

Programmation et Algorithmes numériques 2

Intégration numérique d'équations différentielles ordinaires

D. Cornu

S4-2019

Un **problème différentiel** s'exprime selon une équation différentielle scalaire d'ordre n

$$\frac{d^n y}{dt^n} = f \left(t, y, \frac{dy}{dt}, \dots, \frac{d^{n-1}y}{dt^{n-1}} \right)$$

f est la fonction **second membre**.

Donne une **famille** de solutions $y(t)$ à n paramètres.

Pour résoudre ce type de système il faut donner n conditions imposées
→ nous donne **une** solution dans la famille

Type de problème

Conditions initiales données pour une seule valeur t_0 de t :

$$y(t_0) = y_0, y'(t_0) = y'_0, \dots, y^{n-1}(t_0) = y_0^{n-1}$$

Problème dit de **conditions initiales** ou de Cauchy

Conditions données pour des valeurs spécifiques de la variable indépendante t :

$$y(t_0) = y_0, y(t_1) = y_1, \dots, y(t_{n-1}) = y_{n-1}$$

Problème dit de **conditions aux limites**

La plupart des problèmes physiques conduisent à des (systèmes d')
équations différentielles.

Désintégration radioactive

$$\frac{d}{dt}N(t) = -kN(t)$$

$N(t)$ nb d'atomes au
temps t

C.I : $N(0) = N_0$

Equation d'onde

$$\frac{1}{c^2} \frac{d^2}{dt^2} E(x, t) = \frac{d^2}{dx^2} E(x, t)$$

$E(x, t)$: champ électrique

Mécanique

$$m \frac{d^2}{dt^2} x(t) = -kx(t)$$

C.I : $x(0) = x_0$ &
 $v(0) = \left. \frac{dx}{dt} \right|_{t=0} = v_0$

Très souvent on ne sait pas résoudre ces systèmes de manière analytique :
→ **résolution numérique**

Equation différentielle scalaire d'ordre 1

Le problème général se simplifie dans le cas des EDO du **premier ordre**
Donne une famille de solution $y(t)$ à **un** paramètre (y_0)

$$\frac{dy}{dt} = f(t, y(t)) \quad \text{avec} \quad y(t_0) = y_0$$

Dans le cas où le membre de droite ne dépend pas de $y(t)$ on a :

$$\frac{dy}{dt} = f(t) \quad \text{solution :} \quad y(t) = y_0 + \int_{t_0}^t f(t) dt$$

Auquel cas on peut simplement utiliser une méthode d'intégration de type rectangles ou trapèzes, ...

Equation différentielle scalaire d'ordre 1

$$\frac{dy}{dt} = f(t, y(t)) \quad \text{avec} \quad y(t_0) = y_0$$

Exemple du cas général : particules dans un champ variable

$$m \frac{dv}{dt} = \sum \text{forces} = f(t, v(t))$$

On voit que la force dépend de la vitesse v et que le champ de force dépend explicitement du temps.

*Note : un système d'EDO d'ordre supérieur n se ramène à **des systèmes différentiels couplés** de n équationss du première ordre \rightarrow EDO vectorielles d'ordre 1*

Résolution Numérique

Dans le cas d'une résolution numérique, on opère une discrétisation par découpage d'un intervalle $[t_0, t_0 + L]$ longueur L selon un pas constant h .

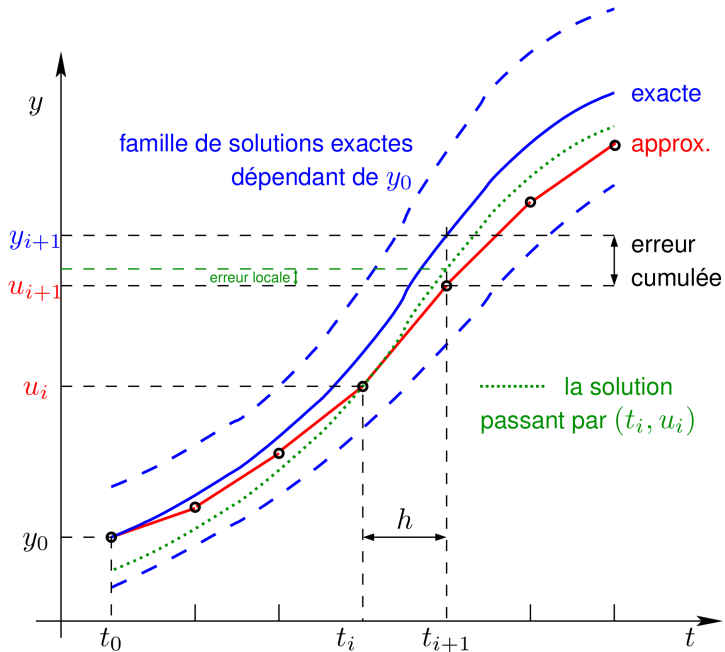
Echantillonnage aux instants $t_i = y_0 + ih$ pour $1 \leq i \leq n$

Numériquement pour un problème avec condition initiale, cela revient à faire une boucle sur les t_i pour calculer l'approximation u_{i+1} à t_{i+1}

C'est une approximation **de proche en proche** de la solution sur l'intervalle L .

⇒ accumulation des erreurs dans la boucle, on parle de **dérive** ou d'erreur cumulée

- Si calcul sur la dernière valeur calculée u_i : **méthodes à un pas**
- Si calcul sur plusieurs valeurs précédentes u_{i-k} ($k \geq 0$) : **méthodes à plusieurs pas** (initialisation le plus souvent à un pas)



Méthodes à un pas

L'estimation de la fonction au pas suivant se résume à un développement limité de Taylor :

$$y(t_i + h) = y(t_i) + h \frac{dy}{dt}(t_i) + \frac{h^2}{2} \frac{d^2y}{dt^2}(t_i) + \dots$$

L'**ordre** n de la méthode est la plus grande puissance de h prise en compte dans l'approximation.

On distingue deux types d'erreur qui se cumulent :

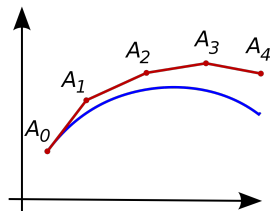
Erreur de troncature locale

- Somme des **termes négligés**
 $\propto h^{n+1}$.
- Déterministe : Augmente si le pas h augmente et est proportionnelle à l'ordre de la méthode +1

Erreur d'arrondi

- **Erreur de précision (finie) des opérations** sur les réels.
- "Aleatoire" : Augmente si les calculs se compliquent, augmente pour un pas plus fin (h petit)

Taille des pas, "résolution/Sampling" de la méthode



Méthode d'ordre 1

La dérive de l'erreur est plus forte si la fonction change rapidement.

Il faut adapter la taille du pas h afin de garantir que la fonction soit bien résolue.

L'Ordre de la méthode permet pour **une même taille de pas** de moins dériver.

D'une manière générale on considère que réduire le pas permet une meilleure solution au prix d'un coût de calcul élevé.

Attention cependant à ne pas devenir dominé par les erreurs d'arrondie ! Il s'agit en réalité d'un **compromis** !

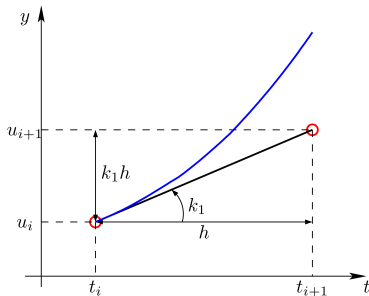
Méthode d'Euler explicite

On exprime la dérivée comme une différence finie :

$$\frac{dy}{dt} = f(t, y(t)) \approx \frac{y(t+h) - y(t)}{h}$$

$$\frac{u_{i+1} - u_i}{h} = f(t_i, u_i)$$

$$\Rightarrow u_{i+1} = u_i + hf(t_i, u_i)$$



La méthode d'Euler néglige les termes en h^2

$$\Rightarrow y(t_i + h) = y(t_i) + hf(t_i, y(t_i)) + O(h^2)$$

On intègre sur L avec un pas h on a donc L/h pas d'intégration \Rightarrow la méthode est donc uniquement **d'ordre 1**

Note : Il existe aussi une méthode d'Euler **Implicite** (ou rétrograde) mais qui nécessite de connaître une solution analytique de u_{i+1}

Exemple de la méthode d'Euler

Rappel objectif : $\frac{dy}{dt} = f(t, y(t))$

$$u_{i+1} = u_i + hf(t_i, u_i)$$

$$y'(t) = t + y + 1$$

Solution exacte :

$$y(t) = 2e^t - t - 2$$

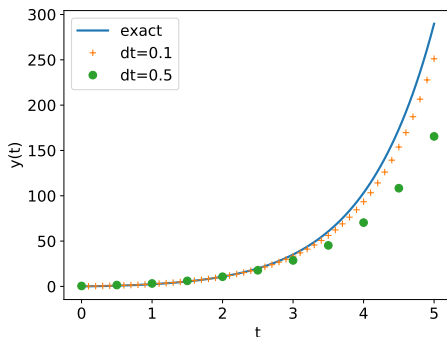
```
n = 10      #nb de pas  
h = 0.5     #intervalle (dt)
```

```
t = 0  
y = 0
```

```
data = np.zeros((n+1,2))
```

```
for i in range(0,n+1):  
    t = i * h  
    y = y + (t + y + 1) * h  
    data[i,0] = t  
    data[i,1] = y
```

```
...  
plot...
```



Exemple 2 de la méthode d'Euler

Cas d'un projectile lancé avec une vitesse v_0 dans un repère x, y

Position : $\vec{r} = (x, y)$ et : $\vec{v} = (v_x, v_y)$ Les équations de newton nous donnent :

$$\frac{d}{dt}\vec{r} = \vec{v}$$

$$\frac{d}{dt}\vec{v} = \frac{\vec{F}}{m}$$

$$x' = v_x$$

$$y' = v_y$$

$$v'_x = F_x/m = 0$$

$$v'_y = F_y/m = -g$$

C'est donc un système de **4 équations différentielles couplées !**

Via la méthode d'Euler : $u_{i+1} = u_i + hf(t_i, u_i)$

$$\vec{y} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \quad \vec{f} = \begin{bmatrix} v_x \\ v_y \\ 0 \\ -g \end{bmatrix}$$

$$x_{i+1} = x_i + v_x \times h$$

$$y_{i+1} = y_i + v_y \times h$$

$$v_{xi+1} = v_{xi} + 0 \times h$$

$$v_{yi+1} = v_{yi} - g \times h$$

Attention : l'ordre des calculs importe ! Ici il faut bien prendre les vitesses en i .

Si on met à jour les vitesses PUIS les positions avec les nouvelles méthode il s'agit d'un **leapfrog**

Méthode d'ordre 2

Pour **augmenter l'ordre de la méthode** on va bien évidemment tenter d'ajouter des termes du développement de Taylor :

$$y(t_i + h) = y(t_i) + h \frac{dy}{dt}(t_i) + \frac{h^2}{2} \frac{d^2y}{dt^2}(t_i) + \dots$$

$$\frac{dy}{dt} = f(t, y(t)) \quad \Rightarrow \quad \frac{d^2y}{dt^2} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} \frac{dy}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f$$

En pratique il est souvent difficile d'obtenir une évaluation correcte des dérivées partielles de f ...

On général on préférera une méthode capable d'évaluer **le second membre de f** en plusieurs points adaptés.



Méthode du point milieu

On centre l'évaluation de la **dérivée** en $t_m = (t_i + t_{i+1})/2$, appelé **point milieu** :

$$y(t_i + h) = y(t_m) + \frac{h}{2} \frac{dy}{dt}(t_m) + \frac{1}{2} \frac{h^2}{4} \frac{d^2y}{dt^2}(t_m) + O(h^3)$$
$$y(t_i) = y(t_m) - \frac{h}{2} \frac{dy}{dt}(t_m) + \frac{1}{2} \frac{h^2}{4} \frac{d^2y}{dt^2}(t_m) + O(h^3)$$

Par différence entre ces deux termes on trouve la solution suivante :

$$y(t_i + h) - y(t_i) = h \frac{dy}{dt}(t_m) + O(h^3)$$

Méthode du point milieu

Pour parler d'une méthode de second ordre on doit faire l'évaluation de f en 2 points, en (t_i, u_i) et **au milieu** $(t_{i+1/2} = t_i + h/2, u_{i+1/2})$

$$u_{i+1} = u_i + hf\left(t_i + \frac{h}{2}, u_i + \frac{h}{2}f(t_i, u_i)\right)$$

On peut alors écrire les coefficients k_i tels que :

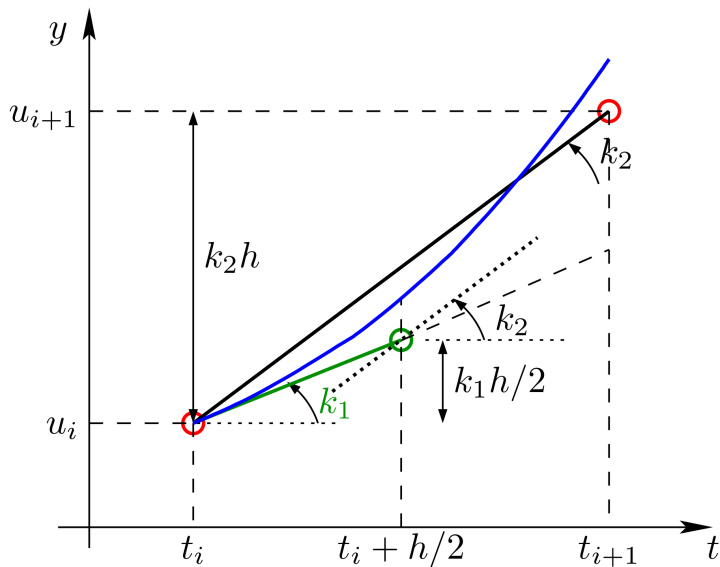
$$k_1 = f(t_i, u_i)$$

$$k_2 = f\left(t_i + \frac{h}{2}, u_i + k_1 \frac{h}{2}\right)$$

$$u_{i+1} = u_i + hk_2$$

Les termes k_1 et k_2 sont évalués l'un après l'autre et le dernier d'entre eux permet d'évaluer u_{i+1}

Illustration de la méthode du point milieu



Méthode de Runge-Kutta d'ordre 2 (RK2)

La méthode RK2 (ou Euler Modifiée) reprend les équations en t_i et t_{i+1} et fait **la moyenne des dérivées aux extrémités** (plutôt que la somme).
(similaire à la méthode des trapèzes)

$$\frac{dy}{dt}(t_i) + \frac{dy}{dt}(t_{i+1}) = 2\frac{dy}{dt}(t_m) + O(h^2)$$

ce qui évite une approximation via le point milieu t_m

$$u_{i+1} = u_i + \frac{h}{2} [f(t_i, u_i) + f(t_{i+1}, u_{i+1})]$$

C'est une méthode dite **implicite**, plus stable que les précédentes (mais plus lourde)

Le plus souvent on se sert tout de même de la méthode d'Euler pour évaluer u_{i+1}

Méthode de Runge-Kutta d'ordre 2 (RK2)

On parle de méthode : **prédicteur-correcteur**

$$k_1 = f(t_i, u_i)$$

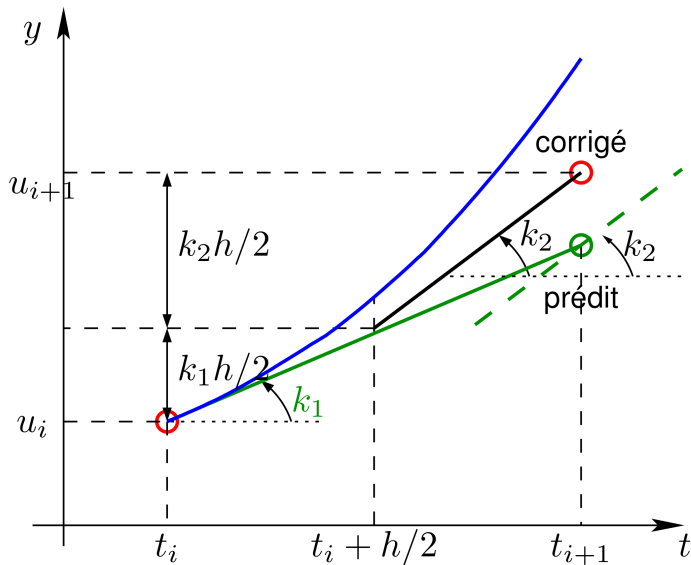
$$k_2 = f(t_{i+1}, u_{i+1})$$

$$u_{i+1} = u_i + \frac{h}{2} [k_1 + k_2]$$

Revient à faire un demi-pas avec chacune des estimation u_i et u_{i+1} effectuée avec Euler progressif.

- C'est une méthode d'ordre 2 comme le point milieu mais pas d'évaluation hors grille. - On peut itérer sur la correction jusqu'à ce que celle-ci devienne négligeable.

Illustration de la méthode RK2



Méthode de RK4

Les méthodes RK(N) fonctionnent toutes selon le même principe de **prédiction-correction** mais avec plusieurs évaluations en différents points entre u_i et u_{i+1} , avec N le nombre d'évaluations nécessaires.

N peut être très élevé, pour un coût de calcul de plus en plus important. En pratique la méthode la plus utilisée est **RK4** (car la plus polyvalente)

$$k_1 = f(t_i, u_i)$$

$$k_2 = f\left(t_i + \frac{h}{2}, u_i + k_1 \frac{h}{2}\right)$$

$$k_3 = f\left(t_i + \frac{h}{2}, u_i + k_2 \frac{h}{2}\right)$$

$$k_4 = f(t_i + h, u_i + k_3 h)$$

$$u_{i+1} = (k_1 + 2k_2 + 2k_3 + k_4) \frac{h}{6}$$

Méthode de Velocity-Verlet

Algorithme spécifiquement
conçu pour l'intégration des
équations du mouvement :

⇒

$$\begin{aligned}\frac{d}{dt}\vec{r} &= \vec{v} \\ \frac{d}{dt}\vec{v} &= \frac{\vec{F}(t, \vec{r}(t), \vec{v}(t))}{m}\end{aligned}$$

Développement de Taylor
associé :

⇒

$$\begin{aligned}\vec{r}(t+h) &= \vec{r}(t) + \vec{v}(t)h + \frac{h^2}{2}\vec{a}(t) \\ \vec{v}(t+h) &= \vec{v}(t) + \vec{a}(t)h + \frac{h^2}{2}\frac{d\vec{a}}{dt}\end{aligned}$$

Différences finies :

⇒

$$\frac{d\vec{a}}{dt} = \frac{\vec{a}(t+h) - \vec{a}}{h}$$

Méthode de Velocity-Verlet

⇒

$$\begin{aligned}\vec{r}(t+h) &= \vec{r}(t) + \vec{v}(t)h + \frac{h^2}{2}\vec{a}(t) \\ \vec{v}(t+h) &= \vec{v}(t) + \frac{\vec{a}(t) + \vec{a}(t+h)}{2}h\end{aligned}$$

$$\begin{aligned}\vec{r}(t+h) &= \vec{r}(t) + \vec{v}(t)h + \frac{h^2}{2}\vec{a}(t) \\ \vec{v}(t+h) &= \vec{v}(t) + \frac{\vec{a}(t) + \vec{a}(t+h)}{2}h\end{aligned}$$

Cet intégrateur est d'ordre 2 sur les vitesses et d'ordre 3 sur les positions.

Il est *simple* à programmer.

C'est un intégrateur **symplectique** : l'évolution des erreurs est périodique \Rightarrow il conserve l'énergie à long terme, **pas de dérive**.

Erreurs des méthodes précédentes

Méthode	Ordre	Erreur locale	Erreur globale	Symplectique
Euler explicite	1	$\propto h^2$	$\propto h$	non
Point milieu	2	$\propto h^3$	$\propto h^2$	non
Runge-Kutta 2	2	$\propto h^3$	$\propto h^2$	non
Runge-Kutta 3	3	$\propto h^4$	$\propto h^3$	non
Runge-kutta 4	4	$\propto h^5$	$\propto h^4$	non
Velocity-Verlet	2+	$\propto h^3+$	$\propto h^2+$	oui

TABLE – Table des erreurs de troncature

L'erreur **d'arrondi** locale est indépendante de $h \rightarrow$ erreur d'arrondi globale $\propto \frac{1}{h}$