programmer le cas de l'oscillateur harmonique.

Q10. Sauver le fichier `volterra`, et travailler sur un fichier très inspiré du précédent,

`TD 2 oscillateur.py` : on prendra $\omega_0=1$.

On affichera les deux courbes, de la position et de la vitesse, avec les CI que l'on souhaite… on prendra un nombre de points en accord avec les capacités de la machine, et attention à ajuster à chaque fois la date maximale pour obtenir de jolies courbes.

- Tester Euler sur l'oscillateur harmonique et conclure.
- Visualiser (avec une méthode correcte) quelques régimes libres, de types différents (selon les valeurs de λ : retrouver les types d'abord théoriquement).
- Visualiser quelques régimes excités sinusoïdalement à la pulsation ω, qui peut être prise notablement supérieure, notablement inférieure, ou bien égale à ω_0 .

Exigibles numériques

- Création d'un vecteur

  En Python, un vecteur peut être implémenté indifféremment par une liste [], un tuple () qui est une liste non modifiable, ou un tableau `numpy`, structure qui permet des calculs plus aisés (sans boucles) que sur les listes (addition, fonction appelée sur chaque élément).

  Ici, nous n'aurons jamais besoin de modifier le contenu d'un vecteur, donc on utilisera les tuples.

  Le code à écrire est donc tout simplement, à partir de deux variables `x` et `y` :

  ```
  Y=(x,y)
  ```

  et Python accepte la syntaxe raccourcie suivante qui produit le même résultat, un tuple :

  ```
  Y=x,y
  ```

- Récupération des coordonnées d'un vecteur dans 2 variables

  Le codage intuitif est `x=Y[0]` puis `y=Y[1]`, mais le codage direct suivant, qu'on utilisera, est beaucoup plus clair et rapide à écrire :

  ```
  (x,y)=Y
  ```

  ou bien sûr, encore plus simple :

  ```
  x,y=Y
  ```

  Mais attention ! la longueur de Y doit être exactement 2, sinon une erreur sera produite.

  (si l'on n'en est pas certain, on peut assurer le code en écrivant `x,y=Y[:2]`, qui tronque la liste, de l'indice 0 sous-entendu à l'indice 2 exclu, donc qui garde les indices 0 et 1).