Objectif du chapitre: Résolution numérique des équations différentielles.

De manière générale, on s'intéresse à trouver une fonction $y : \mathbb{R} \to \mathbb{R}$ telle que

$$y^{(m)}(t) = f\left(t, y(t), y'(t), \ldots, y^{(m-1)}(t)\right), \quad m \geq 1, t \in [t_0, t_f]$$

Définition (EDO)

Une Équation Différentielle Ordinaire (EDO) est une relation faisant intervenir une fonction inconnue (ici y) d'une seule variable indépendante (ici t) et ses dérivées. **L'ordre** d'une EDO est déterminé par la dérivée la plus élevée de la fonction inconnue apparaissant dans l'équation.

L'unicité de la solution à ce type de problème est garantie par l'ajout de conditions supplémentaires.

En général il s'agit de conditions portant sur la valeur de l'inconnue en certains points. Le nombre de conditions nécessaires étant déterminées par l'ordre de l'EDO.

La variable indépendante t représente souvent (mais pas toujours !) le temps.

Quelques exemples d'EDO:

$$mg - k(v(t))^2 - v'(t) = 0, \text{ EDO d'ordre 1 (non linéaire)}$$
$$y^{(4)}(t) - ty(t) = 9, \text{ EDO d'ordre 4 (linéaire)}$$
$$(y''(x))^3 - y(x) = x^5, \text{ EDO d'ordre 2 (non linéaire)}$$

Dans un premier temps, nous allons nous intéresser aux EDO d'ordre 1.

Faisant toujours référence au temps, la condition que l'on ajoute pour pouvoir résoudre est souvent prise en t_0 et est dite **condition initiale**. On parlera de condition finale si la condition est prise en t_f.

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y_0 \end{cases} \tag{1}$$

Le système (1) est dit problème de Cauchy.

Exemple:

$$\begin{cases} ty'(t) = 2y(t) + t^3 ln(t),\ t \geq 1 \\ y(1) = 0 \end{cases} \tag{2}$$

Vous avez vu (MAT-1900) un procédé de résolution «à la main»:

- Résolution de l'équation homogène (sans second membre)

$$ty'(t) - 2y(t) = 0 \Rightarrow y_h(t) = Ct^2$$

- Recherche d'une solution particulière (variation de la constante)

$$y_p(t) = \frac{t^2}{4}\left(ln(t^2) - 1\right)t^2$$

- On en déduit la forme générale des solutions

$$y(t) = y_h(t) + y_p(t) = t^2\left(C + \frac{t^2}{4}(ln(t^2) - 1)\right)$$

La condition initiale de (2) permet de fixer la constante C est d'obtenir une unique solution.

Comme toujours, on souhaite une approche de résolution systématique du problème, viable numériquement.

Naturellement au niveau numérique il n'est pas possible de construire une solution pour toutes les valeurs possibles de t.

On ne décrira $y(t)$ que pour un nombre fini de points $\rightarrow$ discrétisation de l'EDO
Plus généralement, on se donnera

- N, un nombre de pas de temps,
- t_i, $i = 0, \ldots, N-1$, les valeurs du temps où la solution sera approchée,
- y_i sera l'approximation de la solution au temps t_i, *i.e*

$$\boldsymbol{y_i \approx y(t_i)},$$

  L'erreur commise au temps t_i sera alors $|y_i - y(t_i)|$. Il n'y a pas d'erreur au temps t_0, c'est la condition initiale ! $\rightarrow$ $y(t_0) = y_0$
- $h_i = t_{i+1} - t_i$, les pas de temps. **On aura fréquemment un pas de temps constant noté $\boldsymbol{h}$.**

Première méthode : **Euler explicite**. Commençons par l'approche géométrique

On connaît la solution au temps t_0, on souhaite une approximation au temps $t_1 = t_0 + h$.

On connaît également la pente à y en t_0

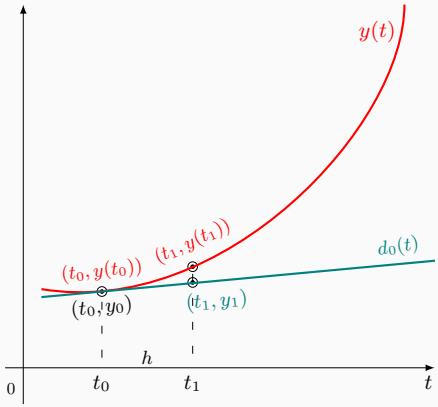
$$y'(t_0) = f(t_0, y(t_0)) = f(t_0, y_0)$$

On suit la droite passant par (t_0, y_0) de pente $f(t_0, y_0)$,

$$d_0(t) = y_0 + f(t_0, y_0)(t - t_0)$$

On prend $y_1 \approx y(t_1)$ comme étant

$$y_1 = d_0(t_1) = y_0 + hf(t_0, y_0)$$



On recommence à partir de t_1

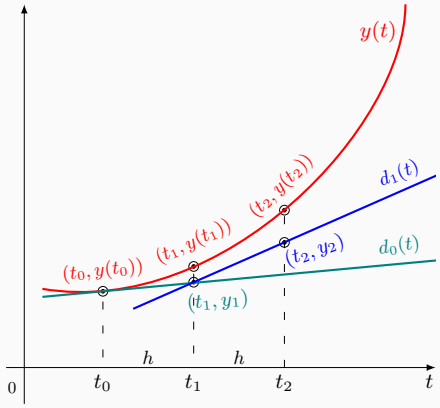
$$y'(t_1) = f(t_1, y(t_1)) \approx f(t_1, y_1)$$

on obtient

$$d_1(t) = y_1 + f(t_1, y_1)(t - t_1)$$

et l'on prend

$$y_2 = d_1(t_2) = y_1 + hf(t_1, y_1) \approx y(t_2)$$



L'erreur introduite lors du calcul de y_1 est réintroduite dans le calcul de $y_2 \rightarrow$ les erreurs vont se propager d'une itération à l'autre.

De manière générale, avec Euler explicite on obtient y_{n+1} à l'aide de la formule

$$y_{n+1} = y_n + hf(t_n, y_n)$$

Définition (Schéma explicite à un pas)

Un schéma numérique explicite à un pas est une relation de la forme

$$y_{n+1} = y_n + h\phi(t_n, y_n)$$

Autrement dit, pour calculer la solution au pas de temps suivant, on se sert uniquement de la solution au pas de temps précédent (connue).

Le schéma d'Euler explicite est un schéma à un pas avec $\phi(t, y) = f(t, y)$. Par opposition aux schémas explicites, il y a les schémas implicites

Définition (Schéma implicite)

Un schéma numérique implicite est une relation de la forme

$$y_{n+1} = y_n + h\phi(t_{n+1}, y_{n+1})$$

Cela implique de résoudre une équation à chaque pas de temps.

En résumé, pour la méthode d'Euler explicite, on a l'algorithme suivant

Algorithme d'Euler explicite, `script_euler_explicite.m`

Données :

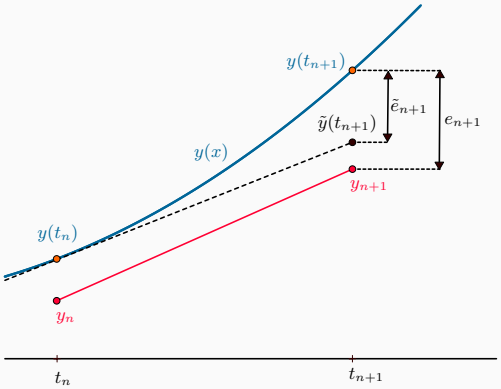
- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Une variable (vecteur) contenant la solution à chaque pas de temps

1 **tant que** $0 \leq n \leq N$ **faire**
2 $\quad y_{n+1} = y_n + hf(t_n, y_n)$;
3 $\quad t_{n+1} = t_n + h$;
4 **fin**

Quid de l'erreur ?

Quand l'algorithme converge t-il vers la solution du problème ?



$$\tilde{y}(t_{n+1}) = y(t_n) + hf(t_n, y(t_n))$$

On a deux sources d'erreurs,

- Une erreur locale, $\tilde{e}_{n+1}$

$$\tilde{e}_{n+1} = y(t_n + 1) - \tilde{y}(t_{n+1})$$

- Une erreur tenant compte de la propagation

$$\tilde{y}(t_{n+1}) - y_{n+1}$$

Au final, l'erreur $e_{n+1} = y(t_{n+1}) - y_{n+1}$ est donnée par

$$e_{n+1} = (y(t_n + 1) - \tilde{y}(t_{n+1})) + (\tilde{y}(t_{n+1}) - y_{n+1})$$

On sait quantifier la première erreur. Pour Euler explicite, le développement de Taylor nous donne

$$\begin{aligned} & y(t_{n+1}) = y(t_n + h) = y(t_n) + hy'(t_n) + O(h^2) \\ \iff & y(t_{n+1}) = \underbrace{y(t_n) + hf(t_n, y(t_n))}_{\tilde{y}(t_{n+1})} + O(h^2) \\ \iff & y(t_{n+1}) - \tilde{y}(t_{n+1}) = O(h^2) \end{aligned}$$

À partir de ce développement de Taylor, on définit également l'**erreur de troncature** $\tau_{n+1}(h)$

$$\begin{aligned} \tau_{n+1}(h) = & \frac{y(t_{n+1}) - y(t_n)}{h} - f(t_n, y(t_n)) = O(h) \\ \iff \tau_{n+1}(h) = & \frac{y(t_{n+1}) - y(t_n)}{h} - y'(t_n) = O(h) \end{aligned}$$

☢ Attention !

On utilise bien $y(t_n)$ et non y_n. On cherche à mesurer l'erreur introduite par l'utilisation d'une différence finie.

L'erreur de troncature est l'erreur «locale» en t_n.

On dit également que la méthode d'Euler explicite est **consistante à l'ordre 1**, car $\tau_{n+1}(h) = O(h)$.

Cette erreur ne tient pas compte des erreurs introduites auparavant.

Le contrôle de cette erreur n'est pas suffisant pour assurer la convergence de la solution numérique vers la solution exacte. Il faut en plus que la méthode soit **stable**.

Définissons d'abord ce que l'on entend par convergence d'un schéma numérique pour une EDO.

Définition (Convergence ordre p)

On dit qu'un schéma converge à l'ordre p si

$$\max_{1 \leq n \leq N} |y(t_n) - y_n| = O(h^p)$$

La convergence à l'ordre p est le fruit de deux éléments : **la consistance** à l'ordre p **et la stabilité.**

- La consistance mesure l'erreur locale à un pas de temps t_n, c'est l'erreur commise en remplaçant y' par une différence finie à un pas donné.
  $\rightarrow$ Assure que la solution exacte des équations discrétisées tende vers la solution exacte des équations continues (quand $h \rightarrow 0$).
- La stabilité d'un schéma assure que l'erreur de propagation d'un pas de temps à l'autre, est bornée. L'étude de la stabilité n'est pas au programme du cours.

Théorème (de Lax)

Si un schéma numérique est **stable et consistant** à l'ordre p alors il est convergent à l'ordre p.

Quelques remarques supplémentaires sur la stabilité. La stabilité dépend:

- De l'EDO considérée (de la fonction f),
- Du schéma numérique. Un schéma numérique peut être :
  - conditionnellement stable, *i.e* stable sous une condition faisant intervenir le pas de discrétisation h,
  - inconditionnellement stable, *i.e* stable peu importe le pas de discrétisation h,
  - instable, *i.e* la propagation d'erreur dégradera (beaucoup) la solution à chaque pas de temps.

Le développement de Taylor ouvre la voie à des schémas d'ordre plus élevé. Voyons la méthode de Taylor du second ordre.

Reprenons le développement de Taylor, et poussons d'un ordre de plus

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + hy'(t_n) + \frac{y''(t_n)h^2}{2} + O(h^3) \\ &= y(t_n) + hf(t_n, y(t_n)) + \frac{f'(t_n, y(t_n))h^2}{2} + O(h^3) \end{aligned} \tag{3}$$

Or,

$$\begin{aligned} f'(t, y(t)) &= \frac{\partial f(t, y(t))}{\partial t} + \frac{\partial f(t, y(t))}{\partial y} y'(t) \\ &= \frac{\partial f(t, y(t))}{\partial t} + \frac{\partial f(t, y(t))}{\partial y} f(t, y(t)) \end{aligned}$$

et donc

$$y(t_{n+1}) \approx y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2}\left(\frac{\partial f(t_n, y(t_n))}{\partial t} + \frac{\partial f(t_n, y(t_n))}{\partial y} f(t_n, y(t_n))\right)$$

En remplaçant $y(t_n)$ par y_n, on obtient l'algorithme de Taylor d'ordre 2.

Algorithme de Taylor d'ordre 2, pseudo-code

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Variable (vecteur) contenant la solution à chaque pas de temps.

1 **tant que** $0 \leq n \leq N$ **faire**

2 $\quad y_{n+1} = y_n + hf(t_n, y_n) + \frac{h^2}{2}\left(\frac{\partial f(t_n, y_n)}{\partial t} + \frac{\partial f(t_n, y_n)}{\partial y} f(t_n, y_n)\right)$;

3 $\quad t_{n+1} = t_n + h$;

4 **fin**

- Comme Euler explicite, c'est un schéma à un pas, avec

$$\phi(t_n, y_n) = f(t_n, y_n) + \frac{h}{2}\left(\frac{\partial f(t_n, y_n)}{\partial t} + \frac{\partial f(t_n, y_n)}{\partial y} f(t_n, y_n)\right)$$

- On montre que $\tau_{n+1}(h) = O(h^2)$, le schéma est consistant à l'ordre 2.

Inconvénient: nécessite le calcul des dérivées de la fonction f.

On souhaite avoir le même ordre (voir plus) sans avoir à calculer les dérivées de f. Reprenons (encore) notre développement de Taylor,

$$\begin{aligned} y(t_{n+1}) =& y(t_n) + hf(t_n, y(t_n)) \\ &+ \frac{h^2}{2}\left(\frac{\partial f(t_n, y(t_n))}{\partial t} + \frac{\partial f(t_n, y(t_n))}{\partial y} f(t_n, y(t_n))\right) + O(h^3) \end{aligned}$$

L'idée est de chercher à remplacer les dérivées de f par des évaluations de f en certains points bien choisis.

On cherche des poids α_1 et α_2 et des points β_1 et β_2 tels que l'expression

$$y(t_{n+1}) = y(t_n) + \alpha_1 hf(t_n, y(t_n)) + \alpha_2 hf(t_n + \beta_1 h, y(t_n) + \beta_2 h) \tag{4}$$

soit une **approximation d'ordre 3**. Un développement de Taylor en deux variables nous donne

$$f(t_n + \beta_1 h, y(t_n) + \beta_2 h) = f(t_n, y(t_n)) + \beta_1 h \frac{\partial f(t_n, y(t_n))}{\partial t} + \beta_2 h \frac{\partial f(t_n, y(t_n))}{\partial y} + O(h^2)$$

et donc

$$\begin{aligned} y(t_{n+1}) =& y(t_n) + (\alpha_1 + \alpha_2) hf(t_n, y(t_n)) \\ &+ \alpha_2 \beta_1 h^2 \frac{\partial f(t_n, y(t_n))}{\partial t} + \alpha_2 \beta_2 h^2 \frac{\partial f(t_n, y(t_n))}{\partial y} + O(h^3) \end{aligned}$$

On a deux expressions d'ordre 3:

$$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2}\left(\frac{\partial f(t_n, y(t_n))}{\partial t} + \frac{\partial f(t_n, y(t_n))}{\partial y} f(t_n, y(t_n))\right) + O(h^3)$$

$$y(t_{n+1}) = y(t_n) + (\alpha_1 + \alpha_2)hf(t_n, y(t_n)) + \alpha_2\beta_1 h^2 \frac{\partial f(t_n, y(t_n))}{\partial t} + \alpha_2\beta_2 h^2 \frac{\partial f(t_n, y(t_n))}{\partial y} + O(h^3)$$

Par identifications des coefficients respectifs, on obtient le système

$$\begin{cases} 1 = \alpha_1 + \alpha_2 \\ \frac{1}{2} = \alpha_2\beta_1 \\ \frac{f(t_n, y(t_n))}{2} = \alpha_2\beta_2 \end{cases}$$

On obtient 3 équations pour 4 inconnues → pas de solution unique, il y a donc plusieurs combinaisons possibles ! On va voir deux solutions populaires.

Méthode d'Euler modifiée: On prend $\alpha_1 = \alpha_2 = \frac{1}{2}$, $\beta_1 = 1$ et $\beta_2 = f(t_n, y(t_n))$. Cela nous donne dans (4)

$$y(t_{n+1}) = y(t_n) + \frac{h}{2}\Big(f(t_n, y(t_n)) + f\big(t_n + h, y(t_n) + hf(t_n, y(t_n))\big)\Big) + O(h^3)$$

Ne nécessite plus l'évaluation des dérivées ! Au niveau discret, en remplaçant $y(t_n)$ par son approximation y_n, on obtient

$$y_{n+1} = y_n + \frac{h}{2}\Big(f(t_n, y_n) + f\big(t_n + h, \underbrace{y_n + hf(t_n, y_n)}_{\text{Euler explicite}}\big)\Big)$$

À chaque itération (pas de temps) on décompose souvent le calcul en deux étapes:

1. Calcul de $\hat{y} = y_n + hf(t_n, y_n) \rightarrow$ prédiction,
2. Calcul de $y_{n+1} = y_n + \frac{h}{2}\Big(f(t_n, y_n) + f\big(t_n + h, \hat{y}\big)\Big) \rightarrow$ correction.

Au final, on a l'algorithme suivant,

Algorithme d'Euler modifié, pseudo-code

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Une variable (vecteur) contenant la solution à chaque pas de temps

1 **tant que** $0 \leq n \leq N$ **faire**

2 $\quad \hat{y} = y_n + hf(t_n, y_n)$;

3 $\quad y_{n+1} = y_n + \dfrac{h}{2}\Big(f(t_n, y_n) + f\big(t_n + h, \hat{y}\big)\Big)$;

4 $\quad t_{n+1} = t_n + h$;

5 **fin**

Méthode du point milieu: On prend $\alpha_1 = 0$, $\alpha_2 = 1$, $\beta_1 = \frac{1}{2}$ et $\beta_2 = \frac{f(t_n, y(t_n))}{2}$. Cela nous donne dans (4)

$$y(t_{n+1}) = y(t_n) + hf\left(t_n + \frac{h}{2}, y_n + \frac{hf(t_n, y(t_n))}{2}\right) + O(h^3)$$

Au niveau discret, en remplaçant $y(t_n)$ par son approximation y_n, on obtient

$$y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{hf(t_n, y_n)}{2}\right)$$

À chaque itération (pas de temps) on décompose souvent le calcul en deux étapes:

1. Calcul de $k_1 = hf(t_n, y_n)$,
2. Calcul de $y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$.

Au final, on a l'algorithme dit du point milieu,

Algorithme du point milieu, pseudo-code

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Une variable (vecteur) contenant la solution à chaque pas de temps

1 **tant que** $0 \leq n \leq N$ **faire**

2 $\quad k_1 = hf(t_n, y_n)$;

3 $\quad y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$;

4 $\quad t_{n+1} = t_n + h$;

5 **fin**

D'autres combinaisons de coefficients α_1, α_2, β_1 et β_2 sont possibles. Ces deux schémas numériques (Euler modifié et point milieu) sont dites méthodes de **Runge-Kutta d'ordre 2**.

Nous allons continuer à augmenter l'ordre du développement de Taylor (3), afin d'obtenir des schémas convergeant à l'ordre 4.

On pousse le développement de Taylor à l'ordre 5.

On introduit 4 poids et 6 points, un raisonnement similaire à celui des méthodes de Runge-Kutta d'ordre 2 nous donne un système de 8 équations pour 10 inconnues.

Parmi toute les possibilités pour le choix des coefficients, un choix est particulièrement populaire. Voici le schéma «RK 4». À chaque itération (pas de temps), on effectue

1. Calcul de $k_1 = hf(t_n, y_n)$,
2. Calcul de $k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$,
3. Calcul de $k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$,
4. Calcul de $k_4 = hf(t_n + h, y_n + k_3)$,
5. Calcul de $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$.

Cela nous donne l'algorithme de Runge Kutta d'ordre 4,

Algorithme de Runge-Kutta ordre 4, pseudo-code `script_RK4.m`

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Une variable (vecteur) contenant la solution à chaque pas de temps

tant que $0 \leq n \leq N$ **faire**

$k_1 = hf(t_n, y_n)$;

$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$;

$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$;

$k_4 = hf(t_n + h, y_n + k_3)$;

$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$;

$t_{n+1} = t_n + h$;

fin

Le nombre et la complexité des calculs augmentent avec l'ordre:

- Euler explicite : 1 évaluation de la fonction f à chaque itération,
- RK 2 : 2 évaluations de la fonction f à chaque itération,
- RK 4 : 4 évaluations de la fonction f à chaque itération.

Cependant plus l'ordre est grand plus l'erreur diminue vite quand h diminue [1].

Rappel

- Si la convergence est d'ordre 2, alors l'erreur diminue par un facteur 2^2 quand h est divisé par deux,
- Si la convergence est d'ordre 4, alors l'erreur diminue par un facteur $2^4 = 16$ quand h est divisé par deux,

De plus, pour h fixé, l'erreur sera plus petite avec une méthode d'ordre 2 qu'avec une méthode d'ordre 1, et ainsi de suite.

Y'a-t-il un équilibre à respecter entre effort de calcul et précision ? Pour éclaircir ce point, effectuons une comparaison.

[1] voir `script_comparaison_ordre_EDO.m` pour exemples numériques.

À «coût» égal (nombre d'évaluation de la fonction f), comparons la précision pour Euler explicite, Euler modifié et Runge-Kutta ordre 4 sur un exemple.

L'EDO est la suivante

$$\begin{cases} y'(t) = y + t + 1, \ t \in [0, 1] \\ y(t_0) = 1 \end{cases}$$

40 évaluations de la fonction f représente:

- 40 pas de temps avec Euler explicite $\Rightarrow h = \frac{1}{40}$
- 20 pas de temps avec Euler modifié $\Rightarrow h = \frac{1}{20}$
- 10 pas de temps avec Runge-Kutta ordre 4 $\Rightarrow h = \frac{1}{10}$

Méthode	h	N	$\max_{1\leq n\leq N} \lvert y(t_n) - y_n \rvert$
Euler explicte (ordre 1)	$1/40$	40	9.9654×10^{-2}
Euler modifiée (ordre 2)	$1/20$	20	3.2723×10^{-3}
RK 4 (ordre 4)	$1/10$	10	6.25297×10^{-6}

Moralité: Pour un coût de calcul à peu prés équivalent, il est préférable de prendre un pas de temps «grand» avec un schéma d'ordre élevé plutôt qu'un pas de temps petit avec un schéma d'ordre moins élevé !

Pour une précision comparable à RK4 avec Euler explicite il faudrait $10\,000$ fois plus de pas de temps !

La méthode de Runge-Kutta d'ordre 4 est la méthode la plus utilisée en raison de sa grande précision.

Les méthodes vues jusqu'à présent concernent les EDO d'ordre 1.

Considérons à présent une équation d'ordre m, avec m conditions initiales,

$$\begin{cases} y^{(m)}(t) = f\left(t, y(t), y'(t), \ldots, y^{(m-1)}(t)\right), \\ y(t_0) = y_0 \\ y'(t_0) = y_1 \\ y''(t_0) = y_2 \\ \vdots \\ y^{(m-1)}(t_0) = y_{m-1} \end{cases} \tag{5}$$

Plan de bataille:

1. Résolution d'un système d'EDO d'ordre 1 (en utilisant les méthodes déjà vues).
2. Ré-écriture de (5) en m EDO d'ordre 1 $\rightarrow$ système d'EDO d'ordre 1,

• **Résolution d'un système d'EDO d'ordre 1**:

Plaçons nous dans le cas général, consistant à trouver $y_1, y_2, \ldots, y_m$ tels que

$$\begin{cases} y_1'(t) = f_1(t, y_1(t), y_2(t), \ldots, y_m(t)) & y_1(t_0) = y_{1,0} \\ y_2'(t) = f_2(t, y_1(t), y_2(t), \ldots, y_m(t)) & y_2(t_0) = y_{2,0} \\ \vdots & \vdots \\ y_{m-1}'(t) = f_{m-1}(t, y_1(t), y_2(t), \ldots, y_m(t)) & y_{m-1}(t_0) = y_{m-1,0} \\ y_m'(t) = f_m(t, y_1(t), y_2(t), \ldots, y_m(t)) & y_m(t_0) = y_{m,0} \end{cases}$$

Que l'on ré-écrit sous forme vectorielle

$$\begin{cases} \vec{Y}'(t) = \vec{F}(t, \vec{Y}(t)) \\ \vec{Y}(t_0) = \vec{Y_0} \end{cases}$$

Où

$$\begin{aligned} &\overrightarrow{Y(t)} = (y_1(t), y_2(t), \ldots, y_m(t)), \\ &\vec{F}(t, \vec{Y}(t)) = (f_1(t, \vec{Y}(t)), f_2(t, \vec{Y}(t)), \ldots, f_m(t, \vec{Y}(t))) \\ &\vec{Y_0} = (y_{1,0}, y_{2,0}, \ldots, y_{m,0}). \end{aligned}$$

Les méthodes vues dans ce chapitre se généralisent bien aux systèmes.

On note $\vec{Y}_i = (y_{1,i}, y_{2,i}, \dots, y_{m,i}) \approx \vec{Y}(t_i) = (y_1(t_i), y_2(t_i), \dots, y_m(t_i))$

• **Euler Explicite:**

$$\vec{Y}_{n+1} = \vec{Y}_n + h\vec{F}(t_n, \vec{Y_n})$$

Soit encore

$$\begin{cases} y_{1,n+1} = y_{1,n} + hf_1(t_n, y_{1,n}, y_{2,n}, \dots, y_{m,n}) \\ y_{2,n+1} = y_{2,n} + hf_2(t_n, y_{1,n}, y_{2,n}, \dots, y_{m,n}) \\ \qquad \vdots \\ y_{m,n+1} = y_{m,n} + hf_m(t_n, y_{1,n}, y_{2,n}, \dots, y_{m,n}) \end{cases}$$

À chaque pas de temps, on fait m Euler explicite «standard» avant de passer au pas de temps suivant.

On en déduit l'algorithme d'Euler explicite pour un système,

Algorithme d'Euler explicite pour les systèmes d'EDO, pseudo-code

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale $\overrightarrow{Y_0} = (y_{1,0}, y_{2,0}, \dots, y_{m,0})$.

Résultat : Une variable (matrice) contenant la solution à chaque pas de temps, pour chaque y_i.

tant que $0 \leq n \leq N$ **faire**
  tant que $1 \leq i \leq m$ **faire**
    $y_{i,n+1} = y_{i,n} + h f_i(t_n, y_{1,n}, y_{2,n}, \dots, y_{m,n})$;
  fin
  $t_{n+1} = t_n + h$;
fin

On peut faire la même chose pour Euler modifié, ou la méthode du Point milieu. Voyons directement la généralisation de la méthode de Runge-Kutta d'ordre 4.

• **Runge-Kutta ordre 4 pour les systèmes:** À chaque pas de temps, on va calculer $4m$ coefficients $k_{i,1}$, $k_{i,2}$, $k_{i,3}$, $k_{i,4}$, $i = 1, \ldots, m$. On part de $(t_0, \vec{Y_0})$:

- On détermine $\vec{k}_1 = (k_{i,1})_{1 \leq i \leq m}$,

$$k_{i,1} = h f_i(t_n, y_{1,n}, y_{2,n}, \cdots, y_{m,n})$$

- On détermine $\vec{k}_2 = (k_{i,2})_{1 \leq i \leq m}$,

$$k_{i,2} = h f_i(t_n + \frac{h}{2}, y_{1,n} + \frac{k_{1,1}}{2}, y_{2,n} + \frac{k_{2,1}}{2}, \cdots, y_{m,n} + \frac{k_{m,1}}{2})$$

- On détermine $\vec{k}_3 = (k_{i,3})_{1 \leq i \leq m}$,

$$k_{i,3} = h f_i(t_n + \frac{h}{2}, y_{1,n} + \frac{k_{1,2}}{2}, y_{2,n} + \frac{k_{2,2}}{2}, \cdots, y_{m,n} + \frac{k_{m,2}}{2})$$

- On détermine $\vec{k}_4 = (k_{i,4})_{1 \leq i \leq m}$,

$$k_{i,4} = h f_i(t_n + h, y_{1,n} + k_{1,3}, y_{2,n} + k_{2,3}, \cdots, y_{m,n} + k_{m,3})$$

- Et finalement, on calcule

$$y_{i,n+1} = y_{i,n} + \frac{1}{6}(k_{i,1} + 2k_{i,2} + 2k_{i,3} + k_{i,4})$$

Il faut calculer les m constantes $k_{i,1}$ avant de passer au calcul des constantes $k_{i,2}$ et ainsi de suite.

- **Revenons à présent aux équations d'ordre m :**

$$\begin{cases} y^{(m)}(t) = f\left(t, y(t), y'(t), \ldots, y^{(m-1)}(t)\right), \\ y(t_0) = y_0 \\ y'(t_0) = y_1 \\ y''(t_0) = y_2 \\ \vdots \\ y^{(m-1)}(t_0) = y_{m-1} \end{cases}$$

On va transformer cette EDO d'ordre m en un système de m EDO d'ordre 1, pour pouvoir utiliser les méthodes vues précédemment.

Pour écrire l'EDO d'ordre m comme un système de m EDO d'ordre 1, posons

$$
\begin{aligned}
u_0(t) &= y(t) \\
u_1(t) &= y'(t) \\
u_2(t) &= y''(t) \\
&\vdots \\
u_{m-1}(t) &= y^{(m-1)}(t) \\
(u_m(t) &= y^{(m)}(t))
\end{aligned}
$$

On a alors

$$
\begin{aligned}
u_0'(t) &= y'(t) = u_1(t) \\
u_1'(t) &= y''(t) = u_2(t) \\
&\vdots \\
u_{m-2}'(t) &= y^{(m-1)}(t) = u_{m-1}(t) \\
u_{m-1}'(t) &= y^{(m)}(t) = f\left(t, u_0(t), u_1(t), \ldots, u_{m-1}(t)\right)
\end{aligned}
$$

Avec ces notations, l'EDO d'ordre m (5) s'écrit comme un système de m d'EDO d'ordre 1

$$\begin{cases} u_0'(t) = u_1(t) \ (= y'(t)) & u_0(t_0) = y_0 \\ u_1'(t) = u_2(t) \ (= y''(t)) & u_1(t_0) = y_1 \\ u_2'(t) = u_3(t) \ (= y'''(t)) & u_2(t_0) = y_2 \\ \vdots & \vdots \\ u_{m-2}'(t) = u_{m-1}(t) \ (= y^{(m-1)}(t)) & u_{m-2}(t_0) = y_{m-2} \\ u_{m-1}'(t) = f\left(t, u_0(t), \ldots, u_{m-1}(t)\right) \ (= y^{(m)}(t)) & u_{m-1}(t_0) = y_{m-1} \end{cases} \tag{6}$$

✓ Ne pas oublier que c'est surtout $u_0(t) = y(t)$ qui nous intéresse, c'est la solution de l'EDO !

La résolution d'une EDO d'ordre m est donc un cas particulier de la résolution d'un système d'EDO d'ordre 1 avec

$$\begin{aligned}\vec{Y}(t) &= (u_0(t), u_1(t), \dots, u_{m-1}(t)),\\ \vec{F}(t, \vec{Y}(t)) &= (u_1(t), u_2(t), \dots, u_{m-1}(t), \underbrace{f\left(t, u_0, u_1, \dots, u_{m-1}\right)}_{y^{(m)}(t)})\end{aligned}$$

$$\vec{Y}_0 = (y_0, y_1, y_2, \dots, y_{m-1})$$

En conclusion, la stratégie de résolution d'une EDO d'ordre m est la suivante[2]:

- Ré-écriture de l'EDO d'ordre m (5) en un système d'EDO d'ordre 1 de la forme (6),
- Résolution du système à l'aide de l'une des méthodes vue au début du chapitre, adaptée à la résolution d'un système d'équations.

[2]Pour un exemple numérique, voir le problème de chute libre `script_parachutiste.m` (nécessite `affiche_bonhomme.m`). Pour visualiser le résultat, cliquer **ici**.

Je dois être capable de :

- Comprendre le concept de schéma en temps,
- Comprendre la construction d'un schéma basé sur le développement de Taylor,
- Comprendre l'origine du concept de famille de schéma pour Runge-Kutta,
- Appliquer les différents schémas pour une EDO d'ordre 1,
- Comprendre le concept d'erreur locale ainsi que l'ordre de convergence,
- Transformer une EDO d'ordre supérieur en un système d'EDO,
- Appliquer les schémas sur un système d'EDO d'ordre 1,
- Comprendre qu'avec les méthodes vues, pour avoir stabilité du schéma, le pas de temps est important et ne doit pas être choisit n'importe comment.

Autre approche de résolution: on intègre l'EDO sur l'intervalle $[t_n, t_{n+1}]$

$$\int_{t_n}^{t_{n+1}} y'(t)dt = \int_{t_n}^{t_{n+1}} f(t, y(t))\, dt$$

$$\iff y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y(t))\, dt$$

La question est comment intégrer le terme de droite $\rightarrow$ on utilise l'interpolation de Newton de la fonction f.

Dans la suite, on note $f_n = f(t_n, y_n) \approx f(t_n, y(t_n))$.

L'idée pour construire le polynôme d'interpolation est d'utiliser des quantités connues provenant des pas de temps précédents.

Par exemple, à partir des 3 pas de temps précédents, la table de différences divisées s'écrit

Table de différences divisées

t_n	f_n			
		$f[t_n, t_{n-1}]$	$f[t_n, t_{n-1}, t_{n-2}]$	
t_{n-1}	f_{n-1}			
		$f[t_{n-1}, t_{n-2}]$		$f[t_n, t_{n-1}, t_{n-2}, t_{n-3}]$
t_{n-2}	f_{n-2}		$f[t_{n-1}, t_{n-2}, t_{n-3}]$	
		$f[t_{n-2}, t_{n-3}]$		
t_{n-3}	f_{n-3}			

Le polynôme d'interpolation de degré 3 est:

$$p_3(t) = f_n + f[t_n, t_{n-1}](t - t_n) + f[t_n, t_{n-1}, t_{n-2}](t - t_n)(t - t_{n-1}) + f[t_n, t_{n-1}, t_{n-3}](t - t_n)(t - t_{n-1})(t - t_{n-2})$$

En utilisant des polynômes de différents degrés, on obtient différents schémas.

- Degré 0, $p_0(t) = f_n = f(t_n, y_n) \approx f(t, y(t))$ et on obtient

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f_n dt = y_n + hf(t_n, y_n)$$

  On retrouve Euler explicite.

- Degré 1, $p_1(t) = f_n + f[t_n, t_{n-1}](t - t_n) \approx f(t, y(t))$ et on obtient

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_n + f[t_n, t_{n-1}](t - t_n)\, dt \\ &= y_n + \frac{h}{2}(3f(t_n, y_n) - f(t_{n-1}, y_{n-1})) \end{aligned}$$

  C'est une méthode à deux pas, on se sert de y_n et y_{n-1} pour calculer y_{n+1}.

On peut continuer ainsi de suite, chaque fois que l'on augmente de 1 le degré du polynôme d'interpolation, on augmente de 1 l'ordre de convergence.

BONUS ! Pas à l'examen

Toutes ces méthodes sont toujours explicites, seuls les pas de temps précédents sont nécessaires.

En utilisant d'autres points pour construire la table de différence divisées, en particulier en ajoutant (t_{n+1}, y_{n+1}) pour construire le polynôme d'interpolation, on obtient des méthodes dites **implicites**.

- Degré 0 (Euler implicite [3]), $p_0(t) = f_{n+1} = f(t_{n+1}, y_{n+1}) \approx f(t, y(t))$

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f_{n+1}\, dt = y_n + hf(t_{n+1}, y_{n+1})$$

- Degré 1, $p_1(t) = f_{n+1} + f[t_{n+1}, t_n](t - t_{n+1}) \approx f(t, y(t))$

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_{n+1} + f[t_{n+1}, t_n](t - t_{n+1})\, dt \\ &= y_n + \frac{h}{2}(f(t_{n+1}, y_{n+1}) + f(t_n, y_n)) \end{aligned}$$

Pour les méthodes implicites, cela implique de résoudre une équation à chaque pas de temps.

[3] voir `script_euler_implicite.m`