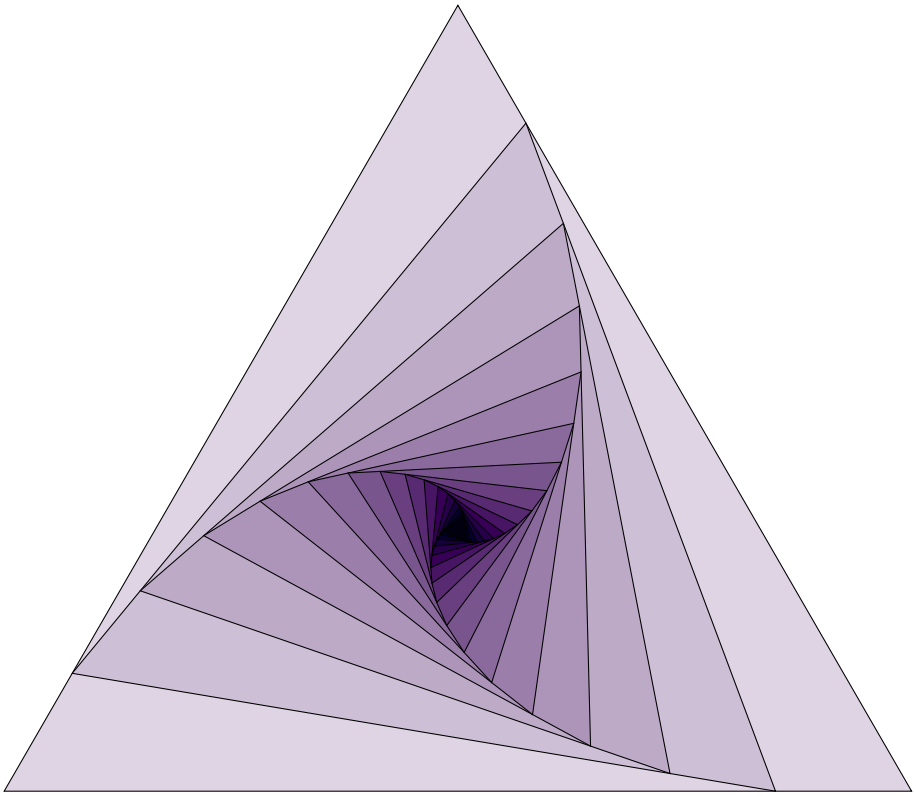
# Méthodes numériques II

*Notes de cours*

Sup Galilée, Ingénieurs Energétique 1ère année



Francois Cuvelier
Université Paris XIII / Institut Galilée
L.A.G.A./Département de Mathématiques
http://www.math.univ-paris13.fr/∼cuvelier

# Table des matières


**1 Langage Algorithmique** **1**
1.1 Pseudo-langage algorithmique . . . 1
1.1.1 Données et constantes . . . 1
1.1.2 Variables . . . 1
1.1.3 Opérateurs . . . 2
1.1.4 Expressions . . . 2
1.1.5 Instructions . . . 3
1.1.6 Fonctions . . . 4
1.2 Méthodologie d'élaboration d'un algorithme . . . 6
1.2.1 Description du problème . . . 6
1.2.2 Recherche d'une méthode de résolution . . . 6
1.2.3 Réalisation d'un algorithme . . . 7
1.2.4 Exercices . . . 7
1.3 Principes de «bonne» programmation pour attaquer de «gros» problèmes . . . 9

**2 Dérivation numérique** **11**

**3 Introduction à la résolution d'E.D.O.** **21**
3.1 Introduction . . . 21
3.1.1 Exemple en météorologie : modèle de Lorentz . . . 21
3.1.2 Exemple en biologie . . . 23
3.1.3 Exemple en chimie : La réaction de Belousov-Zhabotinsky . . . 23
3.1.4 Exemple en mécanique . . . 25
3.2 Problème de Cauchy . . . 26
3.3 Différences finies pour les E.D.O. . . . 29
3.3.1 Différences finies pour le problème de Cauchy en dimension $m = 1$ . . . 29
3.3.2 Différences finies pour le problème de Cauchy en dimension $m$ . . . 33
3.4 Méthodes à un pas ou à pas séparés . . . 33
3.5 Méthodes de Runge-Kutta . . . 35
3.5.1 Principe . . . 36
3.5.2 Formules explicites de Runge-Kutta d'ordre 2 . . . 36
3.5.3 Méthodes de Runge-Kutta d'ordre 4 . . . 39
3.6 Méthodes à pas multiples . . . 41
3.6.1 Exemple : schéma de point milieu . . . 41

3.6.2 Le principe . . . . . 41
3.6.3 Méthodes explicites d'Adams-Bashforth . . . . . 43
3.6.4 Méthodes implicites d'Adams-Moulton . . . . . 44
3.6.5 Schéma prédicteur-correcteur . . . . . 45

**4 Introduction à la résolution d'E.D.P. 47**

# Chapitre 1

## Langage Algorithmique

## 1.1 Pseudo-langage algorithmique

Pour uniformiser l'écriture des algorithmes nous employons, un pseudo-langage contenant l'indispensable :

- variables,
- opérateurs (arithmétiques, relationnels, logiques),
- expressions,
- instructions (simples et composées),
- fonctions.

Ce pseudo-langage sera de fait très proche du langage de programmation de Matlab.

### 1.1.1 Données et constantes

Une donnée est une valeur introduite par l'utilisateur (par ex. une température, une vitesse, ... Une constante est un symbole ou un identificateur non modifiable (par ex. $\pi$, la constante de gravitation,...)

### 1.1.2 Variables

**Definition 1.1**

Une variable est un objet dont la valeur est modifiable, qui possède un nom et un type (entier, charactère, réel, complexe, ...). Elle est rangée en mémoire à partir d'une certaine adresse.

### 1.1.3 Opérateurs

**Opérateurs arithmétiques**

| Nom | Symbole | example |
|---|---|---|
| **addition** | $+$ | $a+b$ |
| **soustraction** | $-$ | $a-b$ |
| **opposé** | $-$ | $-a$ |
| **produit** | $*$ | $a*b$ |
| **division** | $/$ | $a/b$ |
| **puissance** $a^b$ | ^ | $a\hat{}b$ |

Table 1.1: Opérateurs arithmétiques

**Opérateurs relationnels**

| Nom | Symbole | example | Commentaires |
|---|---|---|---|
| **identique** | $==$ | $a == b$ | vrai si $a$ et $b$ ont même valeur, faux sinon. |
| **différent** | $\sim=$ | $a \sim= b$ | faux si $a$ et $b$ ont même valeur, vrai sinon. |
| **inférieur** | $<$ | $a < b$ | vrai si $a$ est plus petit que $b$, faux sinon. |
| **supérieur** | $>$ | $a > b$ | vrai si $a$ est plus grand que $b$, faux sinon. |
| **inférieur ou égal** | $<=$ | $a <= b$ | vrai si $a$ est plus petit ou égal à $b$, faux sinon. |
| **supérieur ou égal** | $>=$ | $a >= b$ | vrai si $a$ est plus grand ou égal à $b$, faux sinon. |

Table 1.2: Opérateurs relationnels

**Opérateurs logiques**

| Nom | Symbole | example | Commentaires |
|---|---|---|---|
| **négation** | $\sim$ | $\sim a$ | vrai si $a$ est faux (ou nul), faux sinon. |
| **ou** | $\|$ | $a\|b$ | vrai si $a$ **ou** $b$ est vrai (non nul), faux sinon. |
| **et** | & | $a\&b$ | vrai si $a$ **et** $b$ sont vrais (non nul), faux sinon. |

Table 1.3: Opérateurs logiques

**Opérateur d'affectation**

| Nom | Symbole | example | Commentaires |
|---|---|---|---|
| **affectation** | $\leftarrow$ | $a \leftarrow b$ | On affecte à la variable $a$ le contenu de $b$ |

Table 1.4: Opérateurs d'affectation

### 1.1.4 Expressions

**Definition 1.2**

Une expression est un groupe d'opérandes (i.e. nombres, constantes, variables, ...) liées par certains opérateurs pour former un terme algébrique qui représente une valeur (i.e. un élément de donnée simple)

**Exemple 1.3**

- Voici un exemple classique d'expression numérique :

$$(b*b-4*a*c)/(2*a).$$

On appelle **opérandes** les identifiants $a$, $b$ et $c$, et les nombres 4 et 2. Les symboles $*$, $-$ et $/$ sont les **opérateurs**.

- Voici un exemple classique d'expression booléenne (logique) :

$$(x < 3.14)$$

Dans cette expression, $x$ est une variable numérique et 3.14 est un nombre réel. Cette expression prendra la valeur vrai (i.e. 1) si $x$ est plus grand que 3.14. Sinon, elle prendra la valeur faux (i.e. 0)

### 1.1.5 Instructions

**Definition 1.4**

Une **instruction** est un ordre ou un groupe d'ordres qui déclenche l'exécution de certaines actions par l'ordinateur. Il y a deux types d'instructions : simple et structuré.
Les **instructions simples** sont essentiellement des ordres seuls et inconditionnels réalisant l'une des tâches suivantes :

1. affectation d'une valeur a une variable.
2. appel d'une fonction (procedure, subroutine, ... suivant les langages).

Les **instructions structurées** sont essentiellement :

1. les instructions composées, groupe de pulsieurs instructions simples,
2. les instructions répétitives, permettant l'exécution répétée d'instructions simples, (i.e. boucles «pour», «tant que»)
3. les instructions conditionnelles, lesquels ne sont exécutées que si une certaine condition est respectée (i.e. «si»)

Les exemples qui suivent sont écrits dans un pseudo langage algorithmique mais sont facilement transposable dans la plupart des langages de programmation.

#### Instructions simples

Voici un exemple de l'*instruction simple d'affectation* :

1: $a \leftarrow 3.14 * R$

On évalue l'expression $3.14 * R$ et affecte le résultat à la variable $a$.
Un autre exemple est donné par l'*instruction simple d'affichage* :

```
affiche('bonjour')
```

Affiche la chaine de caractères `'bonjour'` à l'écran. Cette instruction fait appel à la fonction `affiche`.

#### Instructions composées

#### Instructions répétitives, boucle «pour»

**Algorithme 1.1** Exemple de boucle «pour»

**Données :** $n$ : un entier.

1: $S \leftarrow 0$
2: **Pour** $i \leftarrow 1$ **à** $n$ **faire**
3: $\quad S \leftarrow S + \cos(i^2)$
4: **Fin Pour**

**Instruction répétitive, boucle «tant que»**

**Algorithme 1.2** Exemple de boucle «tant que»

```
1: i ← 0, x ← 1
2: Tantque i < 1000 faire
3:     x ← x + i * i
4:     i ← i + 1
5: Fin Tantque
```

**Instruction répétitive, boucle «répéter ...jusqu'à»**

**Algorithme 1.3** Exemple de boucle «répéter ...jusqu'à»

```
1: i ← 0, x ← 1
2: Répéter
3:     x ← x + i * i
4:     i ← i + 1
5: jusqu'à i>=1000
```

**Instructions conditionnelles «si»**

**Algorithme 1.4** Exemple d'instructions conditionnelle «si»

**Données :** *age* : un réel.

```
1: Si age >= 18 alors
2:     affiche('majeur')
3: Sinon Si age >= 0 alors
4:     affiche('mineur')
5: Sinon
6:     affiche('en devenir')
7: Fin Si
```

### 1.1.6 Fonctions

Les fonctions permettent

- d'automatiser certaines tâches répétitives au sein d'un même programme,
- d'ajouter à la clarté d'un programme,
- l'utilisation de portion de code dans un autre programme,
- ...

**Fonctions prédéfinies**

Pour faciliter leur usage, tous les langages de programmation possèdent des fonctions prédéfinies. On pourra donc supposer que dans notre langage algorithmique un grand nombre de fonctions soient prédéfinies : par exemple, les fonctions mathématiques sin, cos, exp, abs, $\cdots$ (pour ne citer quelles)

**Syntaxe**

On utilise la syntaxe suivante pour la définition d'une fonction

**Fonction** $[args_1, \ldots, args_n] \leftarrow$ NomFonction( $arge_1, \ldots, arge_m$ )
instructions
**Fin Fonction**

La fonction se nomme NomFonction . Elle admet comme paramètres d'entrée (données) les $m$ arguments $arge_1, \ldots, arge_m$ et comme paramètres de sortie (résultats) les $n$ arguments $args_1, \ldots, args_n$. Ces derniers doivent être déterminés dans le corps de la fonction (partie instructions).
Dans le cas ou la fonction n'admet qu'un seul paramètre de sortie, l'écriture se simplifie :

**Fonction** $args \leftarrow$ NomFonction( $arge_1, \ldots, arge_m$ )
instructions
**Fin Fonction**

### Ecrire ses propres fonctions

Pour écrire une fonction «propre», il faut tout d'abord déterminer exactement ce que devra faire cette fonction.

Puis, il faut pouvoir répondre à quelques questions :

1. Quelles sont les données (avec leurs limitations)?
2. Que doit-on calculer ?

Et, ensuite il faut la **commenter** : expliquer son usage, type des paramètres, ....

### Exemple : résolution d'une équation du premier degré

Nous voulons écrire une fonction calculant la solution de l'équation

$$ax + b = 0,$$

où nous supposons que $a \in \mathbb{R}^*$ et $b \in \mathbb{R}$. La solution de ce problème est donc

$$x = -\frac{b}{a}.$$

Les données de cette fonction sont $a \in \mathbb{R}^*$ et $b \in \mathbb{R}$. Elle doit retourner $x = -\frac{b}{a}$ solution de $ax + b = 0$.

**Algorithme 1.5** Exemple de fonction : Résolution de l'équation du premier degré $ax + b = 0$.

**Données :** $a$ : nombre réel différent de 0
$b$ : nombre réel.
**Résultat :** $x$ : un réel.

1: **Fonction** $x \leftarrow$ REPD( $a, b$ )
2: $x \leftarrow -b/a$
3: **Fin Fonction**

**Remarque 1.5** Cette fonction est très simple, toutefois pour ne pas «alourdir» le code nous n'avons pas vérifié la validité des données fournies.

**Exercice 1.1.1**

Ecrire un algorithme permettant de valider cette fonction.

**Exemple : résolution d'une équation du second degré**

Nous cherchons les solutions réelles de l'équation

$$ax^2 + bx + c = 0, \tag{1.1}$$

où nous supposons que $a \in \mathbb{R}^*$, $b \in \mathbb{R}$ et $c \in \mathbb{R}$ sont donnés.

Mathématiquement, l'étude des solutions réelles de cette équation nous amène à envisager trois cas suivant les valeurs du discriminant $\Delta = b^2 - 4ac$

- si $\Delta < 0$ alors les deux solutions sont complexes,
- si $\Delta = 0$ alors la solution est $x = -\frac{b}{2a}$,
- si $\Delta > 0$ alors les deux solutions sont $x_1 = \frac{-b-\sqrt{\Delta}}{2*a}$ et $x_2 = \frac{-b-\sqrt{\Delta}}{2*a}$.

**Exercice 1.1.2**

1. Ecrire la fonction `discriminant` permettant de calculer le discriminant de l'équation (1.1).
2. Ecrire la fonction `RESD` permettant de résoudre l'équation (1.1) en utilisant la fonction `discriminant`.
3. Ecrire un programme permettant de valider ces deux fonctions.

**Exercice 1.1.3**

Même question que précédemment dans le cas complexe (solution et coefficients).

## 1.2 Méthodologie d'élaboration d'un algorithme

### 1.2.1 Description du problème

- Spécification d'un ensemble de données
  Origine : énoncé,hypothèses, sources externes, ...
- Spécification d'un ensemble de buts à atteindre
  Origine : résultats, opérations à effectuer, ...
- Spécification des contraintes

### 1.2.2 Recherche d'une méthode de résolution

- Clarifier l'énoncé.
- Simplifier le problème.
- Ne pas chercher à le traiter directement dans sa globalité.
- S'assurer que le problème est soluble (sinon problème d'indécidabilité!)
- Recherche d'une stratégie de construction de l'algorithme
- Décomposer le problème en sous problèmes partiels plus simples : raffinement.
- Effectuer des raffinements successifs.
- Le niveau de raffinement le plus élémentaire est celui des instructions.

### 1.2.3 Réalisation d'un algorithme

Il doit être conçu indépendamment du langage de programmation et du système informatique (sauf cas très particulier)

- L'algorithme doit être exécuté en un nombre fini d'opérations.
- L'algorithme doit être spécifié clairement, sans la moindre ambiguïté.
- Le type de données doit être précisé.
- L'algorithme doit fournir au moins un résultat.
- L'algorithme doit être effectif : toutes les opérations doivent pouvoir être simulées par un homme en temps fini.

Pour écrire un algorithme détaillé, il faut tout d'abord savoir répondre a quelques questions :

- Que doit-il faire ? (i.e. Quel problème est-il censé résoudre?)
- Quelles sont les données necessaires à la résolution de ce problème?
- Comment résoudre ce problème «à la main» (sur papier)?

Si l'on ne sait pas répondre à l'une de ces questions, l'écriture de l'algorithme est fortement comprise.

### 1.2.4 Exercices

**Exercice 1.2.1: Algorithme pour une somme**

Ecrire un algorithme permettant de calculer

$$S(x) = \sum_{k=1}^{n} k \sin(2 * k * x)$$

**Correction Exercice 1.2.1** L'énoncé de cet exercice est imprécis. On choisi alors $x \in \mathbb{R}$ et $n \in \mathbb{N}$ pour rendre possible le calcul. Le problème est donc de calculer

$$\sum_{k=1}^{n} k \sin(2kx).$$

Toutefois, on aurait pu choisir $x \in \mathbb{C}$ ou encore un tout autre problème :

$$\text{Trouver } x \in \mathbb{R} \text{ tel que } S(x) = \sum_{k=1}^{n} k \sin(2kx)$$

où $n \in \mathbb{N}$ et $S$, fonction de $\mathbb{R}$ à valeurs réelles, sont les données!

**Algorithme 1.6** Calcul de $S = \sum_{k=1}^{n} k \sin(2kx)$

**Données :** $x$ : nombre réel,
$n$ : nombre entier.
**Résultat :** $S$ : un réel.

1: $S \leftarrow 0$
2: **Pour** $k \leftarrow 1$ **à** $n$ **faire**
3: $\quad S \leftarrow S + k * \sin(2 * k * x)$
4: **Fin Pour**

◇

### Exercice 1.2.2: Algorithme pour un produit

Ecrire un algorithme permettant de calculer

$$P(z) = \prod_{n=1}^{k} sin(2 * k * z/n)^k$$

**Correction Exercice 1.2.2** L'énoncé de cet exercice est imprécis. On choisi alors $z \in \mathbb{R}$ et $k \in \mathbb{N}$ pour rendre possible le calcul.

**Algorithme 1.7** Calcul de $P = \prod_{n=1}^{k} \sin(2kz/n)^k$

**Données :** $z$ : nombre réel,
$k$ : nombre entier.
**Résultat :** $P$ : un réel.

```
1: P ← 1
2: Pour n ← 1 à k faire
3:     P ← P * sin(2 * k * z/n)^k
4: Fin Pour
```

◇

### Exercice 1.2.3: Série de Fourier

Soit la série de Fourier

$$x(t) = \frac{4A}{\pi}\left\{\cos\omega t - \frac{1}{3}\cos 3\omega t + \frac{1}{5}\cos 5\omega t - \frac{1}{7}\cos 7\omega t + \cdots\right\}.$$

Ecrire la fonction SFT permettant de calculer $x_n(t)$.

**Correction Exercice 1.2.3** Nous devons écrire la fonction permettant de calculer

$$x_n(t) = \frac{4A}{\pi}\sum_{k=1}^{n}(-1)^{k+1}\frac{1}{2k-1}\cos((2k-1)\omega t)$$

Les données de la fonction sont $A \in \mathbb{R}$, $\omega \in \mathbb{R}$, $n \in \mathbb{N}^*$ et $t \in \mathbb{R}$.

Grâce a ces renseignements nous pouvons déjà écrire l'entête de la fonction :

**Algorithme 1.8** En-tête de la fonction SFT retournant valeur de la série de Fourier en $t$ tronquée au $n$ premiers termes de l'exercice 1.2.3.

**Données :** $t$ : nombre réel,
$n$ : nombre entier strictement positif
$A$, $\omega$ : deux nombres réels.
**Résultat :** $x$ : un réel.

```
1: Fonction x ← SFT( t, n, A, ω )
2:     ...
3: Fin Fonction
```

Maintenant nous pouvons écrire progressivement l'algorithme pour aboutir au final à une version ne contenant que des opérations élémentaires.

**Algorithme 1.9** $\mathcal{R}_0$

1: $x \leftarrow \frac{4A}{\pi} \sum_{k=1}^{n} \left( \begin{array}{c} (-1)^{k+1} \frac{1}{2k-1} \times \\ \cos((2k-1)\omega t) \end{array} \right)$

**Algorithme 1.9** $\mathcal{R}_1$

1: $S \leftarrow \sum_{k=1}^{n} (-1)^{k+1} \frac{1}{2k-1} \cos((2k-1)\omega t)$
2: $x \leftarrow \frac{4A}{\pi} S$

**Algorithme 1.9** $\mathcal{R}_1$

1: $S \leftarrow \sum_{k=1}^{n} (-1)^{k+1} \frac{1}{2k-1} \cos((2k-1)\omega t)$
2: $x_n(t) \leftarrow \frac{4A}{\pi} S$

**Algorithme 1.9** $\mathcal{R}_2$

1: $S \leftarrow 0$
2: **Pour** $k = 1$ **à** $n$ **faire**
3: $\quad S \leftarrow S + (-1)^{k+1} \frac{1}{2k-1} * \cos((2k-1)\omega t)$
4: **Fin Pour**
5: $x_n(t) \leftarrow \frac{4A}{\pi} S$

Finalement la fonction est

**Algorithme 1.9** Fonction SFT retournant la valeur de la série de Fourier en $t$ tronquée au $n$ premiers termes de l'exercice **??**.

**Données :** $t$ : nombre réel,
$n$ : nombre entier strictement positif
$A,\ \omega$ : deux nombres réels.
**Résultat :** $x$ : un réel.

```
1: Fonction x ← SFT( t, n, A, ω )
2:     S ← 0
3:     Pour k = 1 à n faire
4:         S ← S + ((−1)^(k + 1)) ∗ cos((2 ∗ k − 1) ∗ ω ∗ t)/(2 ∗ k − 1)
5:     Fin Pour
6:     S ← 4 ∗ A ∗ S/π
7: Fin Fonction
```

◇

**Exercice 1.2.4**

Reprendre les trois exercices précédents en utilisant les boucles «tant que».

## 1.3 Principes de «bonne» programmation pour attaquer de «gros» problèmes

Tous les exemples vus sont assez courts. Cependant, il peut arriver que l'on ait des programmes plus longs à écrire (milliers de lignes, voir des dizaines de milliers de lignes). Dans l'industrie, il arrive que des équipes produisent des codes de millions de lignes, dont certains mettent en jeux des vies humaines (contrôler un avion de ligne, une centrale nucléaire, ...). Le problème est évidemment d'écrire des programmes sûrs. Or, *un programme à* 100% *sûr, cela n'existe pas!* Cependant, plus un programme est simple, moins le risque d'erreur est grand : c'est sur cette remarque de bon sens que se basent les «bonnes» méthodes. Ainsi :

*Tout problème compliqué doit être découpé en sous-problèmes plus simples*

Il s'agit, lorsqu'on a un problème $P$ à résoudre, de l'analyser et de le décomposer en un ensemble de problèmes $P_1,\ P_2,\ P_3, \ldots$ plus simples. Puis, $P_1$, est lui-même analysé et décomposé en $P_{11}, P_{12}, \ldots$, et

$P_2$ en $P_{21}, P_{22}$, etc. On poursuit cette analyse jusqu'à ce qu'on n'ait plus que des problèmes élémentaires à résoudre. Chacun de ces problèmes élémentaires est donc traité séparément dans un module, c'est à dire un morceau de programme relativement indépendant du reste. Chaque module sera *testé et validé* séparément dans la mesure du possible et naturellement *largement documenté*. Enfin ces modules élémentaires sont assemblés en modules de plus en plus complexes, jusqu'à remonter au problème initiale. A chaque niveau, il sera important de bien réaliser les phases de test, validation et documentation des modules.

Par la suite, on s'évertue à écrire des algorithmes!
**Ceux-ci ne seront pas optimisés[a]!**

[a] améliorés pour minimiser le nombre d'opérations élémentaires, l'occupation mémoire, ..

# Chapitre 2

## Dérivation numérique

Soit $y : [a,b] \longrightarrow \mathbb{R}$, de classe $\mathcal{C}^1$ et $t \in [a,b]$. La dérivée $y'(t)$ est donnée par

$$\begin{aligned} y'(t) &= \lim_{h\to 0^+} \frac{y(t+h)-y(t)}{h} \\ &= \lim_{h\to 0^+} \frac{y(t)-y(t-h)}{h} \\ &= \lim_{h\to 0} \frac{y(t+h)-y(t-h)}{2h} \end{aligned}$$

**Definition 2.1**

Soit $N \in \mathbb{N}^*$. On appelle **discrétisation régulière de** $[a,b]$ **à** $N$ **pas ou** $N+1$ **points** l'ensemble des points $a+nh$, $n \in [\![0,N]\!]$ où le pas $h$ est donné par $h=(b-a)/N$.

Soient $\{t^n\}_{n\in[\![0,N]\!]}$ une discrétisation régulière de $[a,b]$ et $(Dy)_n$ une approximation de $y'(t^n)$. On appelle

- **différence finie progressive** l'approximation

$$(Dy)_n^P = \frac{y(t^{n+1})-y(t^n)}{h},\ \forall n \in [\![0,N-1]\!] \tag{2.1}$$

- **différence finie rétrograde** l'approximation

$$(Dy)_n^R = \frac{y(t^n)-y(t^{n-1})}{h},\ \forall n \in [\![1,N]\!] \tag{2.2}$$

- **différence finie centrée** l'approximation

$$(Dy)_n^C = \frac{y(t^{n+1})-y(t^{n-1})}{2h},\ \forall n \in [\![1,N-1]\!] \tag{2.3}$$

Pour calculer l'erreur commise par ces approximations, on rappelle le developpement de Taylor-Lagrange

**Proposition 2.2: Développement de Taylor-Lagrange**

Soit $f$ une application $f : [a,b] \longrightarrow \mathbb{R}$. Si $f \in \mathcal{C}^{r+1}([a,b])$ alors

- $\forall (x,y) \in [a,b]^2$ il existe un $\xi \in ]x,y[$ tel que

$$f(x) = f(y) + \sum_{k=1}^{r} \frac{f^{(k)}(y)}{k!}(x-y)^k + \frac{f^{(r+1)}(\xi)}{(r+1)!}(x-y)^{r+1} \tag{2.4}$$

- $\forall x \in [a,b]$, $\forall h \in \mathbb{R}^*$ vérifiant $x+h \in [a,b]$, il existe $\xi \in ]\min(x,x+h), \max(x,x+h)[$ tel quel

$$f(x+h) = f(x) + \sum_{k=1}^{r} \frac{f^{(k)}(x)}{k!}h^k + \frac{f^{(r+1)}(\xi)}{(r+1)!}h^{r+1} \tag{2.5}$$

*Brook Taylor* 1685-1731, Mathématicien anglais

(a) *Joseph-Louis Lagrange* 1736-1813, Mathématicien italien(sarde) puis naturalisé français

**Definition 2.3**

Soit $g$ une fonction. On dit que $g$ **se comporte comme un grand O de** $h^q$ quand h tend vers 0 si et seulement si il existe $H > 0$ et $C > 0$ tel que

$$|g(h)| \leqslant Ch^q, \quad \forall h \in ]-H, H[.$$

On note alors $g(h) = \mathcal{O}(h^q)$.

Avec cette notation le développement de Taylor (2.5) s'écrit aussi

$$f(x+h) = f(x) + \sum_{k=1}^{r} \frac{f^{(k)}(x)}{k!}h^k + \mathcal{O}(h^{r+1}). \tag{2.6}$$

**Exercice 2.0.1**

**Q. 1** *Soit $y \in \mathcal{C}^2([a,b])$.*

1. *Montrer qu'il existe $\eta_P^n \in ]t^n, t^{n+1}[$ et $\eta_R^n \in ]t^{n-1}, t^n[$ tels que*

$$(Dy)_n^P = y^{(1)}(t^n) + \frac{h}{2}y^{(2)}(\eta_P^n)$$

*et*

$$(Dy)_n^R = y^{(1)}(t^n) - \frac{h}{2}y^{(2)}(\eta_R^n)$$

2. *En déduire que*

$$|y^{(1)}(t^n) - (Dy)_n^P| \leqslant C_1 h, \quad \text{avec} \quad C_1 = \frac{1}{2}\max_{t\in[t^n,t^{n+1}]}|y^{(2)}(t)|$$

*et*

$$|y'(t^n) - (Dy)_n^R| \leqslant C_2 h, \quad \textit{avec } C_2 = \frac{1}{2} \max_{t\in[t^{n-1},t^n]} |y^{(2)}(t)|$$

**Q. 2** *Soit $y \in \mathcal{C}^3([a,b])$.*

1. *Montrer qu'il existe $\eta_1^n \in ]t^n, t^{n+1}[$ et $\eta_2^n \in ]t^{n-1}, t^n[$ tels que*

$$(Dy)_n^C = y^{(1)}(t^n) - \frac{h^2}{12}(y^{(3)}(\eta_1^n) + y^{(3)}(\eta_2^n))$$

2. *En déduire que*

$$|y^{(1)}(t^n) - (Dy)_n^C| \leqslant Eh^2, \quad \textit{avec} \quad E = \frac{1}{6} \max_{t\in[t^{n-1},t^{n+1}]} |y^{(3)}(t)|$$

**Correction Exercice 3.2.1**

**Q. 1** 1. Soit $n \in [\![0, N-1]\!]$, on a $t^{n+1} = t^n + h$ et

$$(Dy)_n^P = \frac{y(t^{n+1}) - y(t^n)}{h}.$$

D'après la formule de Taylor (2.5), il existe $\eta_P^n \in [t^n, t^{n+1}]$ tel que

$$y(t^{n+1}) = y(t^n) + h\, y^{(1)}(t^n) + \frac{h^2}{2!}\, y^{(2)}(\eta_P^n) \tag{2.7}$$

d'où

$$(Dy)_n^P = \frac{y(t^{n+1}) - y(t^n)}{h} = y^{(1)}(t^n) + \frac{h}{2!}\, y^{(2)}(\eta_P^n). \tag{2.8}$$

Soit $n \in [\![1, N]\!]$, on a $t^{n-1} = t^n - h$ et

$$(Dy)_n^R = \frac{y(t^n) - y(t^{n-1})}{h}.$$

D'après la formule de Taylor (2.5), il existe $\eta_R^n \in [t^{n-1}, t^n]$ tels que

$$y(t^{n-1}) = y(t^n) - h\, y^{(1)}(t^n) + \frac{(-h)^2}{2!}\, y^{(2)}(\eta_R^n) \tag{2.9}$$

d'où

$$(Dy)_n^R = \frac{y(t^n) - y(t^{n-1})}{h} = y^{(1)}(t^n) - \frac{h}{2}\, y^{(2)}(\eta_R^n) \tag{2.10}$$

2. Soit $n \in [\![0, N-1]\!]$, comme $\eta_P^n \in [t^n, t^{n+1}]$ on a

$$|y^{(2)}(\eta_P^n)| \leqslant \max_{t\in[t^n,t^{n+1}]} |y^{(2)}(t)|$$

d'où, en utilisant (2.8),

$$|(Dy)_n^P - y^{(1)}(t^n)| = \frac{h}{2}|y^{(2)}(\eta_P^n)| \leqslant \frac{h}{2} \max_{t\in[t^n,t^{n+1}]} |y^{(2)}(t)|.$$

De même, comme $\eta_R^n \in [t^{n-1}, t^n]$ on a

$$|y^{(2)}(\eta_R^n)| \leqslant \max_{t\in[t^{n-1},t^{nn}]} |y^{(2)}(t)|$$

d'où, en utilisant (2.10),

$$|(Dy)_n^R - y^{(1)}(t^n)| = \frac{h}{2}|y^{(2)}(\eta_R^n)| \leqslant \frac{h}{2} \max_{t\in[t^{n-1},t^n]} |y^{(2)}(t)|.$$

**Q. 2** 1. Soit $n \in [\![0, N-1]\!]$, on a

$$(Dy)_n^C = \frac{y(t^{n+1}) - y(t^{n-1})}{2h}.$$

D'après la formule de Taylor (2.5), il existe $\eta_1^n \in [t^n, t^{n+1}]$ et $\eta_2^n \in [t^{n-1}, t^n]$ tels que

$$y(t^{n+1}) = y(t^n) + h\, y^{(1)}(t^n) + \frac{h^2}{2!}\, y^{(2)}(t^n) + \frac{h^3}{3!}\, y^{(3)}(\eta_1^n) \tag{2.11}$$

et

$$y(t^{n-1}) = y(t^n) - h\, y^{(1)}(t^n) + \frac{h^2}{2!}\, y^{(2)}(t^n) - \frac{h^3}{3!}\, y^{(2)}(\eta_2^n) \tag{2.12}$$

En soustrayant (2.12) à (2.11), on obtient

$$y(t^{n+1}) - y(t^{n-1}) = 2h\, y^{(1)}(t^n) + \frac{h^3}{6}(y^{(3)}(\eta_1^n) + y^{(3)}(\eta_2^n))$$

d'où

$$y^{(1)}(t^n) = \frac{y(t^{n+1}) - y(t^{n-1})}{2h} - \frac{h^2}{12}(y^{(3)}(\eta_1^n) + y^{(3)}(\eta_2^n)).$$

2. Comme $\eta_1^n \in [t^n, t^{n+1}] \subset [t^{n-1}, t^{n+1}]$, on en déduit que

$$|\, y^{(3)}(\eta_1^n)| \leqslant \max_{t \in [t^{n-1}, t^{n+1}]} |\, y^{(3)}(t)|.$$

De même, comme $\eta_2^n \in [t^{n-1}, t^n] \subset [t^{n-1}, t^{n+1}]$ on a

$$|\, y^{(3)}(\eta_2^n)| \leqslant \max_{t \in [t^{n-1}, t^{n+1}]} |\, y^{(3)}(t)|.$$

◇

**Definition 2.4**

La différence $|y'(t^n) - (Dy)_n|$ est appelée **erreur de troncature au point $t^n$.** On dira que $|y'(t^n) - (Dy)_n|$ est d'ordre $p > 0$ si il existe une constante $C > 0$ telle que

$$|y'(t^n) - (Dy)_n| \leqslant Ch^p \iff |y'(t^n) - (Dy)_n| = \mathcal{O}(h^p).$$

Les erreurs de troncature des différences finies progressive et rétrograde sont d'ordre 1 et l'erreur de troncature de la différence finie centrée est d'ordre 2.

On a démontré le lemme suivant

**Lemme 2.5**

Si $y \in \mathcal{C}^2([a, b])$ alors

$$\left| y'(t^n) - \frac{y(t^{n+1}) - y(t^n)}{h} \right| = \mathcal{O}(h), \quad \forall n \in [\![0, N-1]\!], \tag{2.13}$$

$$\left| y'(t^n) - \frac{y(t^n) - y(t^{n-1})}{h} \right| = \mathcal{O}(h), \quad \forall n \in [\![1, N]\!]. \tag{2.14}$$

Si $y \in \mathcal{C}^3([a, b])$ alors

$$\left| y'(t^n) - \frac{y(t^{n+1}) - y(t^{n-1})}{2h} \right| = \mathcal{O}(h^2), \quad \forall n \in [\![1, N-1]\!]. \tag{2.15}$$

**Exercice 2.0.2**

Soit $f \in \mathcal{C}^3([a,b];\mathbb{R})$. On note $t^n$, $n \in [\![0,N]\!]$, une discrétisation **régulière** de $[a,b]$ de pas $h$. On note $\boldsymbol{F} \in \mathbb{R}^{N+1}$ le vecteur défini par $F_{n+1} = f(t^n)$, $\forall n \in [\![0,N]\!]$.

**Q. 1** *1. Déterminer en fonction de $h$ et $\boldsymbol{F}$, un vecteur $\boldsymbol{V} \in \mathbb{R}^{N+1}$ vérifiant*

$$V_{n+1} = f'(t^n) + \mathcal{O}(h), \quad \forall n \in [\![0,N]\!]$$

*2. Ecrire une fonction algorithmique permettant, à partir du vecteur $\boldsymbol{F}$ et de la discrétisation régulière, de calculer le vecteur $\boldsymbol{V}$ précédant.*

**Q. 2** *1. Connaissant uniquement le vecteur $\boldsymbol{F}$, déterminer un vecteur $\boldsymbol{V} \in \mathbb{R}^{N+1}$ vérifiant*

$$\boldsymbol{W}_n = f'(t^n) + \mathcal{O}(h^2).$$

*2. Ecrire une fonction algorithmique permettant, à partir du vecteur $\boldsymbol{F}$ et de la discrétisation régulière, de calculer le vecteur $\boldsymbol{W}$ précédant.*

**Correction Exercice 2.0.2**

**Q. 1** 1. On a $h = (b-a)/N$ et $t^n = a + nh$, $\forall n \in [\![0,N]\!]$. Par la formule de Taylor, on obtient respectivement les formules progressives et régressives d'ordre 1 suivantes (voir Lemme 2.5)

$$\frac{f(t+h)-f(t)}{h} = f'(t) + \mathcal{O}(h) \quad \text{et} \quad \frac{f(t)-f(t-h)}{h} = f'(t) + \mathcal{O}(h).$$

On va utiliser ces formules en $t = t^n$. On note que la formule progressive n'est pas utilisable en $t = t^N$ (car $t^N + h = b + h \notin [a,b]$!) et que la formule régressive n'est pas utilisable en $t = t^0$ (car $t^0 - h = a - h \notin [a,b]$!). Plus précisement, on a

$$\frac{f(t^{n+1})-f(t^n)}{h} = f'(t^n) + \mathcal{O}(h), \quad \forall n \in [\![0,N[\![, \tag{2.16}$$

$$\frac{f(t^n)-f(t^{n-1})}{h} = f'(t^n) + \mathcal{O}(h), \quad \forall n \in ]\!]0,N]\!] \tag{2.17}$$

On peut alors construire le vecteur $\boldsymbol{V}$ en prenant

$$V_1 \overset{\text{def}}{=} \frac{f(t^1)-f(t^0)}{h} = f'(t^0) + \mathcal{O}(h), \quad \text{formule progressive (2.16) avec } n = 0$$

$$V_{N+1} \overset{\text{def}}{=} \frac{f(t^N)-f(t^{N-1})}{h} = f'(t^N) + \mathcal{O}(h), \quad \text{formule régressive (2.17) avec } n = N$$

et pour les points strictement intérieurs les deux formules sont possible :

$$V_n \overset{\text{def}}{=} \frac{f(t^{n+1})-f(t^n)}{h} = f'(t^n) + \mathcal{O}(h), \quad \text{formule progressive (2.16) avec } n \in ]\!]0,N[\![$$

$$V_n \overset{\text{def}}{=} \frac{f(t^n)-f(t^{n-1})}{h} = f'(t^n) + \mathcal{O}(h), \quad \text{formule régressive (2.17) avec } n \in ]\!]0,N[\![$$

En choisissant par exemple la formule progressive pour les points intérieurs, on obtient en fonction de $\boldsymbol{F}$ et $h$

$$\begin{aligned} V_1 &\overset{\text{def}}{=} \frac{F_2 - F_1}{h} && = f'(t^0) + \mathcal{O}(h) \\ \forall n \in ]\!]0,N[\![,\ V_n &\overset{\text{def}}{=} \frac{F_{n+1} - F_n}{h} && = f'(t^n) + \mathcal{O}(h) \\ V_{N+1} &\overset{\text{def}}{=} \frac{F_{N+1} - F_N}{h} && = f'(t^N) + \mathcal{O}(h). \end{aligned}$$

2. La fonction algorithmique peut s'écrire sous la forme :

**Algorithme 2.1** Calcul de dérivées d'ordre 1

**Données :** $\boldsymbol{F}$ : vecteur de $\mathbb{R}^{N+1}$.
$h$ : nombre réel strictement positif.
**Résultat :** $\boldsymbol{V}$ : vecteur de $\mathbb{R}^{N+1}$.

**Fonction** $\boldsymbol{V} \leftarrow$ DeriveOrdre1 ( $h, \boldsymbol{F}$ )
  $V(1) \leftarrow (F(2) - F(1))/h$
  **Pour** $i = 2$ **à** $N$ **faire**
    $V(i) \leftarrow (F(i+1) - F(i))/h$
  **Fin Pour**
  $V(N+1) \leftarrow (F(N+1) - F(N))/h$
**Fin Fonction**

D'autres façons de présenter le problème sont possibles mais les données peuvent différer de celles de l'énoncé. Par exemple une autre fonction algorithmique pourrait être

**Algorithme 2.2** Calcul de dérivées d'ordre 1

**Données :** $f$ : $f : [a, b] \longrightarrow \mathbb{R}$
$a, b$ : deux réels, $a < b$,
$N$ : nombre de pas de discrétisation
**Résultat :** $\boldsymbol{V}$ : vecteur de $\mathbb{R}^{N+1}$ tel que $V_{n+1} = f'(t^n) + \mathcal{O}(h)$
avec $(t^n)_{n=0}^N$ discrétisation régulière de $[a, b]$.

**Fonction** $\boldsymbol{V} \leftarrow$ DeriveOrdre1 ( $f, a, b, N$ )
  $t \leftarrow$ DisReg$(a, b, N)$ ▷ fonction retournant la discrétisation régulière...
  $h \leftarrow (b - a)/N$
  $V(1) \leftarrow (f(t(2)) - f(t(1))/h$
  **Pour** $i = 2$ **à** $N$ **faire**
    $V(i) \leftarrow (f(t(i+1)) - f(t(i)))/h$
  **Fin Pour**
  $V(N+1) \leftarrow (f(t(N+1)) - f(t(N))/h$
**Fin Fonction**

**Q. 2** 1. Du lemme 2.5, on a la formule centrée d'ordre 2

$$\frac{f(t+h) - f(t-h)}{2h} = f'(t) + \mathcal{O}(h^2).$$

On va utiliser cette formule en $t = t^n$. On note que la formule n'est pas utilisable en $t = t^N = b$ et $t = t^0 = a$. On obtient

$$f'(t^n) = \frac{f(t^{n+1}) - f(t^{n-1})}{2h} + \mathcal{O}(h^2), \quad \forall n \llbracket 0, N \llbracket. \tag{2.18}$$

Il reste donc à établir une formule *progressive* en $t^0$ d'ordre 2 (i.e. une formule utilisant les points $t^0$, $t^1$, $t^2$, ...) et une formule *régressive* en $t^N$ d'ordre 2 (i.e. une formule utilisant les points $t^N$, $t^{N-1}$, $t^{N-2}$, ...).

De manière générique pour établir une formule *progressive* en $t$ d'ordre 2 approchant $f'(t)$, on écrit les deux formules de Taylor aux points $t + h$ et $t + 2h$ :

- il existe $\xi_1 \in ]t, t + h[$ tel que

$$f(t+h) = f(t) + hf'(t) + \frac{h^2}{2!} f^{(2)}(t) + \frac{h^3}{3!} f^{(3)}(\xi_1), \tag{2.19}$$

- il existe $\xi_2 \in ]t, t + 2h[$ tel que

$$f(t+2h) = f(t) + (2h)f'(t) + \frac{(2h)^2}{2!} f^{(2)}(t) + \frac{(2h)^3}{3!} f^{(3)}(\xi_2). \tag{2.20}$$

L'objectif étant d'obtenir une formule d'ordre 2 en $t$ pour approcher $f'(t)$, on va éliminer les termes en $f^{(2)}(t)$ en effectuant la combinaison $4 \times (2.19) - (2.20)$. On obtient alors

$$4f(t+h) - f(t+2h) = 3f(t) + 2hf'(t) + 4\frac{h^3}{3!} f^{(3)}(\xi_1) - \frac{(2h)^3}{3!} f^{(3)}(\xi_2)$$

et donc

$$\begin{aligned} f'(t) &= \frac{-3f(t) + 4f(t+h) - f(t+2h)}{2h} - 4\frac{h^2}{3!} f^{(3)}(\xi_1) + \frac{2^3h^2}{3!} f^{(3)}(\xi_2) \\ &= \frac{-3f(t) + 4f(t+h) - f(t+2h)}{2h} + \mathcal{O}(h^2) \end{aligned}$$

Cette dernière formule permet alors l'obtention d'une formule d'ordre 2 en $t = t^0 = a$ :

$$f'(t^0) = \frac{-3f(t^0) + 4f(t^1) - f(t^2)}{2h} + \mathcal{O}(h^2) \tag{2.21}$$

De la même manière pour établir une formule *régressive* en $t$ d'ordre 2 approchant $f'(t)$, on écrit les deux formules de Taylor aux points $t-h$ et $t-2h$ :

- il existe $\xi_1 \in ]t-h, t[$ tel que

$$f(t-h) = f(t) + (-h)f'(t) + \frac{(-h)^2}{2!} f^{(2)}(t) + \frac{(-h)^3}{3!} f^{(3)}(\xi_1), \tag{2.22}$$

- il existe $\xi_2 \in ]t-2h, t[$ tel que

$$f(t-2h) = f(t) + (-2h)f'(t) + \frac{(-2h)^2}{2!} f^{(2)}(t) + \frac{(-2h)^3}{3!} f^{(3)}(\xi_2). \tag{2.23}$$

L'objectif étant d'obtenir une formule d'ordre 2 en $t$ pour approcher $f'(t)$, on va éliminer les termes en $f^{(2)}(t)$ en effectuant la combinaison $4 \times (2.22) - (2.23)$. On obtient alors

$$4f(t-h) - f(t-2h) = 3f(t) - 2hf'(t) - 4\frac{h^3}{3!} f^{(3)}(\xi_1) + \frac{(2h)^3}{3!} f^{(3)}(\xi_2)$$

et donc

$$\begin{aligned} f'(t) &= \frac{3f(t) - 4f(t-h) + f(t-2h)}{2h} - 4\frac{h^2}{3!} f^{(3)}(\xi_1) + \frac{2^3h^2}{3!} f^{(3)}(\xi_2) \\ &= \frac{3f(t) - 4f(t-h) + f(t-2h)}{2h} + \mathcal{O}(h^2) \end{aligned}$$

Cette dernière formule permet alors l'obtention d'une formule d'ordre 2 en $t = t^N = b$ :

$$f'(t^N) = \frac{3f(t^N) - 4f(t^{N-1}) + f(t^{N-2})}{2h} + \mathcal{O}(h^2) \tag{2.24}$$

On peut alors construire le vecteur $\boldsymbol{W}$ en utilisant les formules (2.18), (2.21) et (2.24) :

$$\begin{aligned} W_1 &\stackrel{\text{def}}{=} \frac{-3f(t^0) + 4f(t^1) - f(t^2)}{2h} &&= f'(t^0) + \mathcal{O}(h^2), && \text{formule progressive (2.21)} \\ W_{N+1} &\stackrel{\text{def}}{=} \frac{3f(t^N) - 4f(t^{N-1}) + f(t^{N-2})}{2h} &&= f'(t^N) + \mathcal{O}(h^2), && \text{formule régressive (2.24)} \end{aligned}$$

et pour les points strictement intérieurs

$$W_n \stackrel{\text{def}}{=} \frac{f(t^{n+1}) - f(t^{n-1})}{2h} \qquad = f'(t^n) + \mathcal{O}(h^2), \qquad \text{formule centrée (2.18) avec } n \in ]\!]0, N[\!]$$

On obtient alors en fonction de $\boldsymbol{F}$ et $h$

$$\begin{aligned} W_1 &\overset{\text{def}}{=} \frac{-3F_1 + 4F_2 - F_3}{2h} && = f'(t^0) + \mathcal{O}(h^2) \\ \forall n \in ]\!]0, N[\![,\ W_n &\overset{\text{def}}{=} \frac{F_{n+1} - F_{n-1}}{2h} && = f'(t^n) + \mathcal{O}(h^2) \\ W_{N+1} &\overset{\text{def}}{=} \frac{3F_{N+1} - 4F_N + F_{N-1}}{2h} && = f'(t^N) + \mathcal{O}(h). \end{aligned}$$

2. La fonction algorithmique peut s'écrire sous la forme :

**Algorithme 2.3** Calcul de dérivées d'ordre 2

**Données :** $\boldsymbol{F}$ : vecteur de $\mathbb{R}^{N+1}$.
$h$ : nombre réel strictement positif.
**Résultat :** $\boldsymbol{W}$ : vecteur de $\mathbb{R}^{N+1}$.

```
1: Fonction W ← DeriveOrdre2 ( h, F )
2:     W(1) ← (−3 ∗ F(1) + 4 ∗ F(2) − F(3))/(2 ∗ h)
3:     Pour i = 2 à N faire
4:         W(i) ← (F(i + 1) − F(i − 1))/(2 ∗ h)
5:     Fin Pour
6:     W(N + 1) ← (3 ∗ F(N + 1) − 4 ∗ F(N) + F(N − 1))/(2 ∗ h)
7: Fin Fonction
```

D'autres façons de présenter le problème sont possibles mais les données peuvent différer de celles de l'énoncé. Par exemple une autre fonction algorithmique pourrait être

**Algorithme 2.4** Calcul de dérivées d'ordre 2

**Données :** $f$ : $f : [a, b] \longrightarrow \mathbb{R}$
$a, b$ : deux réels, $a < b$,
$N$ : nombre de pas de discrétisation
**Résultat :** $\boldsymbol{W}$ : vecteur de $\mathbb{R}^{N+1}$ tel que $W_{n+1} = f'(t^n) + \mathcal{O}(h^2)$
avec $(t^n)_{n=0}^N$ discrétisation régulière de $[a, b]$.

```
1: Fonction W ← DeriveOrdre2 ( f, a, b, N )
2:     t ← DisReg(a, b, N)                ▷ fonction retournant la discrétisation régulière...
3:     h ← (b − a)/N
4:     W(1) ← (−3 ∗ f(t(1)) + 4 ∗ f(t(2)) − f(t(3)))/(2 ∗ h)
5:     Pour i = 2 à N faire
6:         W(i) ← (f(t(i + 1)) − f(t(i − 1)))/(2 ∗ h)
7:     Fin Pour
8:     W(N + 1) ← (3 ∗ f(t(N + 1)) − 4 ∗ f(t(N)) + f(t(N − 1)))/(2 ∗ h)
9: Fin Fonction
```

◇

Pour illustrer l'intéret de *monter* en ordre, on représente en Figure 2.2 les dérivées numériques calculées avec les fonctions DeriveOrdre1 et DeriveOrdre2 . On peut noter visuellement la meilleur concordance de la dérivée numérique d'ordre 2 sur la figure de gauche. Sur celle de droite, c'est moins clair car les différentes courbes sont presque confondues.

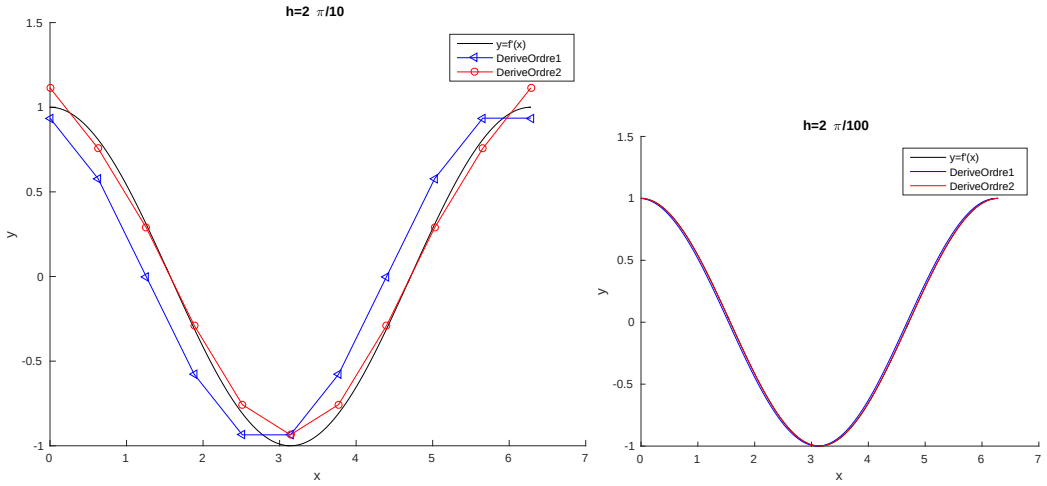



Figure 2.2: Représentation des dérivées numériques avec $f(x) := sin(x)$, $a = 0$, $b = 2\pi$. A gauche $h = \frac{2\pi}{10}$, à droite $h = \frac{2\pi}{100}$.

Pour une meilleur interprétation et connaissant la dérivée de la fonction, on représente en Figure 2.3 les erreurs entre les dérivées numériques et la dérivée exacte.

- Pour la dérivée d'ordre 1, l'erreur maximale pour $h = \frac{2\pi}{10}$ est d'environ 0.3 et pour $h = \frac{2\pi}{100}$ d'environ 0.03 : le pas $h$ a été divisé par 10 et comme la méthode est d'odre 1 l'erreur, qui est en $\mathcal{O}(h)$, est aussi divisée par 10.

- Pour la dérivée d'ordre 2, l'erreur maximale pour $h = \frac{2\pi}{10}$ est d'environ 0.1 et pour $h = \frac{2\pi}{100}$ d'environ 0.001 : le pas $h$ a été divisé par 10 et comme la méthode est d'ordre 2 l'erreur, qui est en $\mathcal{O}(h^2)$, est divisée par 100!

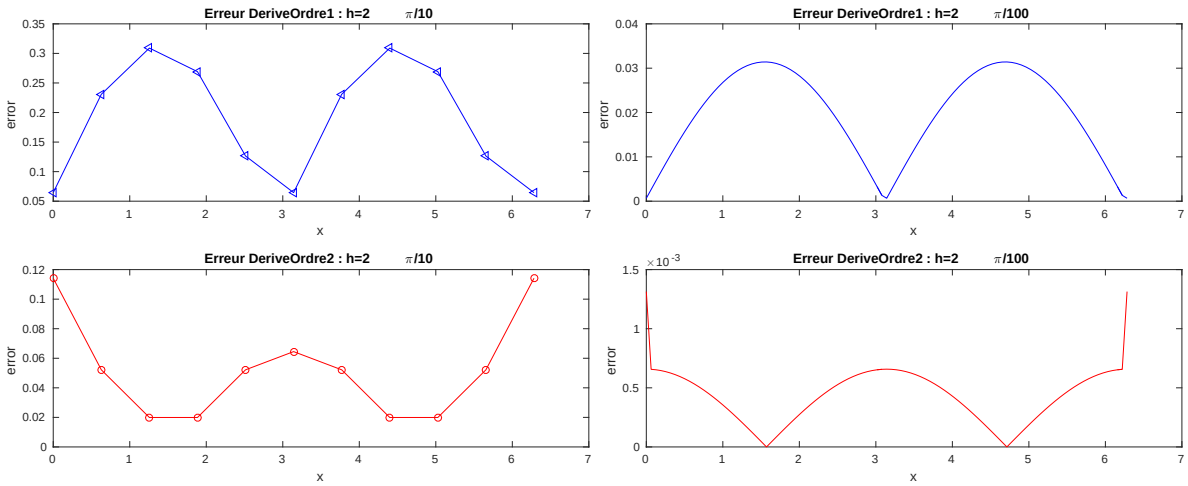



Figure 2.3: Erreur des dérivées numériques numériques avec $f(x) := sin(x)$, $a = 0$, $b = 2\pi$. A gauche $h = \frac{2\pi}{10}$, à droite $h = \frac{2\pi}{100}$.

La Figure 2.4 en échelle logarithmique permet de se rendre compte graphiquement de l'intéret de *monter* en ordre.

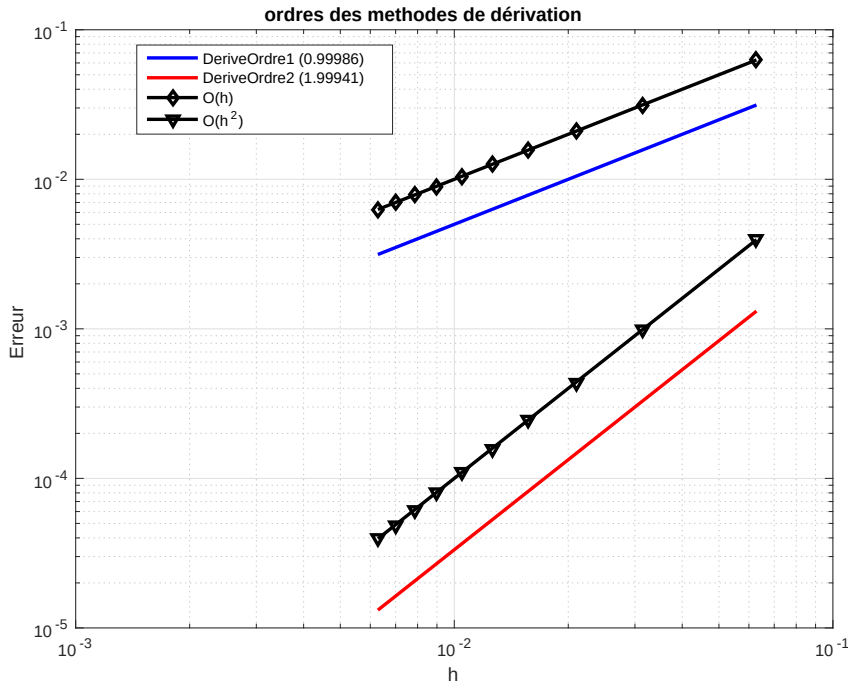



Figure 2.4: Dérivation numérique : mise en évidence de l'ordre des méthodes

# Chapitre 3

# Introduction à la résolution d'E.D.O.

## 3.1 Introduction

Les équations différentielles ordinaires ou E.D.O.[1] sont utilisées pour modéliser un grand nombre de phénomènes mécaniques, physiques, chimiques, biologiques, ...

**Definition 3.1**

On appelle **équation différentielle ordinaire (E.D.O.) d'ordre** $p$ une équation de la forme :

$$\mathcal{F}(t, \boldsymbol{y}(t), \boldsymbol{y}^{(1)}(t), \boldsymbol{y}^{(2)}(t), \ldots, \boldsymbol{y}^{(p)}(t)) = 0.$$

**Definition 3.2**

On appelle **forme canonique d'une E.D.O.** une expression du type :

$$\boldsymbol{y}^{(p)}(t) = \boldsymbol{\mathcal{G}}(t, \boldsymbol{y}(t), \boldsymbol{y}^{(1)}(t), \boldsymbol{y}^{(2)}(t), \ldots, \boldsymbol{y}^{(p-1)}(t)). \tag{3.1}$$

**Proposition 3.3**

Toute équation différentielle d'ordre $p$ sous forme canonique peut s'écrire comme un système de $p$ équations différentielles d'ordre 1.

### 3.1.1 Exemple en météorologie : modèle de Lorentz

Mathématiquement, le couplage de l'atmosphère avec l'océan est décrit par le système d'équations aux dérivées partielles couplées de Navier-Stokes de la mécanique des fluides. Ce système d'équations était beaucoup trop compliqué à résoudre numériquement pour les premiers ordinateurs existant au temps

[1] En anglais, *ordinary differential equations* ou O.D.E.

de Lorenz. Celui-ci eut donc l'idée de chercher un modèle très simplifié de ces équations pour étudier une situation physique particulière : le phénomène de convection de Rayleigh-Bénard. Il aboutit alors à un système dynamique différentiel possédant seulement trois degrés de liberté, beaucoup plus simple à intégrer numériquement que les équations de départ.

(a) *Edward Norton Lorenz* 1917-2008, Mathématicien et météorologiste américain

$$\begin{cases} x'(t) &= -\sigma x(t) + \sigma y(t) \\ y'(t) &= -x(t)y(t) + \rho x(t) - y(t) \\ z'(t) &= x(t)y(t) - \beta z(t) \end{cases} \tag{3.2}$$

avec $\sigma = 10$, $\rho = 28$, $\beta = 8/3$ et les données initales $x(0) = -8, y(0) = 8$ et $z(0) = \rho - 1$. C'est un système de trois E.D.O. d'ordre 1.
Dans ces équations, $\sigma$,$\rho$ et $\beta$ sont trois paramètres réels.

$x(t)$ est proportionnel à l'intensité du mouvement de convection, $y(t)$ est proportionnel à la différence de température entre les courants ascendants et descendants, et $z(t)$ est proportionnel à l'écart du profil de température vertical par rapport à un profil linéaire (Lorenz 1963 p.135).

Pour illustrer le **caractère chaotique** de ce système différentiel, on le résoud numériquement avec les données initiales *exactes* (courbe bleue de la figure 3.2) et avec la donnée initiale $x(0)$ *perturbée* : $x(0) = -8 + 1e - 4$ (courbe rouge de la figure 3.2). Une très légère variation des données initiales engendrent de très forte variation de la solution.

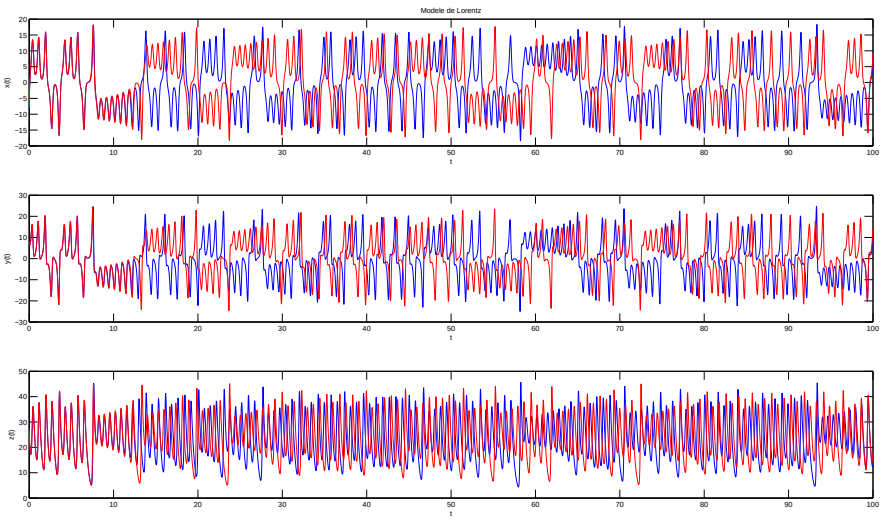



Figure 3.2: Illustration du caractère chaotique du modèle de Lorenz : donnée initiale courbe bleue $x(0) = -8, y(0) = 8, z(0) = 27$, courbe rouge $x(0) = -8 + 1e - 4, y(0) = 8, z(0) = 27$.

En représentant, en Figure 3.3, la courbe paramétré $(x(t), y(t), z(t))$ dans l'espace, on obtient l'*attracteur étrange de Lorenz* en forme d'aile de papillon.

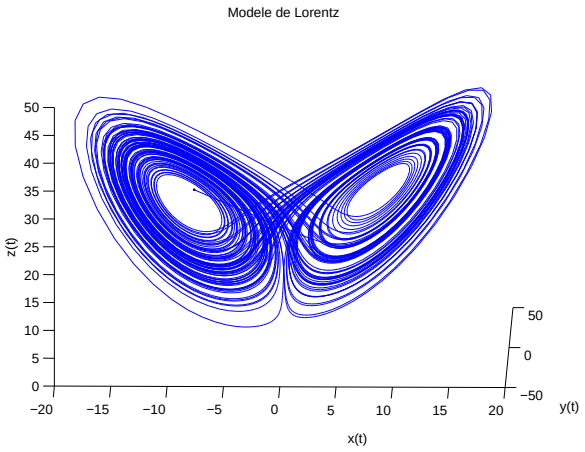



Figure 3.3: Attracteur étrange de Lorenz

### 3.1.2 Exemple en biologie

Considérons une population $y$ d'animaux dans un milieu ambient où au plus $B$ animaux peuvent coexister. On suppose que initialement la population soit $y0 << B$ et que le facteur de croissance des animaux soit égal à une constante $C$. Dans ce cas, l'évolution de la population au cours du temps sera proportionnelle au nombre d'animaux existants, sans toutefois que ce nombre ne dépasse la limite $B$. Cela peut s'exprimer à travers l'équation

$$y'(t) = Cy(t)\left(1 - \frac{y(t)}{B}\right),\ t > 0,\ y(0) = y0. \tag{3.3}$$

La résolution de cette équation permet de trouver l'évolution de la population au cours du temps.

On considère maintenant deux populations, $y_1$ et $y_2$, où $y_1$ sont les proies et $y_2$ sont les prédateurs. L'évolution des deux populations est alors décrite par le système d'équations différentielles

$$\begin{cases} y_1'(t) &= C_1 y_1(t)[1 - b_1 y_1(t) - d_2 y_2(t)], \\ y_2'(t) &= -C_2 y_2(t)[1 - d_1 y_1(t)], \end{cases} \tag{3.4}$$

où $C_1$ et $C_2$ sont les facteurs de croissance des deux populations, $d_1$ et $d_2$ tiennent compte de l'interaction entre les deux populations, tandis que $b_1$ est lié à la quantité de nourriture disponible pour la population des proies $y_1$. Ce système de deux équations différentielles d'odre 1 est connu comme modèle de *Lotka-Volterra*.

### 3.1.3 Exemple en chimie : La réaction de Belousov-Zhabotinsky

Sous certaines conditions, des réactions chimiques peuvent être oscillantes. Par exemple, le mélange d'une solution de bromate de potassium et d'acide sulfurique avec une solution d'acide manolique et de bromure de sodium peut entrainer une oscillation de la couleur de la solution mélange du rouge au bleue avec une période de 7 secondes.

(a) *Boris Pavlovich Belousov* 1893-1970, Chimiste et biophysicien russe

(b) *Anatol Zhabotinsky* 1938-2008, Chimiste russe

(c) *Ilya Prigogine* 1917-2003, Physicien et chimiste belge (origine russe). Prix Nobel de chimie en 1977

Un modéle dérivé est nommé **modèle du brusselator** proposé par I. Prigogine (Université libre de Bruxelle) basé sur les équations chimiques :

$$\begin{array}{lcll} A & \longrightarrow & X & (M_1) \\ 2X+Y & \longrightarrow & 3X & (M_2) \\ B+X & \longrightarrow & Y+C & (M_3) \\ X & \longrightarrow & D & (M_4) \end{array}$$

On note $k_i$, $i \in [\![1,4]\!]$ les vitesses de réactions des équations $(M_i)$. On obtient alors le système différentiel suivant :

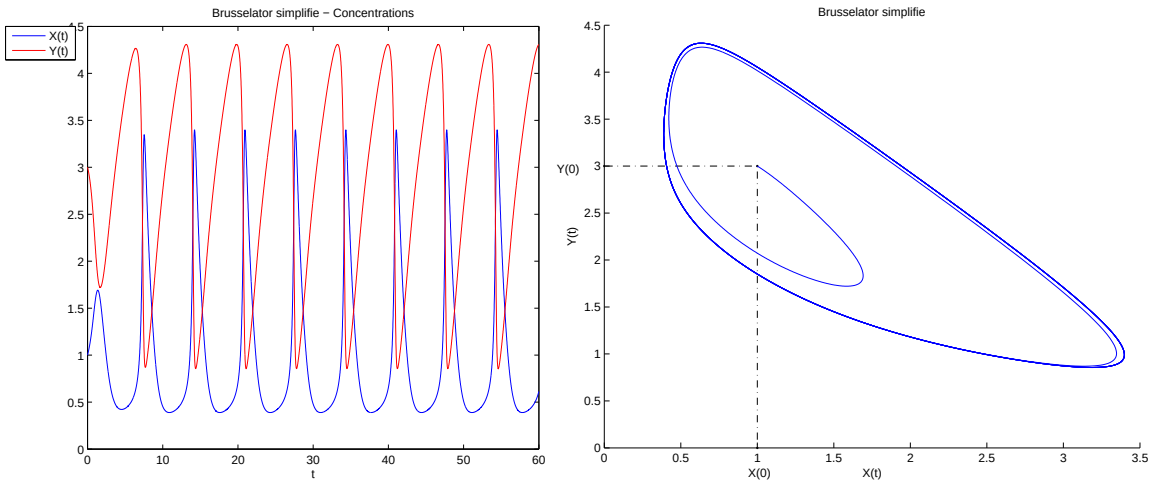
$$\left\{ \begin{array}{lcl} A'(t) & = & -k_1 A(t) \\ B'(t) & = & -k_3 B(t)X(t) \\ X'(t) & = & k_1 A(t) + k_2 X^2(t)Y(t) - k_3 B(t)X(t) - k_4 X(t) \\ Y'(t) & = & -k_2 X^2(t)Y(t) + k_3 B(t)X(t) \end{array} \right.$$

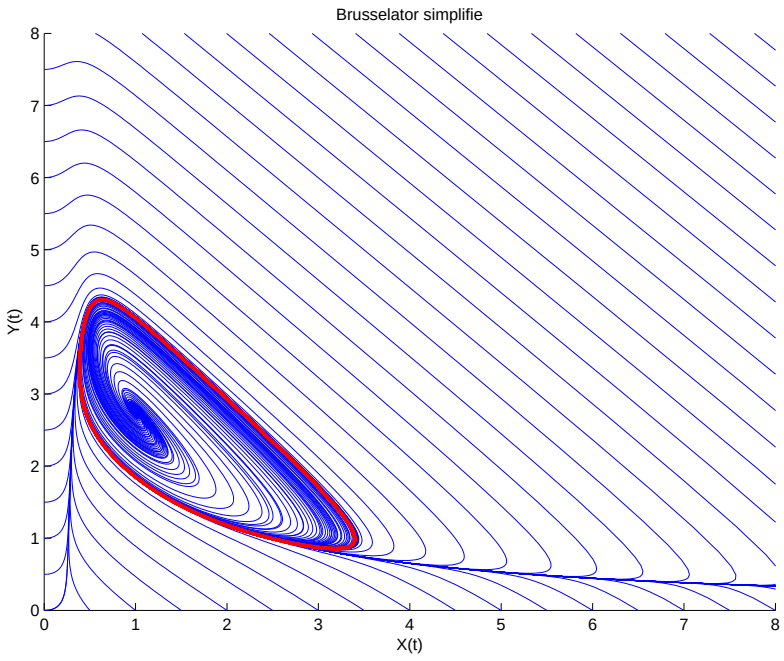
**Exemple 3.4** Dans cet exemple, on étudie le problème simplifié du Bruxelator. Lorsque les réactions de (??) ont des constantes de réactions $k_1, \ldots, k_4$ égales respectivement à 1, $\alpha > 0$, 1 et 1, et que les concentrations de $A$ et $B$ sont constantes, repectivement égales à 1 et $\beta > 0$, on est alors amené à résoudre l'E.D.O. $\forall t \in ]0, T]$,

$$\left\{ \begin{array}{lcl} X'(t) & = & 1 + \alpha X^2(t)Y(t) - (\beta+1)X(t) \\ Y'(t) & = & -\alpha X^2(t)Y(t) + \beta X(t) \end{array} \right. \tag{3.5}$$

C'est un système de deux E.D.O. d'ordre 1.

Par exemple, avec le jeu de données $\alpha = 1$, $\beta = 3.5$ et les contitions initiales $X(0) = 3$ et $Y(0) = 2$ on obtient des oscillations :



### 3.1.4 Exemple en mécanique

Le pendule pesant le plus simple est constitué d'un petit objet pesant accroché à une tige de masse négligeable devant celle de l'objet. L'autre extrémité de la tige est l'axe de rotation du pendule. On note $\theta(t)$ l'angle que fait le pendule par rapport à l'axe vertical, à un instant $t$, $L$ la longueur de la tige, $M$ sa masse et $g$ l'accélération de la pesanteur. Ce pendule est représenté en Figure 3.5.

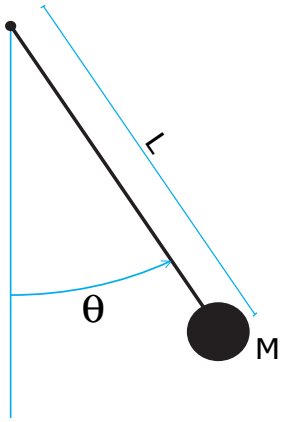



Figure 3.5: Pendule simple

Si le pendule est soumis à un frottement visqueux de coefficient $\nu > 0$, l'équation différentielle est alors

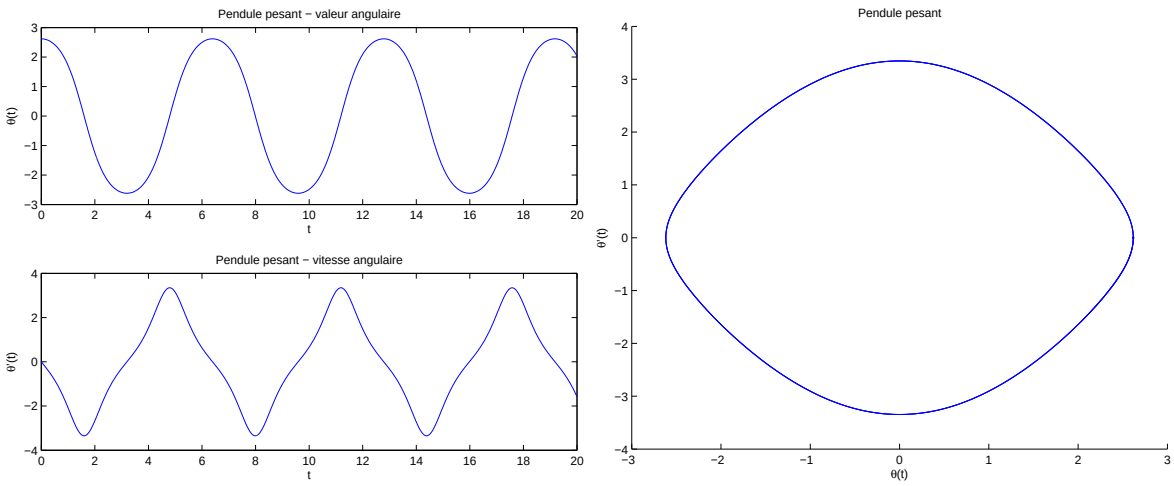
$$\theta''(t) + \frac{g}{L}\sin(\theta(t)) + \frac{\nu}{ML^2}\theta'(t) = 0, \ \forall t \in ]0, T], \tag{3.6}$$

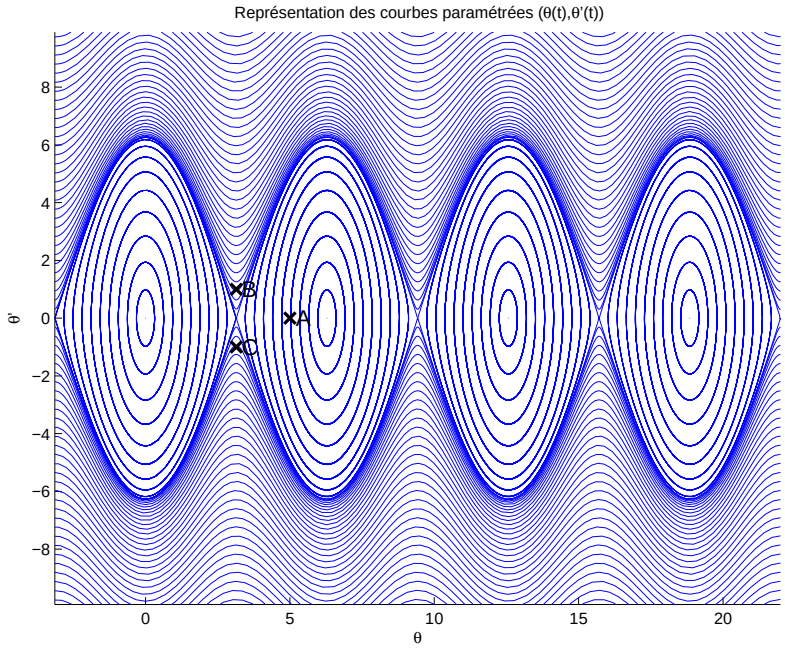
avec les conditions initiales

$$\begin{cases} \theta(0) &= \theta_0, \quad \text{position angulaire initiale,} \\ \theta'(0) &= \theta_0, \quad \text{vitesse angulaire initiale,} \end{cases} \tag{3.7}$$

C'est une E.D.O. d'ordre 2.

Par exemple, dans le cas non visqueux $\nu = 0$, avec le jeu de données $\frac{g}{L} = 3$ et les conditions initiales $\theta_0 = \frac{5\pi}{6}$ et $\theta_0' = 0$ on obtient






## 3.2 Problème de Cauchy

Pour résoudre numériquement une E.D.O., nous allons la réécrire sous une forme plus *générique* : le problème de Cauchy. De très nombreux résultats mathématiques existent sur les problèmes de Cauchy. Nous ne ferons que rappeler le théorème de Cauchy-Lipschitz (19ème siècle). L'ensemble des méthodes numériques que nous allons étudier auront pour but la résolution d'un problème de Cauchy quelconque. Elles pourront donc être utilisées (sans efforts) pour la résolution d'une très grande variété d'E.D.O.

(a) *Augustin Louis Cauchy* 1789-1857, mathématicien français

(b) *Rudolf Lipschitz* 1832-1903, mathématicien allemand

On commence par donner la définition d'un problème de Cauchy :

**Definition 3.5: problème de Cauchy** ★★★★★

Soit $\boldsymbol{f}$ l'application continue définie par

$$\begin{array}{rccc} \boldsymbol{f} & : & [t^0, t^0+T]\times\mathbb{R}^m & \longrightarrow & \mathbb{R}^m \\ & & (t,\boldsymbol{y}) & \longmapsto & \boldsymbol{f}(t,\boldsymbol{y}) \end{array}$$

avec $T \in ]0,+\infty]$. Le **problème de Cauchy** revient à chercher une fonction $\boldsymbol{y}$ définie par

$$\begin{array}{rccc} \boldsymbol{y} & : & [t^0, t^0+T] & \longrightarrow & \mathbb{R}^m \\ & & t & \longmapsto & \boldsymbol{y}(t) \end{array}$$

continue et dérivable, telle que

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t,\boldsymbol{y}(t)), \quad \forall t \in [t^0, t^0+T] \tag{3.8}$$
$$\boldsymbol{y}(t^0) = \boldsymbol{y}^{[0]} \in \mathbb{R}^m. \tag{3.9}$$

**Exercice 3.2.1**

Quelles sont les données du problème de Cauchy (3.8)-(3.9)?

Dans de nombreux cas, la variable $t$ représente le temps et les composantes du vecteur $\boldsymbol{y}$ , une famille de paramètres décrivant l'état d'un système matériel donné. L'équation différentielle (3.8) traduit physiquement la loi d'évolution du système considéré.

En général, il est impossible de trouver analytiquement des solutions à ces problèmes. Il faut alors construire des méthodes numériques pour obtenir des solutions approchées. Toutefois, il serait bon de vérifier l'existence et l'unicité d'une solution.

Par exemple, le problème suivant

$$\left\{\begin{array}{lcl} y'(t) & = & \sqrt[3]{y(t)}, \quad \text{si} \quad t \geqslant 0 \\ y(0) & = & 0 \end{array}\right.$$

peut s'écrire sous la forme d'un problème de Cauchy avec $t^0 = 0$, $T = +\infty$, $m = 1$, $\boldsymbol{y}^{[0]} = 0$ et

$$\begin{array}{rccc} \boldsymbol{f} & : & [t^0, t^0+T] \times \mathbb{R} & \longrightarrow & \mathbb{R} \\ & & (t, v) & \longmapsto & \sqrt[3]{v} \end{array} .$$

Ce problème admet les **trois solutions** suivantes : $y(t) = 0$, $y(t) = \sqrt{8t^3/27}$ et $y(t) = -\sqrt{8t^3/27}$.

Le théorème suivant assure l'existence et l'unicité sous certaines conditions.

**Théorème 3.6: Cauchy-Lipschitz**

Soit le problème de Cauchy donné par la définition 3.5. On suppose que la fonction $\boldsymbol{f}$ est continue sur un ouvert $U$ de $\mathbb{R} \times \mathbb{R}$ et quelle est localement lipschitzienne en $\boldsymbol{y}$ : $\forall (t, \boldsymbol{y}) \in U$, $\exists \mathcal{W}$ voisinage $\boldsymbol{t}$, $\exists \mathcal{V}$ voisinage $\boldsymbol{y}$, $\exists L > 0$ tels que

$$\forall s \in \mathcal{W},\ \forall (\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{V}^2, \quad \|\boldsymbol{f}(s, \boldsymbol{u}) - \boldsymbol{f}(s, \boldsymbol{v})\| \leqslant L \|\boldsymbol{u} - \boldsymbol{v}\| \tag{3.10}$$

Sous ces hypothèses le problème de Cauchy (3.8)-(3.9) admet une unique solution.

**Proposition 3.7**

Si $\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}}(t, \boldsymbol{y})$ est continue et bornée, alors $\boldsymbol{f}$ satisfait la condition de Lipschitz (3.10) en $\boldsymbol{y}$.

**Exercice 3.2.2**

Pour chacune des E.D.O. suivantes écrire le problème de Cauchy associé

$$(a) \left\{\begin{array}{l} x''(t) + \alpha x'(t) + \beta \cos(x(t)) = \sin(t),\ t \in ]0, 2\pi] \\ x(0) = 0,\ x'(0) = 1. \end{array}\right.$$

$$(b) \left\{\begin{array}{l} LCv''(t) + \left(\dfrac{L}{R_2} + R_1 C\right) v'(t) + \left(\dfrac{R_1}{R_2} + 1\right) v(t) = e,\ t \in ]0, 100] \\ v(0) = 0,\ v'(0) = 0. \end{array}\right.$$

$$(c) \left\{\begin{array}{l} x''(t) = \mu(1 - x^2(t))x'(t) - x(t),\ t \in ]0, 10] \\ x(0) = 1,\ x'(0) = 1. \end{array}\right.$$

$$(d) \left\{\begin{array}{l} y^{(3)}(t) - \cos(t) y^{(2)}(t) + 2\sin(t) y^{(1)}(t) - y(t) = 0,\ t \in ]0, T] \\ y(0) = u_0,\ y^{(1)}(0) = v_0,\ y^{(2)}(0) = w_0. \end{array}\right.$$

**Correction Exercice**

(*a*) C'est une E.D.O. d'ordre 2. Pour écrire le problème de Cauchy assosicé, on écrit l'E.D.O. sous la forme d'un système de 2 E.D.O. d'ordre 1 (voir Proposition 3.3) en prenant $m = 2$ et en posant

$$\boldsymbol{y}(t) \stackrel{\text{def}}{=} \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}.$$

On a alors

$$\begin{aligned} \boldsymbol{y}'(t) &= \begin{pmatrix} x'(t) \\ x''(t) \end{pmatrix} = \begin{pmatrix} x'(t) \\ -\alpha x'(t) - \beta\cos(x(t)) + \sin(t) \end{pmatrix} \\ &= \begin{pmatrix} y_2(t) \\ -\alpha y_2(t) - \beta\cos(y_1(t)) + \sin(t) \end{pmatrix} = \boldsymbol{f}(t, \boldsymbol{y}(t)) \end{aligned}$$

Le problème de Cauchy associé est donc

trouver la fonction $\boldsymbol{y} : [0, 2\pi] \longrightarrow \mathbb{R}^2$ vérifiant

$$\begin{aligned} \boldsymbol{y}'(t) &= \boldsymbol{f}(t, \boldsymbol{y}(t)), \quad \forall t \in [0, 2\pi] \\ \boldsymbol{y}(0) &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \in \mathbb{R}^2 \end{aligned}$$

avec

$$\begin{aligned} \boldsymbol{f} \ : \ [0, 2\pi] \times \mathbb{R}^2 &\longrightarrow \mathbb{R}^2 \\ (t, \boldsymbol{z}) &\longmapsto \begin{pmatrix} z_2 \\ -\alpha z_2 - \beta\cos(z_1) + \sin(t) \end{pmatrix} \end{aligned}$$

(*b*) Pour cette E.D.O. on suppose les paramètres physiques $L$, $C$, $R_1$ et $R_2$ donnés. C'est une E.D.O. d'ordre 2. Pour écrire le problème de Cauchy assosicé, on écrit l'E.D.O. sous la forme d'un système de 2 E.D.O. d'ordre 1 (voir Proposition 3.3) en prenant $m = 2$ et en posant

$$\boldsymbol{y}(t) \stackrel{\text{def}}{=} \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ v'(t) \end{pmatrix}.$$

On a alors

$$\begin{aligned} \boldsymbol{y}'(t) &= \begin{pmatrix} v'(t) \\ v''(t) \end{pmatrix} = \begin{pmatrix} v'(t) \\ \frac{1}{LC}\left(e - (\frac{L}{R_2} + R_1 C)v'(t) - (\frac{R_1}{R_2} + 1)v(t)\right) \end{pmatrix} \\ &= \begin{pmatrix} y_2(t) \\ \frac{e}{LC} - (\frac{1}{CR_2} + \frac{R_1}{L})y_2(t) - \frac{1}{LC}(\frac{R_1}{R_2} + 1)y_1(t) \end{pmatrix} = \boldsymbol{f}(t, \boldsymbol{y}(t)) \end{aligned}$$

Le problème de Cauchy associé est donc

trouver la fonction $\boldsymbol{y} : [0, 100] \longrightarrow \mathbb{R}^2$ vérifiant

$$\begin{aligned} \boldsymbol{y}'(t) &= \boldsymbol{f}(t, \boldsymbol{y}(t)), \quad \forall t \in [0, 100] \\ \boldsymbol{y}(0) &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} \in \mathbb{R}^2 \end{aligned}$$

avec

$$\begin{aligned} \boldsymbol{f} \ : \ [0, 100] \times \mathbb{R}^2 &\longrightarrow \mathbb{R}^2 \\ (t, \boldsymbol{z}) &\longmapsto \begin{pmatrix} z_2 \\ \frac{e}{LC} - (\frac{1}{CR_2} + \frac{R_1}{L})z_2 - \frac{1}{LC}(\frac{R_1}{R_2} + 1)z_1 \end{pmatrix} \end{aligned}$$

(*c*) Pour cette E.D.O. on suppose le paramètre $\mu$ donné. C'est une E.D.O. d'ordre 2. Pour écrire le problème de Cauchy assosicé, on écrit l'E.D.O. sous la forme d'un système de 2 E.D.O. d'ordre 1 (voir Proposition 3.3) en prenant $m = 2$ et en posant

$$\boldsymbol{y}(t) \stackrel{\text{def}}{=} \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix} = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}.$$

On a alors

$$\begin{aligned}\boldsymbol{y}'(t) &= \begin{pmatrix} x'(t) \\ x''(t) \end{pmatrix} = \begin{pmatrix} x'(t) \\ \mu(1-x^2(t))x'(t) - x(t) \end{pmatrix} \\ &= \begin{pmatrix} y_2(t) \\ \mu(1-y_1^2(t))y_2(t) - y_1(t) \end{pmatrix} = \boldsymbol{f}(t, \boldsymbol{y}(t))\end{aligned}$$

Le problème de Cauchy associé est donc

trouver la fonction $\boldsymbol{y} : [0,10] \longrightarrow \mathbb{R}^2$ vérifiant

$$\begin{aligned}\boldsymbol{y}'(t) &= \boldsymbol{f}(t, \boldsymbol{y}(t)), \quad \forall t \in [0,10] \\ \boldsymbol{y}(0) &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} \in \mathbb{R}^2\end{aligned}$$

avec

$$\begin{aligned}\boldsymbol{f} : [0,10] \times \mathbb{R}^2 &\longrightarrow \mathbb{R}^2 \\ (t, \boldsymbol{z}) &\longmapsto \begin{pmatrix} z_2 \\ \mu(1-z_1^2)z_2 - z_1 \end{pmatrix}\end{aligned}$$

(*d*) Pour cette E.D.O. on suppose les paramètres $T$, $u_0$, $v_0$ et $w_0$ donnés. C'est une E.D.O. d'ordre 3. Pour écrire le problème de Cauchy assosicé, on écrit l'E.D.O. sous la forme d'un système de 3 E.D.O. d'ordre 1 (voir Proposition 3.3) en prenant $m = 3$ et en posant

$$\boldsymbol{Y}(t) \stackrel{\text{def}}{=} \begin{pmatrix} Y_1(t) \\ Y_2(t) \\ Y_3(t) \end{pmatrix} = \begin{pmatrix} y(t) \\ y'(t) \\ y''(t) \end{pmatrix}.$$

On a noté ici $\boldsymbol{Y}$ au lieu de $\boldsymbol{y}$ pour éviter les confusions! On a alors

$$\begin{aligned}\boldsymbol{Y}'(t) &= \begin{pmatrix} y'(t) \\ y^{(2)}(t) \\ y^{(3)}(t) \end{pmatrix} = \begin{pmatrix} y'(t) \\ y^{(2)}(t) \\ \cos(t)y^{(2)}(t) - 2\sin(t)y^{(1)}(t) + y(t)) \end{pmatrix} \\ &= \begin{pmatrix} Y_2(t) \\ Y_3(t) \\ \cos(t)Y_3(t) - 2\sin(t)Y_2(t) + Y_1(t)) \end{pmatrix} = \boldsymbol{f}(t, \boldsymbol{Y}(t))\end{aligned}$$

Le problème de Cauchy associé est donc

trouver la fonction $\boldsymbol{y} : [0,T] \longrightarrow \mathbb{R}^3$ vérifiant

$$\begin{aligned}\boldsymbol{y}'(t) &= \boldsymbol{f}(t, \boldsymbol{y}(t)), \quad \forall t \in [0,T] \\ \boldsymbol{y}(0) &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} \in \mathbb{R}^2\end{aligned}$$

avec

$$\begin{aligned}\boldsymbol{f} : [0,T] \times \mathbb{R}^3 &\longrightarrow \mathbb{R}^3 \\ (t, \boldsymbol{z}) &\longmapsto \begin{pmatrix} z_2 \\ z_3 \\ \cos(t)z_3 - 2\sin(t)z_2 + z_1) \end{pmatrix}\end{aligned}$$

◇

## 3.3 Différences finies pour les E.D.O.

### 3.3.1 Différences finies pour le problème de Cauchy en dimension $m = 1$

On veut résoudre numériquement le problème de Cauchy (3.8)-(3.9) en utilisant les différences finies progressive, rétrograde ou centrée.

On note $t^n = t^0 + nh$, $n \in \{0, \dots, N\}$ une **discrétisation régulière de** $[t^0, t^0 + T]$ avec $h = T/N$ le **pas de temps**. On a

$$y'(t) = f(t, y(t)),\ \forall t \in [t^0, t^0 + T]$$

ce qui entraine

$$y'(t^n) = f(t^n, y(t^n)),\ \forall n \in [\![0, N]\!]. \tag{3.11}$$

En utilisant la formule des différences finies progressive, on a $\forall n \in [\![0, N-1]\!]$, $\exists \eta_n \in [t^n, t^{n+1}]$ tels que

$$\frac{y(t^{n+1}) - y(t^n)}{h} = f(t^n, y(t^n)) + \frac{h}{2} y''(\eta_n)$$

ou encore

$$y(t^{n+1}) = y(t^n) + hf(t^n, y(t^n)) + \frac{h^2}{2} y''(\eta_n).$$

On note $y^{[n]}$ une approximation de $y(t^n)$ pour $n \in [\![0, N]\!]$.

La méthode d'**Euler progressive** est donnée par le schéma

$$\left\{ \begin{array}{lcl} y^{[n+1]} & = & y^{[n]} + hf(t^n, y^{[n]}),\ \forall n \in [\![0, N-1]\!] \\ y^{[0]} & = & y(t^0) \end{array} \right. \tag{3.12}$$

Ce schéma est **explicite**, car il permet le calcul direct de $y^{[n+1]}$ en fonction de $y^{[n]}$.

De la même manière, la méthode d'**Euler régressive** est donnée par le schéma

$$\left\{ \begin{array}{lcl} y^{[n+1]} & = & y^{[n]} + hf(t^{n+1}, y^{[n+1]}),\ \forall n \in [\![0, N-1]\!] \\ y^{[0]} & = & y(t^0) \end{array} \right. \tag{3.13}$$

Ce schéma est **implicite**, car $y^{[n+1]}$ est définit implicitement en fonction de $y^{[n]}$. Il faut donc résoudre à chaque pas de temps une équation non-linéaire en utilisant des méthodes de point fixe par exemple.

**Exercice 3.3.1**

On veut résoudre numériquement le problème $(\mathcal{P})$ suivant : trouver $y$ telle que

$$(\mathcal{P}) \quad \left\{ \begin{array}{lcl} y'(t) & = & \cos(t) + 1,\ \forall t \in [0, 4\pi] \\ y(0) & = & 0. \end{array} \right.$$

dont la solution exacte est $y(t) = \sin(t) + t$.

On rappelle le schéma d'Euler progressif pour la résolution d'un problème de Cauchy

$$(\mathcal{S}) \quad \left\{ \begin{array}{lcl} y^{(n+1)} & = & y^{(n)} + hf(t^n, y^{(n)}), \\ y^{(0)} & & \text{donné.} \end{array} \right.$$

**Q. 1** *Expliquer en détail comment utiliser le schéma d'Euler progressif pour résoudre le problème $(\mathcal{P})$ en précisant entre autres les données, les inconnues, les dimensions des variables, lien entre $y^{(n)}$ et la fonction $y$, ...*

**Q. 2** *Soit $a, b$, $a < b$ deux réels. Ecrire une fonction* DisReg *retournant une discrétisation de l'intervalle $[a; b]$ avec $N$ pas (constant) de discrétisation.*

**Q. 3** *Ecrire une fonction* REDEP *retournant l'ensemble des couples $(t^n, y^{(n)})$ calculés par le schéma d'Euler progressif.*

**Q. 4** *Ecrire un algorithme complet de résolution de $(\mathcal{P})$ par le schéma d'Euler progressif.*

**Correction Exercice 3.3.1**

**Q. 1** On commence par écrire le problème de cauchy associé à $(\mathcal{P})$ :

$$(\mathcal{PC}) \quad \left\{ \begin{array}{lcl} y'(t) & = & f(t, y(t)), \forall t \in [t^0, t^0 + T] \\ y(t^0) & = & y_0 \in \mathbb{R}. \end{array} \right.$$

avec $t^0 = 0$, $T = 4\pi$, $y_0 = 0$ et

$$\begin{array}{rccc} f & : & [t^0, t^0 + T] \times \mathbb{R} & \longrightarrow & \mathbb{R} \\ & & (t, z) & \longmapsto & \cos(t) + 1 \end{array}.$$

Les données du problème de Cauchy sont donc les réels $t^0$, $T$, $y_0$ et la fonction $f$. L'inconnue est la fonction $y : [t^0, t^0 + T] \longrightarrow \mathbb{R}$.
Pour résoudre numériquement le problème de Cauchy, on utilise le schéma $(\mathcal{S})$ où les données sont celles du problème de Cauchy plus le nombre de discrétisations $N \in \mathbb{N}^*$ On peux alors calculer

- $t^n$, $n \in [\![0, N]\!]$ qui sont les points de la discrétisation régulière à $N$ intervalles :

$$t^n = t^0 + nh, \ \forall n \in [\![0, N]\!], \quad \text{avec } h = \frac{T}{N}.$$

- $y^{(n)}$, $n \in [\![0, N]\!]$ déterminés par le schéma $(\mathcal{S})$. On a $y^{(0)} = y^0$, puis on calcule

$$y^{(n+1)} = y^{(n)} + hf(t^n, y^{(n)}), \quad \text{pour } \ i = 0 \text{ à } N - 1$$

**Q. 2** Une discrétisation régulière de l'intervalle $[a, b]$ avec $N$ pas (constant) de discrétisation est donnée par

$$t^n = a + nh, \ \forall n \in [\![0, N]\!], \quad \text{avec } h = \frac{b - a}{N}.$$

**Algorithme 3.1** Fonction DisReg retournant une discrétisation régulière de l'intervalle $[a, b]$

**Données :** $a, b$ : deux réels, $a < b$
$N$ : un entier non nul (nombre de pas de discrétisation).
**Résultat :** $\boldsymbol{t}$ : vecteur de $\mathbb{R}^{N+1}$

```
1: Fonction t ← DisReg ( a, b, N )
2:     h ← (b − a)/N
3:     Pour n ← 0 à N faire
4:         t(n + 1) ← a + n ∗ h
5:     Fin Pour
6: Fin Fonction
```

**Q. 3** L'algorithme de la fonction REDEP est :

**Algorithme 3.2** Fonction REDEP : résolution d'un problème de Cauchy scalaire par le schéma d'Euler progressif

**Données :** $f$ : $f : [t^0, t^0 + T] \times \mathbb{R} \longrightarrow \mathbb{R}$ fonction d'un problème de Cauchy (scalaire)
$t^0$ : réel, temps initial
$T$ : réel $> 0$
$y^0$ : réel, donnée initiale
$N$ : un entier non nul (nombre de pas de discrétisation).
**Résultat :** $\boldsymbol{t}$ : vecteur de $\mathbb{R}^{N+1}$, $\boldsymbol{t}(n) = t^{n-1}$, $\forall n \in [\![1, N + 1]\!]$
$\boldsymbol{Y}$ : vecteur de $\mathbb{R}^{N+1}$, $\boldsymbol{Y}(n) = y^{(n-1)}$, $\forall n \in [\![1, N + 1]\!]$

```
1: Fonction [t,Y] ← REDEP ( f, t^0, T, y^0, N )
2:     t ← DisReg(t^0, t^0 + T, N)
3:     h ← (b − a)/N
4:     Y(1) ← y^0
5:     Pour n ← 1 à N faire
6:         Y(n + 1) ← Y(n) + h ∗ f(t(n), Y(n))
7:     Fin Pour
8: Fin Fonction
```

**Q. 4** Il faut tout d'abord écrire la fonction fCauchy correspondant à la fonction $f$:

**Algorithme 3.3** Fonction FCAUCHY : fonction $f$ du problème de Cauchy associé à $(\mathcal{P})$

**Données :** $t$ : un réel
$z$ : un réel
**Résultat :** $w$ : un réel

1: **Fonction** $w \leftarrow$ FCAUCHY $(t, y)$
2: $w \leftarrow \cos(t) + 1$
3: **Fin Fonction**

L'algorithme de résolution est :

**Algorithme 3.4** résolution numérique du problème $(\mathcal{P})$

1: $t^0 \leftarrow 0$
2: $T \leftarrow 4\pi$
3: $y^0 \leftarrow 0$
4: $[\boldsymbol{t}, \boldsymbol{Y}] \leftarrow$ REDEP$(fCauchy, t^0, T, y^0, 500)$

◇

**Exemple**

Soit l'E.D.O. suivante

$$\begin{cases} y'(t) &= y(t) + t^2 y^2(t), \quad \text{pour} \quad t \in [0,5], \\ y(0) &= -1 \end{cases}$$

de solution exacte

$$y(t) = 1/(e^{-t} - t^2 + 2t - 2).$$

Les résolutions numériques avec les schémas d'Euler progressif et rétrograde sont représentés en figure 3.7.

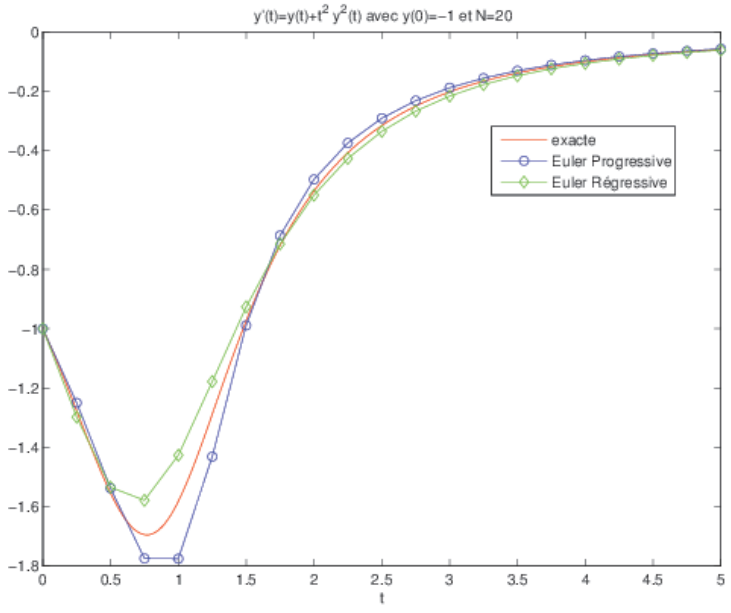



Figure 3.7: résolutions avec Euler progressif et rétrograde

### 3.3.2 Différences finies pour le problème de Cauchy en dimension $m$

On veut résoudre le problème de Cauchy :

$$(\mathcal{PC}) \quad \begin{cases} \boldsymbol{y}'(t) &= \boldsymbol{f}(t,\boldsymbol{y}(t)), \quad \forall t \in [t^0, t^0+T] \\ \boldsymbol{y}(t^0) &= \boldsymbol{y}_0 \in \mathbb{R}^m. \end{cases}$$

La fonction $\boldsymbol{f}$ étant définie par

$$\begin{array}{rccl} \boldsymbol{f} & : & [t^0,t^0+T]\times\mathbb{R}^m & \longrightarrow & \mathbb{R}^m \\ & & (t,\boldsymbol{z}) & \longmapsto & \boldsymbol{w}=\boldsymbol{f}(t,\boldsymbol{z})=\begin{pmatrix}\boldsymbol{f}_1(t,\boldsymbol{z})\\ \vdots \\ \boldsymbol{f}_d(t,\boldsymbol{z})\end{pmatrix} \end{array}$$

En notant

$$\boldsymbol{y}(t)=\begin{pmatrix}y_1(t)\\ \vdots \\ y_m(t)\end{pmatrix}\in\mathbb{R}^m \quad \text{et} \quad \boldsymbol{f}(t,\boldsymbol{y}(t))=\begin{pmatrix}f_1(t,\boldsymbol{y}(t))\\ \vdots \\ f_m(t,\boldsymbol{y}(t))\end{pmatrix}\in\mathbb{R}^m$$

le problème de Cauchy peut aussi s'écrire sous la forme

$$\begin{cases} y_1'(t) &= f_1(t,\boldsymbol{y}(t)), \\ \vdots & \\ y_d'(t) &= f_d(t,\boldsymbol{y}(t)), \quad \forall t\in[t^0,t^0+T] \\ \text{avec} & \quad \boldsymbol{y}(t^0)=\boldsymbol{y}_0\in\mathbb{R}^m. \end{cases}$$

Après discrétisation et utilisation de la formule des différences finies progressive on obtient

$$\begin{cases} y_1^{[n+1]} &= y_1^{[n]}+hf_1(t^n,\boldsymbol{y}^{[n]}) \\ \vdots & \\ y_d^{[n+1]} &= y_d^{[n]}+hf_d(t^n,\boldsymbol{y}^{[n]}) \\ \text{avec} & \quad \boldsymbol{y}(t^0)=\boldsymbol{y}_0\in\mathbb{R}^d. \end{cases}$$

où $\boldsymbol{y}^{[n]}=\begin{pmatrix}y_1^{[n]}\\ \vdots \\ y_d^{[n]}\end{pmatrix}$ et $\boldsymbol{y}^{[n]})\approx\boldsymbol{y}(t^n)$.

On obtient alors la méthode d'**Euler progressive** sous forme vectorielle :

$$\begin{cases} \boldsymbol{y}^{[n+1]} &= \boldsymbol{y}^{[n]}+h\boldsymbol{f}(t^n,\boldsymbol{y}^{[n]}), \ \forall n\in[\![0,N-1]\!] \\ \boldsymbol{y}^{[0]} &= \boldsymbol{y}(t^0) \end{cases} \tag{3.14}$$

Ce schéma est **explicite**, car il permet le calcul direct de $\boldsymbol{y}^{[n+1]}$ en fonction de $\boldsymbol{y}^{[n]}$.

De la même manière, la méthode d'**Euler régressive** sous forme vectorielle est donnée par le schéma

$$\begin{cases} \boldsymbol{y}^{[n+1]} &= \boldsymbol{y}^{[n]}+h\boldsymbol{f}(t^{n+1},\boldsymbol{y}^{[n+1]}), \ \forall n\in[\![0,N-1]\!] \\ \boldsymbol{y}^{[0]} &= \boldsymbol{y}(t^0) \end{cases} \tag{3.15}$$

Ce schéma est **implicite**, car $\boldsymbol{y}^{[n+1]}$ est définit implicitement en fonction de $\boldsymbol{y}^{[n]}$.

## 3.4 Méthodes à un pas ou à pas séparés

Les méthodes proposées dans cette section ont objectif la résolution du problème de Cauchy (voir Definition 3.5, page 26).

On note $(t^n)_{n=0}^N$ la discrétisation régulière de l'intervalle $[t^0,t^0+T]$. Il est toutefois possible de prendre une discrétisation où le pas n'est pas constant et vérifiant $t^0=a<t^1<\ldots<t^N=b$ mais cette possibilité ne sera pas étudiée dans ce cours.

Les méthodes sont dites **à un pas** car le calcul d'une approximation $\boldsymbol{y}^{[n+1]}$ de $\boldsymbol{y}(t^{n+1})$ ne nécessite que la connaissance de la valeur $\boldsymbol{y}^{[n]}\approx\boldsymbol{y}(t^n)$.

De manière générique, ces schémas sont donnés par

**Definition 3.8: Méthodes à un pas**

Les méthodes à un pas utilisent la formule générale:

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + h\boldsymbol{\Phi}(t^n, \boldsymbol{y}^{[n]}, h) \tag{3.16}$$

Le schéma (3.16) converge sur l'intervalle $[t^0, t^0 + T]$ si, pour la suite des $\boldsymbol{y}^{[n]}$ calculés, l'écart maximum avec la solution exacte diminue quand le pas $h$ diminue:

$$\lim_{h=\frac{T}{N}\to 0} \max_{n\in\{0,\dots,N\}} \left\| \boldsymbol{y}^{[n]} - \boldsymbol{y}(t^n) \right\| = 0$$

Pour la méthode d'Euler progressif, la fonction $\boldsymbol{\Phi}(t, \boldsymbol{y}, h)$ est $\boldsymbol{f}(t, \boldsymbol{y})$.

**Definition 3.9: consistance**

Le schéma de calcul (3.16) est consistant avec le problème de Cauchy (3.8)-(3.9) si

$$\lim_{h=\frac{T}{N}\to 0} \max_{n} \left\| \frac{\boldsymbol{y}(t^{n+1}) - \boldsymbol{y}(t^n)}{h} - \boldsymbol{\Phi}(t^n, \boldsymbol{y}(t^n), h) \right\| = 0$$

Cela signifie que le schéma doit être une approximation vraisemblable, bien construite.

**Théorème 3.10: consistance (admis)**

Le schéma (3.16) est consistant avec le problème de Cauchy (3.8)-(3.9) si $\Phi(t, \boldsymbol{y}, 0) = f(t, \boldsymbol{y})$.

**Definition 3.11: stabilité**

La méthode est stable si une petite perturbation sur $\boldsymbol{y}^{[0]}$ ou $\boldsymbol{\Phi}$ n'entraîne qu'une petite perturbation sur la solution approchée, et cela quel que soit le pas $h$.

Souvent , lorsque la méthode n'est pas stable, l'amplification des erreurs est exponentielle et, au bout de quelques calculs, les résultats entraînent des dépassements de capacité.

**Théorème 3.12: stabilité (admis)**

Si $\boldsymbol{\Phi}(t, \boldsymbol{y}, h)$ vérifie la condition de Lipschitz en $\boldsymbol{y}$ alors la méthode est stable.

**Théorème 3.13: convergence (admis)**

Si la méthode est **stable et consistante**, alors elle **converge** pour n'importe quelle valeur initiale.

**Definition 3.14: Ordre d'un schéma**

Le schéma (3.16) est d'ordre $p$ si la solution $\boldsymbol{y}$ du problème de Cauchy (3.8)-(3.9) vérifie

$$\max_{n} \left\| \frac{\boldsymbol{y}(t^{n+1}) - \boldsymbol{y}(t^n)}{h} - \boldsymbol{\Phi}(t^n, \boldsymbol{y}(t^n), h) \right\| = \mathcal{O}(h^p)$$

**Lemme 3.15: (admis)**

Soient $\boldsymbol{y}$ la solution du problème de Cauchy (3.8)-(3.9). et $(\boldsymbol{y}^{[n]})_{n\in[\![0,N]\!]}$ donnés par un schéma à un pas (3.16) d'ordre $p$ avec $\boldsymbol{y}^{[0]} = \boldsymbol{y}(t^0)$. On a alors

$$\max_{n\in[\![0,N]\!]} \left\| \boldsymbol{y}(t^n) - \boldsymbol{y}^{[n]} \right\| = \mathcal{O}(h^p) \tag{3.17}$$

**Proposition 3.16: (admis)**

Le schéma d'Euler progressif est une méthode à un pas d'ordre 1.

Il est possible de vérifier/retrouver numériquement l'ordre du schéma d'Euler progressif. Pour celà on choisi un problème de Cauchy dont la solution exacte est connue et on calcule pour différentes valeurs de $h$ (et donc différentes valeurs de $N$) le maximum de l'erreur commise entre la solution exacte et la solution numérique donnée par le schéma d'Euler progressif :

$$E(h) = \max_{n\in[\![0,N]\!]} \left\| \boldsymbol{y}(t^n) - \boldsymbol{y}^{[n]} \right\|$$

On représente ensuite la fonction $h \mapsto E(h)$. La méthode d'Euler progressive étant d'ordre 1, on a alors $E(h) = \mathcal{O}(h) \approx Ch$ quand $h$ est suffisament petit : la courbe obtenue doit donc être une droite.

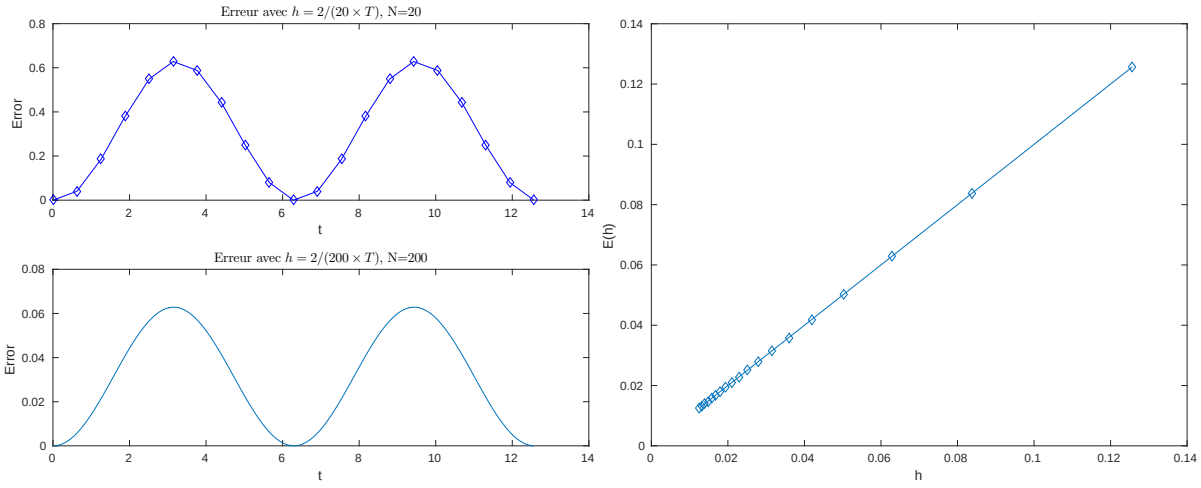



Figure 3.8: Méthode d'Euler progressive : vérification numérique de l'ordre

Dans la section suivante on étudie une classe de méthodes à un pas très usitées.

## 3.5 Méthodes de Runge-Kutta

Les méthodes de Runge-Kutta sont des méthodes à un pas dédiées à la résolution de problèmes de Cauchy (3.8)-(3.9).

(a) *Carle Runge* 1856-1927, mathématicien et physicien allemand

(b) *Martin Wilhelm Kutta* 1867-1944, Mathématicien allemand

(c) *John C. Butcher* 1933, Mathématicien appliqué néozélandais

### 3.5.1 Principe

Pour simplifier, on suppose $h_n = h$. L'idée fondamentale des méthodes de Runge-Kutta est d'intégrer l'équation (3.8)) sur $[t^n, t^{n+1}]$ et de calculer:

$$\boldsymbol{y}(t^{n+1}) = \boldsymbol{y}(t^n) + \int_{t^n}^{t^{n+1}} \boldsymbol{f}(t, \boldsymbol{y}(t))dt,$$

en utilisant une formule d'intégration numérique à $q$ points intermédiaires pour évaluer l'intégrale.

La fonction $\boldsymbol{\Phi}$ associée à une méthode de Runge-Kutta à $q$ évaluations de $\boldsymbol{f}$ peut s'écrire sous la forme :

$$\boldsymbol{\Phi}(t, \boldsymbol{y}, h) = \sum_{i=1}^{q} c_i \boldsymbol{k}^{[i]}(t, \boldsymbol{y}, h)$$

avec

$$\boldsymbol{k}^{[i]}(t, \boldsymbol{y}, h) = \boldsymbol{f}\left(t + ha_i, y + h\sum_{j=1}^{q} b_{i,j}\boldsymbol{k}^{[j]}(t, \boldsymbol{y}, h)\right),\ 1 \leqslant i \leqslant q$$

que l'on peut représenter sous la forme d'un tableau dit **tableau de Butcher** :

$$\begin{array}{c|c} \boldsymbol{a} & \mathbb{B} \\ \hline & \boldsymbol{c}^t \end{array} \tag{3.18}$$

avec $\mathbb{B} = (b_{i,j})_{i,j\in[\![1,q]\!]} \in \mathcal{M}q, q(\mathbb{R})$, $\boldsymbol{a} = (a_i)_{i\in[\![1,q]\!]} \in \mathbb{R}^q$ et $\boldsymbol{c} = (c_i)_{i\in[\![1,q]\!]} \in \mathbb{R}^q$

**Proposition 3.17: (admis)**

1. Les méthodes de Runge-Kutta explicites sont stables si $\boldsymbol{f}$ est contractante en $\boldsymbol{y}$.
2. Une méthode de Runge-Kutta est d'ordre 0 si
$$a_i = \sum_{j=1}^{q} b_{ij}.$$
3. Une méthode de Runge-Kutta est d'ordre 1 (et donc consistante) si elle est d'ordre 0 et si
$$\sum_{i=1}^{q} c_i = 1.$$
4. Une méthode de Runge-Kutta est d'ordre 2 si elle est d'ordre 1 et si
$$\sum_{i=1}^{q} c_i a_i = 1/2.$$
5. Une méthode de Runge-Kutta est explicite si la matrice $\mathbb{B}$ est triangulaire inférieure à diagonale nulle :
$$\forall (i,j) \in [\![1,q]\!],\ j \geqslant i, \quad b_{ij} = 0.$$

### 3.5.2 Formules explicites de Runge-Kutta d'ordre 2

Le tableau de Butcher associé aux méthodes de Runge-Kutta d'ordre 2 s'écrit sous la forme

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2\alpha} & \frac{1}{2\alpha} & 0 \\ \hline & 1-\alpha & \alpha \end{array} \tag{3.19}$$

Pour la méthode de Runge-Kutta d'ordre 2, la fonction $\boldsymbol{\Phi}$ associée au schéma général (3.16) est donnée par

$$\boldsymbol{\Phi}(t, \boldsymbol{y}, h) = (1-\alpha)\boldsymbol{f}(t, \boldsymbol{y}) + \alpha \boldsymbol{f}(t + \frac{h}{2\alpha}, \boldsymbol{y} + \frac{h}{2\alpha}\boldsymbol{f}(t, \boldsymbol{y})).$$

- Avec $\alpha = \frac{1}{2}$, on obtient la **méthode de Heun** :

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{2}\boldsymbol{f}(t^n, \boldsymbol{y}^{[n]}) + \frac{h}{2}\boldsymbol{f}\left(t^{n+1}, \boldsymbol{y}^{[n]} + h\boldsymbol{f}(t^n, \boldsymbol{y}^{[n]})\right).$$

- Avec $\alpha = 1$, on obtient la **méthode d'Euler modifiée** ou **méthode du point milieu**:

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + h\boldsymbol{f}\left(t^n + \frac{h}{2}, \boldsymbol{y}^{[n]} + \frac{h}{2}\boldsymbol{f}(t^n, \boldsymbol{y}^{[n]})\right).$$

**Exercice 3.5.1**

la **méthode de Heun** est donnée par

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{2}\boldsymbol{f}(t^n, \boldsymbol{y}^{[n]}) + \frac{h}{2}\boldsymbol{f}\left(t^{n+1}, \boldsymbol{y}^{[n]} + h\boldsymbol{f}(t^n, \boldsymbol{y}^{[n]})\right).$$

**Q. 1** *Ecrire la fonction algorithmique* REDHEUNVEC *permettant de résoudre un problème de Cauchy (vectoriel par la méthode de Heun en utilisant au plus* $2N$ *évaluation de* $\boldsymbol{f}$.

**Q. 2** *Ecrire un programme algorithmique permettant de retrouver numériquement l'ordre de cette méthode.*

**Correction Exercice 3.5.1**

**Q. 1** Le schéma de Heun peut s'écrire sous la forme

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{2}(\boldsymbol{k}_1^n + \boldsymbol{k}_2^n)$$

avec

$$\begin{aligned} \boldsymbol{k}_1^n &= \boldsymbol{f}(t^n, \boldsymbol{y}^{[n]}) \\ \boldsymbol{k}_2^n &= \boldsymbol{f}(t^{n+1}, \boldsymbol{y}^{[n]} + h\boldsymbol{k}_1) \end{aligned}$$

L'algorithme de la fonction REDHEUNVEC s'écrit alors :

**Algorithme 3.5** Fonction REDHEUNVEC : résolution d'un problème de Cauchy par le schéma de Heun

**Données :** $\boldsymbol{f}$ : $\boldsymbol{f} : [t^0, t^0+T] \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ fonction d'un problème de Cauchy (scalaire)
$t^0$ : réel, temps initial
$T$ : réel $> 0$
$\boldsymbol{y}^0$ : un vecteur de $\mathbb{R}^d$, donnée initiale
$N$ : un entier non nul (nombre de pas de discrétisation).

**Résultat :** $\boldsymbol{t}$ : vecteur de $\mathbb{R}^{N+1}$, $\boldsymbol{t}(n) = t^{n-1}$, $\forall n \in [\![1, N+1]\!]$
$\boldsymbol{Y}$ : matrice réelle de dimension $d \times (N+1)$, $\boldsymbol{Y}(:,n) = \boldsymbol{y}^{(n-1)}$, $\forall n \in [\![1, N+1]\!]$

```
Fonction [t,Y] ← REDHEUNVEC ( f, t^0, T, y^0, N )
    t ← DisReg(t^0, t^0 + T, N)
    h ← (b − a)/N
    Y(:,1) ← y^0
    Pour n ← 1 à N faire
        k_1 ← f(t(n), Y(:,n))
        k_2 ← f(t(n+1), Y(:,n) + h k_1)
        Y(:,n+1) ← Y(:,n) + (h/2) ∗ (k_1 + k_2)
    Fin Pour
Fin Fonction
```

**Q. 2** Il est possible de vérifier/retrouver numériquement l'ordre du schéma de Heun. Pour celà on choisi un problème de Cauchy dont la solution exacte est connue et on calcule pour différentes valeurs de $h$ (et

donc différentes valeurs de $N$) le maximum de l'erreur commise entre la solution exacte et la solution numérique donnée par le schéma de Heun :

$$E(h) = \max_{n\in[\![0,N]\!]} \left\| \boldsymbol{y}(t^n) - \boldsymbol{y}^{[n]} \right\|$$

On représente ensuite la fonction $h \mapsto E(h)$. La méthode de Heun étant d'ordre 2, on a alors $E(h) = \mathcal{O}(h^2) \approx Ch^2$ quand $h$ est suffisament petit. On utilise alors une échelle logarithmique pour représenter la courbe. En effet, on a

$$\log E(h) \approx \log(Ch^2) = \log(C) + 2\log(h)$$

En posant $X = \log(h)$ et $Y = \log E(h)$, coordonnées en échelle logarithmique, on a

$$Y \approx \log(C) + 2X$$

qui est l'équation d'une droite. La pente de cette droite est donc l'ordre de la méthode.

1: $t^0 \leftarrow 0$, $T \leftarrow 4\pi$,
2: $f : t, z \longrightarrow \cos(t) + 1$, $y_{ex} : t \longrightarrow \sin(t) + t$
3: $y^0 \leftarrow y_{ex}(t^0)$
4: $LN \leftarrow 100 : 50 : 1000$
5: $nLN \leftarrow$ LENGTH$(LN)$
6: $H \leftarrow \mathbb{0}_{nLN}$, ▷ pour stocker les $h$
7: $E \leftarrow \mathbb{0}_{nLN}$, ▷ pour stocker les erreurs
8: **Pour** $k \leftarrow 1$ **à** $nLN$ **faire**
9: $N \leftarrow LN(k)$
10: $[t, y] \leftarrow$ REDHEUNVEC$(f, t^0, T, \boldsymbol{y}^0, N)$
11: $E(k) \leftarrow$ MAX(ABS$(y - y_{ex}(t))$)
12: $H(k) \leftarrow T/N$
13: **Fin Pour**
14: ... ▷ Representation graphique

◇

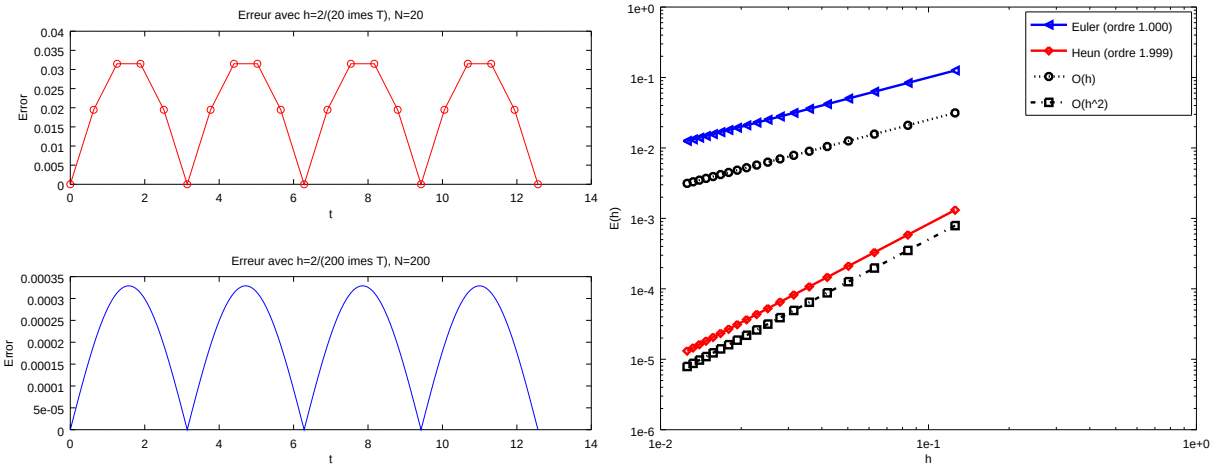



Figure 3.10: Méthode de Heun : vérification numérique de l'ordre et comparaison avec la méthode d'Euler progressive

Les méthodes d'ordre $p > 1$ sont plus précises que la méthode d'Euler, et d'autant plus que $p$ augmente. On se limite dans la pratique à $p = 4$. Cette méthode est connue sous le nom de *Runge-Kutta 4*. On peut noter que pour ces méthodes le pas peut être adapté à chaque itération.

### 3.5.3 Méthodes de Runge-Kutta d'ordre 4

La méthode explicite la plus utilisée est donnée par le tableau de Buchler suivant

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array} \tag{3.20}$$

Ce qui donne le schéma explicite de Runge-Kutta d'ordre 4 :

$$\begin{array}{rcl} \boldsymbol{k}_1^{[n]} & = & \boldsymbol{f}(t^n, \boldsymbol{y}^{[n]}) \\ \boldsymbol{k}_2^{[n]} & = & \boldsymbol{f}(t^n + \frac{h}{2}, \boldsymbol{y}^{[n]} + \frac{h}{2}\boldsymbol{k}_1^{[n]}) \\ \boldsymbol{k}_3^{[n]} & = & \boldsymbol{f}(t^n + \frac{h}{2}, \boldsymbol{y}^{[n]} + \frac{h}{2}\boldsymbol{k}_2^{[n]}) \\ \boldsymbol{k}_4^{[n]} & = & \boldsymbol{f}(t^n + h, \boldsymbol{y}^{[n]} + h\boldsymbol{k}_3^{[n]}) \\ \hline \boldsymbol{y}^{[n+1]} & = & \boldsymbol{y}^{[n]} + \frac{h}{6}(\boldsymbol{k}_1^{[n]} + 2\boldsymbol{k}_2^{[n]} + 2\boldsymbol{k}_3^{[n]} + \boldsymbol{k}_4^{[n]}). \end{array} \tag{3.21}$$

**Exercice 3.5.2**

la **méthode de Runge-Kutta d'ordre 4** est donnée par

$$\begin{array}{rcl} \boldsymbol{k}_1^{[n]} & = & \boldsymbol{f}(t^n, \boldsymbol{y}^{[n]}) \\ \boldsymbol{k}_2^{[n]} & = & \boldsymbol{f}(t^n + \frac{h}{2}, \boldsymbol{y}^{[n]} + \frac{h}{2}\boldsymbol{k}_1^{[n]}) \\ \boldsymbol{k}_3^{[n]} & = & \boldsymbol{f}(t^n + \frac{h}{2}, \boldsymbol{y}^{[n]} + \frac{h}{2}\boldsymbol{k}_2^{[n]}) \\ \boldsymbol{k}_4^{[n]} & = & \boldsymbol{f}(t^n + h, \boldsymbol{y}^{[n]} + h\boldsymbol{k}_3^{[n]}) \\ \hline \boldsymbol{y}^{[n+1]} & = & \boldsymbol{y}^{[n]} + \frac{h}{6}(\boldsymbol{k}_1^{[n]} + 2\boldsymbol{k}_2^{[n]} + 2\boldsymbol{k}_3^{[n]} + \boldsymbol{k}_4^{[n]}). \end{array}$$

**Q. 1** *Ecrire la fonction algorithmique* REDRK4VEC *permettant de résoudre un problème de Cauchy (vectoriel par la méthode de Runge-Kutta d'ordre 4.*

**Q. 2** *Ecrire un programme algorithmique permettant de retrouver numériquement l'ordre de cette méthode.*

**Correction Exercice 3.5.2**

**Q. 1** L'algorithme de la fonction REDRK4VEC s'écrit alors :

**Algorithme 3.6** Fonction REDRK4VEC : résolution d'un problème de Cauchy par le schéma de RK4

**Données :** $f$ : $\boldsymbol{f} : [t^0, t^0 + T] \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ fonction d'un problème de Cauchy (scalaire)
$t^0$ : réel, temps initial
$T$ : réel $> 0$
$\boldsymbol{y}^0$ : un vecteur de $\mathbb{R}^d$, donnée initiale
$N$ : un entier non nul (nombre de pas de discrétisation).
**Résultat :** $\boldsymbol{t}$ : vecteur de $\mathbb{R}^{N+1}$, $\boldsymbol{t}(n) = t^{n-1}$, $\forall n \in [\![1, N+1]\!]$
$\boldsymbol{Y}$ : matrice réelle de dimension $d \times (N+1)$, $\boldsymbol{Y}(:, n) = \boldsymbol{y}^{(n-1)}$, $\forall n \in [\![1, N+1]\!]$

1: **Fonction** $[\boldsymbol{t}, \boldsymbol{Y}] \leftarrow$ REDRK4VEC $(\ f, t^0, T, \boldsymbol{y}^0, N\ )$
2: $\quad \boldsymbol{t} \leftarrow DisReg(t^0, t^0 + T, N)$
3: $\quad h \leftarrow (b - a)/N$
4: $\quad \boldsymbol{Y}(:, 1) \leftarrow \boldsymbol{y}^0$
5: $\quad$ **Pour** $n \leftarrow 1$ **à** $N$ **faire**
6: $\quad\quad \boldsymbol{k}_1 \leftarrow \boldsymbol{f}(\boldsymbol{t}(n), \boldsymbol{Y}(:, n))$
7: $\quad\quad \boldsymbol{k}_2 \leftarrow \boldsymbol{f}(\boldsymbol{t}(n) + h/2, \boldsymbol{Y}(:, n) + (h/2)\boldsymbol{k}_1)$
8: $\quad\quad \boldsymbol{k}_3 \leftarrow \boldsymbol{f}(\boldsymbol{t}(n) + h/2, \boldsymbol{Y}(:, n) + (h/2)\boldsymbol{k}_2)$
9: $\quad\quad \boldsymbol{k}_4 \leftarrow \boldsymbol{f}(\boldsymbol{t}(n) + h, \boldsymbol{Y}(:, n) + h\boldsymbol{k}_3)$
10: $\quad\quad \boldsymbol{Y}(:, n+1) \leftarrow \boldsymbol{Y}(:, n) + (h/6) * (\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4)$
11: $\quad$ **Fin Pour**
12: **Fin Fonction**

**Q. 2** *voir aussi correction Exercice 3.5.1-Q2 : l'ordre 2 étant remplacé par 4 ici!* Il est possible de vérifier/retrouver numériquement l'ordre du schéma de Runge-Kutta 4. Pour celà on choisi un problème de Cauchy dont la solution exacte est connue et on calcule pour différentes valeurs de $h$ (et donc différentes valeurs de $N$) le maximum de l'erreur commise entre la solution exacte et la solution numérique donnée par le schéma de Runge-Kutta 4 :

$$E(h) = \max_{n\in[\![0,N]\!]} \left\| \boldsymbol{y}(t^n) - \boldsymbol{y}^{[n]} \right\|$$

On représente ensuite la fonction $h \mapsto E(h)$. La méthode de Runge-Kutta 4 étant d'ordre 4, on a alors théoriquement $E(h) = \mathcal{O}(h^4) \approx Ch^4$ quand $h$ est suffisament petit. On utilise alors une échelle logarithmique pour représenter la courbe. En effet, on a

$$\log E(h) \approx \log(Ch^4) = \log(C) + 4\log(h)$$

En posant $X = \log(h)$ et $Y = \log E(h)$, coordonnées en échelle logarithmique, on a

$$Y \approx \log(C) + 4X$$

qui est l'équation d'une droite. La pente de cette droite est donc l'ordre de la méthode.

1: $t^0 \leftarrow 0$, $T \leftarrow 4\pi$,
2: $f : t, z \longrightarrow \cos(t) + 1$, $y_{ex} : t \longrightarrow \sin(t) + t$
3: $y^0 \leftarrow y_{ex}(t^0)$
4: $LN \leftarrow 100 : 50 : 1000$
5: $nLN \leftarrow$ LENGTH$(LN)$
6: $H \leftarrow \mathbb{0}_{nLN}$, ▷ pour stocker les $h$
7: $E \leftarrow \mathbb{0}_{nLN}$, ▷ pour stocker les erreurs
8: **Pour** $k \leftarrow 1$ **à** $nLN$ **faire**
9: $N \leftarrow LN(k)$
10: $[t, y] \leftarrow$ REDRK4VEC$(f, t^0, T, \boldsymbol{y}^0, N)$
11: $E(k) \leftarrow$ MAX(ABS$(y - y_{ex}(t))$)
12: $H(k) \leftarrow T/N$
13: **Fin Pour**
14: ... ▷ Representation graphique

◇

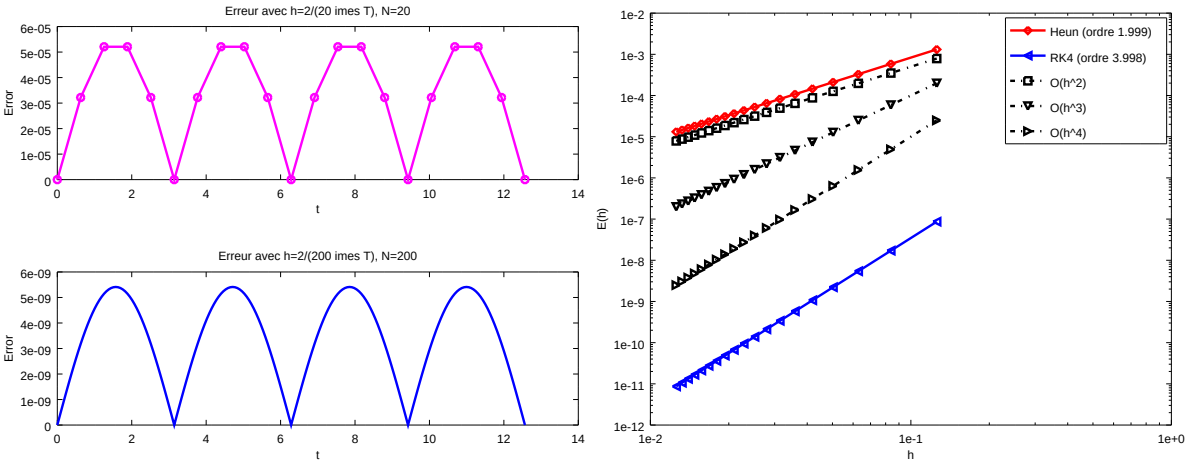



Figure 3.11: Méthode de Runge-Kutta 4 : vérification numérique de l'ordre et comparaison avec la méthode de Heun

Lorsque l'on diminue encore le pas on obtient la Figure 3.12. L'erreur engendrée par la méthode de Runge-Kutta 4 se *stabilise* autour de la valeur $1e-14$ : la précision machine est atteinte!

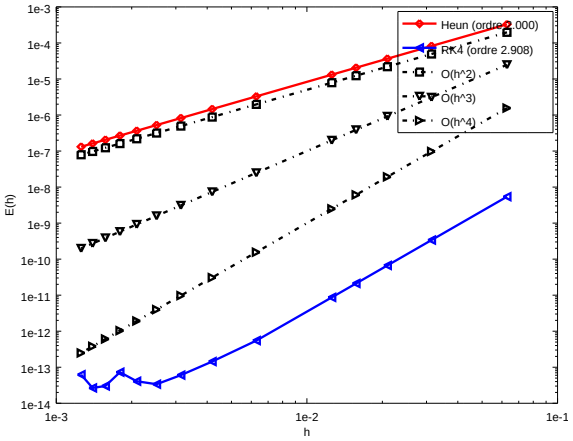



Figure 3.12: Méthode de Runge-Kutta 4 : vérification numérique de l'ordre avec précision machine atteinte

## 3.6 Méthodes à pas multiples

### 3.6.1 Exemple : schéma de point milieu

Considérons la méthode à deux pas définie par la récurrence:

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n-1]} + 2h\boldsymbol{f}(t^n, \boldsymbol{y}^{[n]}). \tag{3.22}$$

Cette méthode est d'ordre 2.

De manière évidente, ces méthodes à pas multiples posent des problèmes de démarrage: ici, on ne peut calculer directement $\boldsymbol{y}^{[1]}$. Les premières valeurs doivent être calculées par un autre schéma.

### 3.6.2 Le principe

Dans tous les schémas présentés, la valeur de $\boldsymbol{y}^{[n+1]}$ était déterminée à chaque pas de façon explicite en fonction uniquement du point $\boldsymbol{y}^{[n]}$. On peut tenir compte de plusieurs valeurs $\boldsymbol{y}^{[i]}$ précédemment calculées et ainsi travailler sur un nombre de pas multiples.

**Definition 3.18: Méthodes à pas multiples**

Les méthodes à pas multiples s'écrivent sous la forme générale:

$$\sum_{i=0}^{k} \alpha_i \boldsymbol{y}^{[n+i]} = h \sum_{i=0}^{k} \beta_i \boldsymbol{f}(t^{n+i}, \boldsymbol{y}^{[n+i]}) \tag{3.23}$$

où $k$ est le nombre de pas, $\alpha_k \neq 0$ et $|\alpha_0| + |\beta_0| > 0$.

**Remarque 3.19** Si $\beta_k = 0$ le schéma est explicite, sinon il est implicite.

**Definition 3.20: ordre**

Soit $\boldsymbol{y}$ la solution d'un problème de Cauchy (3.8)-3.9 et $\boldsymbol{y}^{[n+k]}$ le terme obtenu par le schéma (3.23) en prenant $\boldsymbol{y}^{[n+i]} = \boldsymbol{y}(t^{n+i})$, $\forall i \in [\![0, k-1]\!]$. Alors, l'erreur locale est

$$\tau(n+k) = \left\| \boldsymbol{y}(t^{n+k}) - \boldsymbol{y}^{[n+k]} \right\|_\infty .$$

Le schéma (3.23) est alors d'**ordre** $p$ si

$$\tau(n+k) = \mathcal{O}(h^{p+1}).$$

**Théorème 3.21: ordre schémas à pas multiples (admis)**

Un schéma à pas multiples de type (3.23) est d'ordre $p$ si et seulement si

$$\begin{aligned} \sum_{i=0}^{k} \alpha_i &= 0, \\ \sum_{i=0}^{k} \alpha_i i^q &= q \sum_{i=0}^{k} \beta_i i^{q-1}, \quad \forall q \in [\![1, p]\!]. \end{aligned}$$

**Propriété 3.22: stabilité schémas à pas multiples (admis)**

Soit une méthode à pas multiples donnée par (3.23). On note $P$ le polynôme défini par

$$P(\lambda) = \sum_{i=0}^{k} \alpha_i \lambda^i.$$

La méthode à pas multiples est **stable**, si

1. toutes les racines de $P$ sont de module inférieur ou égal à 1,
2. une racine de module égal à 1 est une racine simple de $P$.

Pour le schéma (3.22), on a $k = 2$ et

$$\left\{ \begin{array}{lcl} \alpha_0 &=& -1 \\ \alpha_1 &=& 0 \\ \alpha_2 &=& 1 \end{array} \right. \qquad \left\{ \begin{array}{lcl} \beta_0 &=& 0 \\ \beta_1 &=& 2 \\ \beta_2 &=& 0 \end{array} \right.$$

On obtient donc $P(\lambda) = -1 + \lambda^2 = (\lambda + 1)(\lambda - 1)$ : le schéma (3.22) est stable.

**Théorème 3.23: convergence (admis)**

On suppose que les $k$ valeurs initiales vérifient,

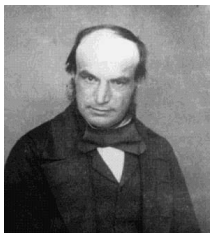
$$\left\| \boldsymbol{y}(t^i) - \boldsymbol{y}^{[i]} \right\| \leqslant C_0 h^p, \ \forall i \in [\![0, k-1]\!].$$

Si le schéma (3.23) est **stable et d'ordre** $p$, alors il est **convergent** d'ordre $p$ :

$$\left\| \boldsymbol{y}(t^n) - \boldsymbol{y}^{[n]} \right\| \leqslant C h^p, \ \forall n \in [\![0, N]\!].$$

**Remarque 3.24** Pour obtenir, à partir d'un schéma à $k$ pas, un schéma d'ordre $p$ il faut obligatoirement initialiser les $k$ premiers termes $(\boldsymbol{y}^{[n]})_{n=0}^{k-1}$ à l'aide d'un schéma d'ordre $p$ au moins pour conserver l'ordre.

Nous allons maintenant donner quelques schémas explicites et implicites à pas multiples développer par :



*John Couch Adams* 1819-1892, mathématicien et astronome britannique

*Francis Bashforth* 1819-1912, mathématicien appliqué britannique

(a) *Forest Ray Moulton* 1872-1952, mathématicien et astronome américain

### 3.6.3 Méthodes explicites d'Adams-Bashforth

On note en abrégé $\boldsymbol{f}^{[n]} = \boldsymbol{f}(t^n, \boldsymbol{y}^{[n]})$. Voici trois schémas :

- schéma explicite d'Adams-Bashforth d'ordre 2 à 2 pas :

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{2}\left(3\boldsymbol{f}^{[n]} - \boldsymbol{f}^{[n-1]}\right). \tag{3.24}$$

- schéma explicite d'Adams-Bashforth d'ordre 3 à 3 pas :

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{12}\left(23\boldsymbol{f}^{[n]} - 16\boldsymbol{f}^{[n-1]} + 5\boldsymbol{f}^{[n-2]}\right). \tag{3.25}$$

- schéma explicite d'Adams-Bashforth d'ordre 4 à 4 pas :

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{24}\left(55\boldsymbol{f}^{[n]} - 59\boldsymbol{f}^{[n-1]} + 37\boldsymbol{f}^{[n-2]} - 9\boldsymbol{f}^{[n-3]}\right). \tag{3.26}$$

Ces schémas sont **explicites** et leur ordre correspond au nombre de pas.

**Exercice 3.6.1**

La **méthode de Adam-Bashforth d'ordre 4** explicite est donnée par

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \tfrac{h}{24}\left(55\boldsymbol{f}^{[n]} - 59\boldsymbol{f}^{[n-1]} + 37\boldsymbol{f}^{[n-2]} - 9\boldsymbol{f}^{[n-3]}\right). \tag{3.27}$$

avec $\boldsymbol{f}^{[n]} = \boldsymbol{f}(t^n, \boldsymbol{y}^{[n]})$.

**Q. 1** *Ecrire la fonction algorithmique* REDAB4VEC *permettant de résoudre un problème de Cauchy (vectoriel) par cette méthode.*

**Correction Exercice 3.6.1**

**Q. 1** Soit $(t^{[n]})_{n=0}^{N}$ la discrétisation régulière de $[t^0, t^0+T]$ avec $h = T/N$. On a donc $t^{[n]} = t^0 + nh$, $\forall n \in [\![0, N]\!]$.

On ne peut *utiliser* le schéma à pas multiples (3.27) que pour $n \geqslant 3$. On va alors utiliser un schéma à un pas d'ordre (au moins) 4 pour calculer les 4 premiers termes $\boldsymbol{y}^{[0]}$, $\boldsymbol{y}^{[1]}$, $\boldsymbol{y}^{[2]}$ et $\boldsymbol{y}^{[3]}$ nécessaire pour *démarrer* le schéma (3.27). On choisi par exemple la méthode de Runge-Kutta d'ordre 4 pour initialiser ces 4 termes. Deux possibilités :

- on réécrit le schéma de Runge-Kutta d'ordre 4 pour ces 4 premiers termes

```
1: Y(:,1) ← y^0
2: Pour n ← 1 à 3 faire
3:     k_1 ← f(t(n), Y(:,n))
4:     k_2 ← f(t(n) + h/2, Y(:,n) + (h/2)k_1)
5:     k_3 ← f(t(n) + h/2, Y(:,n) + (h/2)k_2)
6:     k_4 ← f(t(n) + h, Y(:,n) + hk_3)
7:     Y(:,n+1) ← Y(:,n) + (h/6) * (k_1 + 2k_2 + 2k_3 + k_4)
8: Fin Pour
```

- on utilise la fonction REDRK4VEC avec ses paramètres d'entrées judicieusement choisis :

```
1: [t_ini, Y_ini] ← REDRK4VEC(f, t^0, t^0 + 3 * h, y^0, 3)
2: Pour n ← 1 à 4 faire
3:     Y(:, n) ← Y_ini(:, n)
4: Fin Pour
```

En choisissant cette dernière solution, l'algorithme de la fonction REDAB4VEC s'écrit alors :

**Algorithme 3.7** Fonction REDAB4VEC : résolution d'un problème de Cauchy par le schéma explicite d'Adams-Bashforth d'ordre 4

**Données :** $\boldsymbol{f}$ : $\boldsymbol{f} : [t^0, t^0+T] \times \mathbb{R}^d \longrightarrow \mathbb{R}^d$ fonction d'un problème de Cauchy (scalaire)
$t^0$ : réel, temps initial
$T$ : réel $> 0$
$\boldsymbol{y}^0$ : un vecteur de $\mathbb{R}^d$, donnée initiale
$N$ : un entier non nul (nombre de pas de discrétisation).

**Résultat :** $\boldsymbol{t}$ : vecteur de $\mathbb{R}^{N+1}$, $\boldsymbol{t}(n) = t^{n-1},\ \forall n \in [\![1, N+1]\!]$
$\boldsymbol{Y}$ : matrice réelle de dimension $d \times (N+1)$, $\boldsymbol{Y}(:,n) = \boldsymbol{y}^{(n-1)},\ \forall n \in [\![1, N+1]\!]$

```
1: Fonction [t,Y] ← REDAB4VEC ( f, t^0, T, y^0, N )
2:     t ← DISREG(t^0, t^0 + T, N)
3:     h ← (b − a)/N
4:     [t_ini, Y_ini] ← REDRK4VEC(f, t^0, t^0 + 3 * h, y^0, 3)
5:     Pour n ← 1 à 4 faire
6:         Y(:, n) ← Y_ini(:, n)
7:     Fin Pour
8:     k_1 ← f(t(3), Y(:, 3))
9:     k_2 ← f(t(2), Y(:, 2))
10:    k_3 ← f(t(1), Y(:, 1))
11:    Pour n ← 4 à N faire
12:        k_0 ← f(t(n), Y(:, n))
13:        Y(:, n + 1) ← Y(:, n) + (h/24) * (55 * k_0 − 59 * k_1 + 37 * k_2 − 9 * k_3)
14:        k_3 ← k_2, k_2 ← k_1, k_1 ← k_0
15:    Fin Pour
16: Fin Fonction
```

◇

## 3.6.4 Méthodes implicites d'Adams-Moulton

On note en abrégé $\boldsymbol{f}^{[n]} = \boldsymbol{f}(t^n, \boldsymbol{y}^{[n]})$. Voici trois schémas :

- schéma d'Adams-Moulton d'ordre 2 à 1 pas :

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{2}\left(\boldsymbol{f}^{[n+1]} + \boldsymbol{f}^{[n]}\right). \tag{3.28}$$

- schéma d'Adams-Moulton d'ordre 3 à 2 pas :

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{12}\left(5\boldsymbol{f}^{[n+1]} + 8\boldsymbol{f}^{[n]} - \boldsymbol{f}^{[n-1]}\right) \tag{3.29}$$

- schéma d'Adams-Moulton d'ordre 4 à 3 pas :

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{24}\left(9\boldsymbol{f}^{[n+1]} + 19\boldsymbol{f}^{[n]} - 5\boldsymbol{f}^{[n-1]} + \boldsymbol{f}^{[n-2]}\right) \tag{3.30}$$

Ces schémas sont **implicites** et leur ordre correspond au nombre de pas plus un.

### 3.6.5 Schéma prédicteur-correcteur

**Principe**

Il s'agit là d'une des méthodes les plus employées. Une méthode de prédiction-correction procède en deux étapes à chacune des itérations:

- **Prédiction** : on calcule une approximation de $\boldsymbol{y}(t_{n+1})$ notée $\overline{\boldsymbol{y}}^{[n+1]}$ à l'aide d'un **schéma explicite**
- **Correction** :

on fournit explicitement une valeur approchée de la solution au $n^{ième}$ pas (soit $\overline{\boldsymbol{y}}^{[n+1]}$), puis on calcule la valeur correspondante de $\boldsymbol{f}(t^{[n+1]}, \overline{\boldsymbol{y}}^{[n+1]})$ que l'on note $\overline{\boldsymbol{f}}^{[n+1]}$ et enfin, on subsitue cette valeur dans un schéma implicite (on obtient alors une valeur *corrigée*).

1: **Pour** $n \leftarrow 0$ **à** $N$ **faire**
2: $\quad \overline{\boldsymbol{y}}^{[n+1]} \leftarrow$ donné par un **schéma explicite**
3: $\quad \boldsymbol{y}^{[n+1]} \leftarrow$ donné par **un schéma implicite**, inconnue $\boldsymbol{y}^{[n+1]}$ remplacée par $\overline{\boldsymbol{y}}^{[n+1]}$
4: **Fin Pour**

**Exemple**

Choisissons la méthode d'Euler explicite pour prédicteur et la méthode implicite des trapèzes comme correcteur.

**Euler explicite :** $\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + h\boldsymbol{f}(t^n, \boldsymbol{y}^{[n]})$
**Trapèze implicite :** $\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{2}(\boldsymbol{f}(t^n, \boldsymbol{y}^{[n]}) + \boldsymbol{f}(t^{n+1}, \boldsymbol{y}^{[n+1]}))$

On obtient :

$$\left\{\begin{array}{lcl} \boldsymbol{f}^{[n]} & = & \boldsymbol{f}(t^n, \boldsymbol{y}^{[n]}); \\ \overline{\boldsymbol{y}}^{[n+1]} & = & \boldsymbol{y}^{[n]} + h\boldsymbol{f}^{[n]}; \\ \overline{\boldsymbol{f}}^{[n+1]} & = & \boldsymbol{f}(t^{n+1}, \overline{\boldsymbol{y}}^{[n+1]}); \\ \boldsymbol{y}^{[n+1]} & = & \boldsymbol{y}^{[n]} + \frac{h}{2}(\overline{\boldsymbol{y}}^{[n+1]} + \boldsymbol{f}^{[n]}) \end{array}\right.$$

**Remarque 3.25** On retrouve ici, pour ce cas simple, une formule de Runge-Kutta 2.

En pratique, on peut utiliser un schéma d'Adams explicite (??) pour la prédiction et un autre implicite (??) pour la correction.

**Exercice 3.6.2**

On pose $\boldsymbol{f}^{[n]} = \boldsymbol{f}(t^n, \boldsymbol{y}^{[n]})$. La **méthode de Adams-Bashforth d'ordre 4** explicite est donnée par

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{24}\left(55\boldsymbol{f}^{[n]} - 59\boldsymbol{f}^{[n-1]} + 37\boldsymbol{f}^{[n-2]} - 9\boldsymbol{f}^{[n-3]}\right)$$

et la **méthode de Adams-Moulton d'ordre 4** implicite par

$$\boldsymbol{y}^{[n+1]} = \boldsymbol{y}^{[n]} + \frac{h}{24}\left(9\boldsymbol{f}^{[n+1]} + 19\boldsymbol{f}^{[n]} - 5\boldsymbol{f}^{[n-1]} + \boldsymbol{f}^{[n-2]}\right)$$

avec $\boldsymbol{f}^{[n]} = \boldsymbol{f}(t^n, \boldsymbol{y}^{[n]})$.
**Q. 1** *Ecrire la fonction algorithmique* REDPRECOR4VEC *permettant de résoudre un problème de Cauchy (vectoriel) par une méthode de prédiction-correction utilisant ces deux schémas.*

**Remarque 3.26**
1. La stabilité du prédicteur intervient peu.
2. Le choix du pas dans ces méthodes est un problème difficile.

**Comparaison avec Runge-Kutta**

L'intérêt d'une méthode de résolution numérique d'équations différentielles se mesure principalement suivant deux critères:

- son coût pour obtenir une précision donnée (c'est à dire le nombre d'évaluations de fonctions par étapes).
- sa stabilité.

La caractéristique des méthodes de Runge-Kutta est que le pas est assez facile à adapter, la mise en oeuvre informatique plus aisée. Mais pour des méthodes du même ordre de consistance, les méthodes de Runge-Kutta exigent plus d'évaluations de fonctions. Quand on sait que la solution de l'équation n'est pas sujette à de brusques variations de la dérivée, on prendra une méthode de type prédicteur-correcteur mais, si le pas doit être adapté plus précisément, on préfèrera une méthode de Runge-Kutta.

# Chapitre 4

## Introduction à la résolution d'E.D.P.

# Liste des algorithmes


1.1 Exemple de boucle «pour» . . . 3
1.2 Exemple de boucle «tant que» . . . 4
1.3 Exemple de boucle «répéter ...jusqu'à» . . . 4
1.4 Exemple d'instructions conditionnelle «si» . . . 4
1.5 Exemple de fonction : Résolution de l'équation du premier degré $ax + b = 0$. . . . 5
1.6 Calcul de $S = \sum_{k=1}^{n} k \sin(2kx)$ . . . 7
1.7 Calcul de $P = \prod_{n=1}^{k} \sin(2kz/n)^k$ . . . 8
1.8 En-tête de la fonction SFT retournant valeur de la série de Fourier en $t$ tronquée au $n$ premiers termes de l'exercice 1.2.3. . . . 8
1.9 Fonction SFT retournant la valeur de la série de Fourier en $t$ tronquée au $n$ premiers termes de l'exercice **??**. . . . 9
2.1 Calcul de dérivées d'ordre 1 . . . 16
2.2 Calcul de dérivées d'ordre 1 . . . 16
2.3 Calcul de dérivées d'ordre 2 . . . 18
2.4 Calcul de dérivées d'ordre 2 . . . 18
3.1 Fonction DisReg retournant une discrétisation régulière de l'intervalle $[a, b]$ . . . 31
3.2 Fonction REDEP : résolution d'un problème de Cauchy scalaire par le schéma d'Euler progressif . . . 31
3.3 Fonction fCauchy : fonction $f$ du problème de Cauchy associé à $(\mathcal{P})$ . . . 32
3.4 résolution numérique du problème $(\mathcal{P})$ . . . 32
3.5 Fonction REDHeunVec : résolution d'un problème de Cauchy par le schéma de Heun . . . 37
3.6 Fonction REDRK4Vec : résolution d'un problème de Cauchy par le schéma de RK4 . . . 39
3.7 Fonction REDAB4Vec : résolution d'un problème de Cauchy par le schéma explicite d'Adams-Bashforth d'ordre 4 . . . 44