UPMC **Master P&A/SDUEE**

UE MU4PY209

Méthodes Numériques et Calcul Scientifique

Résolution numérique des équations différentielles ordinaires (EDO)

2019–2020

Jacques.Lefrere@upmc.fr

Table des matières

1 Introduction

1.1 Problème différentiel

— équation différentielle scalaire d'ordre n

$$\boxed{\frac{\mathrm{d}^n y}{\mathrm{d}t^n} = f\left(t,\ y,\ \frac{\mathrm{d}y}{\mathrm{d}t},\ \ldots,\ \frac{\mathrm{d}^{n-1} y}{\mathrm{d}t^{n-1}}\right)}$$

où f est la fonction second membre donnée

$\Rightarrow$ **famille** de solutions $y(t)$ à n paramètres

— ensemble de n conditions imposées

$\Rightarrow$ choix d'**une** solution dans la famille

1.2 Deux types de problèmes différentiels à résoudre

— Conditions initiales données pour une seule valeur t_0 de t, par exemple

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

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

— Conditions données pour des valeurs distinctes de la variable indépendante t, par exemple :

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

Problème de **conditions aux limites** (non traité, sauf problème de tir).

1.3 Équations différentielles scalaires du 1er ordre

Étudier d'abord les équations différentielles scalaires **du premier ordre**.
$\Rightarrow$ **famille de solutions** $y(t)$ à **un** paramètre (y_0)

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t,\ y(t)) \quad \text{avec} \quad y(t_0) = y_0$$ condition initiale

Les EDO d'ordre supérieur se ramènent à des systèmes différentiels couplés du premier ordre (EDO vectorielles du premier ordre).

1.4 Unicité et problème bien posé : conditions suffisantes

La **condition de Lipschitz**

$$|f(t,y_2) - f(t,y_1)| \leqslant K\ |y_2 - y_1|$$

assure l'**unicité** de la solution.

$\left|\frac{\partial f}{\partial y}(t,y)\right| \leqslant K$ dans un domaine convexe $\Rightarrow$ condition de Lipschitz vérifiée.

Les erreurs d'arrondi amènent à **toujours résoudre un problème perturbé**.

Problème bien posé si : le problème faiblement perturbé (second membre ou condition initiale) possède une solution proche de celle du problème original.
La condition de Lipschitz assure que le problème est bien posé.

1.5 Méthodes de résolution numérique et notations

Résolution numérique approchée sur l'intervalle $[t_0, t_0 + L]$ de longueur L

Discrétisation par découpage de l'intervalle de longueur L selon un pas constant h

Échantillonnage de la solution aux instants $t_i = t_0 + ih$ pour $1 \leqslant i \leqslant n$.
Solution numérique : $u_i =$ approximation de $y(t_i)$

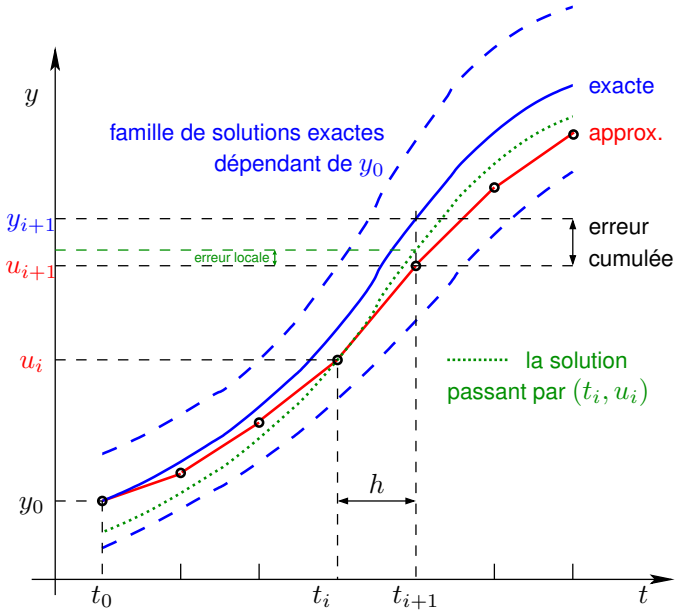
À partir de la condition initiale $u_0 = y(t_0)$ imposée,
faire une **boucle** sur les abscisses t_i pour calculer l'approximation u_{i+1} à t_{i+1}
$\rightarrow$ approximer ainsi de proche en proche la solution sur l'intervalle L.

$\Rightarrow$ accumulation des erreurs dans la boucle

À chaque pas de la boucle, pour calculer u_{i+1}, on peut s'appuyer :

— sur **la dernière valeur** calculée u_i : **méthodes à un pas**

— sur **plusieurs valeurs** $u_{i-k} (k \geqslant 0)$ antérieurement calculées :
méthodes à plusieurs pas (initialisation nécessaire par méthode à un pas)



Méthode à pas constant

Découpage de l'intervalle de longueur L selon un pas fixe $h = L/n$.

$u_i =$ approximat. de $y(t_i)$

Un pas :
$t_i \rightarrow t_{i+1}$
$u_i \rightarrow u_{i+1}$

2 Méthodes à un pas

Constituent l'algorithme de base qui permet d'estimer la valeur de la solution à l'instant $t_{i+1} = t_i + h$, connaissant seulement u_i, celle à t_i.

La valeur à estimer peut être approchée par un développement limité de Taylor :

$$y(t_i + h) = y(t_i) + h\frac{\mathrm{d}y}{\mathrm{d}t}(t_i) + \frac{h^2}{2}\frac{\mathrm{d}^2y}{\mathrm{d}t^2}(t_i) + \cdots \qquad (1)$$

Ordre n de la méthode = plus grande puissance de h prise en compte dans l'approximation.

— Somme des termes négligés = **erreur de troncature locale** $\propto h^{n+1}$ déterministe, augmente si le pas h augmente et si l'ordre de la méthode diminue

— Précision finie des opérations sur les réels $\Rightarrow$ **erreur d'arrondi** aléatoire augmente lorsque les calculs se compliquent, en particulier si le pas h diminue.

Indépendamment du coût (en temps de calcul) des opérations, et des cas où la fonction est tabulée, **ne pas croire que diminuer le pas améliore toujours** la qualité du résultat : un **compromis** doit être trouvé entre ces deux types d'erreurs.

2.1 Méthodes du premier ordre

2.1.1 Méthode d'Euler progressive (explicite)

Méthode du premier ordre d'intérêt pédagogique, à éviter en pratique

$$\boxed{u_{i+1} = u_i + hf(t_i, u_i)} \qquad (2)$$

Exemple : stabilité

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -\frac{y}{\tau} \quad \Rightarrow \quad \text{solution analytique} \quad y = y_0 e^{-t/\tau} \Rightarrow y_n = y_0(e^{-h/\tau})^n$$

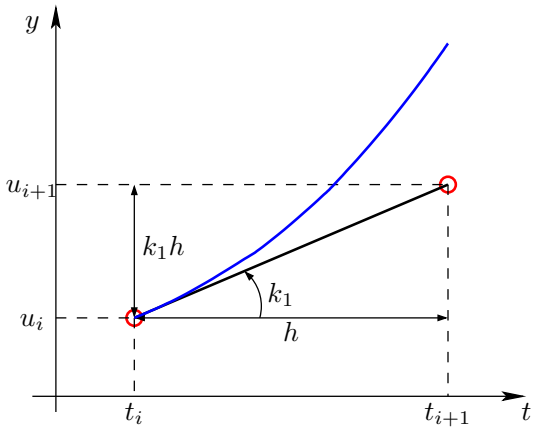
$$u_{i+1} = u_i - \frac{h}{\tau}u_i \quad \Rightarrow \quad \text{solution numérique} \quad u_n = y_0\,(1 - h/\tau)^n$$

Si $\tau > 0$, la solution exacte vérifie $y(\infty) = 0$,

Mais pour l'approximation, $u_n \to 0 \iff |1 - h/\tau| < 1 \iff 0 < h < 2\tau$.

Condition de **stabilité** : $h < 2\tau$ (pas h petit)

Mais, si $h > \tau$, alors $(1 - h/\tau) < 0$: alternance de signe de la solution u_n.



Méthode d'Euler

Méthode **explicite** qui ne nécessite qu'une seule évaluation de la fonction second membre f par pas :

$k_1 = f(t_i, u_i)$

facilement **instable**

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

voir dérivée avant

2.1.2 Méthode d'Euler rétrograde (implicite)

$$\boxed{u_{i+1} = u_i + hf(t_{i+1}, \boldsymbol{u_{i+1}})} \qquad (3)$$

Méthode **implicite** : résolution itérative, plus difficile à mettre en œuvre, sauf si la forme de $f(t, u)$ permet le calcul analytique de u_{i+1} à partir de l'équation (3).

Avantage : meilleure **stabilité** que la méthode progressive explicite.

Exemple : stabilité

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -\frac{y}{\tau} \quad \Rightarrow \quad \text{solution analytique} \quad y = y_0 e^{-t/\tau} \Rightarrow y_n = y_0(e^{-h/\tau})^n$$

$$u_{i+1} = u_i - \frac{h}{\tau}u_{i+1} \quad \Rightarrow \quad \text{solution numérique} \quad u_{i+1} = \frac{u_i}{1 + h/\tau}$$

$$u_n = \frac{y_0}{(1 + h/\tau)^n}$$

Si $\tau > 0,\ y(\infty) = 0$, et aussi $u_n \to 0 \quad \forall \tau > 0,\ \forall h > 0$ solution **stable**

Mise en œuvre de la méthode d'Euler rétrograde : résolution de l'équation implicite par itération

$$\boxed{u_{i+1} = u_i + hf(t_{i+1}, \boldsymbol{u_{i+1}})}$$

Itérer l'application g pour rechercher son **point fixe**

$$v_2' = g(v_2) = u_i + hf(t_2, v_2)$$

Ce point fixe est la solution de l'équation implicite.

— Utilise plusieurs évaluations du second membre, sans calcul de ses dérivées.

— Très peu d'itérations nécessaires

Initialisation par le prédicteur avec Euler progressif

$$\begin{aligned} t_2 &= t_i + h \\ k_1 &= f(t_i, u_i) \\ v_2 &= u_i + hk_1 \end{aligned}$$

Boucle pour recherche du point fixe de $g(v_2) = v_2' = u_i + hf(t_2, v_2)$

$$\begin{aligned} k_2 &= f(t_2, v_2) \\ v_2' &= u_i + hk_2 \\ \delta v_2 &= v_2' - v_2 \\ &\text{arrêt si} \quad |\delta v_2|^2 \leqslant \alpha^2 |v_2|^2 \\ v_2 &= v_2' \end{aligned}$$

La fonction g est contractante si $g'(v_2) = h\left|\frac{\partial f}{\partial v_2}\right| \leqslant 1$
ce qui est vérifié si le pas est assez faible.
Rappel : la condition de Lipschitz est $\left|\frac{\partial f}{\partial v_2}\right| \leqslant K$

Critère d'arrêt : choisir α faible, mais $\alpha > \varepsilon$.

2.2 Méthodes du deuxième ordre

Première idée : augmenter le nombre de termes du développement de Taylor :
rarement utilisé, car nécessite l'évaluation des dérivées partielles de f.

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t,\ y(t)) \qquad \Rightarrow \qquad \frac{\mathrm{d}^2y}{\mathrm{d}t^2} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} = \frac{\partial f}{\partial t} + f\frac{\partial f}{\partial y} \qquad (6)$$

Préférer utiliser **plusieurs évaluations du second membre** f en des points adaptés.

Centrer l'évaluation de la dérivée au point milieu $t_m = (t_i + t_{i+1})/2$.

$$y(t_i + h) = y(t_m) + \frac{h}{2}\frac{\mathrm{d}y}{\mathrm{d}t}(t_m) + \frac{1}{2}\frac{h^2}{4}\frac{\mathrm{d}^2y}{\mathrm{d}t^2}(t_m) + O(h^3) \qquad (7a)$$

$$y(t_i) = y(t_m) - \frac{h}{2}\frac{\mathrm{d}y}{\mathrm{d}t}(t_m) + \frac{1}{2}\frac{h^2}{4}\frac{\mathrm{d}^2y}{\mathrm{d}t^2}(t_m) + O(h^3) \qquad (7b)$$

Par différence, (approximation locale parabolique, voir aussi dérivée centrée à 2 termes)

$$y(t_i + h) - y(t_i) = h\frac{\mathrm{d}y}{\mathrm{d}t}(t_m) + O(h^3)$$

2.2.1 Méthode du point milieu

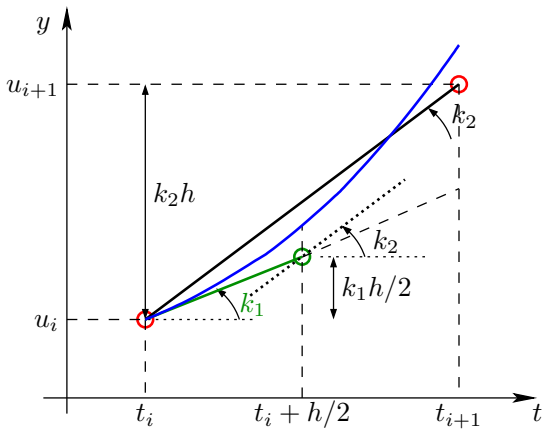
Nécessite l'évaluation du second membre f en 2 points :
en (t_i, u_i) et **au milieu** $(t_{i+1/2} = t_i + h/2,\ u_{i+1/2})$ d'un pas (hors grille).

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

$$k_1 = f(t_i,\ u_i) \qquad (8a)$$

($u_{i+1/2}$ calculé via Euler) $\qquad k_2 = f(t_i + \frac{h}{2},\ u_i + k_1\frac{h}{2}) \qquad (8b)$

$$u_{i+1} = u_i + hk_2 \qquad (8c)$$



Méthode du point milieu

Méthode **explicite** qui nécessite deux évaluations du second membre par pas dont une hors grille.

2.2.2 Méthode d'Euler modifiée

En appliquant 7a et 7b à la dérivée et en faisant la somme, on peut remplacer la dérivée au milieu par la moyenne des dérivées aux extrémités de l'intervalle (voir méthode de quadrature dite des trapèzes) :

$$\frac{\mathrm{d}y}{\mathrm{d}t}(t_i) + \frac{\mathrm{d}y}{\mathrm{d}t}(t_{i+1}) = 2\frac{\mathrm{d}y}{\mathrm{d}t}(t_m) + O(h^2)$$

D'où une approximation n'utilisant pas la valeur de f au point milieu t_m :

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

De nouveau, méthode a priori **implicite**, plus stable, mais plus lourde.

$\Rightarrow$ Contournement du problème en utilisant l'approximation d'Euler explicite (voir 2) pour évaluer u_{i+1} intervenant dans f.

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

Bilan : méthode de type **prédicteur-correcteur** équivalent à

— un demi-pas avec la pente initiale k_1

— et un demi-pas avec la pente k_2 du point prédit par Euler progressif.

$$k_1 = f(t_i,\ u_i) \tag{9a}$$

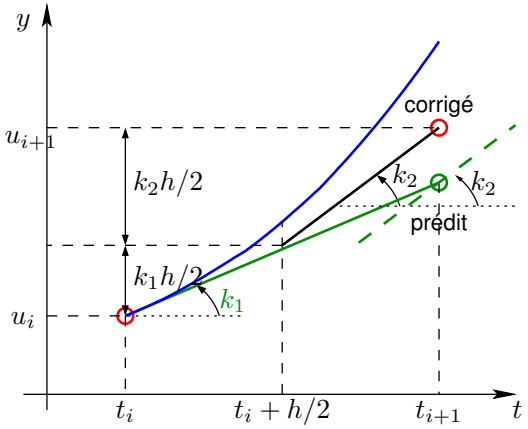
$$k_2 = f(t_{i+1},\ u_i + k_1 h) \tag{9b}$$

$$u_{i+1} = u_i + \frac{h}{2}\left[k_1 + k_2\right] \tag{9c}$$

Remarques

— deuxième ordre comme point milieu mais sans évaluation hors grille

— la résolution de l'équation implicite peut se faire en itérant la correction jusqu'à ce qu'elle devienne négligeable.



Méthode d'Euler modifiée

Méthode **explicite** qui nécessite deux évaluations de la fonction par pas en des points de la grille.

2.2.3 Méthode de Heun

$$k_1 = f(t_i,\ u_i) \tag{10a}$$
$$k_2 = f(t_i + \frac{2}{3}h,\ u_i + \frac{2}{3}k_1 h) \tag{10b}$$
$$u_{i+1} = u_i + \frac{h}{4}\left[k_1 + 3k_2\right] \tag{10c}$$

2.3 Méthodes de Runge Kutta

Plus généralement, avec r évaluations de f, on peut atteindre une méthode d'ordre r si $r \leqslant 4$. Pour atteindre l'ordre 5, six évaluations sont nécessaires.

$\Longrightarrow$ la méthode de Runge Kutta d'ordre 4 est très utilisée.

Les méthodes de Runge-Kutta sont **stables**.

2.3.1 Méthode de Runge Kutta d'ordre 3

$$k_1 = f(t_i, u_i) \tag{11a}$$
$$k_2 = f(t_i + \frac{h}{2},\ u_i + k_1\frac{h}{2}) \tag{11b}$$
$$k_3 = f(t_i + h,\ u_i + (2k_2 - k_1)h) \tag{11c}$$
$$u_{i+1} = u_i + (k_1 + 4k_2 + k_3)\frac{h}{6} \tag{11d}$$

2.3.2 Méthode de Runge Kutta d'ordre 4

$$k_1 = f(t_i,\ u_i) \tag{12a}$$
$$k_2 = f(t_i + \frac{h}{2},\ u_i + k_1\frac{h}{2}) \tag{12b}$$
$$k_3 = f(t_i + \frac{h}{2},\ u_i + k_2\frac{h}{2}) \tag{12c}$$
$$k_4 = f(t_i + h,\ u_i + k_3 h) \tag{12d}$$
$$u_{i+1} = u_i + (k_1 + 2k_2 + 2k_3 + k_4)\frac{h}{6} \tag{12e}$$

2.4 Erreur absolue en fonction du pas et de l'ordre

nombre de pas $= L/h$ $\Longrightarrow$ erreur globale $\sim$ erreur locale $\times\ L/h$

TABLE 1 – Erreur de **troncature seule**

Méthode	ordre	erreur locale	erreur globale
Euler explicite	1	$\propto h^2$	$\propto h$
Point milieu – Euler modifiée	2	$\propto h^3$	$\propto h^2$
Runge-Kutta 3	3	$\propto h^4$	$\propto h^3$
Runge-Kutta 4	4	$\propto h^5$	$\propto h^4$

Erreur d'**arrondi** locale indépendante de $h \Rightarrow$ erreur d'arrondi globale $\propto 1/h$

2.5 Exemple de l'équation logistique $\frac{\mathrm{d}y}{\mathrm{d}t} = y(1 - y/2)$

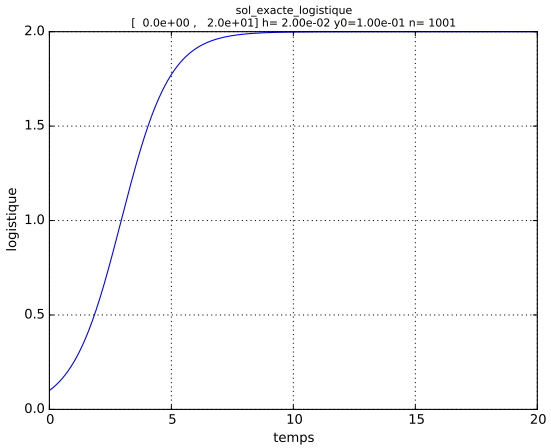


FIGURE 1 – Solution analytique de l'équation logistique pour $t_0 = 0$, $y(t_0) = 0.1$, $a = 1$ et $k = 2$.

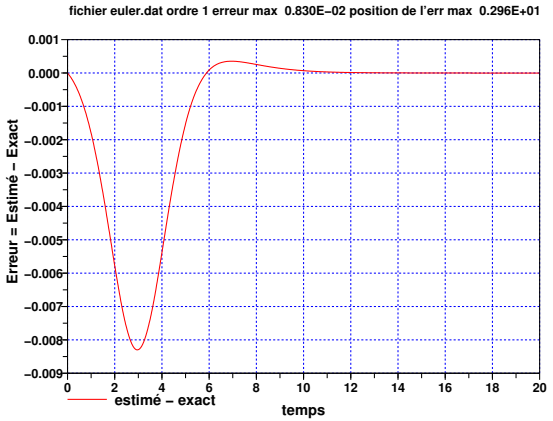


FIGURE 2 – Erreur dans l'intégration de l'équation logistique avec la méthode d'Euler pour $h = 0,02$. L'allure régulière montre que l'**erreur de troncature** domine.
Erreur de troncature locale liée à la courbure de la solution

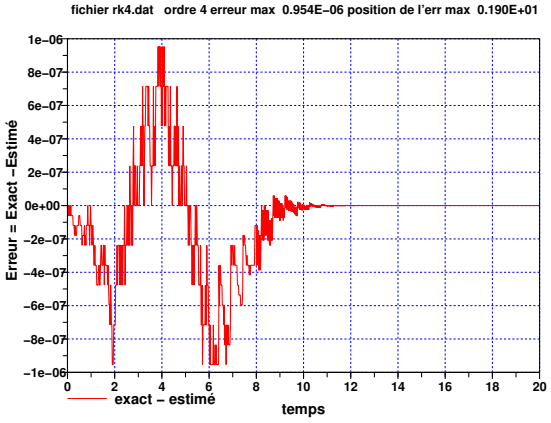
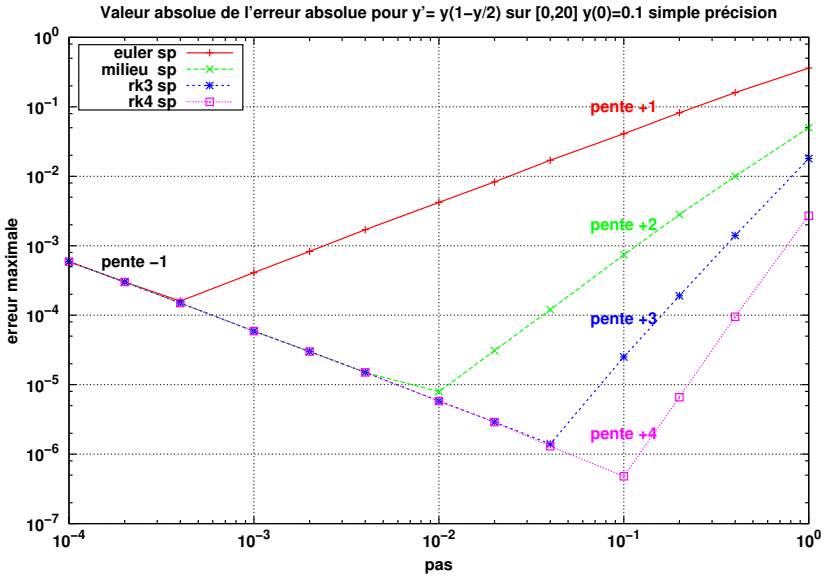
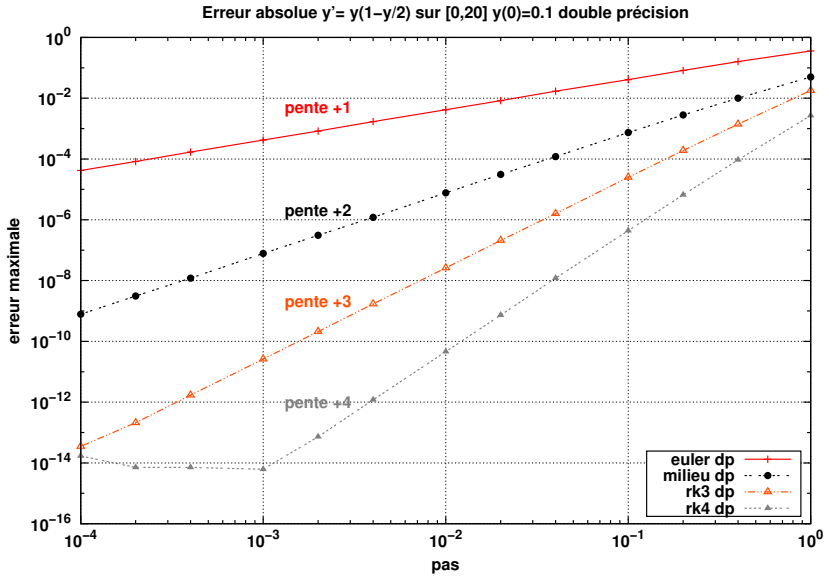


FIGURE 3 – Erreur dans l'intégration de l'équation logistique avec Runge Kutta d'ordre 4 pour $h = 0,02$. L'allure **bruitée** est caractéristique de l'**erreur d'arrondi** et on retrouve les niveaux de quantification des réels sur 32 bits.

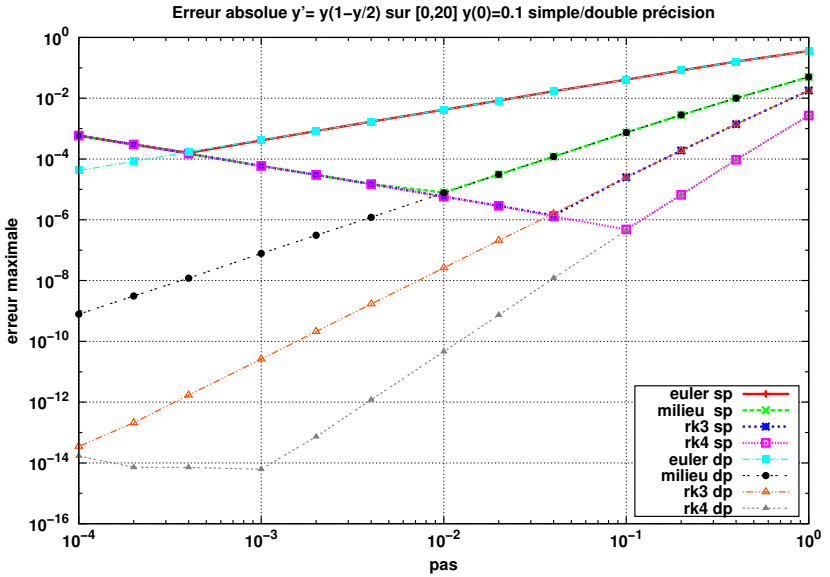
2.5.1 Exemple d'erreur totale maximale en simple précision (32 bits)



2.5.2 Exemple d'erreur totale maximale en double précision (64 bits)



2.5.3 Comparaison des erreurs maximales simple/double précision



3 Méthodes à plusieurs pas

3.1 Méthodes d'Adams

Principe : les erreurs augmentent avec l'intégration, les points les plus proches de la valeur initiale ont tendance à être plus fiables. Pour calculer u_{i+1}, on peut s'appuyer non seulement sur la dernière valeur estimée u_i, mais sur les m précédentes.

— si le calcul invoque la pente au point recherché $f(t_{i+1}, u_{i+1})$, la méthode est **implicite** (voir 2.1.2) : ADAMS-MOULTON

— sinon elle est **explicite** : ADAMS-BASHFORTH .

Dans les deux cas, il faut **initialiser** le calcul par une méthode à un pas sur les m premiers points.

Le calcul réutilise les évaluations antérieures du second membre.

$\Longrightarrow$ stocker ces valeurs pour économiser les calculs

3.1.1 Adams Bashforth : explicite, pas de terme en $f(t_{i+1}, u_{i+1})$

Adams-Bashforth utilise m points à gauche de t_{i+1} :

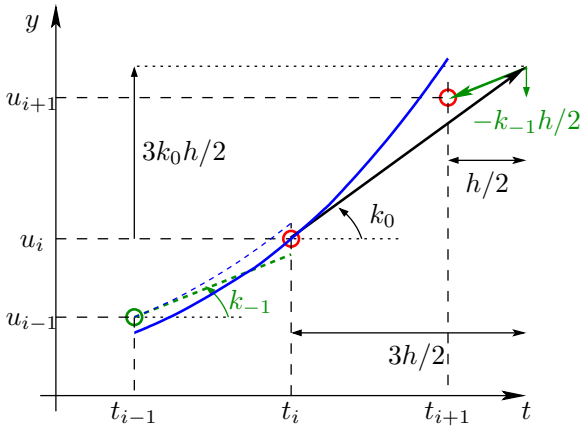
$$u_{i+1} = u_i + \beta h \sum_{j=1}^{m} \alpha_j f(t_{i-j+1}, u_{i-j+1}) \quad \text{avec} \quad \beta \sum_{j=1}^{m} \alpha_j = 1$$

méthode d'ordre m, mais une seule nouvelle évaluation du second membre à chaque pas

m	β	α_1	α_2	α_3	α_4
1	1	1			
2	1/2	3	−1		
3	1/12	23	−16	5	
4	1/24	55	−59	37	−9

Exemple : Adams-Bashforth à deux pas

$$u_{i+1} = u_i + \frac{h}{2}\left[3f(t_i, u_i) - f(t_{i-1}, u_{i-1})\right]$$



Adams-Bashforth à deux pas

Méthode **explicite d'ordre deux** mais seulement une évaluation nouvelle du second membre à chaque pas

$\Rightarrow$ mémoriser les seconds membres.

3.1.2 Adams Moulton : implicite, terme en $f(t_{i+1}, u_{i+1})$

Adams-Moulton utilise $m+1$ points dont t_{i+1} donc **implicite** :

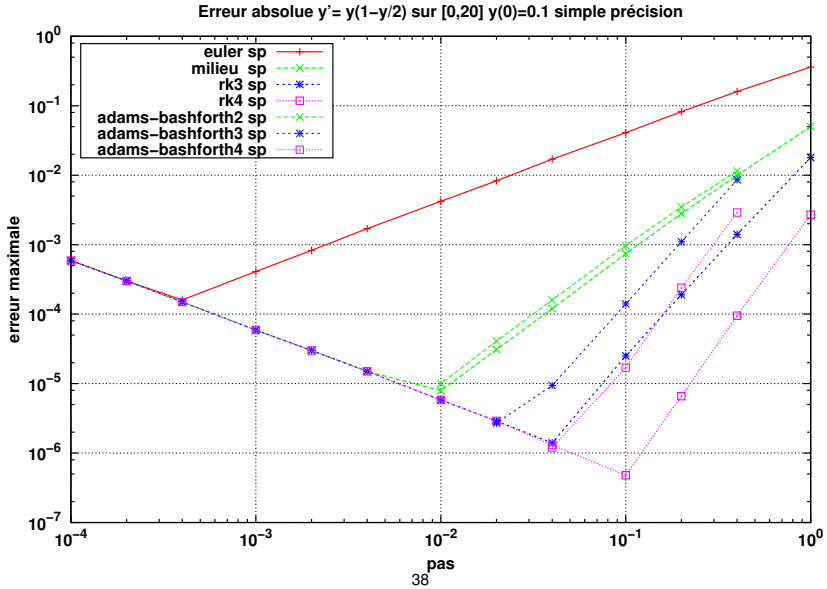
$$u_{i+1} = u_i + \beta h \sum_{j=0}^{m} \alpha_j f(t_{i-j+1}, u_{i-j+1}) \quad \text{avec} \quad \beta \sum_{j=0}^{m} \alpha_j = 1$$

méthode d'ordre $m+1$ (nombre d'évaluations du second membre)

m	β	α_0	α_1	α_2	α_3
0	1	1			
1	1/2	1	1		
2	1/12	5	8	−1	
3	1/24	9	19	−5	1

Éviter les difficultés de l'implicite en utilisant un prédicteur explicite de u_{i+1}, injecté ensuite dans l'expression d'Adams-Moulton vue comme correcteur.

3.1.3 Comparaison méthodes à un pas et Adams explicite



3.1.4 Méthodes de prédicteur correcteur

Principe : bénéficier des qualités d'une méthode implicite mais l'appliquer à une estimation obtenue par une méthode explicite du même ordre (voir Euler modifiée).

— **prédiction** de u_{i+1} par une méthode **explicite**

— **correction** de u_{i+1} par une formule **implicite** où $f(t_{i+1}, y(t_{i+1}))$ a été approximé par la prédiction $f(t_{i+1}, u_{i+1})$.

Exemple : méthode d'Euler modifiée

Une itération de la partie correction est possible.

L'ordre est celui du correcteur, mais la stabilité dépend plus du prédicteur.

Ces méthodes permettent d'**estimer l'erreur de troncature** à partir de la différence entre prédicteur et correcteur $\Longrightarrow$ adaptation du pas

3.2 Méthodes adaptatives

Principe : ajuster le pas localement pour obtenir une précision imposée.

Estimer l'erreur de troncature par l'écart entre deux solutions numériques :

u_i, d'ordre n $$\eta_i = \frac{y(t_{i+1}) - u_{i+1}}{h} \propto h^n \quad (13a)$$

$u_i^\star$, d'ordre $n+1$ $$\eta_i^\star = \frac{y(t_{i+1}) - u_{i+1}^\star}{h} \propto h^{n+1} \quad (13b)$$

or $h \ll 1$ donc $$\eta_i \approx \frac{u_{i+1}^\star - u_{i+1}}{h} \propto h^n \quad (13c)$$

Modification du pas d'un facteur q

$$\eta_i(hq) \approx q^n \eta_i(h) \approx \frac{q^n}{h}(u_{i+1}^\star - u_{i+1}) \quad (14)$$

Pour obtenir une **précision** globale $\Delta y \approx L\delta$, imposer localement :

$$|\eta_i(hq)| \approx \delta \quad (15)$$

d'où le facteur q à appliquer au pas :

$$q \approx \left(\frac{h\delta}{u_{i+1}^\star - u_{i+1}}\right)^{1/n} \quad (16)$$

— Si $q < 1$, refuser u_{i+1} et diminuer le pas

— Si $q > 1$, accepter u_{i+1} et augmenter le prochain pas

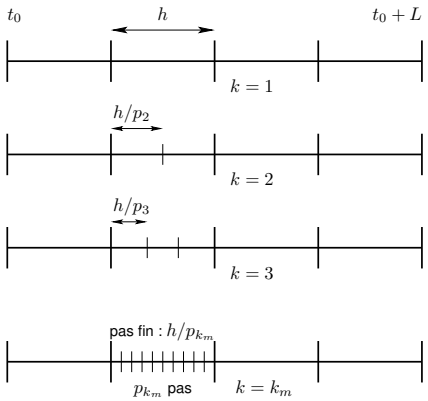
⚠ En pratique, limiter le pas à un intervalle raisonnable et éviter des variations brutales.
On suppose ici que le pas choisi permet de négliger l'erreur d'arrondi.

3.2.1 Exemple : méthode de Runge Kutta Fehlberg

Une méthode de Runge Kutta d'**ordre 5** (6 évaluations de f)
permet de contrôler la précision obtenue par un Runge Kutta d'**ordre 4**
utilisant les évaluations de f aux **mêmes points** que celle d'ordre 5
(les poids ne sont pas ceux de la méthode d'ordre 4 classique).

3.3 Méthodes d'extrapolation de Gragg

3.3.1 Principe de l'extrapolation



Subdivisions successives

Découpage de l'intervalle de **pas grossier** h en sous-intervalles de **pas fin** $h_k = h/p_k$ de plus en plus petits.
Développement polynomial de l'erreur de troncature en fonction du pas pour extrapoler au pas nul ($h_k \to 0$).

Exemple de la méthode d'Euler : l'erreur de troncature **globale sur un pas grossier** h est du premier ordre en fonction du **pas fin** h/p_k.
Par exemple, pour les subdivisions p_1 et p_2 :

$$y(t+h) = v_1 + a_1\left(\frac{h}{p_1}\right) + a_2\left(\frac{h}{p_1}\right)^2 + \cdots \quad (17)$$

$$y(t+h) = v_2 + a_1\left(\frac{h}{p_2}\right) + a_2\left(\frac{h}{p_2}\right)^2 + \cdots \quad (18)$$

Combinaison linéaire de ces deux estimateurs v_1 et v_2
$\Rightarrow$ **éliminer le terme d'ordre 1** de l'erreur (a_1 inconnu)
$\Rightarrow$ nouvel estimateur $w_{2,2}$ **d'ordre 2** tel que :

$$y(t+h) = w_{2,2} + b_2\left(\frac{h}{p_2}\right)^2 + \cdots$$

$$w_{2,2} = \frac{(p_2/p_1)\,w_{2,1} - w_{1,1}}{p_2/p_1 - 1} = w_{2,1} + \frac{w_{2,1} - w_{1,1}}{p_2/p_1 - 1} \quad (19)$$

Autre exemple avec la méthode du point milieu :

— l'erreur de troncature globale sur h est d'ordre 2 en fonction du pas h/p_k ;
— pas de termes d'ordre impair dans le développement de l'erreur.

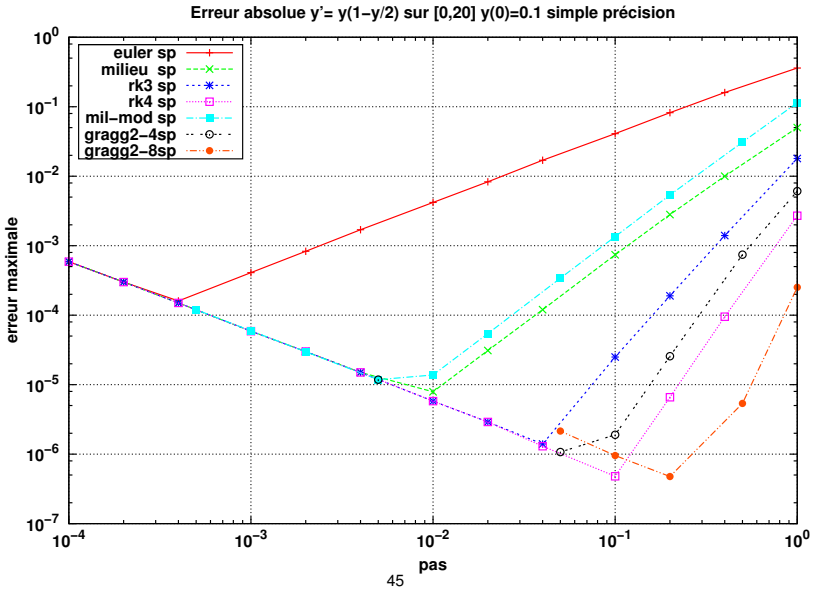
Combinaison linéaire de deux estimateurs avec des pas fins différents :
$\Rightarrow$ élimination du terme d'ordre 2 de l'erreur de troncature
$\Rightarrow$ erreur de troncature d'ordre 4

Itérer le processus avec une suite de subdivisions et de combinaisons linéaires d'estimateurs pour augmenter l'ordre de l'erreur de troncature.

⚠ **Mais** amélioration limitée par l'erreur d'**arrondi**...

L'écart $w_{k+1,k+1} - w_{k,k}$ donne une estimation de l'erreur de troncature si on retient la solution $w_{k,k}$. En ajustant, pour chaque intervalle de largeur h, le nombre k_m de subdivisions pour respecter une erreur absolue imposée, on obtient une **version adaptative de la méthode de Gragg**.

3.3.2 Comparaison méthodes à un pas et extrapolation de Gragg



4 Les EDO du premier ordre en pratique

4.1 Échelles de temps et problèmes raides

Ne pas oublier que chaque problème différentiel possède une ou plusieurs **échelles de temps propres** (périodes ou pseudo-périodes, constantes de temps).
La solution ne peut être représentée correctement qu'avec un pas assez inférieur au plus petit de ces temps propres.
Cette analyse impose donc une valeur maximale pour le pas.

Certains problèmes différentiels qualifiés de **raides** comportent des échelles de temps très différentes : leur intégration numérique s'avère délicate et coûteuse (pas faible pour respecter le temps court, mais nombreux pour accéder au temps long).
Il existe des méthodes spécifiques des EDO raides qui ne sont pas présentées ici.

4.2 Validation des résultats

Validation via une solution analytique d'un problème simplifié

Lorsqu'une solution analytique est disponible (par exemple pour certaines valeurs de paramètres qui permettent de simplifier l'EDO), sa comparaison avec la solution numérique permet de tester la méthode. Le calcul de l'erreur dans le domaine où la troncature domine permet d'extrapoler l'effet d'un changement de pas connaissant l'ordre de la méthode.

Validation sans solution analytique

Dans le cas où aucune solution analytique de référence n'est disponible, la validation s'appuie sur les mêmes outils que les méthodes adaptatives :

— diminution du pas (division par 2)
— augmentation de l'ordre de la méthode
— extrapolation de Gragg
— calcul d'invariants (énergie par exemple)

4.3 Structure des programmes de résolution d'EDO du 1er ordre

1. un algorithme de base (appliquant une méthode d'ordre 1, 2, 3 ou 4 à la fonction second membre f passée en argument) permettant d'**avancer d'un pas** dans l'intégration de l'équation différentielle
2. éventuellement une procédure qui choisit le pas le plus grand possible compatible avec la précision attendue et contrôle la progression de l'intégration (elle pourrait comporter un algorithme adaptatif)
3. un programme d'**interface avec l'utilisateur** qui choisit la méthode, le second membre, lit les paramètres (conditions initiales par ex.), déclenche et arrête **la boucle d'intégration** et stocke les résultats.
4. un module comportant **les fonctions seconds membres** de l'équation différentielle et les éventuelles solutions analytiques exactes ou approchées
5. un module d'**utilitaires** notamment pour écrire les résultats dans un fichier pour visualisation ultérieure.

5 Systèmes d'équations différentielles du 1er ordre

5.1 Extension des méthodes scalaires explicites aux vecteurs

Système de n équations différentielles couplées du premier ordre associées à n conditions initiales

considérer les **vecteurs** $\vec{\mathbf{y}}$ et $\vec{\mathbf{f}}$.

$$\begin{aligned}
\frac{\mathrm{d}y_1}{\mathrm{d}t} &= f_1(t,\ y_1, y_2, \ldots, y_n)\\
\frac{\mathrm{d}y_2}{\mathrm{d}t} &= f_2(t,\ y_1, y_2, \ldots, y_n)\\
\ldots &= \ldots\\
\frac{\mathrm{d}y_n}{\mathrm{d}t} &= f_n(t,\ y_1, y_2, \ldots, y_n)
\end{aligned}
\qquad
\begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{pmatrix}
\quad \text{et} \quad
\begin{pmatrix} f_1 \\ f_2 \\ \ldots \\ f_n \end{pmatrix}$$

Les **méthodes explicites** de résolution des équations différentielles scalaires du premier ordre **s'appliquent aux systèmes**.

$$\frac{\mathrm{d}\vec{\mathbf{y}}}{\mathrm{d}t} = \vec{\mathbf{f}}(t,\ \vec{\mathbf{y}})$$

À chaque étape, effectuer les calculs **sur chaque composante** avant de passer à l'étape suivante : exemple avec point milieu

Étape 1 : vecteur des pentes au bord gauche de l'intervalle

$$\vec{\mathbf{k_1}} = \vec{\mathbf{f}}(t_1,\ \vec{\mathbf{y_1}})$$

$$\begin{aligned}
k_{1,1} &= f_1(t_1,\ y_{1,1},\ y_{1,2},\ \ldots,\ y_{1,n})\\
k_{1,2} &= f_2(t_1,\ y_{1,1},\ y_{1,2},\ \ldots,\ y_{1,n})\\
\ldots &= \ldots\\
k_{1,n} &= f_n(t_1,\ y_{1,1},\ y_{1,2},\ \ldots,\ y_{1,n})
\end{aligned}$$

avant de calculer...

Étape 2 : vecteur des pentes au point milieu prédit

$$\vec{\mathbf{k_2}} = \vec{\mathbf{f}}(t_1 + h/2,\ \vec{\mathbf{y_1}} + \vec{\mathbf{k_1}}h/2)$$

$$\begin{aligned}
k_{2,1} &= f_1(t_1 + h/2,\ y_{1,1} + k_{1,1}h/2,\ y_{1,2} + k_{1,2}h/2,\ \ldots,\ y_{1,n} + k_{1,n}h/2)\\
k_{2,2} &= f_2(t_1 + h/2,\ y_{1,1} + k_{1,1}h/2,\ y_{1,2} + k_{1,2}h/2,\ \ldots,\ y_{1,n} + k_{1,n}h/2)\\
\ldots &= \ldots\\
k_{2,n} &= f_n(t_1 + h/2,\ y_{1,1} + k_{1,1}h/2,\ y_{1,2} + k_{1,2}h/2,\ \ldots,\ y_{1,n} + k_{1,n}h/2)
\end{aligned}$$

Étape 3 : vecteur résultat au bord droit de l'intervalle

$$\vec{\mathbf{u}}_{i+1} = \vec{\mathbf{u}_i} + h\ \vec{\mathbf{k_2}}$$

5.2 Exemple de système non-linéaire couplé du premier ordre : équations de Lotka-Volterra

Deux populations en conflit : modèle proies (y_1) – prédateurs (y_2)

$a_1 = 1/\tau_1$ = taux de croissance de y_1 (proies) en l'absence de y_2 (prédateurs)

$a_2 = 1/\tau_2$ = taux de décroissance de y_2 (prédateurs) en l'absence de y_1 (proies)

Termes de couplage non-linéaires en $y_1 y_2$ (rencontre des 2 espèces)

$\frac{a_1}{k_2} y_2$ = taux de destruction des proies par les prédateurs

$\frac{a_2}{k_1} y_1$ = taux de croissance des prédateurs au détriment des proies

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = +a_1 y_1 \left(1 - \frac{y_2}{k_2}\right) \qquad (20a)$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = -a_2 y_2 \left(1 - \frac{y_1}{k_1}\right) \qquad (20b)$$

Solutions **périodiques**

Lotka-Volterra : cycle dans le plan de phase

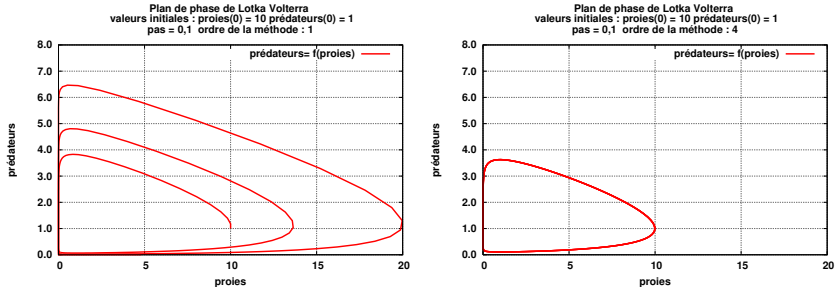
En éliminant le temps, on obtient un invariant, donc des solutions périodiques :

$$\frac{\mathrm{d}y_2}{\mathrm{d}y_1} = -\frac{a_2 y_2}{a_1 y_1}\,\frac{1 - y_1/k_1}{1 - y_2/k_2} \quad \Rightarrow \quad \boxed{y_1^{a_2} y_2^{a_1} \mathrm{e}^{-a_1 y_2/k_2 - a_2 y_1/k_1} = \mathrm{C^{te}}}$$

Tangentes horizontales pour $y_1 = k_1$ (ou $y_2 = 0$) : équilibre des prédateurs

Tangentes verticales pour $y_2 = k_2$ (ou $y_1 = 0$) : équilibre des proies

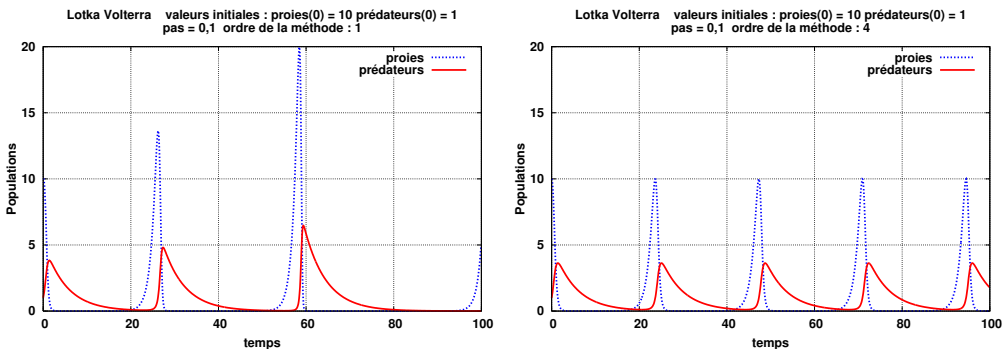
$$k_1 = k_2 = 1, a_1 = 1, a_2 = 0,2$$



Méthode d'Euler : non périodique — Runge Kutta d'ordre 4 : cycle correct

Résolution numérique de Lotka-Volterra : $k_1 = k_2 = 1, a_1 = 1, a_2 = 0,2, h = 0,1$

Échelles **linéaires**

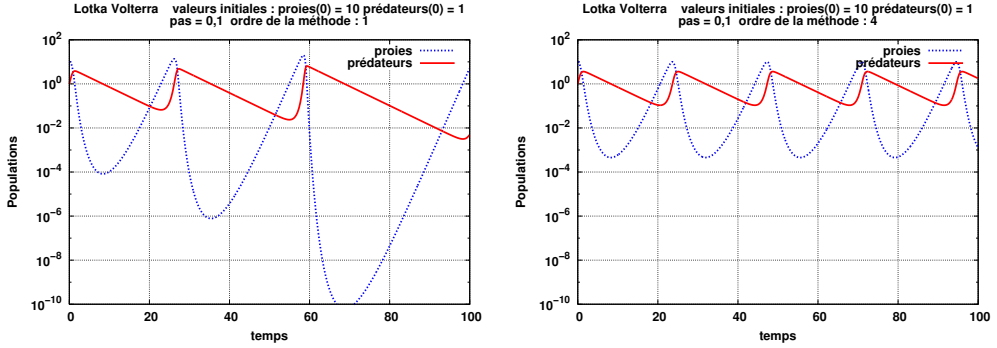


Méthode d'Euler progressive : **Les solutions divergent**

Méthode de Runge Kutta d'ordre 4 : **Cycle stable**

Résolution numérique de Lotka-Volterra : $k_1 = k_2 = 1, a_1 = 1$ et $a_2 = 0,2$.

Échelle **log** en ordonnée



Méthode d'Euler progressive : **divergente** — Méthode de Runge Kutta d'ordre 4 : **stable**

Pente avec peu de proies : $\frac{\mathrm{d}\ln y_2}{\mathrm{d}t} \approx -1/\tau_2$ d'où facteur $e^{-4} \approx 1/54$ sur une durée de $20 = 4\tau_2$.

Pente avec peu de prédateurs : $\frac{\mathrm{d}\ln y_1}{\mathrm{d}t} \approx 1/\tau_1$ d'où facteur 100 sur durée de $4,6 = 4,6\tau_1$

Changement de variables : passage au logarithme des populations

En normalisant les populations par leur valeurs à l'équilibre et en prenant le logarithme, on introduit les variables $z_1 = \ln(y_1/k_1)$ et $z_2 = \ln(y_2/k_2)$.
Le système différentiel prend alors la forme **séparable** :

$$\frac{\mathrm{d}z_1}{\mathrm{d}t} = \frac{1}{\tau_1}\left(1 - \exp(z_2)\right) \quad (21)$$

$$\frac{\mathrm{d}z_2}{\mathrm{d}t} = -\frac{1}{\tau_2}\left(1 - \exp(z_1)\right) \quad (22)$$

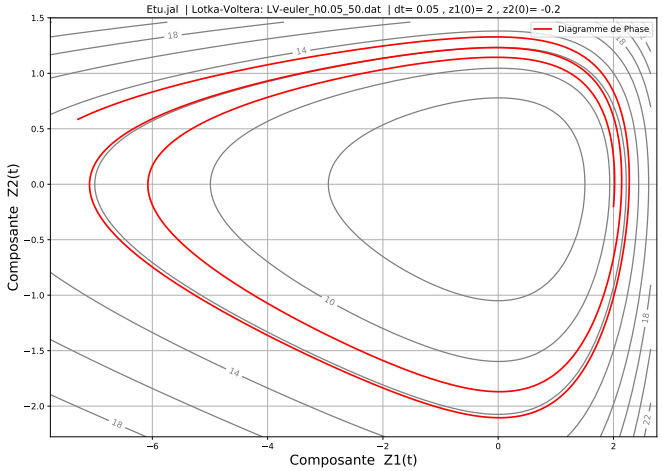
où la dérivée de z_1 ne dépend que de z_2 et réciproquement.

Le système transformé possède un invariant Λ

$$\Lambda(z_1, z_2) = \tau_1\left(\exp(z_1) - z_1\right) + \tau_2\left(\exp(z_2) - z_2\right) \quad (23)$$

qui ne dépend donc que des conditions initiales. Il est constant sur une orbite (trajectoire fermée) dans le plan de phase.

Exemple de résolution avec Euler progressive :



La trajectoire de la solution coupe des iso-Λ. Elle ne respecte pas l'invariant.

6 Équations différentielles d'ordre supérieur

$$\frac{\mathrm{d}^n y}{\mathrm{d}t^n} = f\left(t,\ y,\ \frac{\mathrm{d}y}{\mathrm{d}t},\ \ldots,\ \frac{\mathrm{d}^{n-1}y}{\mathrm{d}t^{n-1}}\right)$$

Une EDO scalaire d'ordre n se ramène à un système de n équations différentielles du premier ordre couplées en posant :

$$\begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{pmatrix} = \begin{pmatrix} y \\ y' \\ \ldots \\ y^{(n-1)} \end{pmatrix} \Longrightarrow \begin{pmatrix} y_1' \\ y_2' \\ \ldots \\ y_n' \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \\ \ldots \\ f\left(t, y_1, y_2, \ldots, y_n\right) \end{pmatrix}$$

6.1 Exemple

Système linéaire du second ordre avec excitation $h(t)$

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} = a\frac{\mathrm{d}y}{\mathrm{d}t} + by + h(t)$$

Poser

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y \\ y' \end{pmatrix} \Rightarrow \begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ ay_2 + by_1 + h(t) \end{pmatrix}$$

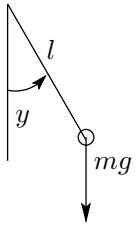
Condition initiale vectorielle : position $y(t_0)$ et vitesse $y'(t_0)$

Remarque Système différentiel d'ordre p de dimension n
$\Rightarrow$ système différentiel couplé du premier ordre à np dimensions.

6.2 Exemple d'EDO d'ordre 2 : le pendule

Pendule non linéaire (y = position angulaire)

$$\boxed{\frac{\mathrm{d}^2y}{\mathrm{d}t^2} = -k^2\sin(y)} \quad \text{où} \quad k^2 = g/l \qquad (24)$$



Pendule linéarisé (cas des petites amplitudes) : $\sin(y) \approx y$

$$\frac{\mathrm{d}^2y}{\mathrm{d}t^2} = -k^2y \qquad (25)$$

l'équation linéarisée admet une solution analytique en $A\cos(kt) + B\sin(kt)$.

Exprimer cette EDO non linéaire du second ordre sous la forme d'un système différentiel couplé de dimension 2 mais du premier ordre.

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y \\ y' \end{pmatrix} \implies \begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ -k^2\sin(y_1) \end{pmatrix}$$

Résolution système non-linéaire, avec le vecteur des valeurs initiales :

$$\begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} y(0) \\ \dfrac{\mathrm{d}y}{\mathrm{d}t}(0) = a \end{pmatrix} = \begin{pmatrix} \text{position angulaire} \\ \text{vitesse angulaire} \end{pmatrix}$$

Énergie mécanique conservée (après division par ml^2) :

$$\frac{1}{2}\left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)^2 + k^2(1-\cos y) = \text{constante}$$

Cas où $y(0) = 0$ (départ en position d'équilibre stable)

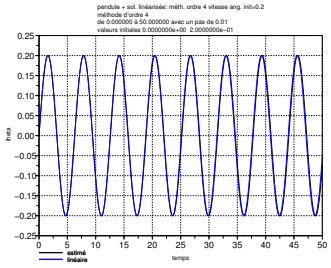
$$\frac{1}{2}\left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)^2 + k^2(1-\cos y) = \frac{1}{2}\left(\frac{\mathrm{d}y}{\mathrm{d}t}(0)\right)^2$$

Vitesse angulaire minimale pour $y = \pi$ (position d'équilibre instable si atteinte).

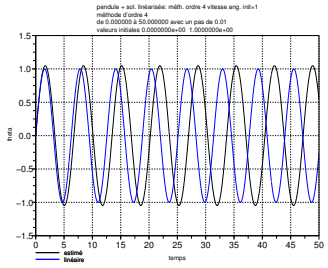
Si $a = \dfrac{\mathrm{d}y}{\mathrm{d}t}(0) > 2k$ (seuil) $\Rightarrow$ la vitesse angulaire ne s'annule pas (apériodique).

Étude de la transition périodique–apériodique selon a dans le cas où $k = 1$

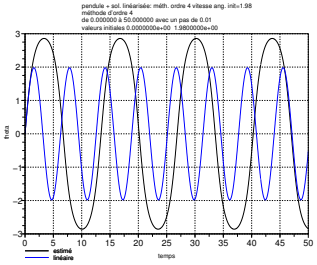
Comparaisons non-linéaire (Runge-Kutta 4)–analytique linéarisé : $y(t)$



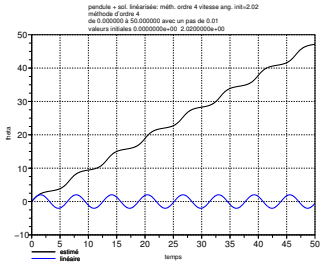
$a = 0.2 \ll 1$ linéarisable



$a = 1$ périodique non sinusoïdal

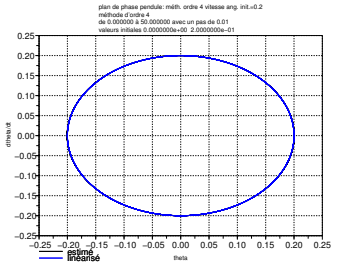


$a = 1.98$ périodique non sinusoïdal

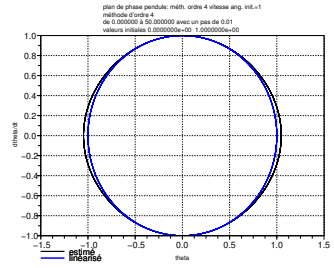


$a = 2.02$ apériodique

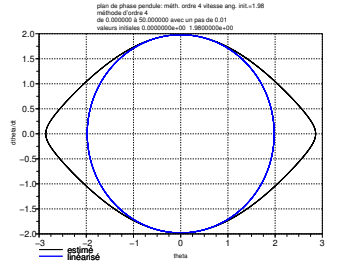
Comparaisons non-linéaire (RK 4)–analytique linéarisé : plan de phase $y'(y)$



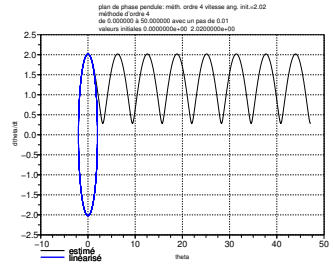
$a = 0.2 \ll 1$ linéarisable



$a = 1$ périodique non sinusoïdal

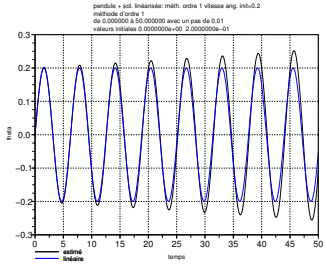


$a = 1.98$ périodique non sinusoïdal

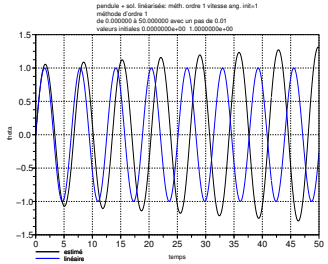


$a = 2.02$ apériodique

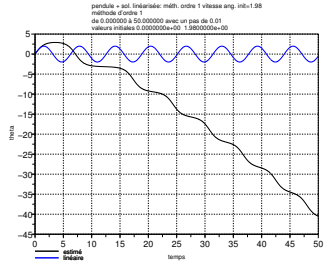
Comparaisons non-linéaire (Euler)–analytique linéarisé $y(t)$



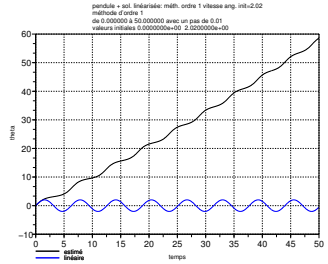
$a = 0.2$



$a = 1$

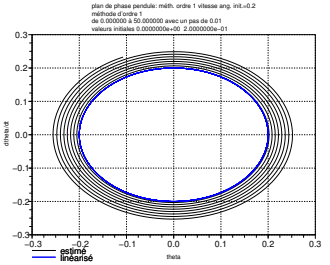


$a = 1.98$ apériodique selon Euler !

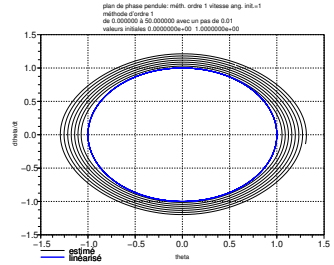


$a = 2.02$

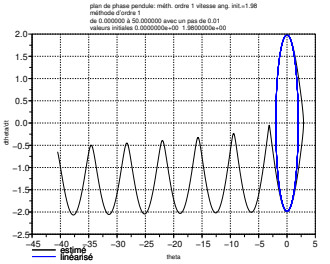
Comparaisons non-linéaire (Euler)–analytique linéarisé : plan de phase $y'(y)$



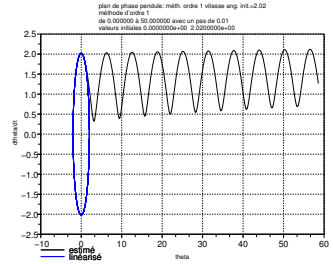
$a = 0.2$



$a = 1$



$a = 1.98$ apériodique selon Euler !



$a = 2.02$

Stabilité à long terme avec Euler progressive et rétrograde

Pendule linéarisé sans frottement représenté dans l'espace des phases : comportement à long terme d'un système non dissipatif

Même méthode sur les 2 composantes (position et vitesse) $h = 0.025$

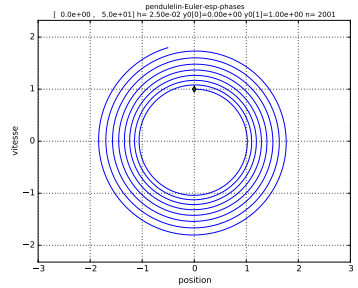


FIGURE 4 – Euler progressive : instable, amplification

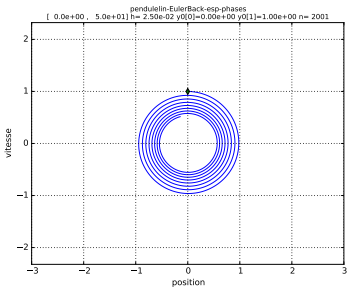


FIGURE 5 – Euler rétrograde : stable, contraction

Méthode d'Euler symplectique Méthodes progressives et méthodes rétrogrades présentent chacune des inconvénients antagonistes : dans le cas d'un système à deux composantes, on peut espérer les compenser en appliquant une méthode progressive sur z_1 et rétrograde sur z_2.

$$z_1(t+h) = z_1(t) + h\frac{\mathrm{d}z_1}{\mathrm{d}t}(t,\ z_1(t),\ z_2(t)) \tag{26}$$

$$z_2(t+h) = z_2(t) + h\frac{\mathrm{d}z_2}{\mathrm{d}t}(t+h,\ z_1(t+h),\ \boxed{z_2(t+h)}\ ) \tag{27}$$

Dans le cas d'un système séparable, la méthode rétrograde ne nécessite plus d'itération car $\frac{\mathrm{d}z_2}{\mathrm{d}t}$ ne dépend pas de z_2 et $z_1(t+h)$ a été calculé avant.

$$z_1(t+h) = z_1(t) + h\frac{\mathrm{d}z_1}{\mathrm{d}t}\,(t,\ \ z_2(t)) \tag{28}$$

$$z_2(t+h) = z_2(t) + h\frac{\mathrm{d}z_2}{\mathrm{d}t}(t+h,\ z_1(t+h)) \tag{29}$$

Exemple de méthode symplectique : alternance des méthodes progressive et rétrograde entre position et vitesse

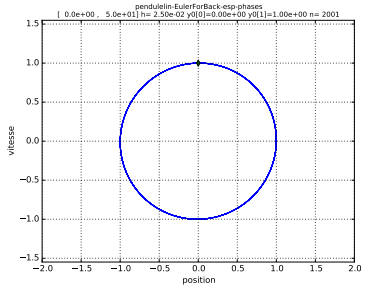


FIGURE 6 – Euler mixte

— progressive sur position,

— rétrograde sur vitesse.

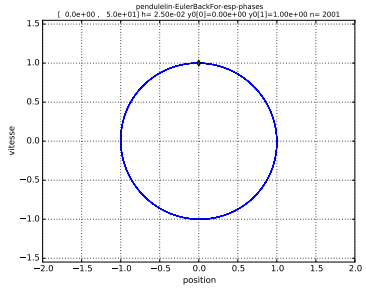


FIGURE 7 – Euler mixte

— rétrograde sur position,

— progressive sur vitesse.

7 Mise en œuvre vectorielle des méthodes à un pas

7.1 Introduction

— Les méthodes d'intégration doivent fonctionner **quelle que soit la taille p des vecteurs** qui représentent la solution $\vec{\mathbf{y}}$ et le second membre $\vec{\mathbf{f}}$ de l'EDO.

— Il en est de même pour l'interface formelle de la fonction second membre en fortran ou le pointeur de fonction second membre en C.

— C'est **le programme principal qui fixera cette taille**.
Il devra donc choisir un second membre de la même dimension.

— Les tailles des tableaux des seconds membres effectifs seront héritées du programme principal et **non déclarées explicitement**.
Mais seules les p composantes effectives de $\vec{\mathbf{f}}$ (2 pour le pendule : dérivée et dérivée seconde) seront calculées à partir des p composantes de $\vec{\mathbf{y}}$.

7.2 En fortran (norme 2003)

Utiliser des **fonctions à argument tableau de rang 1** $\vec{\mathbf{y}}$
d'étendue p déterminée à l'exécution (nombre p d'EDO scalaires d'ordre 1)
et à résultat tableau de même étendue que $\vec{\mathbf{y}}$ pour :

1. **le second membre de l'équation différentielle :**

```
MODULE abstrait
 ABSTRACT INTERFACE ! de la fct générique IInd mb de l'EDO
  FUNCTION fty(t, y) ! dy/dt
   REAL, DIMENSION(:),INTENT(in) :: y ! variable vecteur
   REAL, INTENT(in) :: t
   REAL, DIMENSION(SIZE(y)) :: fty    ! vecteur résultat
  END FUNCTION fty
 END INTERFACE
END MODULE abstrait
```

L'étendue p du vecteur résultat $\vec{\mathbf{f}}$ effectif sera donc fixée par le programme principal via $\vec{\mathbf{y}}$ et non par la fonction second membre.

2. **chacune des méthodes à un pas** (Euler, point milieu et Runge Kutta) :
les pentes locales $\vec{\mathbf{k}_i}$ seront des **tableaux locaux**, par exemple automatiques.

```
FUNCTION u2_rk4(u1, t1, h, f)
  USE abstrait ! où est définie l'interface abstraite fty
  REAL, DIMENSION(:), INTENT(IN) :: u1 ! valeur initiale
  REAL, INTENT(IN) :: t1 ! instant initial
  REAL, INTENT(IN) :: h  !  pas
  PROCEDURE(fty) :: f ! déclaration de l'interface de f
  REAL, DIMENSION(SIZE(u1)) :: u2_rk4 ! valeur estimée à t1+h
  ! variables locales de même étendue que u1
  REAL, DIMENSION(SIZE(u1)) :: k1, k2, k3, k4 ! pentes locales
  ...
  u2_rk4 = u1 + ...
END FUNCTION u2_rk4
```

3. **Dans le programme principal** (et dans la procédure d'écriture sur fichier), les solutions vectorielles (analytique et par intégration) sont représentées par des **tableaux 2D alloués dynamiquement** (n instants, p composantes).

   Les étendues n et p sont donc choisies à l'exécution, sachant que p doit être correspondre au nombre effectif de composantes du second membre étudié.

   Mais la dimension temporelle n'est pas « vue » par les méthodes : elles travaillent sur des vecteurs (d'étendue p) dans un intervalle $[t_i, t_{i+1}]$, à i fixé.

7.3 En C89 avec des tableaux dynamiques sur le tas

Utiliser des fonctions à « argument tableau 1D » $\vec{\mathbf{y}}$ de taille déterminée à l'exécution et rendant un pointeur vers un tableau alloué sur le tas de même taille que $\vec{\mathbf{y}}$.

1. **le second membre de l'équation différentielle** sera alloué par la fonction $\vec{\mathbf{f}}\left(\vec{\mathbf{y}}, t\right)$ qui rend le pointeur vers ce tableau, dont la fonction appelante (la méthode d'intégration) devra prendre en charge la libération ;

```
float * pendule(float t, float *u, int p){
  float *second_membre= NULL; /* tableau 1D */
  second_membre = float1d(p); /* allocation sur le tas */
  second_membre[0] = u[1];
  second_membre[1] = -sin(u[0]);
  return second_membre;       /* valeur de retour = pointeur */
}
```


2. **pour chacune des méthodes à un pas** (Euler, point milieu et Runge Kutta) les pentes locales $\vec{\mathbf{k_i}}$ seront des tableaux alloués et libérés localement, car leur nombre dépend de la méthode ; en revanche, le résultat $\vec{\mathbf{u}}_{i+1}$ qui est aussi vectoriel sera passé en argument (sous forme pointeur plus nombre d'éléments), son allocation et libération prises en charge par l'appelant.

```
void u2_milieu(int p, float *u1, float t1, float h,
               float* (* ptr_f) (float, float*, int),
               float * u2){
/* permettant d'avancer d'un pas en temps*/
  float *k1 = NULL;       /* vecteur pente */
  float *k2 = NULL;       /* vecteur pente */
  float *u12 = NULL;      /* vecteur intermédiaire */
  k1 = (*ptr_f)(t1, u1, p);  /* allocation par la fct IInd membre */
  u12 = float1d (p);      /* allocation locale */
  /* ... */
  float1d_libere(k1);     /* libération des vecteurs des pentes */
  /* ... */
```


3. **Dans le programme principal** (et dans la procédure d'écriture sur fichier), les solutions vectorielles (analytique et par intégration) sont représentées par des tableaux 2D. Mais la dimension temporelle n'est pas « vue » par les méthodes : elles travaillent sur des vecteurs de taille p dans un intervalle $[t_i, t_{i+1}]$. Cela impose que les composantes des vecteurs soient contigües en mémoire, donc **le deuxième indice est celui des composantes, le premier celui du temps**.

```
/* allocation dans le main des tableaux 2D */
  u = float2d(n, p); /* n instants et p equations */
/* appel de la méthode du point milieu par exemple */
  u2_milieu(p, u[i], t[i], h, &pendule, u[i+1]);
/* donc u[i] est un tableau 1D = vecteur des composantes de u_i
```

7.4 En C99 avec des tableaux automatiques

Fonctions à « argument tableau 1D » $\vec{\mathbf{y}}$ de taille p déterminée à l'exécution

Déclaration tardive des tableaux automatiques $\Rightarrow$ éviter les tableaux dynamiques

Mais une fonction ne peut pas rendre un tableau $\Rightarrow$ fonctions à résultat `void`

$\Rightarrow$ **déclaration du tableau argument par l'appelant**

et remplissage par la fonction appelée

1. **la fonction second membre de l'équation différentielle** remplit le tableau `second_mb` de taille p représentant $\vec{\mathbf{f}}(\vec{\mathbf{y}}, t)$ qui a été déclaré par l'appelant (la méthode) avec la taille fixée par le `main`

```
// version C99 avec tableaux automatiques
void pendule(float t, int p, float u[p], float second_mb[p]){
  //  p = 2 ici = dimension des vecteurs  u et second_membre
  second_mb[0] = u[1];
  second_mb[1] = -sin(u[0]);
  return;
}
```


2. **pour chacune des méthodes à un pas** (Euler, point milieu et Runge Kutta) les pentes locales $\vec{\mathbf{k_i}}$ seront des **tableaux locaux automatiques**, car leur nombre dépend de la méthode ; en revanche, le résultat $\vec{\mathbf{u}}_{i+1}$ qui est aussi vectoriel sera passé en argument, sa déclaration étant prise en charge par l'appelant.

```
// C99 avec tableaux automatiques
void u2_milieu(int p, float u1[p], float t1, float h,
                  void (* ptr_f) (float, int, float[], float[]),
                  float u2[p]) {
// methode permettant d'avancer d'un pas en temps
  float k1[p] ;       // vecteur pente à gauche
  float k2[p] ;       // vecteur pente au milieu
  float u12[p];       // vecteur intermédiaire
  (*ptr_f)(t1, p, u1, k1);  //  évaluation du second membre en u1
  // résultat dans le vecteur local k1
  ...
  // tableau u2[p] déclaré dans l'appelant et rempli ici (boucle)
```


3. **Dans le programme principal** (et dans la procédure d'écriture sur fichier), les solutions vectorielles (analytique et par intégration) sont représentées par des tableaux 2D. Mais la dimension temporelle n'est pas « vue » par les méthodes : elles travaillent sur des vecteurs de taille p dans un intervalle $[t_i, t_{i+1}]$. Cela impose que les composantes des vecteurs soient contigües en mémoire, donc **le deuxième indice est celui des composantes, le premier celui du temps**.

```
// tableaux automatiques 2D C99 déclarés dans le main
// p choisi selon la dimension du second membre
 float u[n][p];  // n instants et p equations
// dans la boucle sur les instants i :
// appel de la méthode du point milieu par ex.
 u2_milieu(p, u[i], t[i], h, &pendule, u[i+1]);
// u[i] et u[i+1] : vecteurs à p composantes
// t[i] : scalaire
```


Références

AKAI, TERRENCE J., *Applied Numerical Methods for Engineers*, 410 pages (Wiley, 1994), ISBN 0-471-57523-2.

BURDEN, RICHARD L. et J. DOUGLAS FAIRES, *Numerical Analysis*, 847 pages (Thompson, Brooks/Cole, 2005), huitième édition, ISBN 0-534-40499-5.

DEMAILLY, J.-P., *Analyse numérique et équations différentielles*, 350 pages (EDP Sciences, 2006), troisième édition, ISBN 978-2-86883-891-9.

GUILPIN, CH., *Manuel de calcul numérique appliqué*, 577 pages (EDP Sciences, 1999), ISBN 2-86883-406-X.

RAPPAZ, JACQUES et MARCO PICASSO, *Introduction à l'analyse numérique*, 268 pages (Presses polytechniques et universitaires romandes, 2010), ISBN 978-2-88074-851-7.