Université de Montpellier **Année Universitaire 2023-2024**
Faculté des Sciences
Mathématiques

L3: module HAX604X Analyse numérique des équations différentielles.

TP 4

EDO du second ordre et systèmes différentiels avec Matlab (ou Python au choix).

Matlab et Python disposent de plusieurs fonctions pour résoudre les équations différentielles.
Pour Matlab on renvoie à l'aide en ligne, rubrique 'Ordinary Differential Equations/ Examples and how to.' Voici quelques fonctions matlab pour intégrer numériquement les EDO:

```
ode45 Nonstiff differential equations Runge-Kutta order 4 and 5
ode23 Nonstiff differential equations Runge-Kutta order 3 and 4
ode113 Nonstiff differential equations multistep Adams ordre variable jusqu'à ordre 13
ode23s Stiff differential equations  implicit Rosenbrock
ode23t Moderately stiff differential equations implicit Trapezoidal rule
ode23tb Stiff differential equations TR-BDF2
```

Le solveur générique à utiliser par défaut est `ode45` qui est basé sur la méthode de Dormand-Prince. Cet algorithme repose sur deux schémas de Runge-Kutta emboîtés d'ordre 4 et 5 respectivement, ce qui permet d'estimer l'erreur et d'adapter le pas en conséquence.
Pour les solveurs disponibles en Python, on renvoie à la doc `Scipy` du solveur `scipy.integrate.solve_ivp`.
Les méthodes suivantes sont implémentées:

```
'RK45' (default): Explicit Runge-Kutta method of order 5(4),
'RK23': Explicit Runge-Kutta method of order 3(2),
'DOP853': Explicit Runge-Kutta method of order 8,
'Radau': Implicit Runge-Kutta method of the Radau IIA family of order 5,
'BDF': Implicit multi-step variable-order (1 to 5) method based on a backward differentiation formula for the derivat
'LSODA': Adams/BDF method with automatic stiffness detection and switching.
```

Le solveur par défaut est 'RK45'. Si le solveur fait trop d'itérations, s'il diverge, ou échoue, votre problème est probablement raide et il vaut mieux utiliser 'Radau' ou 'BDF'. 'LSODA' est aussi un bon choix universel, mais il est peut-être moins pratique car il est issu d'un vieux code Fortran.

1 L'oscillateur harmonique.

Ecrire l'équation du second ordre $y'' = -y$, $y(0) = 0$, $y'(0) = 1$ sous forme d'un système du premier ordre. Vérifier que la quantité $y(t)^2 + y'(t)^2$ est constante au cours du temps (c'est une *intégrale première*). Ecrire un code `oscillateur` qui résout numériquement l'équation différentielle sous forme d'un système du premier ordre, en utilisant le solveur Matlab `ode45` ou Python `RK45` et qui trace sur une figure la courbe (y, y').

- Pourquoi obtient-on un cercle dans l'espace (y, y') (appelé plan des phases)?
- Pourquoi le cercle est-il épaissi lorsqu'on simule sur une durée beaucoup plus longue, disons 1000 périodes?
- Augmenter la précision de la résolution de l'EDO. utiliser pour cela les options du solveur (commandes Matlab à adapter pour le solveur Python)

  ```
  options = odeset('RelTol', 1e-5);
  [t,y]=ode45(@(t,y) [y(2);-y(1)], [0,2000*pi],[0;1],options);
  ```

  Est-ce que le problème a disparu?

Pour remédier à ces défauts, il existe des schémas *symplectiques* qui respectent les intégrales premières de l'équation et cela *même à une précision basse.* Coder le schéma d'Euler implicite-explicite suivant, avec un pas de temps $h = 0.01$ ou même le pas plus grossier $h = 0.1$

$$\left\{ \begin{array}{rcrcl} p_{n+1} & = & p_n & - & h \quad q_n \\ q_{n+1} & = & q_n & + & h \quad p_{n+1} \end{array} \right. \tag{1}$$

Ici q_n désigne la position y_n et p_n désigne la vitesse y'_n. Comparer le résultat de la simulation sur 1000 périodes avec le schéma `ode45` ou `RK45`.

2 Un système différentiel en épidémiologie.

On considère le système différentiel SIR (sound, infected, recovered) suivant: (Cf exercice 93 des TD de l'UE calcul différentiel :-)

$$\left\{\begin{array}{rcl} S'(t) &=& -\beta\,\frac{S\cdot I}{N} \\ I'(t) &=& \beta\,\frac{S\cdot I}{N} - (\gamma+\delta)\,I \\ R'(t) &=& \gamma\,I \\ D'(t) &=& \delta\,I \end{array}\right.$$

avec les conditions initiales $I(0)=1,\ R(0)=0,\ D(0)=0,\ S(0)=N-R(0)-I(0)-D(0)$.
Ce système très classique en épidémiologie modélise la propagation d'un virus dans une population. Le nombre N correspond à la population initiale. Les paramètres β,γ,δ correspondent respectivement aux taux de contamination, guérison et décès. Pour commencer on prendra $N=1000\,000$, $\beta=0.2$, $\gamma=0.1$, $\delta=0.01$. L'unité de temps est le jour.

- Montrer que $\forall t,\ S(t)+I(t)+R(t)+D(t)\equiv N$.
- Ecrire un code Matlab ou Python qui utilise le solveur standard 'RK45' (`ode45` en Matlab et `solve_ivp` en Python qui résout le système et trace sur une même figure l'évolution des quatre populations $t\mapsto S(t)$, $t\mapsto I(t)$, $t\mapsto R(t)$, $t\mapsto D(t)$ pendant une durée donnée, par exemple 730 jours.
- Vérifier que vos résultats respectent bien $\forall t,\ S(t)+I(t)+R(t)+D(t)\equiv N$.
- Tracer quelques trajectoires en augmentant le taux de décès δ. Que constatez-vous pour $\delta=0.1$? Expliquez.

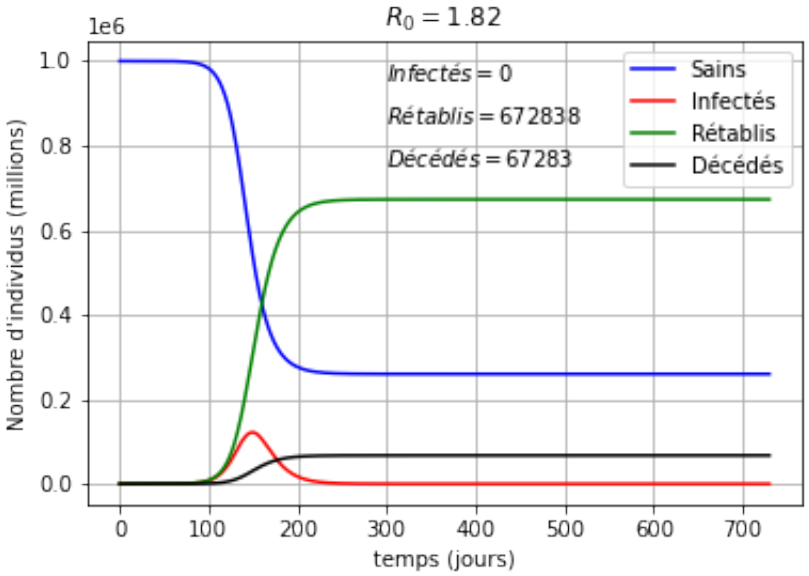

Fig. 1: Simulation d'une vague épidémique.

Pour étudier la virulence initiale du virus on linéarise le système différentiel au voisinage de $t=0, I=0, S=S_0$. On obtient l'équation différentielle linéaire

$$\frac{dI}{dt} = \left(\beta\,\frac{S_0}{N} - (\gamma+\delta)\right)I$$

qu'on écrit plutôt

$$\frac{dI}{dt} = \left(\frac{\beta S_0}{N(\gamma+\delta)} - 1\right)(\gamma+\delta)\,I.$$

Le paramètre sans dimension $R_0 = \frac{\beta S_0}{N(\gamma+\delta)}$ caractérise le régime initial de l'épidémie: si $R_0>1$ (resp. <1) le nombre de personnes infectées croît (resp. décroît) exponentiellement.

- Choisissez un jeu de paramètre de sorte que $R_0=1$. Que constatez-vous?
- *Régime asymptotique.* Que constatez-vous sur le ratio guéris/décédés lorsque $t\to+\infty$?

Pour des compléments, consulter `https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology`