TP 1 : Résolution d'équations différentielles et calcul de trajectoires de planètes

7 mars 2014

1 Introduction

Le but de ce TP est de calculer numériquement les trajectoires de 5 planètes du système solaire, Jupiter, Saturne, Uranus, Neptune et Pluton. Il est tiré du polycopié d'*Analyse Numérique*, de Ernst Hairer.

On considère que les planètes ne sont soumises qu'aux forces gravitationnelles exercées par les autres planètes et par le soleil. La force gravitationnelle $\mathbf{F}_{2\to1}$ exercée par un corps de masse m_2 situé en $\vec{x}_2 = (x_2, y_2, z_2)$ sur un corps de masse m_1 situé en $= \vec{x}_1 = (x_1, y_1, z_1)$ est donnée par l'expression suivante :

$$\vec{F}_{2\to1} \;=\; -Gm_2\, m_1\, \frac{(\vec{x}_1 - \vec{x}_2)}{||\vec{x}_1 - \vec{x}_2||^3},$$

avec G la constante de gravitation universelle. La loi de Newton :

$$\sum F = m_i\, a$$

permet alors d'obtenir les équations du mouvement suivantes pour le corps i considéré

$$\vec{x}_i" \;=\; -G \sum_{j=1, j\neq i}^{6} m_j\, \frac{(\vec{x}_i - \vec{x}_j)}{||\vec{x}_i - \vec{x}_j||^3}$$

pour i=1 à 6, où :

- i=1 correspond à Jupiter, de masse $m_1 = 0.000954786104043$,
- i=2, à Saturne, avec $m_2 = 0.000285583733151$,
- i=3, à Uranus avec $m_3 = 0.0000437273164546$,
- i=4, à Neptune avec $m_4 = 0.0000517759138449$,
- i=5, à Pluton avec $m_5 = 1/(1.3.10^8)$,
- i=6, au soleil avec $m_6 = 1.00000597682$.

Les unités de masse sont relatives au soleil, lui-même de masse différente de 1 après correction pour tenir compte de la masse des planètes proches. Les unités de temps sont en jours terrestres. Le mouvement des planètes est étudié dans le référentiel lié au soleil, qui a donc la position initiale $\vec{x}_6 = 0$. On imposera que le soleil reste fixe, c'est à dire que

$$\vec{x}_6" = 0$$

La constante de gravitation universelle est G = $2.95912208286.10^{-4}$. Les positions initiales des 5 planètes, données en unités astronomiques (1 A.U. = 149597870 km), ainsi que leurs vitesses initiales sont résumées dans le tableau suivant :

	Jupiter	Saturne	Uranus	Neptune	Pluton
x	-3.5023653	9.0755314	8.3101420	11.4707666	-15.5387357
y	-3.8169847	-3.0458353	-16.2901086	-25.7294829	-25.2225594
z	-1.5507963	-1.6483708	-7.2521278	-10.816945	-3.1902382
u	0.00565429	0.00168318	0.00354178	0.00288930	0.00276725
v	-0.00412490	0.00483525	0.00055029	0.00039677	-0.00136504
w	-0.00190589	0.00192462	0.00055029	0.00039677	-0.00136504

FIGURE 1 – Positions et vitesses initiales des planètes

Le système d'équations donné par

$$\begin{cases} \vec{x}_i{}'' &= -G\sum_{j=1,j\neq i}^{6} m_j \frac{(\vec{x}_i-\vec{x}_j)}{||\vec{x}_i-\vec{x}_j||^3} \\ \vec{x}_6{}'' &= 0 \end{cases}$$

est un système d'ordre 2 car on a une dérivée seconde. On peut le mettre sous la forme d'un système d'équations différentielles du premier ordre, en introduisant la variable intermédiaire $\vec{v}_i = \vec{x}_i'$ (qui est simplement la vitesse) :

$$\begin{cases} \vec{x}_i{}' &= \vec{v}_i \\ \vec{v}_i{}' &= -G \sum_{j=1,j\neq i}^{6} m_j \frac{(\vec{x}_i-\vec{x}_j)}{||\vec{x}_i-\vec{x}_j||^3} \\ \vec{x}_6{}' &= \vec{v}_6 \\ \vec{v}_6{}' &= 0 \end{cases}$$

On ré-écrit alors ce système sous une forme vectorielle en introduisant un vecteur X qui regroupe toutes les variables nécessaires à la description du mouvement, c'est-à-dire dans notre cas les positions et les vitesses des 6 corps (on a donc 36 composantes) :

$$X' = f(X)$$

avec le vecteur $f(X)$ qui regroupe la partie droite de l'équation et X un vecteur colonne de longueur 36, les 3 premières valeurs correspondant à la position du corps 1, les 3 suivantes à la vitesse du corps 1, les 3 suivantes à la position du corps 2, les 3 suivantes à la vitesse du corps 2, etc. Dans le fichier `planetes.f90`, vous trouverez une ébauche du code résolvant ce système, à modifier et compléter par la suite. Pour compiler ce fichier, on tapera :

```
gfortran -o run planetes.f90
```

Pour lancer l'exécution, on tapera alors :

```
./run
```

On rappelle que pour compiler en mode de déboggage, il faut taper :

```
gfortran -o run planetes.f90 -g -fbounds-check
```

2 Méthode d'Euler

Question 1

Complétez la sous-routine *fonct* qui calcule la valeur de chacune des composantes du vecteur $f(X)$, à partir du vecteur X. Pour s'assurer que votre fonction est implémentée correctement, vérifiez que :

$$\begin{aligned} f(X_0) \approx (&0.00565429,\ -0.00412490,\ -0.00190589,\ 0.00000656,\ 0.00000714,\ 0.00000290,\\ &0.00168318,\ 0.00483525,\ 0.00192462,\ -0.00000293,\ 0.00000098,\ 0.00000053,\\ &0.00354178,\ 0.00137102,\ 0.00055029,\ -0.00000032,\ 0.00000063,\ 0.00000028,\\ &0.00288930,\ 0.00114527,\ 0.00039677,\ -0.00000012,\ 0.00000028,\ 0.00000012,\\ &0.00276725,\ -0.00170702,\ -0.00136504,\ 0.00000017,\ 0.00000028,\ 0.00000004,\\ &0.00000000,\ 0.00000000,\ 0.00000000,\ 0.00000000,\ 0.00000000,\ 0.00000000) \end{aligned}$$

où X_0 est le vecteur des positions et vitesses des planètes à l'instant initial, récapitulés dans le tableau en introduction. Notez que ce vecteur est déjà initialisé dans le fichier `planetes.f90`.

Question 2

On va maintenant compléter la sous-routine *euler*, qui pour X quelconque à un instant t, renvoie l'itéré X_next à l'instant $t + \Delta t$ par la méthode d'Euler explicite. Cette sous-routine utilisera elle-même la sous-routine *fonct*. On rappelle, que dans le cas général d'un système :

$$\begin{cases} X'(t) &= f(t, X(t)) \\ X(a) &= X_a \end{cases},$$

la méthode d'Euler explicite estime $X_{n+1} \approx X(t_{n+1})$ par la formule :

$$X_{n+1} \ = \ X_n \ + \ \Delta t \, f(t_n, X_n)$$

Question 3

Complétez ensuite le programme principal, qui va utiliser la subroutine *euler*. Son objectif est de calculer (et d'enregistrer dans des fichiers textes) les positions du premier corps (Jupiter), de t=0 à $t_{max} = 100000$. On peut dans un premier temps choisir $\Delta t = 100$. Au début du programme, le vecteur X est initialisé avec les données de l'introduction, et à chaque pas de temps, on écrase les anciennes inconnues par les nouvelles et on les écrit dans un fichier.
Rappel :

```
! Pour ouvrir le fichier
open(unit=11,file='P1.txt',form='formatted',status='unknown',action='write')
! Pour ecrire dedans
write(11,'(3f15.6)') X(1), X(2), X(3)
! Puis pour le fermer
close(11)
```

Vous pouvez ensuite visualiser la trajectoire obtenue à l'aide de gnuplot, en tapant :

```
gnuplot
splot 'P1.txt' w l
```

La trajectoire est-elle périodique ? Diminuez le pas de temps et comparez.

Question 4

Visualisez les trajectoires des 5 planètes (pas seulement Jupiter) sur Gnuplot. Pour visualiser le contenu de plusieurs fichiers sur la même figure sous gnuplot, on utilise des commandes de la forme :

```
splot 'P1.txt' w l, 'P2.txt' w l, 'P3.txt' w l
```

3 Résolution par la méthode de Runge-Kutta

La méthode de Euler donne, à chaque pas de temps, une erreur de terme dominant $C_1\,\Delta t^2$. Au bout de N pas, on a donc une erreur totale de $C_1\,N\,\Delta t^2 = C_1 t_{max}\Delta t$, où t_{max} est le temps maximal atteint dans la simulation.

La méthode de Runge-Kutta d'ordre 4 est bien plus précise puisqu'elle donne une erreur totale de terme principal $C_4\ t_max\ \Delta\ t^4$. Elle s'écrit sous la forme :

$$\begin{cases} k_1 &= f(t_n, X_n) \\ k_2 &= f(t_n + \dfrac{\Delta t}{2}, X_n + \dfrac{\Delta t}{2} k_1) \\ k_3 &= f(t_n + \dfrac{\Delta t}{2}, X_n + \dfrac{\Delta t}{2} k_2) \\ k_4 &= f(t_n + \Delta t, X_n + \Delta t\ k_3) \\ X_{n+1} &= X_n + \dfrac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases}$$

Question 5

Créez une sous-routine RK4, qui pour X quelconque à un instant t, renvoie l'itéré X_next à l'instant $t + \Delta t$ par la méthode RK4. Dans le programme principal, remplacez l'appel de la sous-routine euler par celle de la sous-routine RK4.

Comparez les deux méthodes pour $\Delta t = 100$, et à $t_{max} = 100000$, temps nécessaire pour que les 5 planètes aient effectué une révolution autour du soleil, et vérifiez que RK4 donne de meilleurs résultats.

Question 6

On cherche souvent à évaluer l'erreur numérique commise par une méthode sous la forme :

$$||X_{\text{numerique}} - X_{\text{exact}}|| \;=\; C\,\Delta t^p \;+\; o(\Delta t^p).$$

On appelle p l'ordre de convergence de la méthode. La solution exacte étant inconnue, l'évaluation numérique de l'ordre de convergence peut se faire en remarquant que l'erreur relative

$$E(\Delta t) = ||X_{\text{num}}(\Delta t) - X_{\text{num}}(\frac{\Delta t}{2})|| = C'\Delta t^p + o(\Delta t^p)$$

est aussi d'ordre p. Notez que sous forme logarithmique, l'erreur est linéaire :

$$\log(E) \approx p \times \log(\Delta t) + C''.$$

Déterminez numériquement l'ordre de convergence obtenu par la méthode de Runge-Kutta. Pour cela, fixez $t_{max} = 100$ et calculez $E(100)$, $E(50)$ et $E(25)$. Recopiez ces valeurs à la main dans un fichier `convergence.dat` de la façon suivante :

```
100 E(100)
50  E(50)
25  E(25)
```

Vous pouvez visualiser la courbe de convergence via gnuplot :

```
set logscale
plot 'convegence.dat'
```

Pour évaluer l'ordre de convergence, gnuplot permet de réaliser une régression linéaire avec ces trois points :

```
E(x) = p*x + C
fit E(x) 'converge.dat' u (log($1)):(log($2)) via p, C
replot exp(C)*x**p
```

N.B. : Pour le calcul de l'erreur, on pourra utiliser la norme infinie ou/et la norme 2.