Analyse Numérique
Equations Différentielles Ordinaires
Mouvement keplérien et double pendule

Axel Bruyere, Dorian Fabregue
4ETI-IMI

4 avril 2021

Objectifs – • Compréhension des EDO d'ordre 1 (Problème de Cauchy)
• Implémentation des méthodes d'Euler et de Runge-Kutta (ordre 2 et 4)
• Application de ces méthodes à l'étude du mouvement képlerien et du double pendule
• Compréhension des défauts de ces méthodes pour chaque étude
• Implémentation de nouveaux algorithmes afin d'améliorer les résultats de l'analyse numérique

1 Contexte

Les équations différentielles ordinaires (EDO) interviennent dans de nombreux domaines de la modélisation mathématique. Nous allons nous intéresser ici à deux modèles mécaniques particuliers, à savoir :

- Etude du mouvement képlerien (trajectoire d'un satellite orbitant autour d'un astre)
- Etude du mouvement d'un double pendule non-amorti

Ces cas-là sont malheureusement trop complexes pour être résolus par des méthodes analytiques car ces situations sont modélisées par des systèmes d'équations différentielles non linéaires et ne présentent pas de solutions exactes.

Pour résoudre ces systèmes d'équations, nous allons mettre en place différents modèles de résolution, à savoir la méthode d'Euler et la méthode de Runge-Kutta (ordres 2 et 4). Nous analyserons les caractéristiques de chacun de ces modèles et mettrons en place, si possible, un nouveau modèle de résolution plus performant.

Table des matières

2 Etude du mouvement képlerien

2.1 Introduction

En astronomie, plus précisément en mécanique céleste, le mouvement képlérien correspond à une description du mouvement d'un astre par rapport à un autre respectant les trois lois de Kepler (1571-1630). La première de ces trois lois, la "loi des orbites", énonce que les objets célestes gravitant autour d'une étoile décrivent des trajectoires elliptiques. C'est ce type de mouvement que nous allons essayer de modéliser dans cette partie.

Dans notre cas, nous supposons l'orbite elliptique avec une forte excentricité. Cela signifie que la distance entre la planète et l'étoile varie fortement entre l'aphélie (point le plus éloigné du l'étoile) et le périhélie (point le plus proche). Dans ce cas, le mouvement de la planète $P = (x(t), y(t))$ est modélisé par le système d'équations différentielles suivant :

$$\begin{cases} x''(t) + \mathscr{G}M\dfrac{x(t)}{[x(t)^2 + y(t)^2]^{3/2}} = 0 \\ y''(t) + \mathscr{G}M\dfrac{y(t)}{[y(t)^2 + y(t)^2]^{3/2}} = 0 \end{cases} \tag{1}$$

où $\mathscr{G}$. est la constante de gravitation universelle, elle apparait dans la troisième loi de Kepler :

$$\frac{a^3}{T^2} = \frac{\mathscr{G}M}{4\pi^2} \tag{2}$$

où a est le demi-grand axe de l'orbite elliptique et T est la période de révolution de la planète autour de l'étoile.

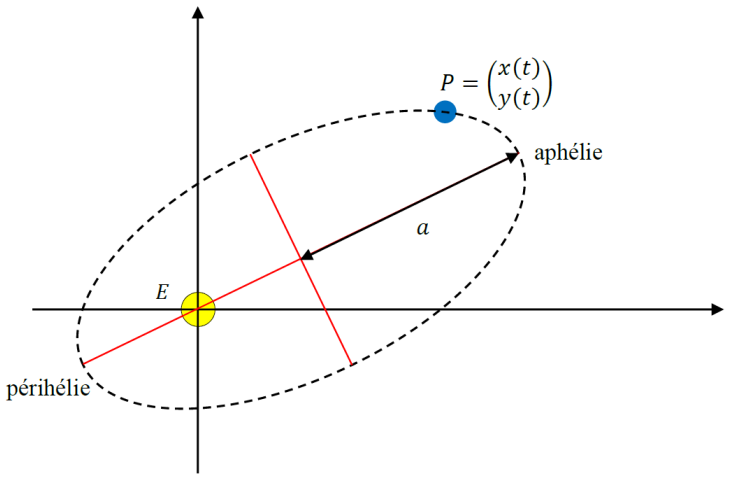

FIGURE 1: Orbite elliptique (avec forte excentricité) d'une planète P autour d'une étoile E

Afin de résoudre ce système d'équations différentielles, nous allons mettre en place un programme utlisant différentes méthodes d'analyse numériques et essayer d'obtenir un résultat satisfaisant, à savoir une ellipse constante comme sur la figure ci-dessus.

2.2 Méthode d'Euler

2.2.1 Problème de Cauchy

Peu importe l'ordre de notre système d'EDO, il peut être ramené à un système d'EDO d'ordre 1. De façon générale, les EDO admettent une infinité de solutions. Ce n'est que lorsqu'on impose une condition initiale $z(t_0) = z_0$ (comme c'est le cas ici avec $(x(t_0), y(t_0)) = (0.5, 0)$) que l'on détermine l'une des solutions de l'EDO. On considèrera alors la solution du problème appelé « problème de Cauchy » qui consiste à déterminer la fonction $z = (x, y) : I \subset \mathbb{R}^2 \to \mathbb{R}^2$ telle que :

$$\begin{cases} z'(t) = u(t, z(t)), \quad \forall t \in I \\ z(t_0) = z_0 \end{cases} \tag{3}$$

Afin de vérifier l'unicité de la solution, on admettra que $u(t, z)$ vérifie deux conditions :
- elle est continue par rapport à la variable t et continue par rapport à la variable $z = (x, y)$
- elle est k-lipschitzienne par rapport à la variable $z = (x, y)$

Les 3 méthodes analytiques que nous allons implémenter pour l'étude du mouvement keplérien seront : la méthode d'Euler, la méthode de Runge-Kutta 4 et la méthode d'Euler-Richardson. Celle-ci devront être adaptées à notre système d'EDO d'orde 2 à deux dimensions.

2.2.2 Méthode d'Euler explicite

On calcule la solution approchée (x_{n+1}, y_{n+1}) au point t_{n+1} à partir de la solution approchée y_n au point t_n. La discrétisation du problème de Cauchy à deux dimensions s'écrit :

$$\begin{cases} f(t_n, x(t_n), y(t_n)) = \dfrac{x(t_{n+1}) - x(t_n)}{t_n + 1 - t_n} (\approx x'(t_n)) \\ g(t_n, x(t_n), y(t_n)) = \dfrac{y(t_{n+1}) - y(t_n)}{t_n + 1 - t_n} (\approx y'(t_n)) \end{cases} \tag{4}$$

ou encore, en approchant $(x(t_n), y(t_n))$ par (x_n, y_n), et en introduisant les notation $f_n = f(t_n, x(t_n), y(t_n))$ et $g_n = g(t_n, x(t_n), y(t_n))$, nous obtenons le schéma explicite de la méthode d'Euler :

$$\begin{cases} x_{n+1} = x_n + h f_n, \quad n = 0, 1, ...N_h - 1 \\ y_{n+1} = y_n + h g_n \end{cases} \tag{5}$$

avec h le résultat de la discrétisation de l'intervalle $I = [a; b]$ divisé en N_h intervalles d'amplitude h suffisemment faible (ici $h = 0.01ua$) par rapport au mouvement étudié.

Sachant que nous avons ici une EDO du second ordre, on introduit de nouvelles variables $(v_x(t_n), v_y(t_n))$ telles que $\forall n \in [0, 1, ..., N_h - 1]$:

$$\begin{cases} v_x(t_n) = x'(t_n) \\ v_y(t_n) = y'(t_n) \\ v_x(0) = x'(0) = 0 \\ v_y(0) = y'(0) = 11.5 \end{cases} \tag{6}$$

Cela nous permet, en résolvant deux EDO d'ordre 2 à chaque itération, d'écrire le programme suivant :

```
function [x,y,t]=fct_Euler_Kepler(x0,y0,vx0,vy0,tmin,tmax,h,f,g)

%%Initialisation
t = tmin:h:tmax;
x = zeros(1,length(t)); x(1) = x0;
y = zeros(1,length(t)); y(1) = y0;
vx = zeros(1,length(t)); vx(1) = vx0;
vy = zeros(1,length(t)); vy(1) = vy0;

%%Calcul des valeurs n+1
for k = 1:length(t)-1
    x(k+1) = x(k) + h*vx(k);
    vx(k+1) = vx(k) + h*f(t(k),x(k),y(k));
    y(k+1) = y(k) + h*vy(k);
    vy(k+1) = vy(k) + h*g(t(k),x(k),y(k));
end
```

2.2.3 Méthode de Runge-Kutta 4 (en 2D)

Les méthodes de RK sont des méthodes numériques à un pas comme la méthode d'Euler dont elles sont une généralisation en considérant cette fois plusieurs évaluations des fonctions (f, g) par intervalle $[t_n; t_{n+1}]$. Nous avons pour RK4 les équations suivantes :

$$\begin{cases} x_{n+1} = x_n + \dfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \quad n = 0, 1, ...N_h - 1 \\ k_1 = f(t_n, x_n) \\ k_2 = f(t_n + \dfrac{h}{2}, x_n + \dfrac{h}{2}k_1) \\ k_3 = f(t_n + \dfrac{h}{2}, x_n + \dfrac{h}{2}k_2) \\ k_4 == f(t_n, x_n + hk_3) \end{cases} \tag{7}$$

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \quad n = 0, 1, ...N_h - 1 \\ k_1 = g(t_n, y_n) \\ k_2 = g(t_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}k_1) \\ k_3 = g(t_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}k_2) \\ k_4 = g(t_n, y_n + hk_3) \end{cases} \tag{8}$$

Sachant que nous avons ici une EDO du second ordre, comme pour la méthode d'Euler, on introduit de nouvelles variables $(v_x(t_n), v_y(t_n))$.Cela nous permet, en résolvant deux EDO d'ordre 2 à chaque itération, d'écrire le programme suivant :

```
function [x,y,t]=fct_RK4_2D_Kepler(x0,y0,vx0,vy0,tmin,tmax,h,f,g)

%%Initialisation
t = tmin:h:tmax;
x = zeros(1,length(t)); x(1) = x0;
y = zeros(1,length(t)); y(1) = y0;
vx = zeros(1,length(t)); vx(1) = vx0;
vy = zeros(1,length(t)); vy(1) = vy0;
```

```
%%Algo keplerien
for k = 1:length(t)-1
    %%k1
    kf1 = f(t(k),x(k),y(k));
    kg1 = g(t(k),x(k),y(k));
    %%k2
    kf2 = f(t(k) + h/2, x(k) + (h/2)*kf1, y(k) + h/2*kg1);
    kg2 = g(t(k) + h/2, x(k) + (h/2)*kf1, y(k) + h/2*kg1);
    %%k3
    kf3 = f(t(k) + h/2, x(k) + (h/2)*kf2, y(k) + h/2*kg2);
    kg3 = g(t(k) + h/2, x(k) + (h/2)*kf2, y(k) + h/2*kg2);
    %%k4
    kf4 = f(t(k) + h, x(k) + h*kf3, y(k) + h*kg3);
    kg4 = g(t(k) + h, x(k) + h*kf3, y(k) + h*kg3);
    %%Calcul des nouvelles valeures n+1
    x(k+1) = x(k) + h*vx(k);
    vx(k+1) = vx(k) + (h/6) * (kf1 + 2*kf2 + 2*kf3 + kf4);
    y(k+1) = y(k) + h*vy(k);
    vy(k+1) = vy(k) + (h/6) * (kg1 + 2*kg2 + 2*kg3 + kg4);

end
```

L'intérêt principal de cette méthode est d'avoir un pas plus flexible que le pas constant de la méthode d'Euler, on se rendra vite compte que cela ne sera pas suffisant pour obtenir une ellipse non-divergente.

2.2.4 Méthode d'Euler-Richardson

Nous prenons ici comme base la méthode d'Euler vue précédemment. Nous y introduisons cependant un pas temporel h adaptatif afin de garantir la qualité de l'analyse numérique. Résumons le principe de l'algorithme (pour plus de détails, consulter *femto-physique.com*) :
Soit la méthode d'Euler avec un pas $h = t_{n+1} - t_n$, envisageons maintenant une évolution en deux étapes de pas $h/2$, elles nous permettent d'obtenir le schéma numérique suivant :

$$\begin{cases} k_n = f'(t_n, Y_n) \\ k'_n = f(t_n + h/2) \\ Y_0 = Y(0) \\ Y_{n+1} = Y_n + hk'_n \end{cases} \tag{9}$$

et d'avoir accès à l'erreur $\varepsilon = \dfrac{h}{2}|k'_n - k_n|$, qui nous servira d'indicateur pour savoir si l'on peut augmenter le pas ou s'il faut le diminuer. Il suffit alors de fixer un seuil de précision ε_{seuil} selon lequel nous augmenterons ou diminuerons la valeur du pas h.

L'algorithme d'Euler-Richardson se déroule de la manière suivante :
1 - Initialisation du pas h, de la durée T et du seuil de précision ε_{seuil},
2 - Initialisation des conditions initiales : $t = 0$ et $y = (0)$
3 - Tant que $t \leq T$, faire :

a. Calcul de $k = f(t, Y)$
b. Calcul de $k' = f(t + h/2, Y + kh/2)$
c. Calcul de $\varepsilon = \frac{h}{2}|k - k'|$ et $\alpha = \frac{\varepsilon}{\varepsilon_{seuil}}$
d. Si $\alpha > 1$, faire $h = 0,9h/\sqrt{\alpha}$
d. Si $\alpha < 1$, faire $h = 0,9h/\sqrt{\alpha}; Y = Y + k'h; t = t + h;$

Dans le cas de notre étude de mouvement keplérien, nous avons :

```
function [X_er,Y_er,t] = fct_Euler_Richardson(x0,y0,vx0,vy0,tmin,tmax,h
    ,f,g,seuil)

%%Initialisation
t = tmin;
Y = [x0;y0;vx0;vy0];

%%Fonction f(y)=[x',y',x'',y'']
fY = @(Yi) [Yi(3);Yi(4);f(1,Yi(1),Yi(2));g(1,Yi(1),Yi(2))];

%%Debut boucle
i=1;
while t(i)<tmax      %Arret quand duree max atteinte
    K = fY(Y(:,i));      %Calcul de K
    Kp = fY(Y(:,i)+0.5*h*K);    %Calcul de K'
    erreur = norm(K-Kp)*h/2;
    alpha = erreur/seuil;

    if alpha>1 %on reitere les calculs precedents avec
        h = 0.9*h/sqrt(alpha); %la nouvelle valeur de h
    else
        Y = [Y, Y(:,i)+h*Kp];
        h = 0.9*h/sqrt(alpha);
        t = [t, t(i)+h];
        i=i+1;
    end
end
X_er = Y(1,:);
Y_er = Y(2,:);
```

2.3 Résultats

2.3.1 Euler

Nous appelons dans notre *main* (cf. annexe) la fonction *fct_Euler_Kepler.m* avec les données de l'énoncé et obtenons la figure suivante :

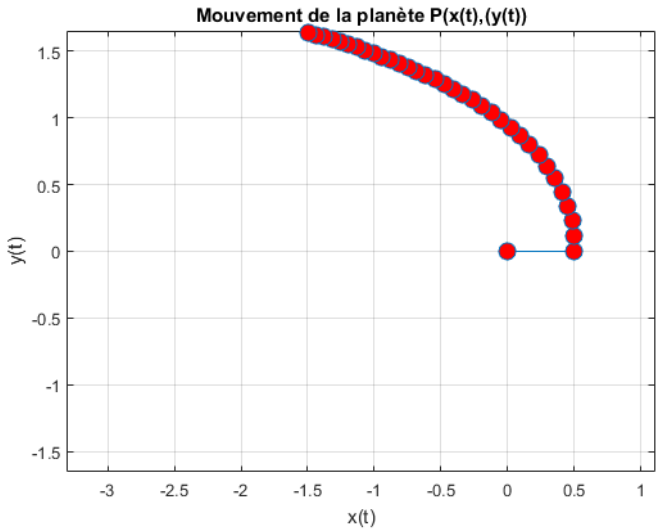

FIGURE 2: Mouvement de l'astre P avec la méthode d'Euler ($h = 0.01$; durée=$2ua$)

On observe que l'astre dévit de son orbite rapidement et ne le rejoindra plus jamais. On pourrait penser corriger cela en réduisant la vitesse initiale $y'(0)$, en se disant qu'il est normal qu'un astre soit ejecté de son orbite s'il a trop de vitesse. On tente le coup avec $y'(0) = 3$ soit une vitesse 4 fois plus faible et obtenons la figure 3 :

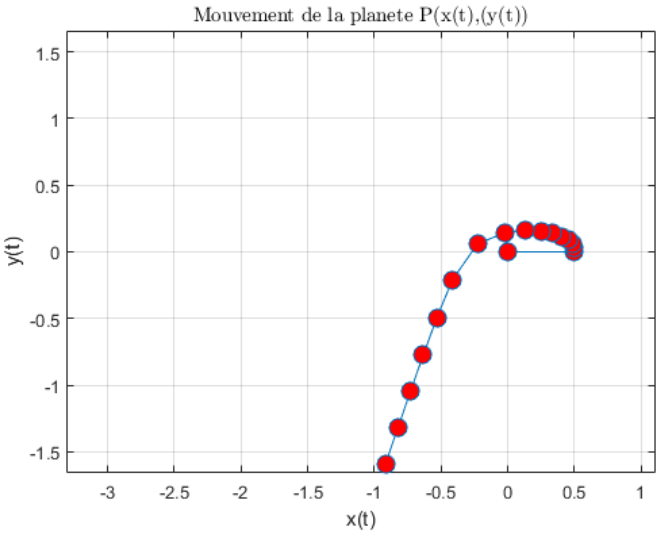

FIGURE 3: Mouvement de l'astre P avec la méthode d'Euler ($y'(0) = 3$) et ($h = 0.01$; durée=$2ua$)

On voit que cela ne fonctionne pas peu importe la vitesse initialle choisie, on en déduit que ce n'est pas un problème de données erronées. On pourrait modifier la position initiale (sans succès évidemment), mais si l'on commence à trop changer les données de l'enoncé, nous perdons de vue l'objectif de départ.

Une donnée sur laquelle on pourrait influer serait celle du pas temporel h, en se disant que notre analyse est incorrecte car trop peu précise. Nous prenons donc $h = 0.001ua$ (soit une valeur 10 fois plus petite) et obtenons la figure 4 :

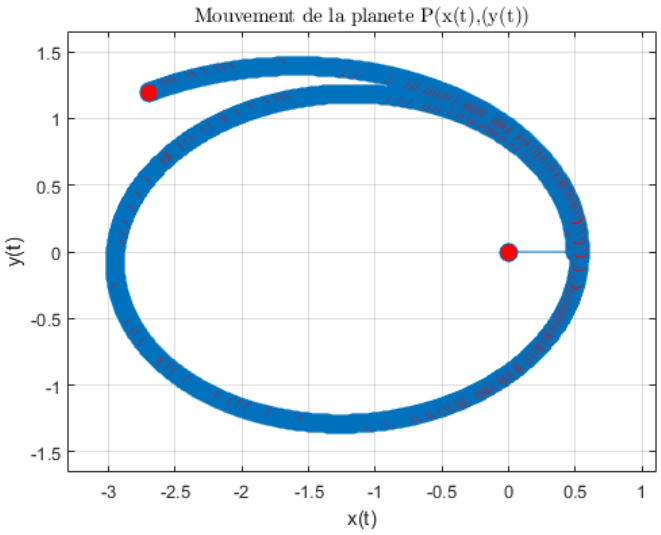

FIGURE 4: Mouvement de l'astre P avec la méthode d'Euler ($h = 0.001ua$ et durée=$3ua$)

En affichant le mouvement de la planète P sur une durée de $3ua$, nous parvenons à obtenir un tracé plus elliptique mais qui finit par diverger au bout d'un moment.
Après toutes les modifications que nous avons effectuées, nous ne sommes pas parvenus à obtenir le tracé elliptique souhaité. Nous en déduisons que la méthode d'Euler, telle quelle, est trop imprécise.

2.3.2 Runge-Kutta 4

Nous appelons dans notre *main* (cf. annexe) la fonction *fct_RK4_2D_kepler.m* avec les données de l'énoncé et obtenons la figure 5 suivante :

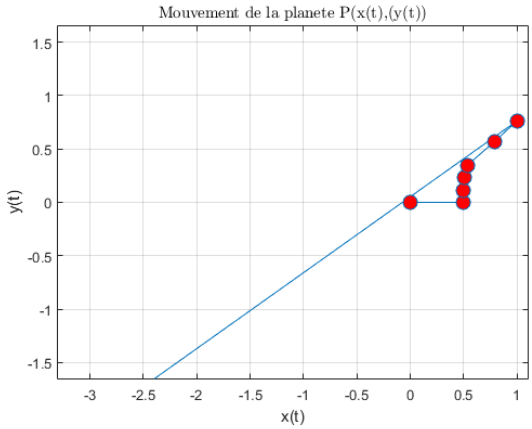

FIGURE 5: Mouvement de l'astre P avec la méthode de Runge-Kutta 4 ($h = 0.01$; durée=$2ua$)

On remarque de suite l'instabilité de l'analyse proposée par cette méthode. On suppose que cette instabilité est causée par des variations trop grandes de valeurs, et proposons donc de prendre un pas $h = 0.001$ pour tenter de contourner ce problème. Nous obtenons la figure 6 suivante :

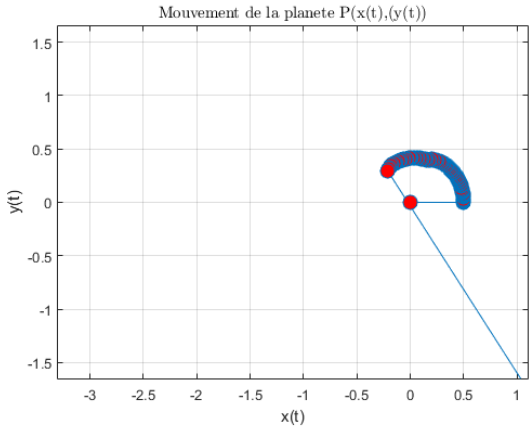

FIGURE 6: Mouvement de l'astre P avec la méthode de Runge-Kutta 4 ($h = 0.001$; durée=$2ua$)

On remarque que l'analyse reste stable plus longtemps, cependant elle diverge au bout d'un certain temps. Nous en déduisons qu'il ne s'agit pas d'une bonne méthode d'analyse pour ce problème. Cependant on remarque, comme avec la méthode d'Euler, qu'influer sur le pas temporel h permet d'améliorer la qualité de l'analyse. C'est pour cela que nous allons analyser ce problème avec la méthode d'Euler-Richardson dans la prochaine partie.

2.3.3 Euler-Richardson

Nous appelons dans notre *main* (cf. annexe) la fonction *fct_Euler_Richardson.m* avec les données de l'énoncé et obtenons la figure 7 suivante :

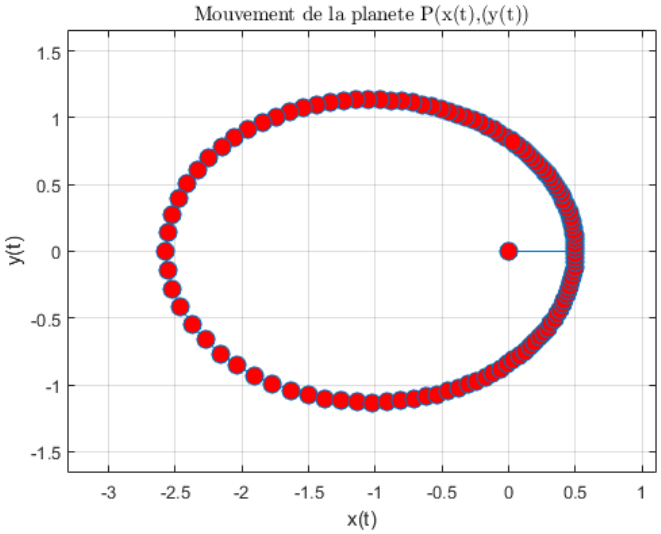

FIGURE 7: Mouvement de l'astre P avec Euler-Richardson (durée=$2ua$; $\varepsilon_{seuil} = 0.01$)

On remarque de suite la précision de l'analyse et obtenons bien une ellipse comme attendu. On peut par ailleurs approximer la valeur de la période du mouvement elliptique $T_{priode} \approx 2ua$. Par précaution, nous vérifions ce résultat pour une durée=$20ua$ et obtenons la figure 8 suivante :

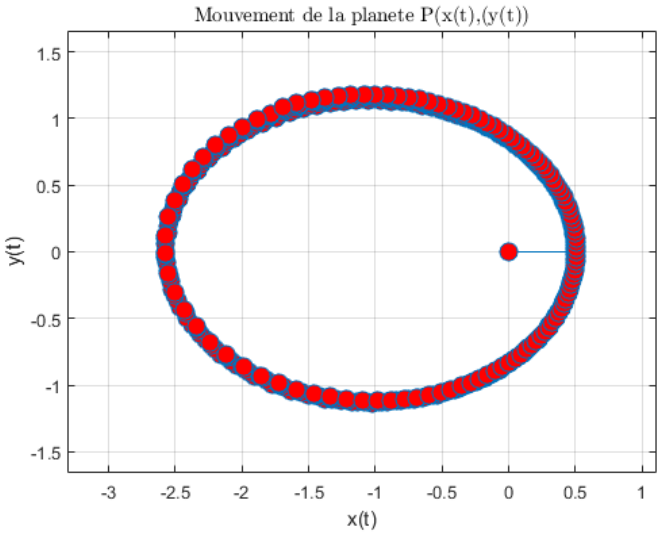

FIGURE 8: Mouvement de l'astre P avec Euler-Richardson (durée=$20ua$; $\varepsilon_{seuil} = 0.01$)

Même après plus de 10 périodes T_{priode}, nous avons une analyse stable avec une valeur de seuil $\varepsilon_{seuil} = 0.01$.

Un autre point à noter est le nombre de points nécessaires au tracé de l'analyse. Avec la méthode d'Euler, pour avoir un tracé un semblant correct sur $2ua$ nous avions besoin de plus de 10000

points, alors qu'avec cette méthode de pas adaptif (et avec $\varepsilon_{seuil} = 0.01$), nous n'avons besoin que de 140 points. On réduit considérablement le temps de d'exécution du programme.

Nous allons maintenant essayer d'afficher le résultat de la figure 7 ($h = 0.01$, durée=$2ua$) avec des seuils d'erreur ε_{seuil} différentes :

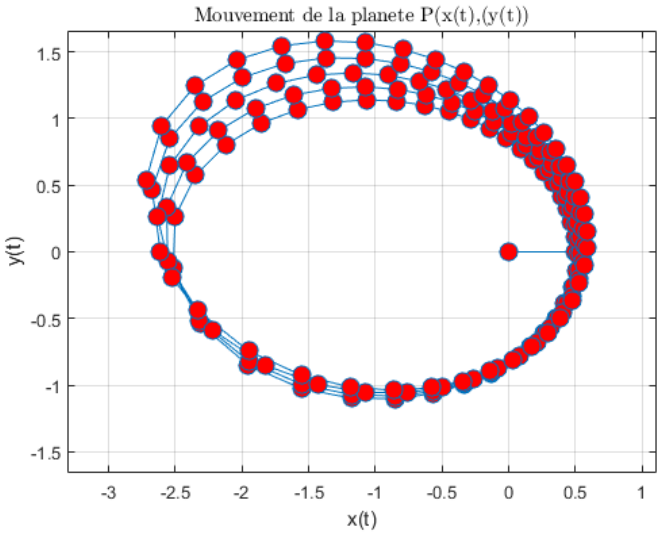

FIGURE 9: Mouvement de l'astre P avec Euler-Richardson (durée=$20ua$; $\varepsilon_{seuil} = 0.1$)

On remarque de suite la divergence qui se crée si on prend $\varepsilon_{seuil} = 0.1$. Ce qui est un résultat attendu.

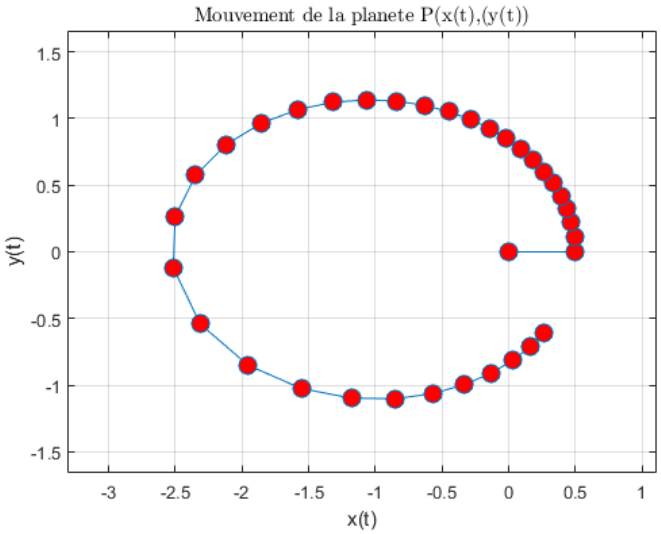

FIGURE 10: Mouvement de l'astre P avec Euler-Richardson (durée=$2ua$; $\varepsilon_{seuil} = 0.1$)

On remarque aussi qu'avec un seuil d'erreur plus important $\varepsilon_{seuil} = 0.1$, l'astre P semble parcourir moins de distance (même pas une ellipse complète) qu'avec $\varepsilon_{seuil} = 0.01$ (cf. figure 7). Il faut garder en tête que dans cette analyse, le temps est discrétisé et est incrémenté à chaque boucle $t = [t, t(i) + h]$; avec un pas adaptatif $h = 0.9h \times \sqrt{\varepsilon_{seuil}/\varepsilon_2}$ qui augmente plus ε_{seuil} est grand.

Essayons maintenant avec $\varepsilon_{seuil} = 0.001$ sur une durée de $2ua$, nous avons la figure 11 suivante :

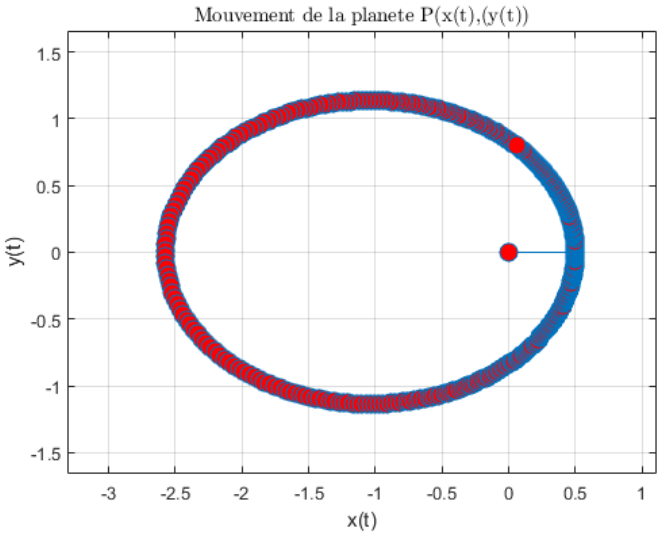

FIGURE 11: Mouvement de l'astre P avec Euler-Richardson (durée=$2ua$; $\varepsilon_{seuil} = 0.001$)

On retrouve un résultat semblable à la figure 7 ($\varepsilon_{seuil} = 0.01$) avec un nombre de points $n = 438$. On aurait pu s'attendre à un nombre de points plus important si l'on pensait qu'avoir une précision ε_{seuil} 10 fois plus fine impliquerait la création de 10 fois plus de points. On va donc visualiser l'évolution du nombre n de points par rapport à la valeur de ε_{seuil} sur une durée de $2ua$:

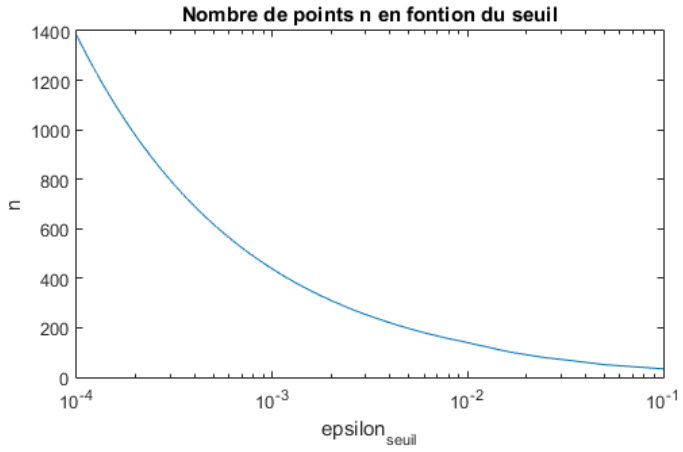

FIGURE 12: Evolution du nombre de points n en fonction de ε_{seuil} (durée=$2ua$)

On remarque que la méthode d'Euler-Richardson nous permet d'accéder à des précisions très élevées ($\varepsilon_{seuil} = 10^{-4}$) sans pour autant augmenter drastiquement le nombre de points n et donc de calculs à effectuer. On notera cependant que la courbe semble suivre une loi qui ressemble à une loi inverse, et que si l'on souhaite des précisions encore plus fines (10^{-8} par exemple) on aurait alors un nombre de points n qui augmenterait drastiquement et on perdrait l'intérêt de cette méthode.

En conclusion, avec des paramètres adaptés, la méthode d'Euler-Richardson reste la plus précise et performante grâce à son pas adaptatif.

3 Double pendule non-amorti

3.1 Introduction

Dans cette partie, nous allons étudier le mouvement d'un double pendule non-amorti. Celui-ci se compose d'un pendule simple, c'est-à-dire une bille de masse *m1* suspendue à un fil de longueur *l1* dont on suppose la masse négligeable, au bout duquel est suspendu un deuxième pendule, de masse *m2* et de longueur *l2*. En coordonnées polaires, les positions de ces deux pendules s'expriment selon des angles θ_1, θ_1.

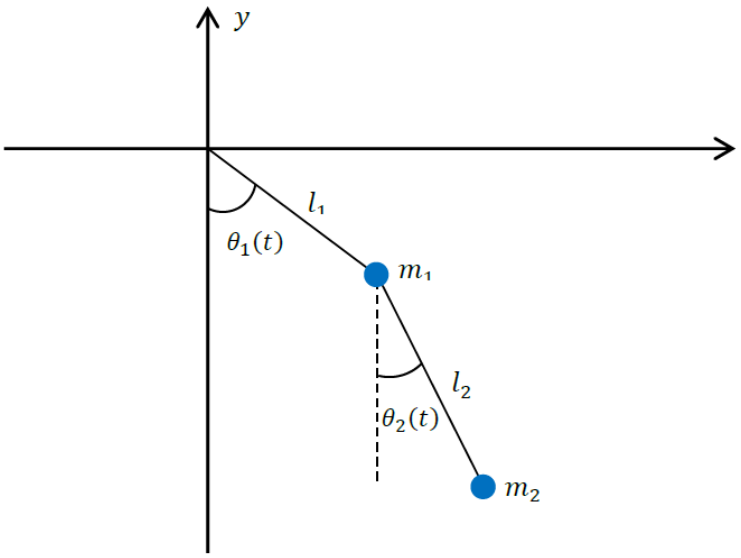

FIGURE 13: Schéma du pendule double

On peut alors démontrer que les angles θ_1 et θ_2 sont décrits par le système d'équations différentielles d'ordre 2 suivant :

$$\begin{cases} (m_1+m_2)l1\theta_1''(t) + m_2 l_2 \theta_2(t)\cos\theta_1(t)-\theta_2(t) + m_2 l_2 \theta_2'(t)^2 \sin\theta_1(t)-\theta_2(t) + (m_1+m_2)g\sin\theta_1(t)-\theta_2(t) = 0 \\ l_1\theta_1''(t)\cos\theta_1(t)-\theta_2(t) + l_2\theta_2''(t) - l_1\theta_1'(t)^2\sin\theta_1(t)-\theta_2(t) + g\sin\theta_1(t)-\theta_2(t) = 0 \end{cases}$$

Il s'agit là d'un système d'équations différentielles non linéaires d'ordre 2n on ne peut donc pas en trouver de solutions analytiques. En revanche, nous pouvons en approcher par des méthodes de résolution numérique. Dans notre cas, nous avons implémenté les méthodes de Runge-Kutta 4 et d' Euler-Richardson.
Pour cela, le système précédent peut être écrit sous la forme du système de 4 équations différentielles linéaires d'ordre 1 suivant :

$$\begin{cases} \theta_1' = f_1(t,\theta_1,\theta_2,z_1,z_2) \\ \theta_2' = f_2(t,\theta_1,\theta_2,z_1,z_2) \\ z_1' = g_1(t,\theta_1,\theta_2,z_1,z_2) \\ z_2' = g_2(t,\theta_1,\theta_2,z_1,z_2) \end{cases}$$

Avec :

$$\begin{cases} f_1(t,\theta_1,\theta_2,z_1,z_2) = z_1 \\ \\ f_2(t,\theta_1,\theta_2,z_1,z_2) = z_2 \\ \\ g_1(t,\theta_1,\theta_2,z_1,z_2) = \frac{g(2m_1+m_2)sin\theta_1+m_2[gsin(\theta_1-2\theta_2)+2(l_2z_2^2+l_1z_1^2cos(\theta_1-\theta_2))sin(\theta_1-\theta_2)]}{2l_1[m_1+m_2sin^2(\theta_1-\theta_2)]} \\ \\ g_2(t,\theta_1,\theta_2,z_1,z_2) = sin(\theta_1-\theta_2)\frac{(m_1+m_2)(l_1z_1^2+gcos\theta_1)+l_2m_2z_2^2cos(\theta_1-\theta_2)}{l_2[m_1+m_2sin^2(\theta_1-\theta_2)]} \end{cases}$$

3.2 Méthode

3.2.1 Méthode de Runge-Kutta 4 (en 4D)

A l'image de ce qui a été fait précédemment, nous étendons la méthode de Runge-Kutta à 4 dimensions. Pour cela, nous introduisons les nouvelles variables u et v. Ainsi, nous nous rendons à résoudre ce nouveau système à 4 inconnues de manière très similaire à celui à deux inconnues.

Ce faisant, nous pouvons écrire le programme suivant :

```
function [x,y,u,v,t]=fct_RK4_4D2(x0,y0,u0,v0,tmin,tmax,h,f1,f2,f3,f4)


t = tmin:h:tmax;
x = zeros(1,length(t));
x(1) = x0;
y = zeros(1,length(t));
y(1) = y0;
u = zeros(1,length(t));
u(1) = u0;
v = zeros(1,length(t));
v(1) = v0;

for k = 1:length(t)-1

    kf1_1 = f1(t(k),x(k),y(k),u(k),v(k));
    kf2_1 = f2(t(k),x(k),y(k),u(k),v(k));
    kf3_1 = f3(t(k),x(k),y(k),u(k),v(k));
    kf4_1 = f4(t(k),x(k),y(k),u(k),v(k));

    kf1_2 = f1(t(k) + h / 2, x(k) + (h / 2) * kf1_1, y(k) + (h / 2) *
        kf2_1, u(k) + (h/2) * kf3_1, v(k) + (h/2) * kf4_1);
    kf2_2 = f2(t(k) + h / 2, x(k) + (h / 2) * kf1_1, y(k) + (h / 2) *
        kf2_1, u(k) + (h/2) * kf3_1, v(k) + (h/2) * kf4_1);
    kf3_2 = f3(t(k) + h / 2, x(k) + (h / 2) * kf1_1, y(k) + (h / 2) *
        kf2_1, u(k) + (h/2) * kf3_1, v(k) + (h/2) * kf4_1);
    kf4_2 = f4(t(k) + h / 2, x(k) + (h / 2) * kf1_1, y(k) + (h / 2) *
        kf2_1, u(k) + (h/2) * kf3_1, v(k) + (h/2) * kf4_1);

```

```
26      kf1_3 = f1(t(k) + h / 2, x(k) + (h / 2) * kf1_2, y(k) + (h / 2) *
            kf2_2, u(k) + (h/2) * kf3_2, v(k) + (h/2) * kf4_2);
27      kf2_3 = f2(t(k) + h / 2, x(k) + (h / 2) * kf1_2, y(k) + (h / 2) *
            kf2_2, u(k) + (h/2) * kf3_2, v(k) + (h/2) * kf4_2);
28      kf3_3 = f3(t(k) + h / 2, x(k) + (h / 2) * kf1_2, y(k) + (h / 2) *
            kf2_2, u(k) + (h/2) * kf3_2, v(k) + (h/2) * kf4_2);
29      kf4_3 = f4(t(k) + h / 2, x(k) + (h / 2) * kf1_2, y(k) + (h / 2) *
            kf2_2, u(k) + (h/2) * kf3_2, v(k) + (h/2) * kf4_2);
30
31      kf1_4 = f1(t(k) + h, x(k) + h * kf1_3, y(k) + h * kf2_3, u(k) + h *
             kf3_3, v(k) + h * kf4_3);
32      kf2_4 = f2(t(k) + h, x(k) + h * kf1_3, y(k) + h * kf2_3, u(k) + h *
             kf3_3, v(k) + h * kf4_3);
33      kf3_4 = f3(t(k) + h, x(k) + h * kf1_3, y(k) + h * kf2_3, u(k) + h *
             kf3_3, v(k) + h * kf4_3);
34      kf4_4 = f4(t(k) + h, x(k) + h * kf1_3, y(k) + h * kf2_3, u(k) + h *
             kf3_3, v(k) + h * kf4_3);
35
36      x(k+1) = x(k) + h / 6 * (kf1_1 + 2 * kf1_2 + 2 * kf1_3 + kf1_4);
37      y(k+1) = y(k) + h / 6 * (kf2_1 + 2 * kf2_2 + 2 * kf2_3 + kf2_4);
38      u(k+1) = u(k) + h / 6 * (kf3_1 + 2 * kf3_2 + 2 * kf3_3 + kf3_4);
39      v(k+1) = v(k) + h / 6 * (kf4_1 + 2 * kf4_2 + 2 * kf4_3 + kf4_4);
40
41  end
```

3.2.2 Méthode d'Euler-Richardson

Pour mettre en oeuvre la méthode d'Euler-Richardson et l'appliquer à notre double pendule, nous avons repris le code de l'étude du mouvement Keplerien auquel nous n'avons apporté que des modifications mineures. Ainsi, la méthode est pratiquement la même que celle décrite précédemment.

3.3 Résultats

Le code du script principal est disponible en annexes.

3.3.1 Méhode de Runge-Kutta 4 - Un seul double pendule

Lorsque nous exécutons notre script principal, nous obtenons la figure suivante :

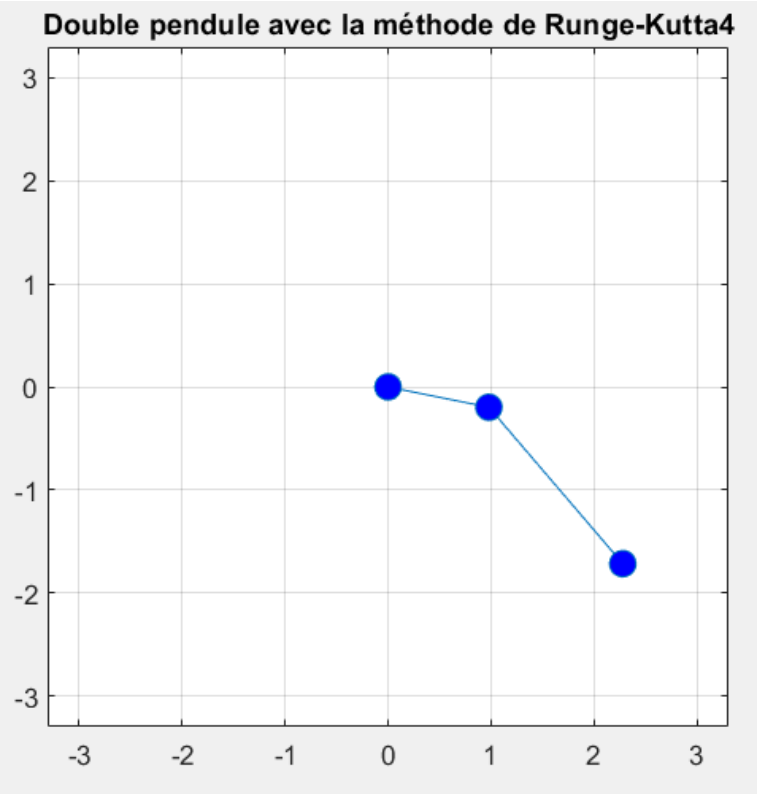

FIGURE 14: Double pendule à un instant t

Un arrêt sur image n'ayant pas vraiment de sens pour montrer le résultat de notre programme, vous pouvez consulter ici une séquence d'exécution.
On constate immédiatement que notre double pendule a un comportement erratique. En effet, la présence du deuxième pendule influe énormément sur le comportement du premier. Il est d'ailleurs à noter qu'en prenant $m_1 >> m_2$, on retrouve une situation de pendule simple, la base de celui-ci étant la bille 1. Pour toute la suite, nous prendrons $m_1 = m_2$ et $l_2 = 2l_1$.
Il faut également noter que si l'on fait tourner notre programme assez longtemps ($\simeq$ 10s), on constate un comportement plus instable, qui se traduit par une accélération et, de fait, une certaine imprécision sur le mouvement. C'est là la limite de la méthode de Runge-Kutta 4 pour étudier ce double pendule.

3.3.2 Méthode d'Euler-Richardson

De la même manière que pour Runge-Kutta 4, vous pouvez consulter ici une séquence d'exécution de notre programme avec la méthode d'Euler-Richardson.
Il apparaît que notre pendule oscille de façon tout aussi erratique que pour la précédente méthode, mais celle-ci présente plusieurs avantages. Pour commencer, le pas de calcul se trouve être adaptatif, ce qui permet au programme d'optimiser le temps d'exécution en fonction du seuil choisi. De plus, cette méthode ne semble pas présenter d'imprécision comme la précédente, ou du moins après un

temps suffisamment long pour que nous ne l'ayons pas détectée. En revanche, et ceci est visible sur la séquence vidéo en lien plus haut, on observe des "ralentissements par accoups", et ceci intrinsèquement au pas, au seuil d'erreur ou au temps de pause entre les tracés.

3.3.3 Superposition d'un deuxième double pendule

Pour cette partie, nous modifions notre code et ajoutons un deuxième pendule, dont la position de départ ne varie que de $\Delta\theta_{02} = 0.001 rad/s$. Vous pouvez consulter la séquence vidéo associée ici.
Au début du parcours, les deux pendules suivent la même trajectoire, à très peu près. En revanche, on constate que les deux trajectoires se séparent rapidement, pour finalement évoluer de manière totalement différente. En réalité, l'expérience du double pendule est un cas d'école de sensibilité aux conditions initiales, théorisée par *Henri Poincaré* au XIX^e siècle, laquelle sera ensuite décrite dans les années 1970 par le terme *chaos* et illustrée par le célèbre *effet papillon*. Cela se traduit par une divergence de nos deux trajectoires, quelles que soient les conditions initiales, pour peu qu'elles diffèrent du plus petits des ϵ. D'après nos recherches, Matlab semble par défaut stocker les variables avec une précision de 10^{-16}. Cela signifie que c'est cette précision que nous devons avoir, si nous souhaitons obtenir exactement les mêmes trajectoires.

3.4 Conclusion

Nous venons d'étudier le pendule double non-amorti, en résolvant les équantions de mouvement de celui-ci à l'aide des méthodes de *Runge-Kutta 4* et d'*Euler-Richardson*. Il en ressort que le choix de la méthode dépend du cahier des charges que l'on fixe. En effet, *Runge-Kutta 4* nous offre un temps de calcul raisonnable, mais une imprécision certaine au bout d'un temps d'exécution donné et dépendant du pas de calcul choisi. Diminuer ce pas implique de rallonger le temps de calcul et ainsi perdre l'avantage que propose cette méthode. La seconde méthode, celle d'*Euler-Richardson*, ne semble pas présenter la même imprécision que *Runge-Kutta 4*. Son pas adaptatif lui permet également de proposer une vitesse d'exécution correcte également. Ainsi, il semble que la méthode d'*Euler-Richardson* soit la plus performante des deux, même si *Runge-Kutta 4* renvoie un résultat satisfaisant.
Quel que soit la méthode utilisée, il apparaît de façon immédiate que le double pendule est extrêmement sensible aux conditions initiales, ce qui, conceptuellement, met en exergue l'importance d'opter pour une méthode la plus précise possible.
Il pourrait également être intéressant d'ajouter un amortissement, et comparer à nouveau les méthodes. Puisqu'un amortissement viendrait complexifier les équations, nous pensons que celui-ci ne ferait que renforcer la différence de performance que nous soulignons.

4 Conclusion

Au cours de ce TP, nous avons étudié les équations du mouvement Keplerien ainsi que celles du mouvement du double pendule non-amorti. Pour cela, il s'est agit de résoudre celles-ci avec diverses méthodes d'analyse numériques.
On remarque que les méthodes d'*Euler* et de *Runge-Kutta 4* ne sont pas suffisamment précises. Les erreurs s'accumulent à chaque itération et on obtient des résultats presque totalement erronés.
Il en ressort que la méthode la plus adaptée s'avère être celle d'*Euler-Richardson* grâce à son pas adaptatif qui permet d'avoir une analyse précise tout en limitant le nombre de calculs nécessaires.

Il existe bien d'autres méthodes de résolution numériques d'équations différentielles. Par exemple, nous pourrions implémenter d'autres algorithme de la famille BDF.

5 Annexes

5.1 Mouvement Keplérien - fonction *main*

```
clear variables;
close all;

% Initialisation de l'intervalle temporel (en ua)
tmin=0;tmax=2;

% parametres du modele keplerien
a=1;
T=1;
GM = 4* pi^2 *T^2 /a^3;

% conditions initiales
x0=0.5*a;
y0=0;
vx0=0;
vy0=11.5;

% second membre de l'equ. diff. (x'',y'')=(vx',vy')=(f(t,x,y),g(t,x,y))
f = @(t,x,y)(-GM * x ./ ((x.^2 + y.^2).^1.5));
g = @(t,x,y)(-GM * y ./ ((x.^2 + y.^2).^1.5));

% pas temporel
h=0.01;

switch 3  %%Choix du modele d'analyse
    case 1 % methode d'Euler
    [X,Y,t] = fct_Euler_Kepler(x0,y0,vx0,vy0,tmin,tmax,h,f,g);

    case 2 % methode RK4
    [X,Y,t] = fct_RK4_2D_Kepler(x0,y0,vx0,vy0,tmin,tmax,h,f,g);

    case 3 % methode Euler-Richardson
    [X,Y,t]= fct_Euler_Richardson(x0,y0,vx0,vy0,tmin,tmax,h,f,g,0.01);
    T_total=t(end);
end

%%Affichage des resultats
figure(1);
xmin=-3*a; xmax=a;
ymin=-1.5*a; ymax=1.5*a;
tic;
for k=1:65:length(t)
x= X * a;
y= Y * a;
plot([0,x],[0,y],'Marker','o','MarkerFacecolor','r','MarkerSize',10);
axis('equal');
```

```
47 axis(1.1*[xmin,xmax,ymin,ymax]);
48 grid 'on';
49 t1=title('Mouvement de la planete P(x(t),(y(t))');
50 xlabel('x(t)')
51 ylabel('y(t)')
52 set(t1,'interpreter','latex');
53 end
54 cputime=toc;
55 fprintf('Duree de la simulation numerique : %1.2f\n',cputime);
```

5.2 Double pendule - fonction *main*

```
clear variables;
close all;
clc;
% parametres physiques

m1=0.7;%masse de la bille 1 (kg)
m2=0.7;%masse de la bille 2 (kg)
r1=0.035;%rayon de la bille (m)
r2=0.035;%rayon de la bille 2
gr=9.8;%acceleration de la pesanteur (m.s^-2)
l1=1;%longueur du fil 1 (m)
l2=2;%longueur du fil 2 (m)

% autres parametres

tmin=0;     % instant initial
tmax=20;  % instant final
pas=0.001;  % pas de calcul
seuil = 0.00002; %seuil d'erreur d'Euler-Richardson
nom_video = 'seq_video3.avi';
fprintf('Duree de l''experience physique : %1.2f\n',tmax-tmin);

% fonctions Y'=F(Y) avec ici Y=(theta,z) et F(Y)=(f,g)
f1=@(t,theta1,theta2,z1,z2)(z1);
f2=@(t,theta1,theta2,z1,z2)(z2);
f3=@(t,theta1,theta2,z1,z2)(-(gr*(2*m1+m2)*sin(theta1)+m2*(gr*sin(
    theta1-2*theta2)+2*(l2*z2^2+l1*z1^2*cos(theta1-theta2))*sin(theta1-
    theta2)))/(2*l1*(m1+m2*(sin(theta1-theta2))^2)));
f4=@(t,theta1,theta2,z1,z2)(sin(theta1-theta2)*((m1+m2)*(l1*z1^2+gr*cos
    (theta1)+l2*m2*z2^2*cos(theta1-theta2)))/(l2*(m1+m2*(sin(theta1-
    theta2))^2)));

% conditions initiales
%Pendule 1
theta01=2*pi/3;  % angle initial (rad)
thetap01=0;      % vitesse angulaire initiale (rad/s)
theta02=2*pi/3;  % angle initial (rad)
thetap02=0;      % vitesse angulaire initiale (rad/s)
%Pendule 2
theta03=2*pi/3;  % angle initial (rad)
thetap03=0;      % vitesse angulaire initiale (rad/s)
theta04=2*pi/3+0.001;  % angle initial (rad)
thetap04=0;      % vitesse angulaire initiale (rad/s)

%Calculs numeriques
switch 2 %Choix de la m thode de r solution
    %1 pour RK4, 2 pour Euler-Richardson
     case 1
```

```
45          [theta1,theta2,z1,z2,t]=fct_RK4_4D2...
46              (theta01,theta02,thetap01,thetap02,tmin,tmax,pas,f1,f2,f3,
                    f4);
47          [theta3,theta4,z3,z4,t]=fct_RK4_4D2...
48              (theta03,theta04,thetap03,thetap04,tmin,tmax,pas,f1,f2,f3,
                    f4);
49          t_pause = 0.001;
50          strTitle='Double pendule avec la m thode de Runge-Kutta4';
51      case 2
52          [theta1,theta2,z1,z2,t] = fct_Euler_Richardson_double_pendule
                ...
53              (theta01,theta02,thetap01,thetap02,tmin,tmax,pas,f3,f4,
                    seuil);
54          t_pause = 0.005;
55          strTitle ='Double pendule avec la m thode d''Euler-Richardson'
                ;
56  end
57
58  % affichage des resultats
59  figure(1);
60  xmin=-(l1+l2);xmax=(l1+l2);
61  ymin=-(l1+l2);ymax=(l1+l2);
62  tic;
63  v=VideoWriter(nom_video);
64  open(v);
65  drawnow;
66  for k=1:65:length(theta1)
67  x1=sin(theta1(k)) * l1;
68  y1= -cos(theta1(k)) * l1;
69  x2=x1+sin(theta2(k)) * l2;
70  y2=y1 -cos(theta2(k)) * l2;
71  % x3=sin(theta3(k)) * l1;
72  % y3= -cos(theta3(k)) * l1;
73  % x4=x3+sin(theta4(k)) * l2;
74  % y4=y3 -cos(theta4(k)) * l2;
75  plotx1 = [0,x1,x2];
76  %plotx2 = [0,x3,x4];
77  ploty1 = [0,y1,y2];
78  %ploty2 = [0,y3,y4];
79  figure(1)
80  plot(plotx1,ploty1,'Marker','o','MarkerFacecolor','b','MarkerSize',10);
81  axis('equal');
82  axis(1.1*[xmin,xmax,ymin,ymax]);
83  grid 'on';
84  t1=title(strTitle);
85  pause(t_pause)
86  thisframe=getframe(gcf);
87  writeVideo(v,thisframe);
88
89  end
```

```
90
91  close(v);
```