TPN 2 : La méthode d'Euler explicite et la mécanique

Exemple sur le pendule plan

MPSI – Lycée Albert Schweitzer – Y. Vadée Le Brun

I Matériel

- Ordinateur avec un IDE pour coder en python (les bibliothèques d'édupython doivent être installées)
- Accès à internet car google est notre ami

II Objectif du TP

Le but de ce TP est de comprendre comment utiliser la méthode d'Euler pour résoudre des problèmes de mécanique. On base ce TP sur l'exemple du pendule plan. En effet, il s'agit d'un problème complexe : on sait le résoudre et calculer sa période d'oscillation dans l'approximation des petits angles. Mais que se passe-t-il pour des angles plus grands ?

III La méthode d'Euler appliquée à un vecteur et une astuce géniale !

On se place dans le cas général d'un problème de mécanique dans lequel la somme des forces dépend de la position du point M (exemple : force de rappel élastique) et/ou de la vitesse du point M (exemple : forces de frottements fluides). Le Principe Fondamental de la Dynamique peut alors s'écrire sous la forme : $m\vec{a} = \vec{F}(\overrightarrow{OM}, \vec{v})$ où $\vec{F}$ représente la résultante des forces. Il convient de réaliser rapidement que cette équation vectorielle équivaut à un ensemble de 3 équations scalaires :

En coordonnées cartésiennes :	En coordonnées cylindriques dans le cas d'un mouvement circulaire de rayon R :
$\begin{cases} m\ddot{x} \\ m\ddot{y} \\ m\ddot{z} \end{cases} = \begin{cases} F_x(x,y,z,\dot{x},\dot{y},\dot{z}) \\ F_y(x,y,z,\dot{x},\dot{y},\dot{z}) \\ F_z(x,y,z,\dot{x},\dot{y},\dot{z}) \end{cases}$	$\begin{cases} -mR\dot{\theta}^2 \\ mR\ddot{\theta} \end{cases} = \begin{cases} F_r(\theta,\dot{\theta}) \\ F_\theta(\theta,\dot{\theta}) \end{cases}$

Résoudre ce système revient donc à trouver les expressions des coordonnées en fonction du temps. Malheureusement il s'agit à priori de 3 équations différentielles du second ordre. La grande astuce est alors de transformer ce système d'équation différentielle d'ordre 2 en un système deux fois plus grand mais d'ordre 1**.** Pour cela, il suffit de considérer la vitesse comme une fonction indépendante. On obtient alors le système suivant :

En coordonnées cartésiennes :	En coordonnées cylindriques dans le cas d'un mouvement circulaire de rayon R, seul θ nous intéresse et on note $v_\theta = \dot{\theta}$ que l'on considère une nouvelle variable :
$\begin{cases} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \end{cases} = \begin{cases} v_x \\ v_y \\ v_z \\ F_x(x,y,z,v_x,v_y,v_z)/m \\ F_y(x,y,z,v_x,v_y,v_z)/m \\ F_z(x,y,z,v_x,v_y,v_z)/m \end{cases}$	$\begin{cases} \dot{\theta} \\ \dot{v_\theta} \end{cases} = \begin{cases} \mathrm{v_\theta} \\ \mathrm{F_\theta(\theta, v_\theta)/mR} \end{cases}$

Ce système d'équation se réécrit sous forme vectorielle : $\underline{\dot{A}} = \underline{G}(\underline{A})$ où la notation barre en bas désigne un vecteur mais de dimension quelconque.

En coordonnées cartésiennes :	En coordonnées cylindriques dans le cas d'un mouvement circulaire de rayon R :
$\underline{A} = \begin{cases} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{cases} et \quad G = \begin{cases} v_x \\ v_y \\ v_z \\ F_x(x,y,z,v_x,v_y,v_z)/m \\ F_y(x,y,z,v_x,v_y,v_z)/m \\ F_z(x,y,z,v_x,v_y,v_z)/m \end{cases}$	$\underline{A} = \begin{cases} \theta \\ v_\theta \end{cases} et\ \underline{G} = \begin{cases} v_\theta \\ F_\theta(\theta, v_\theta)/mR \end{cases}$

Comme cette équation différentielle est d'ordre 1, on peut lui appliquer la méthode d'Euler déjà vue. La seule différence est qu'il faut l'appliquer simultanément à toutes les coordonnées du vecteur A à chaque pas de calcul.

IV Le TP

IV.1 Déroulement du TP

Le TP se fera en deux temps :

- Partie informatique : rédaction du code et évaluation. Critère de précision et de stabilité.
- Partie physique : Utilisation du code pour étudier l'effet de la non linéarité du pendule plan.
  Comparaison avec la formule approchée de Borda : $T = T_0\left(1 + \frac{\theta_0^2}{16}\right)$

Dans les deux parties, des outils complémentaires à python peuvent être utilisés. Par exemple regressi pour traiter les données.

IV.2 Partie informatique

IV.2.1 Ecriture du code et cahier des charges

Dans ce TP, on ne part pas de zéro : un premier TP sur la méthode d'Euler a déjà été fait avec l'exemple du circuit RC. Toutefois, comme nous ferons d'autres TP numériques de mécanique, nous souhaitons produire un code modulaire afin qu'il soit facilement adaptable à l'étude de différents problèmes. Le code est donc divisé et utilise la programmation orientée objet :

integrateur_complet.py : contient une classe nommée ODESolver, cette classe est une classe abstraite qui implémente tout ce dont a besoin un solveur d'équation différentielle du premier ordre. Je vous l'ai implémenté en entier car l'objet de ce TP ne porte pas sur cet aspect technique.

- __init__ est le constructeur de la classe. Il créer le solveur et lui précise l'équation différentielle qu'il devra résoudre en lui donnant f : la fonction qui donne la dérivée (qui correspond à G pour nos problèmes de mécanique).
- solve est le cœur de la classe : il permet de résoudre l'équation différentielle en ayant la même syntaxe et les mêmes capacités que Odeint, la méthode de référence de Scipy. L'essentiel a comprendre est la présence du self.advance(dt) : on passe d'une étape à la suivante en utilisant cette méthode. C'est là que l'algorithme d'Euler sera implémenté.
- return_error est un simple raccourci : c'est une fonction qui résout l'équation différentielle puis compare sa solution à une solution de référence et renvoie, le temps que ça lui à prit et l'erreur commise. Il est possible de l'appeler avec une liste de pas de temps et de récupérer la liste d'erreurs correspondantes.
- Comme différents algorithmes de résolutions sont possibles, on crée une classe pour chaque algorithme.

A FAIRE :
Allez vérifier que la méthode advance dans ForwardEuler correspond bien à la méthode d'Euler vue au précédent TP. Notez que la syntaxe de Python est bien commode : l'aspect vectoriel est totalement transparent dans l'implémentation !

Nous allons donc désormais nous intéresser aux autres fichiers.

IV.2.1.a Objectif 1 : Le fichier pendule_plan_structure.py

Ce fichier contient une classe pendule. Cette classe contient toute la physique d'un pendule et ses paramètres. C'est là que va se trouver le gros du travail de physicien.

A FAIRE : (Pensez à le renommer pendule_plan.py)
Premièrement : il faut remplir la fonction derA pour qu'elle renvoie les bonnes valeurs dans et hors du cadre de l'approximation des petits angles.

Deuxièmement : il faut remplir la fonction A_math pour qu'elle renvoie la solution exacte dans le cadre de l'approximation des petits angles. Il faut qu'elle renvoie l'angle et la vitesse angulaire à n'importe quel instant t.

Troisièmement : il faut remplir la fonction Em pour qu'elle renvoie l'énergie mécanique du pendule connaissant son état (angle et vitesse angulaire) et ce dans et hors du cadre de l'approximation des petits angles.

IV.2.1.b Objectif 2 : Le fichier test_structure.py

A FAIRE : (Pensez à le renommer test.py)
En vous inspirant des fichiers test_pas_de_temps.py et test_integrateurs.py, que j'ai créé pour mes propres besoins, compléter le fichier test.py. Votre objectif est juste de vérifier que votre code fonctionne. Essayez de calculer une solution avec et sans l'approximation des petits angles. Calculez une erreur en vous comparant soit à la solution mathématique, soit à la solution odeint précise. Calculez l'énergie mécanique. Ensuite tracez ces trois grandeurs en fonction du temps.

Testez différents pas de temps afin d'évaluer la précision et la stabilité de votre intégrateur. Comparez vous à odeint. N'hésitez pas à tester odeint seul pour voir comment il se comporte.

L'étude de l'énergie mécanique est importante pour étudier la stabilité : si celle-ci dérive alors votre intégrateur n'est pas stable.

IV.2.1.c Objectif 3 : création d'un fichier Bordas.py

A FAIRE EN CLASSE :
Par la suite nous allons vouloir vérifier la formule de Bordas. Créez un fichier bordas.py qui compare la résolution sans l'approximation des petits angles avec la solution exacte dans le cas des petits angles.

Il faut ensuite mesurer la période pour différents angles initiaux et vérifier la loi de Bordas. Cela peut être fait directement dans python ou à l'aide d'outils extérieurs.

IV.2.2 Evaluation de l'intégrateur numérique

Comparer pour différents pas de temps l'évolution de l'erreur angulaire pour la méthode d'Euler et pour odeint. Vérifier en particulier que le temps de calcul est inversement proportionnel au pas de calcul et que l'erreur angulaire maximale est proportionnelle au pas de calcul pour la méthode d'Euler. Est-ce le cas pour odeint ? Que peut-on conclure en termes de précision et de temps d'exécution pour ces deux méthodes ? En particulier doit on toujours choisir le pas de temps le plus petit possible ?

Comparer pour différents pas de temps l'évolution de l'énergie mécanique pour la méthode d'Euler et pour odeint. N'hésitez pas à observer finement odeint sans Euler en comparaison, testez-le sur des longues durées (par rapport à la durée d'une période). Que peut-on dire en termes de stabilité et de précision pour ces deux intégrateurs ?

IV.3 Partie physique

En prenant soin de bien choisir l'intégrateur numérique et le pas de temps (cf partie évaluation) testez la formule de Bordas. Prédit-elle correctement l'évolution de la période avec l'angle initial ? Le signal est-il réellement sinusoïdal pour des grandes valeurs de l'angle initial ?