TP n° 10 – Méthode d'Euler

D'après un TP de E. Bougnol, J.J. Fleck, M. Heckmann et M. Kostyra, Lycée Kléber.

Contexte général

En astro-informatique, la méthode d'Euler est souvent utilisée pour intégrer numériquement les équations du mouvement des objets en interaction gravitationnelle. On va étudier ici le cas particulier d'une masse m supposée ponctuelle soumise à l'attraction gravitationnelle d'une masse M bien plus importante placée au centre O du repère et fixe dans le référentiel d'étude. Les équations du mouvement s'écrivent alors de manière vectorielle :

En coordonnées cartésiennes :

$$\begin{cases} \ddot{x} &= -\dfrac{G\,M}{(x^2+y^2)^{\frac{3}{2}}}\,x \\ \ddot{y} &= -\dfrac{G\,M}{(x^2+y^2)^{\frac{3}{2}}}\,y \end{cases}$$

$$E = \frac{1}{2}\,m\,(\dot{x}^2+\dot{y}^2) - \frac{G\,M\,m}{\sqrt{x^2+y^2}}$$

En coordonnées polaires :

$$\begin{cases} \ddot{r} - r\,\dot{\theta}^2 &= -\dfrac{G\,M}{r^2} \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} &= 0 \end{cases}$$

$$E = \frac{1}{2}\,m\,(\dot{r}^2 + (r\,\dot{\theta})^2) - \frac{G\,M\,m}{r}$$

En bons astro-informaticiens, nous prendrons les termes constants G, M et m tous égaux à 1 unité SI, de sorte que les équations soient vraiment simples à écrire.

En coordonnées cartésiennes :

$$\begin{cases} \ddot{x} &= -\dfrac{1}{(x^2+y^2)^{\frac{3}{2}}}\,x \\ \ddot{y} &= -\dfrac{1}{(x^2+y^2)^{\frac{3}{2}}}\,y \end{cases}$$

$$E = \frac{1}{2}\,(\dot{x}^2+\dot{y}^2) - \frac{1}{\sqrt{x^2+y^2}}$$

En coordonnées polaires :

$$\begin{cases} \ddot{r} &= -\dfrac{1}{r^2} + r\,\dot{\theta}^2 \\ \ddot{\theta} &= -\frac{2}{r}\,\dot{r}\dot{\theta} \end{cases}$$

$$E = \frac{1}{2}\,(\dot{r}^2 + (r\,\dot{\theta})^2) - \frac{1}{r}$$

Un des buts du TP est de constater que, pour ce problème particulier et en utilisant la méthode d'Euler classique, les calculs en coordonnées cartésiennes sont en général moins précis que les calculs en coordonnées polaires.

I Intégration par la méthode d'Euler

On va intégrer les deux systèmes d'équations par la méthode d'Euler pour vérifier si le choix du repère influence la précision des calculs. On évaluera aussi l'influence du pas de temps en unités arbitraires (par exemples : `dt=1e-2` ou `1e-3` ou `1e-4`) sur les résultats d'intégration. On limitera le nombre total de points à afficher à 1000 en ne stockant que des échantillons prélevés à intervalle régulier entre `0` et `tfinal`.
Outre l'aspect visuel qualitatif des trajectoires obtenues, un critère quantitatif de la précision de nos calculs sera la conservation de l'énergie calculée. On pourra notamment déterminer la fluctuation maximale $100\,\frac{E_{max}-E_{min}}{|E_{min}|}$ de l'énergie calculée entre l'instant initial et la fin de l'intégration par la méthode d'Euler. Pour cela, on calculera également à chaque pas de sortie l'énergie.

L'algorithme à considérer est le suivant :

- Initialisation des valeurs de position et vitesse. Les valeurs `x0=1, y0=0, vx0=0, vy0=1` conduisant à une trajectoire circulaire (si `G=M=m=1`), il est conseillé de commencer avec ce type de conditions initiales pour vérifier que l'intégration fonctionne correctement. De même, pour débuter tout en limitant le temps de calcul, on prendra `tfinal=100` et `dt=1e-2`.
- On boucle le calcul pour aller de l'instant initial à l'instant final. À chaque étape , on met à jour l'accélération, puis la position, puis la vitesse [1], puis le temps (`x = x + vx*dt`, `y = y + vy*dt`, `vx = vx + ax*dt`, `vy = vy + ay*dt`, `t= t + dt`).
- Si `t` correspond à un des millièmes du temps final, on calcule l'énergie et on stocke les valeurs de positions, d'énergie et de temps dans des listes.
- Une fois l'intégration terminée on affiche la fluctuation en énergie pour chacune des méthodes et on trace les courbes.

Il va donc falloir écrire deux fonctions : `euler_cartesien(x0,vy0,dt,tfinal)` et `euler_polaire(x0,vy0,dt,tfinal)` qui acceptent toutes deux en entrée les valeurs de position et de vitesse initiales en coordonnées cartésiennes, la valeur du pas de temps `dt` et le temps final `tfinal`.
Ces deux fonctions doivent renvoyer quatre listes `x,y,t,E` comprenant respectivement les coordonnées cartésiennes x et y, ainsi que les temps t et énergies E pour chacun des 1000 points que l'on utilisera pour l'affichage final.
Pour l'intégration en coordonnées polaires, on rappelle les formules utiles permettant le passage initial de cartésiennes à polaires dans ces conditions simples ($y_0 = 0$ et $v_{x0} = 0$) [2] :

$$r_0 = x_0,\ \theta_0 = 0,\ \dot{r}_0 = 0 \text{ et } \dot{\theta}_0 = \frac{v_{y0}}{r_0}$$

et, réciproquement, à tout instant pour renvoi des listes demandées en fin de procédure :

$$x = r\cos(\theta) \text{ et } y = r\sin(\theta)$$

Une fois l'algorithme implanté en python, modifier les conditions initiales (x_0 et v_{y0}), dt, t_f pour voir leurs effets sur la précision de calcul et sur la trajectoire calculée.

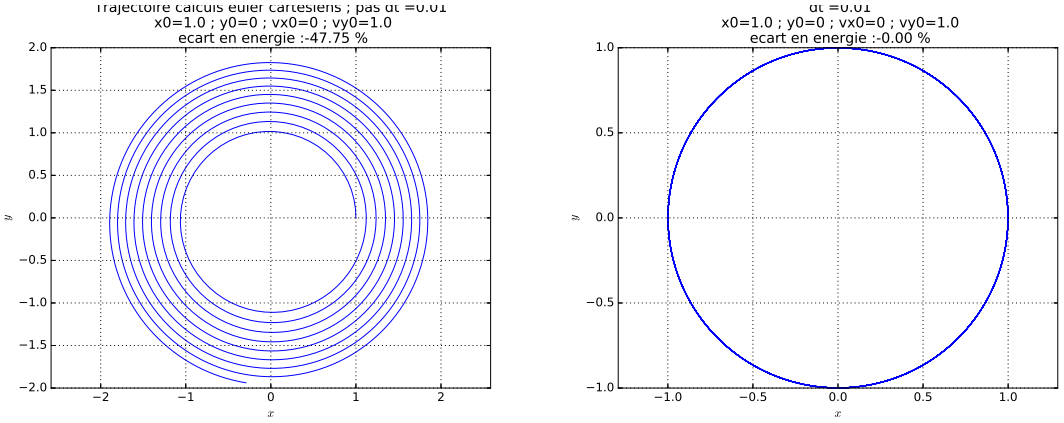

FIGURE 1 – Conditions initiales : $x0 = 1$, $y0 = 0$, $vx0 = 0$ $vy0 = 1$; pas d'intégration $dt = 0,01$. Figure de gauche : calculs en coordonnées cartésiennes. Figure de droite : calculs en coordonnées polaires. La trajectoire est censée être circulaire. On constate qu'en coordonnées polaires l'intégration est plus précise qu'en coordonnées cartésiennes.

1. On prend soin de respecter cet ordre. En effet, $a(t)$ dépend de $x(t)$ et $y(t)$, il faut donc mettre à jour l'accélération avant d'avoir mis à jour la position. De même, $x(t+dt) = x(t) + vx(t)dt$ donc il faut mettre à jour la position avant d'avoir mis à jour la vitesse.
2. Attention, si $y_0 \neq 0$ ou $v_{x0} \neq 0$, les relations de passage sont beaucoup plus compliquées. Voir Annexes.

II Intégration par la méthode Leap-frog

La version « saute-mouton » de l'algorithme d'Euler a plusieurs avantages sur sa version simple : meilleure conservation de l'énergie et invariance par renversement du temps en sont deux facettes. Pour limiter les complications, on se contentera de l'implémenter en coordonnées cartésiennes, mais avec des conditions initiales arbitraires en x et en y pour pouvoir vérifier le caractère invariant par retournement temporel. La méthode d'Euler précédente (à une seule variable) aurait pu s'écrire comme une suite de valeurs telles que :

$$x_{n+1} = x_n + v_n \,\mathrm{d}t \quad \text{et} \quad v_{n+1} = v_n + a_n \,\mathrm{d}t$$

où l'accélération à l'étape n ne dépend que de x_n . En d'autres termes, les nouvelles valeurs des fonctions x et v sont calculées entièrement à partir des anciennes valeurs. Avec la méthode « saute-mouton », vitesse et position ne sont plus évaluées aux mêmes instants, la vitesse étant calculée pour des pas « demi-entiers » de sorte que position et vitesse puissent jouer à « saute-mouton » à tour de rôle l'un par-dessus l'autre :

$$x_{n+1} = x_n + v_{n+\frac{1}{2}} \,\mathrm{d}t \quad \text{et} \quad v_{n+\frac{1}{2}} = v_{n-\frac{1}{2}} + a_n \,\mathrm{d}t$$

L'écriture semble bizarrement équivalente à la précédente mais, sous réserve d'une initialisation correcte, on peut montrer que la méthode a une précision quadratique [3], contrairement à la méthode d'Euler dont la précision est linéaire [4]. La méthode d'initialisation pertinente est la suivante : pour un couple $(x_0,\, v_0)$ de conditions initiales, on initialise bien x à x_0 , mais pour la vitesse, on initialise un demi pas plus tard : $v_{\frac{1}{2}} = v_0 + a_0 \,\frac{\mathrm{d}t}{2}$.
Vous pouvez à présent écrire la fonction `leapfrog(position,vitesse,dt)`, où `position` et `vitesse` sont maintenant des doublets pour la position initiale `(x0,y0)` et la vitesse initiale `(vx0,vy0)`, et la comparer à `euler(position,vitesse,dt)`, notamment au niveau de la conservation de l'énergie et de la réversibilité par renversement du temps (en d'autres termes, si on donne le point d'arrivée en `t = 10` pour les deux procédures et que l'on inverse le signe des vitesses, la particule se retrouve-t-elle à `t = 20` à l'endroit où elle se trouvait à `t = 0` ?).
Une fois l'algorithme implanté en python, modifier les conditions initiales (x_0, y_0, v_{x0} et v_{y0}), dt, t_f pour voir leurs effets sur la précision de calcul et sur la trajectoire calculée.

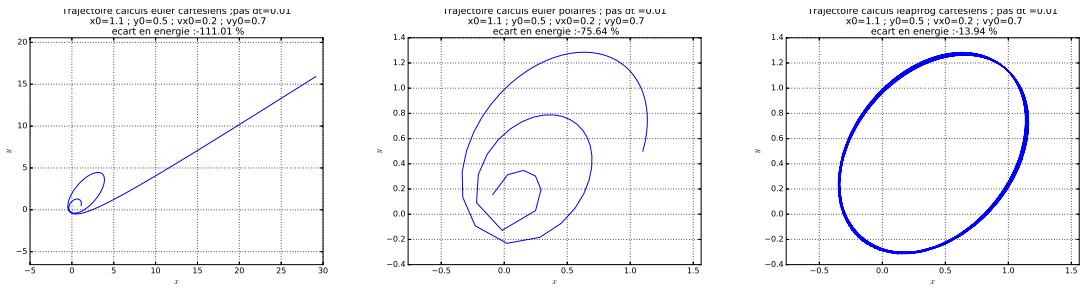

FIGURE 2 – Conditions initiales : $x0 = 1.1$, $y0 = 0.5$, $vx0 = 0.2$, $vy0 = 0.7$; pas d'intégration $dt = 0,01$. Figure de gauche : calculs en coordonnées cartésiennes. Figure du milieu : calculs en coordonnées polaires. Figure de droite : calculs en coordonnées cartésiennes avec la méthode Leap-frog. La trajectoire est censée être elliptique. On constate que, même en coordonnées cartésiennes, la méthode Leap-frog est la plus précise.

3. C'est-à-dire proportionnelle à $(\mathrm{d}t)^2$. Voir Annexes.
4. C'est-à-dire proportionnelle à $\mathrm{d}t$.

Annexes

Relations de passage de cartésiennes à polaires

Dans un cas général, on a :

$$\begin{aligned}\dot{r} &= \frac{\dot{x}x}{\sqrt{x^2+y^2}} + \frac{\dot{y}y}{\sqrt{x^2+y^2}} \\ \dot{\theta} &= \frac{1}{\sqrt{x^2+y^2}}\left(\frac{-\dot{x}y}{\sqrt{x^2+y^2}} + \frac{\dot{y}x}{\sqrt{x^2+y^2}}\right)\end{aligned}$$

Précision de la méthode Leap-frog

Dans la méthode Leap-frog, on itère par pas demi entier :

$$x_{n+\frac{1}{2}} = x_n + \frac{1}{2} v_n \,\mathrm{d}t \tag{1}$$

$$v_{n+1} = v_n + a_{n+\frac{1}{2}} \,\mathrm{d}t \tag{2}$$

$$x_{n+1} = x_{n+\frac{1}{2}} + \frac{1}{2} v_{n+1} \,\mathrm{d}t \tag{3}$$

Si on injecte l'équation (1) dans l'équation (3), on obtient :

$$x_{n+1} = x_n + v_n \,\mathrm{d}t + \frac{1}{2} a_{n+\frac{1}{2}} \,(\mathrm{d}t)^2$$

La précision de l'algorithme semble donc bien quadratique. Néanmoins, la précision théorique de la méthode d'intégration est limitée par l'étape qui a la précision la plus faible. Si on n'y prend pas garde, c'est l'étape d'initialisation qui est le « maillon faible » de l'algorithme. En effet, la toute première demi-étape de mise à jour de la position (de 0 à $\frac{1}{2}$) :

$$x_{\frac{1}{2}} = x_0 + \frac{1}{2} v_0 \,\mathrm{d}t$$

est de précision linéaire et donc malheureusement tout l'algorithme. On remédie à ce problème en utilisant, *pour la toute première demi-étape uniquement*, la mise à jour suivante :

$$x_{\frac{1}{2}} = x_0 + \frac{1}{2} v_{\frac{1}{2}} \,\mathrm{d}t$$

avec

$$v_{\frac{1}{2}} = v_0 + a_0 \,\frac{\mathrm{d}t}{2}$$

ce qui mène à

$$x_{\frac{1}{2}} = x_0 + \frac{1}{2} v_0 \,\mathrm{d}t + \frac{1}{4} a_0 \,(\mathrm{d}t)^2$$

Ce qui prouve la précision quadratique de la première étape. Pour toutes les étapes ultérieures, on utilise les équations (1), (2) et (3) de l'algorithme Leap-frog qui sont équivalentes aux équations $x_{n+1} = x_n + v_{n+\frac{1}{2}} \,\mathrm{d}t$ et $v_{n+\frac{1}{2}} = v_{n-\frac{1}{2}} + a_n \,\mathrm{d}t$ utilisées dans l'énoncé.