TE 5 : Résolution numérique d'équations différentielles ordinaires (EDO)

Objectif

L'objectif est de mettre au point des méthodes générales de résolution numériques des EDO s'appliquant à des systèmes d'EDO couplées quel que soit leur nombre et leur ordre : par exemple les EDO du premier ordre de type Lotka-Volterra régissant l'évolution de plusieurs populations en compétition ou les EDO du second ordre de plusieurs pendules couplés.
On commence par le cas simple des EDO scalaires du premier ordre (II). On convertit ensuite le code scalaire en vectoriel (III), afin de traiter des systèmes d'équations différentielles couplées et, par les mêmes programmes, des EDO d'ordre supérieur. Dans une partie préliminaire (I), on se propose de rechercher les solutions analytiques de quelques EDO pour les comparer aux solutions numériques.

I Résolution analytique d'EDO scalaires du premier ordre

Objectif : résoudre analytiquement les équations différentielles suivantes, puis tracer les solutions sous python. Les solutions d'une EDO du premier ordre dépendent d'un seul paramètre : par exemple la condition initiale, à l'instant $t_0 = 0$, $y(t_0)$ est notée y_0.

I.1 EDO linéaires

L'EDO qui régit l'évolution d'une population (croissance ou extinction) avec une constante de temps τ en présence d'un forçage externe b s'écrit :

$$\tau \frac{\mathrm{d}y}{\mathrm{d}t} + y(t) = b(t) \tag{1}$$

Rechercher la solution analytique dans les cas suivants.

1. Sans forçage, $b = 0$.
2. Avec un forçage constant $b = b_0$.
3. Avec un forçage périodique $b = b_0 \sin(\omega t)$.
   Dans ce cas, il est plus simple de rechercher comme solution particulière la solution permanente de type sinusoïdal $y_p(t) = A\cos(\omega t) + B\sin(\omega t)$. Discuter les cas limites $\omega\tau \ll 1$ et $\omega\tau \gg 1$.

Représenter les solutions dans les trois cas sur l'intervalle $[0,\ 20]$ avec $\tau = 4.$, $b_0 = 8$, $\omega = 0.5$ et $y_0 = 10$.
Dans le cas (1) sans forçage, tracer à la main sur un même graphique, les solutions obtenues pour plusieurs conditions initiales $y_0 = -10, -5, +5, +10$ et (!) $y_0 = 0$.

I.2 EDO scalaire du premier ordre : l'équation logistique

L'équation différentielle (2) régit l'évolution d'une population avec un taux de croissance $a > 0$ en présence d'un mécanisme la faisant tendre vers k supposé positif pour $t \to \infty$.

$$\frac{\mathrm{d}y}{\mathrm{d}t} = ay\left(1 - \frac{y}{k}\right) \tag{2}$$

Montrer que la solution de l'EDO (2) se met sous la forme (3). Discuter le sens d'évolution de $y(t)$ selon la valeur de y_0 supposé positif.
Choisir $a = 1$, $k = 2$, $t_0 = 0$. Représenter la solution dans les cas où $y_0 = 0,1$ puis $y_0 = 4$.

$$y(t) = \frac{k}{1 + \dfrac{k - y_0}{y_0}\exp\left(-a(t - t_0)\right)} \tag{3}$$

I.3 EDO scalaire du premier ordre à solution gaussienne

On recherche la solution de l'équation différentielle (4) avec la condition initiale $y(t_0) = y_0$. On suppose que $k > 0$, $y_0 > 0$ et $a > t_0$.

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -(t-a)\,y/k^2 \tag{4}$$

Montrer que la solution de l'EDO (4) se met sous la forme (5) d'une gaussienne centrée en a et de largeur k.
Représenter la solution dans l'intervalle $[0,\ 20]$. On choisit $a = 4.$, $k = 2.$ et $y_0 = 0,5$.

$$y(t) = y_0 \exp\left[\frac{1}{2}\left(\frac{t_0-a}{k}\right)^2\right] \exp\left[-\frac{1}{2}\left(\frac{t-a}{k}\right)^2\right] \tag{5}$$

Ne pas programmer directement l'expression (5), mais la reformuler pour limiter les risques de dépassement de capacité (liés à l'exponentielle) et les erreurs d'arrondi (dans les différences).

II Résolution numérique des EDO scalaires du premier ordre

L'objectif est d'analyser l'influence de l'ordre de la méthode et de la valeur du pas sur l'erreur de troncature. Pour de très faibles valeurs du pas, on mettra en évidence les erreurs d'arrondi d'aspect aléatoire : elles dépendent peu de l'ordre de la méthode mais augmentent quand le pas diminue jusqu'à devenir dominantes. Le meilleur compromis sera obtenu avec une méthode d'ordre élevé et un pas faible, mais pas trop !

II.1 Mise en œuvre informatique de la résolution en python

Outre le programme principal, le code comporte quatre modules :

1. Le **programme principal** choisit les instants de début `t0` et de fin `tfin`, le nombre `n` d'instants, la méthode de résolution, la valeur initiale `y0`, la fonction second membre à intégrer et la fonction analytique solution si elle est connue. Il **définit** aussi les valeurs des paramètres (`a` et `k` par exemple) des fonctions à tester et les groupe dans un tuple `par`. Il construit les tableaux `tps` des instants et éventuellement `yexact` de la solution analytique. L'ensemble de ces informations hormis `yexact` sont fournies à la fonction d'intégration numérique `integre` qui renvoie le tableau `yapprox` de la solution numérique. Le programme principal peut alors appeler les fonctions utilitaires d'écriture sur fichier et de représentation graphique des solutions et des écarts entre numérique et analytique.
2. La fonction d'**intégration numérique** `integre` initialise la solution avec `y0`. Elle lance ensuite la boucle qui appelle à chaque pas le solveur choisi pour calculer de proche en proche la solution numérique et la stocke dans le tableau `yapprox`. Enfin, elle renvoie le tableau `yapprox` au programme principal.
3. La définition des différentes **fonctions second membre** $f(t,y) = \dfrac{\mathrm{d}y}{\mathrm{d}t}(t,y)$ à intégrer, ainsi que les **solutions analytiques** $y_{\text{exact}}(t, t_0, y_0)$ éventuellement associées. Ces couples de fonctions utilisent des paramètres communs (`a` et `k` par exemple) qui ont été définis dans le programme principal.
4. La définition des **méthodes d'intégration** (solveurs) des différents ordres (Euler, Point milieu, ...) qui permettent de progresser d'un pas, de t_i à t_{i+1}. On leur passe en argument la fonction second membre à intégrer qu'elles évalueront en plusieurs points. Ces méthodes seront conçues de façon à pouvoir traiter une fonction second membre quelconque.
5. Des fonctions **utilitaires** permettant de sauvegarder les résultats dans un fichier texte (avec la fonction `savetxt` de `numpy`) et de les tracer. Le nom du fichier sera choisi en fonction de la méthode et du pas (`logistique-euler-0.1.dat` par exemple signifiant problème logistique avec méthode d'Euler et pas de `0.1`). Le corps du fichier des résultats comportera une ligne par instant sous la forme suivante dans le cas où une solution analytique est fournie :

abscisse	ordonnée estimée	ordonnée analytique	écart
t_i	u_i	$y(t_i)$	$u_i - y(t_i)$

Une fois la méthode mise au point, on insérera dans le fichier de résultats un entête de trois lignes indiquant les paramètres de cette simulation. Ces lignes d'entête commenceront par le caractère `#` ce qui permet à la fonction python `loadtxt` de les ignorer à la relecture du fichier.

1. le nom du fichier et les bornes des abscisses
2. l'ordre de la méthode de résolution, le pas d'échantillonnage et la valeur de l'écart maximal
3. enfin, un libellé succinct pour chaque colonne de valeurs

II.2 Les paramètres du second membre

Les paramètres des EDO (par exemple `a` et `k` pour la logistique et la gaussienne) communs au second membre et à la solution analytique, sont fixés dans le programme principal et utilisés par ces fonctions. Le nombre de paramètres dépend de l'EDO particulière à résoudre. En revanche, les solveurs doivent être indépendants de ces paramètres. Or la fonction second membre n'est appelée que via les solveurs. Une solution sans variables globales consiste à rassembler les paramètres dans un tuple `par` défini dans le programme principal : la taille du tuple dépend de l'EDO et le tuple peut aussi être vide [1] s'il n'y a aucun paramètre. Ce tuple est passé en argument à la fonction d'intégration qui se contente de le transmettre tel quel au solveur, lequel le passe à son tour sans modification à la fonction second membre qui seule l'utilise réellement.
Par rapport à la présentation théorique des méthodes d'intégration, ce tuple constitue dans la pratique un argument supplémentaire de nombreuses fonctions.

II.2.1 Les quatre modules et cinq fichiers du code

Les différentes fonctions seront regroupées dans quatre modules auxquels il faut ajouter le fichier du programme principal.

Programme principal
choix du second membre
choix des paramètres du second membre
choix de la solution analytique éventuelle
choix de l'échantillonnage
(début `t0`, fin `tfin`, nombre de points `n`)
choix de la valeur initiale `y0`
...
choix de la méthode (solveur)
...
Création et remplissage du tableau `tps` des instants et de celui `yexact` de la fonction analytique.
Appel de `integre`
...
écriture sur fichier et tracé

module `fcts.py`
Les fonctions second membre `dydt(par, t, y)` et les solutions analytiques `g(par, t, t0, y0)` associées.
Elles utilisent les paramètres communs passés via le tuple `par`

```
f_logistique(par, t, y)
a = par[0] ; k =par[1]
...
```

Les solutions analytiques associées passant par `(t0, y0)`

```
logistique(par, t, t0, y0)
a = par[0] ; k =par[1]
...
```

module `intedo.py`
La fonction `integre(f, par, tps, y0, meth)`
Initialisation avec `y0`
Boucle de calcul pas à pas

> Appel de la méthode d'intégration `meth` appliquée au second membre particulier `f` pour calculer u_{i+1} et remplir le tableau `yapprox`.

module `methodes.py`
Les méthodes d'intégration sur un pas, de type `milieu(f, par, yi, ti, h)`

1. Euler
2. point milieu
3. Runge Kutta d'ordre 4

Elles appellent plusieurs fois la fonction second membre de l'EDO `f` (de deux variables réelles et à valeur réelle) passée en argument des méthodes d'intégration.
Elles renvoient une estimation u_{i+1} de la solution au bord droit de l'intervalle $[t_i, t_{i+1}]$.

module `utils.py`
Les fonctions d'écriture des tableaux dans des fichiers
`ecrit(nom_fic, par, tps, yapprox, yexact)`
...

1. le nom du fichier doit indiquer l'EDO, la méthode et le pas
2. calcul de l'écart entre estimation et calcul analytique et recherche du maximum de sa valeur absolue ainsi que de l'instant associé.
3. écriture des résultats (un instant par ligne)

Les fonctions de tracé ont les mêmes arguments et les figures sont sauvegardées au format `pdf`.

II.3 Plan de travail pour les EDO scalaires

Pour l'équation logistique, on choisit $a = 1$, $k = 2$, $t_0 = 0$ et on cherche la solution dans l'intervalle $[0,\ 20]$ avec pour valeur initiale $y_0 = 0,1$. Comme cette EDO a été vue en cours, on se contente de la résoudre pour un seul pas $h = 0.4$ avec les différentes méthodes.

1. On rappelle qu'un tuple est délimité par des parenthèses et constitué d'éléments quelconques séparés par des virgules. `par = ()` affecte un tuple vide à `par` ; `par = (a,)` affecte à `par` un tuple d'une seule variable `a`, la virgule étant nécessaire pour indiquer que les parenthèses ne signifient pas un simple groupement.

Pour la gaussienne, on choisit $a = 4$, $k = 2$, $t_0 = 0$ et on cherche la solution dans l'intervalle $[0,\ 20]$ avec pour valeur initiale $y_0 = 0,5$. Pour cette EDO, on analyse en détail les erreurs en fonction du pas comme suit.

1. Rédiger le programme complet avec seulement la méthode d'Euler sans le calcul d'erreur. Visualiser les
1 ⇒ résultats pour un pas de 0,4. figure des solutions `sol-euler-04.pdf`
2. Reprendre l'étude avec un pas de 0,1.
3. Ajouter le calcul d'erreur et visualiser l'erreur en fonction du temps. Contrôler l'erreur maximale et sa position calculées par le programme.
4. Programmer progressivement les méthodes d'ordre supérieur et tester les pas de 0,4 et de 0,1. Avec
2 ⇒ chaque méthode, calculer le rapport entre les erreurs maximales obtenues pour un pas de 0,4 et un pas de 0,1. Le comparer avec le rapport attendu en fonction de l'ordre de la méthode
3 ⇒ figures d'erreur `euler-01.pdf` et `rk4-01.pdf`
5. Reprendre le calcul avec un pas $h = 0,01$ et effectuer la même comparaison entre $h = 0,1$ et $0,01$.
6. Avec la méthode de Runge Kutta d'ordre 4, comparer l'allure des signaux d'écart obtenus pour un pas
4 ⇒ $h = 0,01$ et $h = 0,0001$. En déduire quel type d'erreur domine dans chaque cas.

III Résolution numérique des EDO vectorielles et d'ordre supérieur

III.1 Introduction

Le passage d'un code pour EDO du premier ordre scalaire à un système d'EDO du premier ordre est quasi-immédiat concernant les méthodes de résolution qui manipulent alors des vecteurs au lieu de scalaires. Elles devront pouvoir traiter un nombre p quelconque de composantes. Les modifications les plus importantes concernent le programme principal (où est choisi p) et les utilitaires d'écriture et de tracé.
L'application du code au système couplé des EDO de Lotka-Volterra, du premier ordre à deux composantes est facile. Mais ce système ne dispose pas de solution analytique générale. La représentation des solutions numériques dans le plan de phase peut cependant permettre de vérifier comment est conservé l'invariant.
Maitriser le traitement des EDO d'ordre supérieur avec le code vectoriel des EDO du premier ordre suppose un travail d'interprétation des composantes. C'est pourquoi on choisit de traiter le cas du **pendule**. On rappelle qu'une EDO du second ordre peut être traduite en un système de deux EDO du premier ordre couplées.

III.2 Plan de travail pour les EDO vectorielles

1. Créer un répertoire spécifique pour implémenter les codes d'EDO vectorielles du premier ordre en gardant la même structure que pour le scalaire. On recopiera les codes des méthodes, mais bien sûr, ceux des fonctions devront comporter des affectations pour chaque composante des vecteurs. Les tableaux de résultats seront des tableaux 2D, dont le premier indice représentera le temps et le second les différentes composantes des vecteurs. La condition initiale est ici un vecteur.

   Commencer par mettre au point le programme principal et les fonctions d'une solution analytique avant d'aborder la partie résolution numérique.
2. Traiter d'abord le problème du pendule avec un second membre **linéarisé** $y'' = -k^2 y$.
   (a) Déterminer l'expression générale de la solution analytique en fonction des conditions initiales (C.I.) y_0 (position) et y'_0 (vitesse). Coder cette solution utilisant le vecteur des C.I. et la tracer.
   (b) Écrire le système vectoriel du premier ordre de dimension 2 représentant l'EDO scalaire du second ordre. Implémenter la résolution numérique vectorielle. Calculer l'écart avec la solution analytique (pour la position et la vitesse) et vérifier l'effet d'un changement de pas.
3. Compléter avec le second membre exact, **non-linéaire**.
   (a) Comparer d'abord la solution numérique de l'EDO avec second membre non-linéaire pour des amplitudes très faibles, avec la solution analytique dans le cas linéaire.
   (b) Tester ensuite l'évolution de la solution numérique quand a augmente. Repérer en particulier le passage en apériodique pour $|y'_0| > 2$.
   (c) Calculer l'énergie mécanique du système à chaque pas de temps et vérifier dans quelle mesure elle se conserve. On représentera l'écart relatif à l'énergie initiale.