TP2 : QUELQUES MÉTHODES NUMÉRIQUES

Introduction

Ce cours est destiné à reprendre certaines méthodes numériques, en particulier celles de résolution d'équations différentielles (mais aussi le calcul d'intégrales).
Votre cours de mathématiques aborde la résolution sur un intervalle I des équations différentielles dites linéaires, c'est à dire de la forme

$$Y'(t) = A(t)Y(t) + B(t) \ .$$

Dans cette équation, B est une fonction de I dans $\mathcal{M}_{n,1}(\mathbb{R})$, A est une fonction de I dans $\mathcal{M}_n(\mathbb{R})$ et Y est la fonction inconnue, de I dans $\mathcal{M}_{n,1}(\mathbb{R})$.
Exemple :

$$\begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = \begin{pmatrix} t & -t \\ 1-t^2 & t^3 \end{pmatrix} \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} + \begin{pmatrix} 1 \\ -t \end{pmatrix} \ .$$

Sous certaines conditions (continuité de A et B) cette équation accompagnée de la condition ("initiale") $Y(t_0) = Y_0$, t_0 étant un élément de I, admet une unique solution. C'est le théorème de Cauchy-Lipschitz.
Ce type d'équations est un cas particulier d'équations du type : $Y'(t) = F(Y(t), t)$, où F est une fonction de $\mathbb{R}^n \times \mathbb{R}$ dans $\mathbb{R}^n$ (on a assimilé $\mathbb{R}^n$ à $\mathcal{M}_{n,1}(\mathbb{R})$). Exemple :

$$\begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = \begin{pmatrix} t^2 x(t) + \sin(y(t)) \\ x(t)^3 \end{pmatrix} \ ; \text{ ici } F\left(\begin{pmatrix} x \\ y \end{pmatrix}, t\right) = \begin{pmatrix} t^2 x + \sin(y) \\ x^3 \end{pmatrix} \ .$$

On sait résoudre certaines de ces équations par l'usage de fonctions usuelles, mais dans de nombreuses situations issues de problèmes physiques, cette résolution "exacte" est impossible. On doit alors introduire des méthodes de résolutions dites "approchées" (ou numériques plutôt).
L'objet de ce cours est de rappeler certaines de ces méthodes (celle d'Euler essentiellement) et de les programmer. Cela sera également l'occasion d'utiliser la bibliothèque numpy.

1 Résolution numérique d'une équation différentielle

On l'applique tout d'abord à la résolution d'une équation scalaire (E) : $y'(t) = F(y(t), t)$ sur un intervalle du type $[t_0; t_0 + T]$, avec condition initiale $y(t_0) = y_0$.

1.1 Principe de la méthode d'Euler

La méthode d'Euler consiste à :

- subdiviser l'intervalle en n parties égales. On pose $h = T/n$, appelé pas ; et, pour $0 \leq k \leq n$, $t_k = t_0 + k\ h$.
- approcher, pour chaque k, $y(t_k)$ par une valeur y_k en partant de $k = 0$ (on a $y(t_0) = y_0$). Pour cela, on remarque que

  $$y(t_{k+1}) - y(t_k) = \int_{t_k}^{t_{k+1}} y'(t)\ dt \ .$$

  Pour obtenir y_{k+1} à partir de y_k, on approche, sur l'intervalle $[t_k; t_{k+1}]$, $y'(t)$ par $y'(t_k)$, égal du fait de l'équation à $F(y(t_k), t_k)$, lui même approché par $F(y_k, t_k)$. Ceci nous donne la relation :

  $$y_{k+1} - y_k = F(y_k, t_k) \times h \ ;$$

  et donc le schéma :

  $$y_{k+1} = y_k + F(y_k, t_k) \times h \ .$$

1.2 Mise en œuvre

On souhaite appliquer la méthode d'Euler au calcul d'une solution approchée de l'équation différentielle $y'+y = \sin(t)$ sur l'intervalle $[0, 30]$ avec la condition initiale $y(0) = 1$. On suppose que `matplotlib.pyplot` a été importé comme `plt` ; et `numpy` comme `np` .
Compléter le programme suivant afin qu'il effectue le calcul et trace le graphique représentant la solution (on pourra faire une première version sans numpy et sans fonction F_1 associée au système).

```
#Fonction associ\'ee au syst\`eme
def F1(y,t):
    return ...

# donn\'ees :
T=30.0 ; t0=0 ; y0=1.0 ; n=50
h = T/n


temps = np.linspace(t0,t0+T,n+1)
Sol = np.zeros(n+1)
Sol[0] = ...
for i in range(...):
    ...
plt.plot(... , ...)
```

1.3 Efficacité

La théorie montre que sous certaines hypothèses il existe une constante K telle que, pour tout k,

$$|y(t_k) - y_k| \leq K\,|h| \ .$$

La méthode d'Euler est dite d'ordre 1. Par exemple, pour augmenter la précision d'un facteur 10, il faut diviser h par 10.
Dans ce cours d'informatique, on s'intéressera surtout à la complexité en temps et en espace. Il est clair que la fonction précédente a une complexité linéaire en la taille n de la subdivision.

1.4 Utilisation de odeint

Le module `scipy.integrate` de python dispose d'une méthode d'intégration numérique des équations différentielles (plus efficace que la méthode d'Euler). Sur notre exemple, cela donne (avec `F1`, `y0` et `temps` définis précédemment) :

```
import scipy.integrate as integr

SolScipy = integr.odeint(F1,y0,temps)
plt.plot(temps,SolScipy)
```

1.5 Dimension supérieure

La méthode d'Euler peut être adaptée au cas où la fonction recherchée y est vectorielle : $y(t) = (y_1(t), \ldots, y_m(t))$. La fonction F qui intervient dans l'équation différentielle $y'(t) = F(y(t), t)$ est alors définie sur $\mathbb{R}^m \times \mathbb{R}$ à valeurs dans $\mathbb{R}^m$. Le principe du schéma est alors le même.
Écrire une fonction `EulerDim2(t0,T,Y0,n,F)` dans le cas où la fonction inconnue est une fonction Y à valeurs dans $\mathbb{R}^2$. La fonction `F` prend en arguments un tableau de 2 éléments et un temps `t` et renvoie un tableau de deux éléments. La solution `Y` renvoyée sera un tableau à 2 lignes et $n+1$ colonnes. Compléter :

```
def EulerDim2(t0,T,Y0,n,F):
    h=T/n
    temps = np.linspace(0 , t0+T , n+1)
    list_Y=...
    list_Y[:,0]=...
    for i in range(n):
        ...
    return list_Y
```

On va tester notre fonction sur le système suivant :

$$Y'(t) = AY(t) \qquad \text{avec} \qquad A = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix} .$$

Compléter le programme suivant afin d'obtenir le tracé du diagramme de phase de la solution (c'est à dire le tracé de la courbe paramétrée $t \mapsto (y_1(t), y_2(t))$).

```
def F_2(Y,t):
    return ...

T=3.
t0=0
Y0=[0,1]

n=100
h=T/n

Sol = EulerDim2(t0,T,Y0,n,F)
temps = np.linspace(0 , t0+T , n+1)
X = ...
Y = ...
plt.plot(X , Y , 'r')
```

On pourra à titre d'exercice faire une version de cette résolution sans recours à une fonction associée au système et sans utiliser numpy.

2 Équations d'ordre supérieur

On peut ramener une équation différentielle d'ordre supérieur à 1 à une équation d'ordre 1, d'inconnue une fonction vectorielle.
On explique cela sur l'exemple suivant : $y''(t) + 4y(t) = 0$, avec conditions initiales $y(0) = 0.5$ et $y'(0) = 0.1$.

1. En posant $Y(t) = (y(t), y'(t))$, l'équation différentielle s'écrit sous la forme $Y'(t) = F_3(Y(t), t)$ avec :
$$F_3((y_1, y_2), t) = (y_2, -4y_1) .$$
2. Écrire la fonction `F3(Y,t)` associée au problème.
3. Écrire une suite d'instruction permettant de tracer la solution obtenue avec $T = 10$, $n = 100$ (on pourra réutiliser une fonction précédemment écrite). Tracer ensuite la solution obtenue.

3 Calcul approché d'intégrale

3.1 Méthode des rectangles et des trapèzes

1. Rappelons la formule de calcul approché de intégrale d'une fonction f sur un intervalle $[a;b]$ par la méthode des rectangles (à gauche) avec une subdivision en n intervalles :

$$R_n = \frac{b-a}{n}\sum_{k=0}^{n-1} f\left(a+k\frac{b-a}{n}\right) .$$

Écrire une fonction `intRectangle(a,b,f,n)` qui renvoie cette approximation.
Tester avec la fonction $x \mapsto 1/x$ sur l'intervalle $[1,2]$ (on pourra comparer avec la valeur exacte de l'intégrale).

2. La méthode des trapèzes est donnée par :

$$T_n = \sum_{k=0}^{n-1} \frac{b-a}{n}\,\frac{f\left(a+k\frac{b-a}{n}\right)+f\left(a+(k+1)\frac{b-a}{n}\right)}{2} .$$

Écrire une fonction `intTrapeze(a,b,f,n)` qui renvoie cette approximation. On veillera à ne calculer qu'une seule fois la valeur de f en chaque point.

3.2 Utilisation de scipy.integrate

On obtient une valeur approchée de l'intégrale (ainsi qu'une majoration de l'erreur) par la commande

```
import scipy.integrate as integr
print(integr.quad(f,1,2))
```

3.3 Méthode de Monte-Carlo

Cette méthode consiste à calculer une intégrale en utilisant le hasard. Reprenons l'exemple précédent pour l'illustrer : on souhaite calculer l'aire la surface A sous la courbe de $x \mapsto 1/x$ entre les abscisses 1 et l'abscisses 2. Soit X une variable aléatoire suivant une loi uniforme sur $[1,2]\times[0,1]$. La probabilité $P(X \in A)$ est égale à la surface recherchée. D'après la loi des grands nombres, si (X_n) est une suite de v.a.i.i.d. de même loi que X, la fréquence avec laquelle les X_n tombent dans A tend vers $P(X \in A)$. On va donc simuler une suite de telles variables aléatoires et estimer cette fréquence.
Rappelons qu'après import du module random, une simulation d'une variable aléatoire uniforme sur $[0,1]$ s'obtient par la commande `random()`. On peut donc obtenir un couple selon la loi uniforme sur $[1,2]\times[0,1]$ par les commandes suivantes :

```
import random as rd
x , y = 1+rd.random() , rd.random()
```

Écrire un programme estimant la surface de A en utilisant la méthode expliquée ci-dessus.

4 Complément : un exemple d'équations aux dérivées partielles

On considère un problème de diffusion de chaleur dans une barre homogène de section S, de longueur $l = 1m$, de conductivité thermique λ, de capacité thermique massique c, de masse volumique μ. En l'absence de perte de chaleur latérale, la diffusion de chaleur se fait uniquement dans l'axe de la barre $(0x)$. Si on note $T(x,t)$ la température au point d'abscisse x de la barre à un instant t, cette fonction vérifie l'équation aux dérivées partielle (EDP) :

$$\frac{\partial T}{\partial t} = \frac{\lambda}{\mu c}\,\frac{\partial^2 T}{\partial x^2} .$$

La barre est initialement à une température $T1 = 300K$; au temps $t = 0$, on applique à son extremité une température $T2 = 400K$, tout en maintenant son extrémité $x = 0$ à $T1 = 300K$. Pour l'application numérique, on posera $\alpha = \dfrac{\lambda}{\mu c}$ et on prendra $\alpha = 10^{-5} m^2 s^{-1}$.

On va discrétiser l'équation à la fois en temps et en espace. Pour cela, on choisit un pas `dx` égal à 0.01m et un pas `dt` égal à 1s. On fixe également la durée de la simulation à 60s.

On stockera les valeurs approchées de $T(x,t)$ dans un tableau `T` de sorte que `T[k,l]` soit la valeur approchée de $T(k \times dx, l \times dt)$.

1. Commencer le programme en déclarant les constantes utiles.
2. Faire un schéma du tableau `T` et l'initialiser, y compris les conditions à $t = 0$.
3. On va maintenant établir le schéma d'intégration numérique de cette EDP en s'inspirant du schéma d'Euler.
   (a) Connaissant au temps $t_{l-1} = (l-1) \times dt$ les valeurs approchées en chaque $x_k = k \times dx$ `T[k,l-1]`, par quelle valeur peut-on approcher $\dfrac{\partial T}{\partial x}(x_{k+1}, t_{l-1})$ et fonction de `T[k+1,l-1]` et `T[k,l-1]` ? Par quelle valeur peut-on approcher $\dfrac{\partial T}{\partial x}(x_k, t_{l-1})$ et fonction de `T[k,l-1]` et `T[k-1,l-1]` ?
   (b) Par quelle valeur peut-on alors approcher $\dfrac{\partial^2 T}{\partial x^2}(x_k, t_{l-1})$?
   (c) En utilisant l'EDP, quelle valeur approchée de $\dfrac{\partial T}{\partial t}(x_k, t_{l-1})$ cela nous donne-t-il ?
   (d) Comment calculer, en s'inspirant du schéma d'Euler, `T[k,l]` à partir de ce qui précède ? (Bien observer pour quelles valeurs de k cela est valable.)
4. Implémenter le schéma dégagé à la question précédente.
5. Une fois que cela fonctionne, augmenter la durée de la simulation à une heure et tracer le profil de température en fonction de x toutes les 10 minutes.
6. Quelle doit être la situation à l'équilibre ?

5 Annexe : quelques rappels sur NumPy

On fait juste quelques rappels sur la manipulation des tableaux. Essayer les commandes suivantes et observer leur résultat (Après avoir importé NumPy : `import numpy as np`).

- Pour déclarer un tableau : `A=np.array([[1,2],[0,3]])` ; on peut aussi initialiser avec des zéros : `np.zeros([2,3])` ; une matrice colonne : `B=np.array([[0],[1]])` ; un tableau à une dimension : `C=np.array([3,2])` .
- Pour accéder à un coefficient : `A[1,0]` ; on peut aussi l'affecter : `A[1,0]=5` .
- Pour accéder à une ligne : `A[1,:]` , à une colonne `A[:,0]` ; on peut aussi affecter une colonne : `A[:,0]=[6,7]` .
- Pour multiplier deux matrices : `np.dot(A,B)` . Attention : `A*B` ne fait pas du tout la même chose. On peut bien sûr additionner deux tableaux (de même format) et multiplier un tableau par une constante.
- Pour connaître les dimensions d'un tableau : `np.shape(A)` . Tester également avec `B` , `C` , `A[:,0]` .
- Bien remarquer le format de `A[:,0]` et le fait que `numpy` accepte de faire `np.dot(A,C)` .
- Pour obtenir une subdivision d'un intervalle (pour tracer une courbe par exemple) : `np.linspace(0,1,11)` (lorsque l'on connaît les bornes et le nombre de subdivisions) ou `np.arange(0,1.1,0.1)` (lorsque l'on connaît les bornes et le pas).