Intégration des équations différentielles : méthode du point milieu

1. Introduction

On s'intéresse à la résolution numérique approchée (ou intégration numérique) d'un système d'équations différentielles de la forme :

$$Y'(t) = f(Y, t) \tag{1}$$

où Y est une matrice colonne contenant N fonctions du temps. Lorsqu'une condition initiale $Y(0)$ est fixée, la solution est unique.

La méthode d'Euler est la méthode la plus simple d'intégration numérique de ce type de système. Elle a l'inconvénient d'être seulement d'ordre 1 : le pas de temps doit être divisé par 10 pour améliorer la précision d'un facteur 10.

La méthode du point milieu est une amélioration de la méthode d'Euler qui a sur cette dernière l'avantage d'être d'ordre 2 : si le pas de temps est divisé par 10, la précision augmente d'un facteur 100. En pratique, le même pas de temps conduit à une précision bien meilleure que la méthode d'Euler.

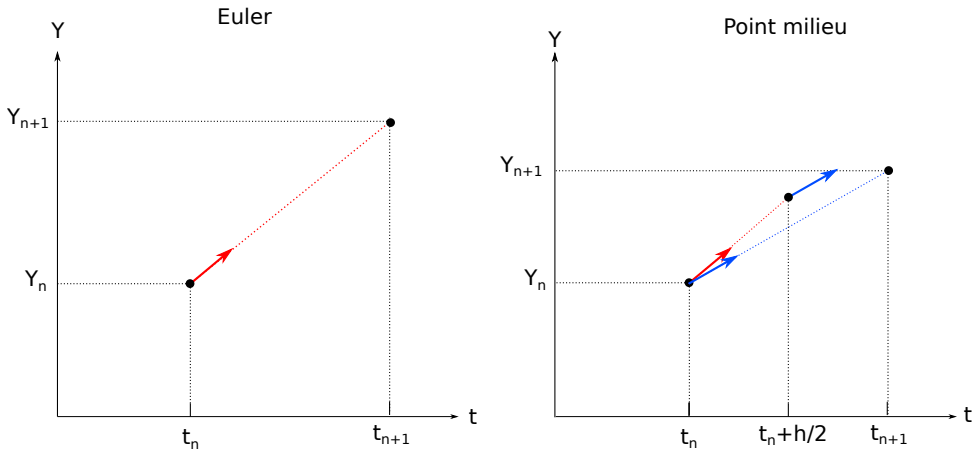
Nous allons définir la méthode du point milieu puis l'implémenter en mettant en œuvre une technique d'adaptation du pas, qui permet de contrôler l'erreur locale grace à une tolérance choisie au départ.

2. Méthode du point milieu

Soit h le pas de temps. Notons Y_n l'approximation obtenue à l'instant t_n. Il s'agit d'obtenir Y_{n+1}, l'approximation à l'instant $t_{n+1} = t_n + h$. La méthode d'Euler consiste à évaluer la dérivée à l'instant t_n (c.a.d. $f(Y_n, t_n)$) puis à assimiler la courbe entre les instants t_n et t_{n+1} à un segment de droite dont la pente est donnée par cette dérivée :

$$Y_{n+1} = Y_n + hf(Y_n, t_n) \tag{2}$$

La figure suivante illustre le principe de la méthode d'Euler. Une flèche partant d'un point représente la dérivée évaluée en ce point.



La méthode du point milieu consiste à tout d'abord estimer Y à l'instant $t_n + h/2$ puis à évaluer la dérivée en se servant de cette estimation ; cette dérivée est alors utilisée pour calculer Y_{n+1}. La méthode s'écrit :

$$k_1 = hf(Y_n, t_n) \tag{3}$$

$$k_2 = hf(Y_n + \frac{1}{2}k_1, t_n + \frac{1}{2}h) \tag{4}$$

$$Y_{n+1} = Y_n + k_2 \tag{5}$$

La méthode du point milieu nécessite deux calculs des dérivées (fonction f) à chaque pas de temps. C'est deux fois plus que la méthode d'Euler, mais ces calculs supplémentaires sont largement compensés par le fait que la méthode est d'ordre 2, c'est-à-dire que l'erreur évolue comme h^2 lorsque le pas de temps est assez petit.

La méthode du point milieu fait partie de la classe des méthodes de Runge-Kutta. Ces méthodes consistent à effectuer plusieurs évaluations successives de la dérivée dans l'intervalle $[t_n, t_n + h]$ et à calculer Y_{n+1} comme une combinaison linéaire de ces dérivées. La méthode du point milieu est une méthode de Runge-Kutta d'ordre 2. Il existe des méthodes de Runge Kutta d'ordre plus élevé, la plus employée étant d'ordre 4. Il est d'usage d'écrire ces méthodes avec des coefficients k_i, qui sont des évaluations des dérivées multipliées par le pas de temps.

3. Adaptation du pas de temps

L'utilisation d'un pas de temps h constant sur la totalité de l'intervalle de temps $[0, T]$ est souvent inefficace car le pas de temps doit être assez petit pour assurer la précision voulue lorsque Y varie très rapidement, mais il peut être alors trop petit lorsque Y varie plus lentement.

Il est donc intéressant d'utiliser une méthode permettant de faire varier le pas de temps, tout en ayant un moyen de contrôler l'erreur locale. Il faut pour cela procéder à chaque pas de temps à une évaluation de l'erreur locale et comparer cette évaluation à une tolérance. L'évaluation de l'erreur locale ne peut se faire qu'à partir d'un critère empirique. La méthode la plus employée consiste à comparer deux valeurs de Y_{n+1}, la première obtenue avec le pas h, la seconde obtenue avec deux fois le pas moitié $h/2$. Cette dernière est toujours plus précise que la première mais ce qui importe ici est la différence entre ces deux estimations :

$$\Delta = Y_{n+1}^{(h)} - Y_{n+1}^{(h/2)} \tag{6}$$

Notons que la seconde nécessite deux fois plus d'évaluations de la dérivée que la première. La différence Δ est en fait une matrice colonne de N valeurs, autrement dit un vecteur. Nous avons besoin d'une valeur associée à ce vecteur ; le plus simple est d'utilisee le maximum des valeurs absolues des composantes :

$$\delta = \mathrm{Max}\left(|\Delta_i|\right)_{1 \leq i \leq N} \tag{7}$$

δ constitue une estimation empirique de l'erreur locale commise lorsqu'on calcule Y_{n+1} avec un pas de temps h. La tolérance est une valeur petite, notée ϵ, que l'on compare à l'estimation de l'erreur. Si l'estimation de l'erreur est supérieure à la tolérance, il faut diminuer le pas de temps. Si au contraire l'estimation de l'erreur est inférieure à la tolérance, il faut augmenter le

pas de temps. Dans un premier temps, on choisit d'augmenter ou de diminuer le pas de 10%. Voici la procédure :

- ▷ Si $\delta < \epsilon$ on augmente le pas en lui donnant la valeur $h' = 1.1h$:
  Dans ce cas, la valeur de Y_{n+1} retenue est celle obtenue avec le pas $h/2$ et la valeur suivante Y_{n+2} est calculée avec le nouveau pas h'.
- ▷ Si $\delta > \epsilon$ on diminue le pas en lui donnant la valeur $h' = 0.9h$:
  Dans ce cas, on doit refaire le calcul de Y_{n+1} avec le nouveau pas h', évaluer la nouvelle erreur puis répéter l'adaptation du pas. Il y a donc une boucle avec réduction itérative du pas jusqu'à obtenir la condition $\delta < \epsilon$.

La tolérance ϵ est calculée en fonction de deux paramètres constants : la tolérance absolue ϵ_a et la tolérance relative ϵ_r, selon la relation :

$$\epsilon = \epsilon_a + \epsilon_r |Y_n| \tag{8}$$

où $|Y_n|$ désigne une norme du vecteur Y_n. On peut prendre par exemple le maximum des valeurs absolues des composantes :

$$|Y_n| = \mathrm{Max}\left(|Y_{n,i}|\right)_{1 \leq i \leq N} \tag{9}$$

Les tolérances absolue et relative sont deux paramètres constants, choisis au début du calcul. Ils permettent de contrôler la précision du résultat, dans le sens où l'abaissement de la tolérance permet d'augmenter la précision. Cependant, les valeurs choisies pour ces tolérances ne permettent en aucun cas de déduire une information sur l'erreur globale (l'erreur à l'instant final T). Pour obtenir cette information, il faudra procéder deux fois à la totalité du calcul sur l'intervalle $[0, T]$, la première fois avec une certaine valeur des tolérances, la second fois avec des tolérances dix fois plus petites.

4. Implémentation en python

4.a. Définition du système différentiel

Comme exemple de système différentiel, on considère le problème de Kepler : un corps de masse m est soumis à l'attraction gravitationnel d'un corps dont la position est fixe dans le référentiel. Soit $(Oxyz)$ un repère lié au référentiel galiléen, le corps attracteur étant supposé fixe et situé au point O. Le corps en mouvement gravitationnel est assimilé à un point matériel de coordonnées $(x, y, 0)$ car son mouvement se fait dans le plan Oxy.

La loi de la dynamique s'écrit :

$$m\frac{d\overrightarrow{v}}{dt} = -K\frac{x\overrightarrow{u}_x + y\overrightarrow{u}_y}{(x^2+y^2)^{\frac{3}{2}}} \tag{10}$$

Pour la résolution numérique, on pose $K/m = 1$. Les variables sont $Y = (x, y, v_x, v_y)$. Le système différentiel s'écrit :

$$\begin{aligned}
\frac{dx}{dt} &= v_x \\
\frac{dy}{dt} &= v_y \\
\frac{dv_x}{dt} &= -\frac{x}{(x^2+y^2)^{\frac{3}{2}}} \\
\frac{dv_y}{dt} &= -\frac{y}{(x^2+y^2)^{\frac{3}{2}}}
\end{aligned}$$

La condition initiale sera prise sous la forme :

$$Y(0) = (1, 0, 0, v_0) \tag{11}$$

où v_o est la vitesse initiale, comprise entre 1 (mouvement circulaire) et $\sqrt{2}$ (mouvement parabolique).

Le vecteur Y_n sera stocké dans un tableau `numpy.ndarray`, ce qui permettra d'utiliser les opérateurs sur ce type de tableaux pour faire les calculs (par exemple l'addition de deux tableaux). On utilisera la fonction `A.max()` pour obtenir la valeur maximale d'un tableau.

[1] Écrire la fonction `systeme(Y,t)` pour ces équations. Cette fonction prend en arguments le tableau des valeurs de Y et l'instant t. Elle renvoie les valeur des dérivées, c'est-à-dire $f(Y, t)$, sous la forme d'un tableau `numpy.ndarray`.

4.b. Méthode du point milieu à pas constant

[2] Écrire une fonction `pas_rk2(systeme,h,tn,Yn)` qui effectue le calcul de Y_{n+1} avec la méthode du point milieu (Runge-Kutta d'ordre 2).

[3] Écrire une fonction `rk2(systeme,Yi,T,h)` qui effectue le calcul jusqu'à l'instant `T` avec un pas de temps constant `h`. La tableau `Yi` contient les valeurs initiales. La fonction `rk2` renvoie deux tableaux : le tableau contenant les instants t_n et le tableau bidimensionnel contenant les Y_n, dont chaque ligne correspond à une valeur de n, c'est-à-dire à l'instant t_n. La construction de ces deux tableaux se fera à l'aide de piles standard (listes python et fonction `append`).

[4] Tester avec le code suivant, qui permet en principe d'obtenir une trajectoire elliptique de forte excentricité :

```
T=100
h=0.01
Yi=[1,0,0,1.3]
(t,tab_y) = rk2(systeme,Yi,T,h)
x = tab_y[:,0]
y = tab_y[:,1]
fig,ax=plt.subplots()
plt.plot(x,y)
plt.grid()
ax.set_aspect('equal')
plt.show()
```

Faire le calcul avec des pas de temps de plus en plus grand et observer la baisse de la précision.

4.c. Méthode du point milieu avec adaptation du pas

[5] Écrire une fonction `pas_rk2_adapt(systeme,h,t,Yn,tolA,tolR)` qui effectue le calcul de Y_{n+1} avec l'adaptation du pas. Les arguments `tolA,tolR` sont les tolérances absolue et relative. La fonction doit renvoyer `(h,t,Y)` où `h` et le nouveau pas de temps, `t` l'instant t_{n+1} et `Y` le tableau contenant Y_{n+1}.

[6] Écrire une fonction `rk2_adapt(systeme,Yi,T,h,tolA,tolR)` qui effectue le calcul jusqu'à l'instant `T` avec `h` comme pas initial. La fonction renvoie les mêmes tableaux que la fonction `rk2` avec en plus le tableaux des valeurs du pas de temps.

[7] Tester avec le code suivant :

```
T=100
h=0.1
Yi=[1,0,0,1.3]
tolA=tolR=1e-5
(t,tab_y,liste_h) = rk2_adapt(systeme,Yi,T,h,tolA,tolR)
x = tab_y[:,0]
y = tab_y[:,1]
fig,ax=plt.subplots()
plt.plot(x,y)
plt.grid()
ax.set_aspect('equal')

plt.figure()
plt.plot(numpy.log10(liste_h))
plt.plot(x)
plt.grid()
plt.show()
```

[8] Dans quelle partie de la trajectoire le pas de temps est-il le plus petit ?

[9] Refaire le calcul avec une durée T plus grande et avec différentes tolérances.

[10] Faire le calcul avec les valeurs suivantes :

```
T = 1000
tolA=tolR=1e-4
```

Que se passe-t-il ?

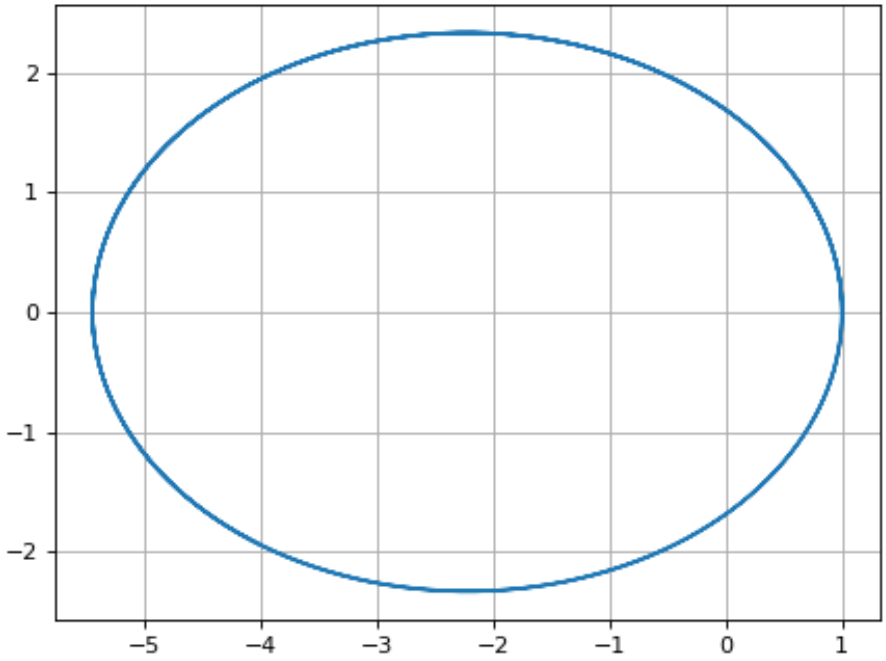
[11] Effectuer une estimation de l'erreur à l'instant $t = 100$ lorsque $\epsilon_r = \epsilon_a = 10^{-4}$ en faisant le calcul avec ces tolérances puis avec des tolérances 10 fois plus petites.

[12] Modifier le code afin qu'il calcule le nombre d'appels de la fonction `pas_rk2` puis affiche ce nombre à la fin du calcul. Tester différents facteurs d'augmentation et de réduction du pas.

5. Solution

5.a. Méthode du point milieu à pas constant

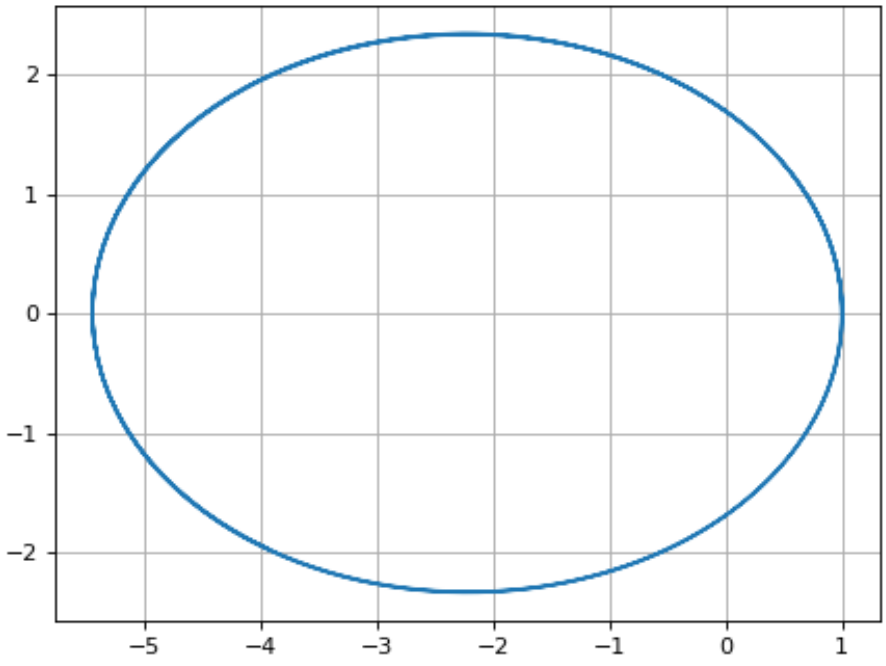
```
import numpy
from matplotlib.pyplot import *

def kepler(Y,t):
    f = numpy.power(Y[0]**2+Y[1]**2,-1.5)
    return numpy.array([Y[2],Y[3],-Y[0]*f,-Y[1]*f])

def pas_rk2(systeme,h,tn,Yn):
    deriv1 = systeme(Yn,tn)
    deriv2 = systeme(Yn+h*deriv1*0.5,tn+0.5*h)
    return Yn+h*deriv2

def rk2(systeme,Yi,T,h):
    Y = Yi
    t = 0.0
    liste_y = [Y]
    liste_t = [t]
    while t<T:
        Y = pas_rk2(systeme,h,t,Y)
        t += h
        liste_y.append(Y)
        liste_t.append(t)
    return (numpy.array(liste_t),numpy.array(liste_y))

T=100
h=0.01
Yi=[1,0,0,1.3]
(t,tab_y) = rk2(kepler,Yi,T,h)
x = tab_y[:,0]
y = tab_y[:,1]
figure()
plot(x,y)
grid()
```



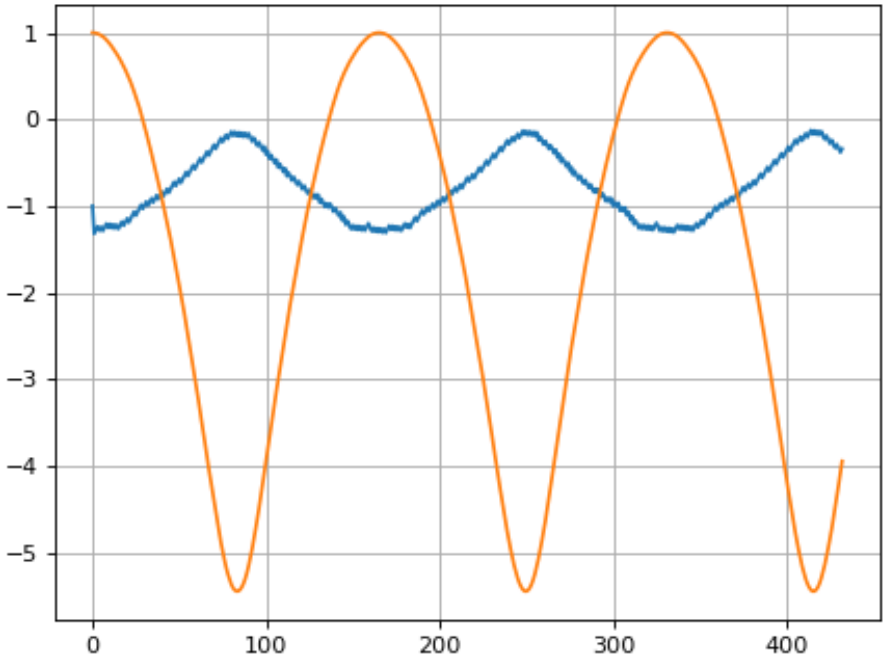
5.b. Méthode du point milieu avec adaptation du pas

```
def pas_rk2_adapt(systeme,h,tn,Yn,tolA,tolR,neval):
    Ya = pas_rk2(systeme,h,tn,Yn)
    Yb = pas_rk2(systeme,h/2,tn,Yn)
    Yc = pas_rk2(systeme,h/2,tn+h/2,Yb)
    neval[0] += 1
    delta = numpy.absolute(Yc-Ya).max()
    epsilon = tolA+tolR*numpy.absolute(Ya).max()
    if delta <= epsilon:
        tn += h
        h=h*1.1
    else:
        while delta > epsilon:
            h=h*0.9
            Ya = pas_rk2(systeme,h,tn,Yn)
            Yb = pas_rk2(systeme,h/2,tn,Yn)
            Yc = pas_rk2(systeme,h/2,tn+h/2,Yb)
            neval[0] += 1
            delta = numpy.absolute(Yc-Ya).max()
            epsilon = tolA+tolR*numpy.absolute(Ya).max()
        tn += h
    return (h,tn,Yc)

def rk2_adapt(systeme,Yi,T,h,tolA,tolR):
    Y = Yi
    t = 0.0
    liste_y = [Y]
    liste_t = [t]
    liste_h = [h]
    neval = [0]
```

```
    while t<T:
        (h,t,Y) = pas_rk2_adapt(systeme,h,t,Y,tolA,tolR,neval)
        liste_y.append(Y)
        liste_t.append(t)
        liste_h.append(h)
    print(neval[0])
    return (numpy.array(liste_t),numpy.array(liste_y),numpy.array(liste_h))

T=100
h=0.1
Yi=[1,0,0,1.3]
tolA=tolR=1e-5
(t,tab_y,liste_h) = rk2_adapt(kepler,Yi,T,h,tolA,tolR)
x = tab_y[:,0]
y = tab_y[:,1]
figure()
plot(x,y)
grid()
```




```
figure()
plot(numpy.log10(liste_h))
plot(x)
grid()
```



Le pas de temps diminue au voisinage du périastre et augmente au voisinage de l'apoastre.

```
T = 100
tolA=tolR=1e-4
Yi=[1,0,0,1.3]
(t,tab_y,liste_h) = rk2_adapt(kepler,Yi,T,h,tolA,tolR)
x1 = tab_y[-1:,0]
(t,tab_y,liste_h) = rk2_adapt(kepler,Yi,T,h,tolA/10,tolR/10)
x2 = tab_y[-1:,0]
(t,tab_y,liste_h) = rk2_adapt(kepler,Yi,T,h,tolA/100,tolR/100)
x3 = tab_y[-1:,0]



print(x2-x1)
--> array([-0.09047352])

print(x3-x2)
--> array([-0.0568701])
```