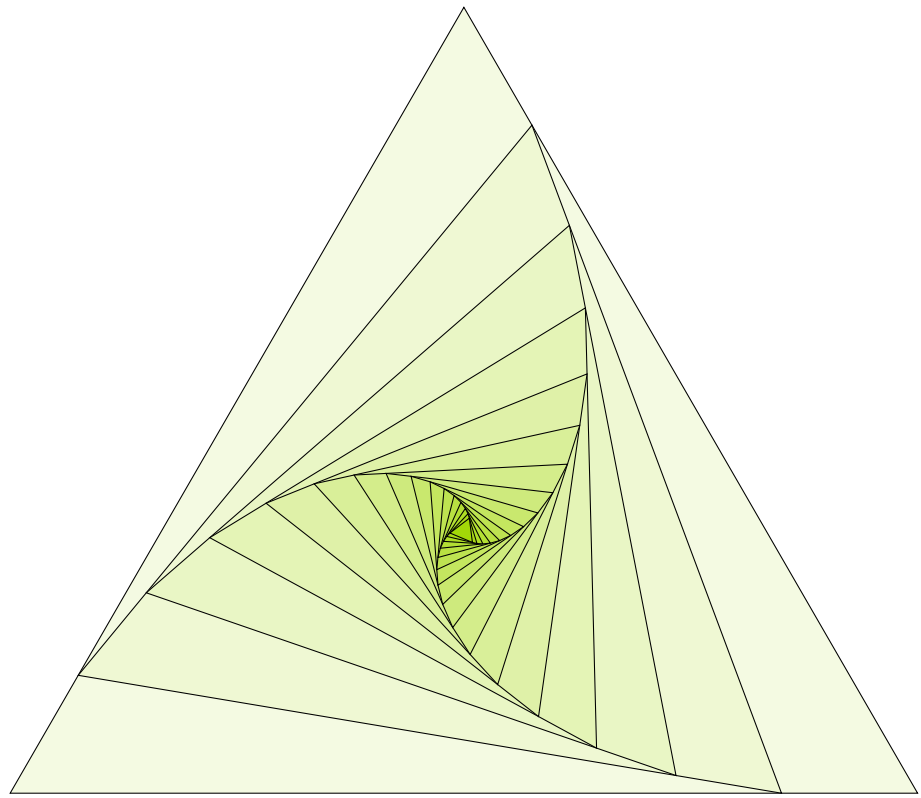


# Algorithmique numérique

*Notes de cours*

Sup Galilée, Ingénieurs 1ère année



Francois Cuvelier 3  
Université Paris XIII / Institut Galilée 4  
L.A.G.A./Département de Mathématiques 5  
6  
7

---



# Table des matières

<b>1</b>	<b>Algorithmique Numérique</b>	<b>1</b>	<b>2</b>
1.1	Introduction . . . . .	1	3
1.1.1	Exemple 1 : permutation de deux voitures . . . . .	1	4
1.1.2	Exemple 2 : résolution d'une équation . . . . .	3	5
1.1.3	caractéristiques . . . . .	3	6
1.1.4	Première approche méthodologique . . . . .	3	7
1.2	Pseudo-langage algorithmique . . . . .	3	8
1.2.1	Données et constantes . . . . .	4	9
1.2.2	Variables . . . . .	4	10
1.2.3	Opérateurs . . . . .	4	11
1.2.4	Expressions . . . . .	4	12
1.2.5	Instructions . . . . .	5	13
1.2.6	Fonctions . . . . .	6	14
1.4	Méthodologie . . . . .	8	15
1.4.1	Description du problème . . . . .	8	16
1.4.2	Recherche d'une méthode de résolution . . . . .	8	17
1.4.3	Réalisation d'un algorithme . . . . .	8	18
1.5	Principes de «bonne» programmation pour attaquer de «gros» problèmes . . . . .	11	19
<b>2</b>	<b>Méthodes Numériques</b>	<b>13</b>	<b>20</b>
2.1	Polynôme d'interpolation de Lagrange . . . . .	13	21
2.1.1	Définition . . . . .	13	22
2.1.2	Erreur de l'interpolation . . . . .	15	23
2.1.3	Points de Chebyshev . . . . .	17	24
2.2	Dérivation numérique . . . . .	17	25
2.3	Intégration numérique . . . . .	21	26
2.3.1	Méthodes simpliste . . . . .	22	27
2.3.2	Méthodes de Newton-Cotes . . . . .	23	28
2.3.3	Méthodes composites . . . . .	24	29
2.3.4	Formule composite des trapèzes . . . . .	25	30
2.3.5	Formule composite de Simpson . . . . .	25	31
2.3.6	Autres méthodes . . . . .	26	32
2.3.7	Intégrales multiples . . . . .	26	33
2.4	Algèbre linéaire . . . . .	27	34



# Chapitre 1

## Algorithmique Numérique

### 1.1 Introduction 2

Le but ici est d'acquérir une méthodologie permettant de *mettre sur le papier* la résolution d'un problème donné (bien posé!). Pour cela, nous utiliserons un ensemble d'instructions dont l'application permet de résoudre le problème en un nombre fini d'opérations (ou d'actions). 3  
4  
5

Ceci est l'objet de l'algorithmique : 6

**Définition 1.1** (Petit Robert 97) **Algorithmique** : Enchaînement d'actions nécessaires à l'accomplissement d'une tâche. 7  
8

#### 1.1.1 Exemple 1 : permutation de deux voitures 9

**Tâche** : Permuter deux voitures sur un parking de trois places numérotées de  $P_1$  à  $P_3$  et ceci sans gêner la circulation. La voiture B, est sur l'emplacement  $P_2$ , la voiture R, est sur l'emplacement  $P_3$ . 10

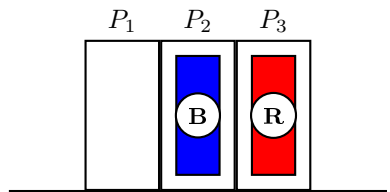


Figure 1.1: Avant permutation

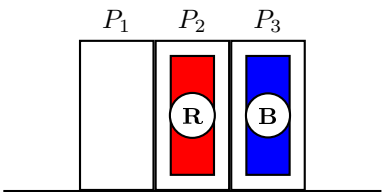


Figure 1.2: Après permutation

Voici, l'«algorithme» basique permettant de réaliser cette tâche : 11  
12

- 1: Déplacer la voiture R de l'emplacement  $P_3$  à  $P_1$ .
- 2: Déplacer la voiture B de l'emplacement  $P_2$  à  $P_3$ .
- 3: Déplacer la voiture R de l'emplacement  $P_1$  à  $P_2$ .

1 Malheureusement, cet algorithme souffre de nombreuses lacunes et ambiguïtés :

- 2 • Au départ, l'emplacement  $P_1$  est-il libre?
- 3 • Que veut-dire l'action **déplacer**? Si les voitures sont non-roulantes, prendre une grue? sinon a-t'on
- 4 les clés des deux véhicules?
- 5 • Dans l'énoncé, il est dit «sans gêner la circulation»! Donc il y a de la circulation et à tout moment
- 6 un véhicule extérieur au problème peut venir occuper un emplacement libre!
- 7 • ...

8 En fait, les lacunes et ambiguïtés proviennent pour la plupart d'une mauvaise description de la tâche à  
9 réaliser : le problème est mal posé.

10 Cependant nous pouvons tout de même écrire un algorithme mais en restreignant son champ d'application  
11 : nous allons faire des **hypothèses** sur les données du problème.

12 Tout d'abord il faut préciser les **données** du problème :

**Données :**

- 13 | Parking de trois emplacements, numérotés de  $P_1$  à  $P_3$ .
- 14 | Deux voitures, l'une notée B et l'autre R.

14 Ensuite, nous précisons les **hypothèses** sur les données

15 **Hypothèses sur les données :**

- 16 • La voiture B est sur l'emplacement  $P_2$ .
- 17 • La voiture R est sur l'emplacement  $P_3$ .
- 18 • Les deux voitures sont «roulantes».

19 Ensuite, nous donnons des hypothèses plus générales

**Hypothèses générales :**

- 20 | 1. Une seule personne réalise la tâche.
- 21 | 2. Celle-ci a son permis de conduire et les clés des deux voitures.
- 22 | 3. Lors du déplacement d'un véhicule d'un emplacement à un autre
- 23 | aucune voiture extérieur ne vient sur le parking.
- 24 | 4. Une voiture extérieur ne reste qu'un temps fini sur un emplacement

21 Nous considérons que la phase de déplacement d'un véhicule d'un emplacement à un autre (libre) ne pose  
22 aucun problème au titulaire d'un permis! Et enfin, nous précisons le résultat voulu :

23 **Résultats :**

- 24 | 1. La voiture B est sur l'emplacement  $P_3$ .
- 25 | 2. La voiture R est sur l'emplacement  $P_2$ .

25 Nous obtenons alors l'algorithme suivant :

- 1: Aller au volant du véhicule R (emplacement  $P_3$ ).
  - 2: **Tantque** emplacement  $P_1$  est occupé **faire**
  - 3: Attendre.
  - 4: **Fin Tantque**
  - 5: Déplacer le véhicule R de l'emplacement  $P_3$  à  $P_1$
  - 6: Aller au volant du véhicule B (emplacement  $P_2$ ).
  - 7: **Tantque** emplacement  $P_3$  est occupé **faire**
  - 8: Attendre.
  - 9: **Fin Tantque**
  - 10: Déplacer le véhicule B de l'emplacement  $P_2$  à  $P_3$
  - 11: Aller au volant du véhicule R (emplacement  $P_1$ ).
  - 12: **Tantque** emplacement  $P_2$  est occupé **faire**
  - 13: Attendre.
  - 14: **Fin Tantque**
  - 15: Déplacer le véhicule R de l'emplacement  $P_1$  à  $P_2$

26

### 1.1.2 Exemple 2 : résolution d'une équation

**Tâche :** Résoudre l'équation  $ax = b$ .

Voici, l'«algorithme» basique permettant de réaliser cette tâche :

```

1: Si  $a$  différent de 0 alors
2:   La solution est  $x = b/a$ .
3: Sinon
4:   Il n'y a pas de solution
5: Fin Si

```

Une nouvelle fois, l'énoncé de la tâche à effectuer est trop imprécise! En effet, rien ne nous permet d'affirmer que  $x$  est l'inconnue ou que  $a$  n'est pas une matrice. Il faut alors palier au manque d'informations en ajoutant des hypothèses pour **clarifier le problème** :

Soient  $a \in \mathbb{R}^*$  et  $b \in \mathbb{R}$  donnés. Le problème est

trouver  $x \in \mathbb{R}$  tel que  
 $ax = b$ .

Ce problème est bien posé : sa résolution ne souffre d'aucune ambiguïté.

---

**Algorithme 1.1** Résolution de l'équation du premier degré  $ax = b$ .

---

**Données :**  $a$  : un réel non nul,  
 $b$  : un réel.

**Résultat :**  $x$  : un réel.

1:  $x \leftarrow b/a$

---

### 1.1.3 caractéristiques

Voici en résumé les caractéristiques d'un *bon* algorithme :

- Il ne souffre d'aucune ambiguïté  $\Rightarrow$  très clair.
- Combinaison d'opérations (actions) élémentaires.
- Pour toutes les données d'entrée, l'algorithme doit fournir un résultat en un nombre fini d'opérations.
- Il est adapté au public auquel il est destiné.

### 1.1.4 Première approche méthodologique

Nous présentons ci-dessous quelques éléments méthodologiques pour la réalisation d'un algorithme :

**Etape 1 :** Définir clairement le problème.

**Etape 2 :** Rechercher une méthode de résolution (formules, ...)

**Etape 3 :** Ecrire l'algorithme (par raffinement successif pour des algorithmes *compliqués*).

## 1.2 Pseudo-langage algorithmique

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

- variables,
- opérateurs (arithmétiques, relationnels, logiques),

- 1 • expressions,
- 2 • instructions (simples et composées),
- 3 • fonctions.

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

### 5 1.2.1 Données et constantes

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

### 8 1.2.2 Variables

9 **Définition 1.2** Une variable est un objet dont la valeur est modifiable, qui possède un nom et un type  
10 (entier, caractère, réel, complexe, ...). Elle est rangée en mémoire à partir d'une certaine adresse.

### 11 1.2.3 Opérateurs

#### 12 Opérateurs arithmétiques

Nom	Symbole	Exemple
addition	+	$a + b$
soustraction	-	$a - b$
opposé	-	$-a$
produit	*	$a * b$
division	/	$a/b$
division	^	$a^b$

#### 14 Opérateurs relationnels

Nom	Symbole	Exemple	Commentaires
identique	==	$a == b$	vrai si $a$ et $b$ ont même valeur, faux sinon.
différent	~=	$a ~= b$	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.

#### 16 Opérateurs logiques

Nom	Symbole	Exemple	Commentaires
négation	~	$\sim a$	vrai si $a$ est faux (ou nul), faux sinon.
ou		$a b$	vrai si $a$ <b>ou</b> $b$ est vrai (non nul), faux sinon.
et	&	$a\&b$	vrai si $a$ <b>et</b> $b$ sont vrais (non nul), faux sinon.

#### 18 Opérateur d'affectation

Nom	Symbole	Exemple	Commentaires
affectation	←	$a \leftarrow b$	On affecte à la variable $a$ le contenu de $b$ L'expression à gauche du symbole ← doit être une variable.

### 21 1.2.4 Expressions

22 **Définition 1.3** Une expression est un groupe d'opérandes (i.e. nombres, constantes, variables, ...) liées  
23 par certains opérateurs pour former un terme algébrique qui représente une valeur (i.e. un élément de  
24 donnée simple)

**Exemple 1** • 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)

### 1.2.5 Instructions

**Définition 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 plusieurs 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 ← 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 chaîne de caractères 'bonjour' à l'écran. Cette instruction fait appel à la fonction `affiche`.

#### Instructions composées

#### Instructions répétitives, boucle «pour»

---

#### Algorithme 1.2 Exemple de boucle «pour»

---

**Données :**  $n$  : un entier.

```
1: S ← 0
2: Pour i ← 1 à n faire
3:   S ← S + cos(i2)
4: Fin Pour
```

---

## 1 Instruction répétitive, boucle «tant que»

**Algorithme 1.3** Exemple de boucle «tant que»

---

```

1:  $i \leftarrow 0, x \leftarrow 1$ 
2: Tantque  $i < 1000$  faire
3:    $x \leftarrow x + i * i$ 
4:    $i \leftarrow i + 1$ 
5: Fin Tantque

```

---

## 2 Instruction répétitive, boucle «répéter ...jusqu'à»

**Algorithme 1.4** Exemple de boucle «répéter ...jusqu'à»

---

```

1:  $i \leftarrow 0, x \leftarrow 1$ 
2: Répéter
3:    $x \leftarrow x + i * i$ 
4:    $i \leftarrow i + 1$ 
5: jusqu'à  $i \geq 1000$ 

```

---

## 3 Instructions conditionnelles «si»

**Algorithme 1.5** Exemple d'instructions conditionnelle «si»

---

**Données :** *note* : un réel.

```

1: Si  $note > 12$  alors
2:   affiche('gagne')
3: Sinon Si  $note \geq 8$  alors
4:   affiche('oral')
5: Sinon
6:   affiche('perdu')
7: Fin Si

```

---

4 **1.2.6** Fonctions

5 Les fonctions permettent

- 6 • d'automatiser certaines tâches répétitives au sein d'un même programme,
- 7 • d'ajouter à la clarté d'un programme,
- 8 • l'utilisation de portion de code dans un autre programme,
- 9 • ...

10 **Fonctions prédéfinies**

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

14 **Syntaxe**

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

16

```

Fonction [args1, ..., argsn] ← NOMFONCTION( arge1, ..., argem )
    instructions
Fin Fonction

```

La fonction se nomme NOMFONCTION. Elle admet comme paramètres d'entrée (données) les  $m$  arguments  $arge_1, \dots, arge_m$  et comme paramètres de sortie (résultats) les  $n$  arguments  $args_1, \dots, args_n$ . Ces derniers doivent être déterminés dans le corps de la fonction (partie instructions).

Dans le cas où la fonction n'admet qu'un seul paramètre de sortie, l'écriture se simplifie :

```

Fonction args ← NOMFONCTION( arge1, ..., argem )
    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 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 1.6** 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 ← REPD( a, b )
2:   x ← -b/a
3: Fin Fonction

```

---

**Remarque 1.3** Cette fonction est très simple, toutefois pour ne pas «alourdir» le code nous n'avons pas vérifié la validité des données fournies.

**exercice 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

$$ax^2 + bx + c = 0, \tag{1.1} \text{ {eq-snd-degre}}$$

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  $\Delta = b^2 - 4ac$

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

- 4 **exercice 2** 1. Ecrire la fonction `discriminant` permettant de calculer le discriminant de l'équation  
5 (1.1).
- 6 2. Ecrire la fonction `RESD` permettant de résoudre l'équation (1.1) en utilisant la fonction `discriminant`.
- 7 3. Ecrire un programme permettant de valider ces deux fonctions.
- 8 **exercice 3** Même question que précédemment dans le cas complexe (solution et coefficients).

## 9 1.4 Méthodologie

### 10 1.4.1 Description du problème

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

### 16 1.4.2 Recherche d'une méthode de résolution

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

### 25 1.4.3 Réalisation d'un algorithme

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

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

34 Pour écrire un algorithme détaillé, il faut tout d'abord savoir répondre à quelques questions :  
35

- Que doit-il faire ? (i.e. Quel problème est-il censé résoudre?) 1
- Quelles sont les données nécessaires à la résolution de ce problème? 2
- Comment résoudre ce problème «à la main» (sur papier)? 3

Si l'on ne sait pas répondre à l'une de ces questions, l'écriture de l'algorithme est fortement comprise. 4

#### Exemple : algorithme pour une somme 5

##### Exercice 1.4.1

Ecrire un algorithme permettant de calculer

$$S(x) = \sum_{k=1}^n k \sin(2kx)$$
6

**Correction** L'énoncé de cet exercice est imprécis. On choisit 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! 7

---

**Algorithme 1.7** 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:  $S \leftarrow S + k * \sin(2 * k * x)$
  - 4: **Fin Pour**
- 

◇ 8

#### Exemple : algorithme pour un produit 9

##### Exercice 1.4.2

Ecrire un algorithme permettant de calculer

$$P(z) = \prod_{n=1}^k \sin(2kz/n)^k$$
10

**Correction** L'énoncé de cet exercice est imprécis. On choisit alors  $z \in \mathbb{R}$  et  $k \in \mathbb{N}$  pour rendre possible le calcul. 11

12

**Algorithme 1.8** Calcul de  $P = \prod_{n=1}^k \sin(2kz/n)^k$

**Données :**  $z$  : nombre réel,  
 $k$  : nombre entier.

**Résultat :**  $P$  : un réel.

- 1:  $P \leftarrow 1$
- 2: **Pour**  $n \leftarrow 1$  à  $k$  **faire**
- 3:  $P \leftarrow P * \sin(2 * k * z/n)^k$
- 4: **Fin Pour**

1

◇

## 2 Exemple : série de Fourier

### Exercice 1.4.3

{exo10}

Soit la série de Fourier

{SFT}

$$x(t) = \frac{4A}{\pi} \left\{ \cos \omega t - \frac{1}{3} \cos 3\omega t + \frac{1}{5} \cos 5\omega t - \frac{1}{7} \cos 7\omega t + \dots \right\}. \quad (1.2)$$

Notons  $x_n(t)$  la série tronquée au  $n$ -ième terme. Ecrire la fonction SFT permettant de calculer  $x_n(t)$ .

3

#### **Correction**

Nous devons écrire la fonction permettant de calculer

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

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

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

**Algorithme 1.9** En-tête de la fonction SFT retournant valeur de la série de Fourier en  $t$  tronquée au  $n$  premiers termes de l'exercice 1.4.3.

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

**Résultat :**  $x$  : un réel.

- 1: **Fonction**  $x \leftarrow \text{SFT}(t, n, A, \omega)$
- 2: ...
- 3: **Fin Fonction**

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

#### **Algorithme 1.10** $\mathcal{R}_0$

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

#### **Algorithme 1.10** $\mathcal{R}_1$

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

8

9

Algorithme 1.10  $\mathcal{R}_1$ 

```

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 1.10  $\mathcal{R}_2$ 

```

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

```

Finalement la fonction est

**Algorithme 1.10** Fonction SFT retournant la valeur de la série de Fourier en  $t$  tronquée au  $n$  premiers termes de l'exercice 1.4.3.

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


**Résultat :**  $x$  : un réel.

```

1: Fonction  $x \leftarrow \text{SFT}(t, n, A, \omega)$ 
2:    $S \leftarrow 0$ 
3:   Pour  $k = 1$  à  $n$  faire
4:      $S \leftarrow S + ((-1)^{(k+1)}) * \cos((2 * k - 1) * \omega * t) / (2 * k - 1)$ 
5:   Fin Pour
6:    $S \leftarrow 4 * A * S / \pi$ 
7: Fin Fonction

```

◇

 Exercice 1.4.4

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

## 1.5 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  $P_1, P_2, P_3, \dots$  plus simples. Puis,  $P_1$ , est lui-même analysé et décomposé en  $P_{11}, P_{12}, \dots$ , 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<sup>a</sup>!**

<sup>a</sup>améliorés pour minimiser le nombre d'opérations élémentaires,  
l'occupation mémoire, ..

1

# Chapitre 2

## Méthodes Numériques

### 2.1 Polynôme d'interpolation de Lagrange

#### 2.1.1 Définition

Soient  $n \in \mathbb{N}^*$  et  $(x_i, y_i)_{i \in \llbracket 0, n \rrbracket}$  avec  $(x_i, y_i) \in \mathbb{R}^2$  et les  $x_i$  distincts deux à deux. Le **polynôme d'interpolation de Lagrange** associé aux  $n + 1$  points  $(x_i, y_i)_{i \in \llbracket 0, n \rrbracket}$ , noté  $\mathcal{P}_n$ , est donné par

$$\mathcal{P}_n(x) = \sum_{i=0}^n y_i L_i(t), \quad \forall t \in \mathbb{R} \quad (2.1) \quad \{\text{Lagrange-eq0}\}$$

avec

$$L_i(t) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{t - x_j}{x_i - x_j}, \quad \forall i \in \llbracket 0, n \rrbracket, \quad \forall t \in \mathbb{R}. \quad (2.2) \quad \{\text{Lagrange-eq1}\}$$



#### Théorème 2.1

Le **polynôme d'interpolation de Lagrange**,  $\mathcal{P}_n$ , associé aux  $n + 1$  points  $(x_i, y_i)_{i \in \llbracket 0, n \rrbracket}$ , est l'unique polynôme de degré au plus  $n$ , vérifiant

$$\mathcal{P}_n(x_i) = y_i, \quad \forall i \in \llbracket 0, n \rrbracket. \quad (2.3) \quad \{\text{LagrangePoly}\}$$

A titre d'exemple, on représente, En figure 2.1, le polynôme d'interpolation de Lagrange associé à 7 points donnés.



#### Exercice 2.1.1

Ecrire la fonction LAGRANGE permettant de calculer  $\mathcal{P}_n$  (polynôme d'interpolation de Lagrange associé aux  $n + 1$  points  $(x_i, y_i)_{i \in \llbracket 0, n \rrbracket}$ ) au point  $t \in \mathbb{R}$ .

#### Correction

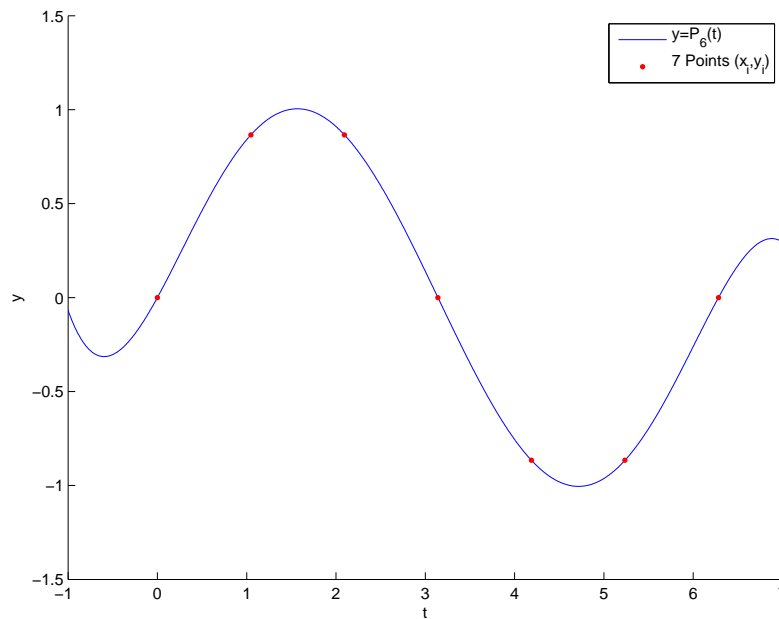


Figure 2.1: Polynôme d'interpolation de Lagrange avec 7 points donnés) <sup>figPoly1</sup>

**But :** Calculer le polynôme  $\mathcal{P}_n(t)$  défini par (2.1)

**Données :**  $\mathbf{X}$  : vecteur/tableau de  $\mathbb{R}^{n+1}$ ,  $X(i) = x_{i-1} \forall i \in \llbracket 1, n+1 \rrbracket$  et  
 $X(i) \neq X(j)$  pour  $i \neq j$ ,

$\mathbf{Y}$  : vecteur/tableau de  $\mathbb{R}^{n+1}$ ,  $Y(i) = y_{i-1} \forall i \in \llbracket 1, n+1 \rrbracket$ ,

$t$  : un réel.

**Résultat :**  $y$  : le réel  $y = \mathcal{P}_n(t)$ .

Algorithme 2.1  $\mathcal{R}_0$

1: Calcul de  $y = \mathcal{P}_n(t) = \sum_{i=1}^{n+1} Y(i)L_{i-1}(t)$

Algorithme 2.1  $\mathcal{R}_1$

1:  $y \leftarrow 0$   
 2: **Pour**  $i \leftarrow 1$  à  $n+1$  **faire**  
 3:  $y \leftarrow y + Y(i) * L_{i-1}(t)$   
 4: **Fin Pour**

Algorithme 2.1  $\mathcal{R}_1$

1:  $y \leftarrow 0$   
 2: **Pour**  $i \leftarrow 1$  à  $n+1$  **faire**  
 3:  $y \leftarrow y + Y(i) * L_{i-1}(t)$   
 4: **Fin Pour**

Algorithme 2.1  $\mathcal{R}_2$

1:  $y \leftarrow 0$   
 2: **Pour**  $i \leftarrow 1$  à  $n+1$  **faire**  
 3:  $L \leftarrow \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{t - X(j)}{X(i) - X(j)}$   
 4:  $y \leftarrow y + Y(i) * L$   
 5: **Fin Pour**

```

Algorithme 2.1  $\mathcal{R}_2$ 
1:  $y \leftarrow 0$ 
2: Pour  $i \leftarrow 1$  à  $n + 1$  faire
3:    $L \leftarrow \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{t - X(j)}{X(i) - X(j)}$ 
4:    $y \leftarrow y + Y(i) * L$ 
5: Fin Pour
    
```

```

Algorithme 2.1  $\mathcal{R}_3$ 
1:  $y \leftarrow 0$ 
2: Pour  $i \leftarrow 1$  à  $n + 1$  faire
3:    $L \leftarrow 1$ 
4:   Pour  $j \leftarrow 1$  à  $n + 1$  faire
5:     Si  $i \sim j$  alors
6:        $L \leftarrow L * (t - X(j)) / (X(i) - X(j))$ 
7:     Fin Si
8:   Fin Pour
9:    $y \leftarrow y + Y(i) * L$ 
10: Fin Pour
    
```

On obtient alors l'algorithme final

**Algorithme 2.1** Fonction LAGRANGE permettant de calculer le polynôme d'interpolation de Lagrange  $\mathcal{P}_n(x)$  défini par (2.1)

**Données :**  $\mathbf{X}$  : vecteur/tableau de  $\mathbb{R}^{n+1}$ ,  $X(i) = x_{i-1} \forall i \in \llbracket 1, n+1 \rrbracket$  et  $X(i) \neq X(j)$  pour  $i \neq j$ ,  
 $\mathbf{Y}$  : vecteur/tableau de  $\mathbb{R}^{n+1}$ ,  $Y(i) = y_{i-1} \forall i \in \llbracket 1, n+1 \rrbracket$ ,  
 $t$  : un réel.

**Résultat :**  $y$  : le réel  $y = \mathcal{P}_n(t)$ .

```

1: Fonction  $y \leftarrow \text{LAGRANGE}(t, X, Y)$ 
2:    $y \leftarrow 0$ 
3:   Pour  $i \leftarrow 1$  à  $n + 1$  faire
4:      $L \leftarrow 1$ 
5:     Pour  $j \leftarrow 1$  à  $n + 1$  faire
6:       Si  $i \sim j$  alors
7:          $L \leftarrow L * (t - X(j)) / (X(i) - X(j))$ 
8:       Fin Si
9:     Fin Pour
10:     $y \leftarrow y + Y(i) * L$ 
11:   Fin Pour
12:   return  $y$ 
13: Fin Fonction
    
```

◇ 3

### 2.1.2 Erreur de l'interpolation

Soit une fonction  $f : [a, b] \rightarrow \mathbb{R}$ . On suppose que les  $y_i$  sont donnés par

$$y_i = f(x_i), \quad \forall i \in \llbracket 0, n \rrbracket. \tag{2.4}$$

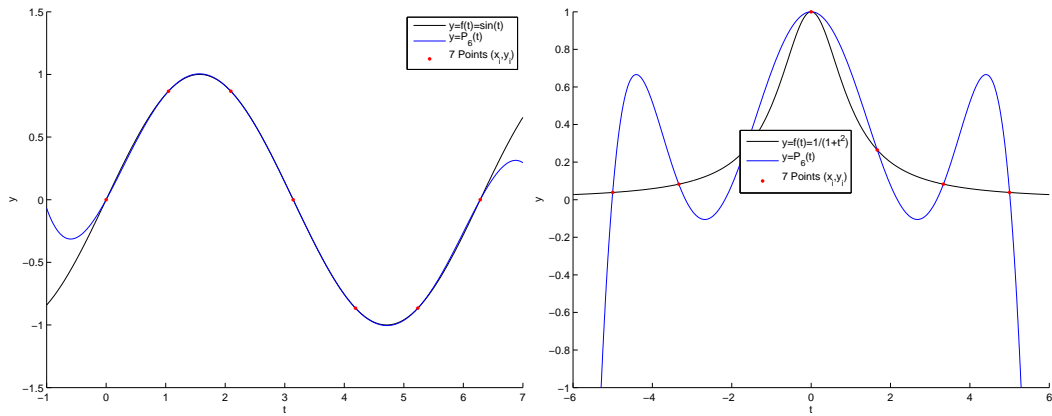
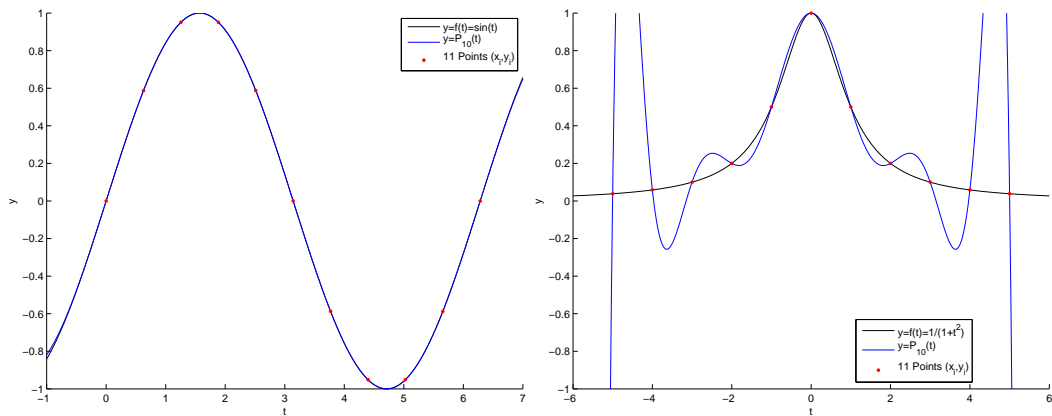
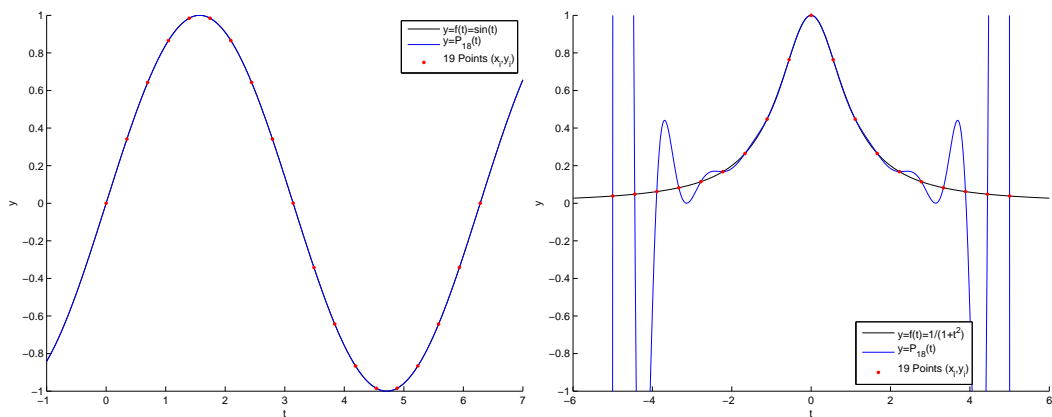
On cherche à évaluer l'erreur  $E_n(t) = f(x) - \mathcal{P}_n(t)$ ,  $\forall t \in [a, b]$ .

On propose en figures 2.2 à 2.4 d'étudier deux exemples. Le premier (figure de gauche) correspond à l'interpolation de la fonction  $\sin(t)$  par le polynôme d'interpolation de Lagrange aux points équidistants  $t_i = a + ih$  avec  $a = 0$  et  $h = 2\pi/n$ . Le second (figure de droite) correspond à l'interpolation de la fonction  $\frac{1}{1+t^2}$  par le polynôme d'interpolation de Lagrange aux points  $x_i = a + ih$  avec  $a = -5$  et  $h = 10/n$ .

Le résultat suivant est dû à Cauchy (1840)



#### Théorème 2.2

Figure 2.2: Erreurs d'interpolation avec  $n = 6$  `figPoly6`Figure 2.3: Erreurs d'interpolation avec  $n = 10$  `figPoly10`Figure 2.4: Erreurs d'interpolation avec  $n = 18$  `figPoly18`

Soit  $f : [a, b] \rightarrow \mathbb{R}$ , une fonction  $(n + 1)$ -fois différentiable et soit  $\mathcal{P}_n(t)$  le polynôme d'interpolation de degré  $n$  passant par  $(x_i, f(x_i))$ ,  $\forall i \in \llbracket 0, n \rrbracket$ . Alors,

{LagrangeCauchy}

$$\forall t \in [a, b], \exists \xi_t \in (\min(x_i, t), \max(x_i, t)), \quad f(t) - \mathcal{P}_n(t) = \frac{f^{(n+1)}(\xi_t)}{(n+1)!} \prod_{i=0}^n (t - x_i) \quad (2.5)$$

1

### 2.1.3 Points de Chebyshev

2

Pour minimiser l'erreur commise lors de l'interpolation d'une fonction  $f$  par un polynôme d'interpolation de Lagrange, on peut, pour un  $n$  donné, "jouer" sur le choix des points  $x_i$  :

3

Trouver  $(\bar{x}_i)_{i=0}^n$ ,  $\bar{x}_i \in [a, b]$ , distincts deux à deux, tels que

4

5

$$\max_{t \in [a, b]} \prod_{i=0}^n |t - \bar{x}_i| \leq \max_{t \in [a, b]} \prod_{i=0}^n |t - x_i|, \quad \forall (x_i)_{i=0}^n, x_i \in [a, b], \text{ distincts 2 à 2} \quad (2.6) \quad \{\text{eqpoly10}\}$$

On a alors le résultat suivant

6



#### Théorème 2.3

Les points réalisant (2.6) sont les points de Chebyshev donnés par

$$\bar{x}_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{(2i+1)\pi}{2n+2}\right), \quad \forall i \in \llbracket 0, n \rrbracket. \quad (2.7) \quad \{\text{eqpoly11}\}$$

7

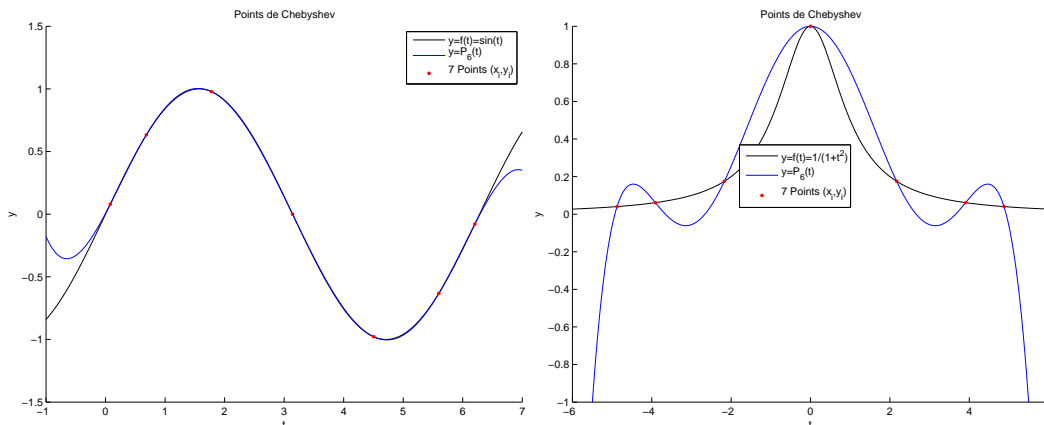


Figure 2.5: Erreurs d'interpolation avec  $n = 6$  figPolyCheb6

## 2.2 Dérivation numérique

8

### Approximations de dérivées premières

9

On propose de chercher une approximation de la dérivée première de  $f$  en un point  $\bar{x} \in ]a, b[$ . On rappelle que la dérivée d'une fonction est définie par

10

11

**Définition 2.4** Une fonction  $f$  définie sur un intervalle  $[a, b]$  est dérivable en un point  $\bar{x} \in ]a, b[$  si la limite suivante existe et est finie

12

13

$$f'(\bar{x}) = \lim_{h \rightarrow 0} \frac{1}{h} (f(\bar{x} + h) - f(\bar{x})) \quad (2.8) \quad \{\text{defderivee}\}$$

On peut donc approcher  $f'(\bar{x})$ , pour  $h$  suffisamment petit par  $\frac{1}{h}(f(\bar{x} + h) - f(\bar{x}))$ . On en déduit alors les deux approximations, pour  $h > 0$ ,

14

15

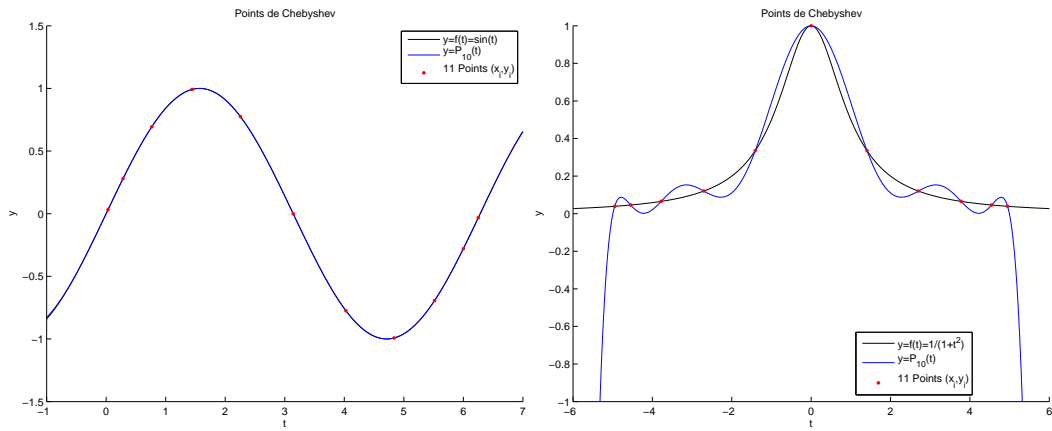


Figure 2.6: Erreurs d'interpolation avec  $n = 10$  `figPolyCheb10`

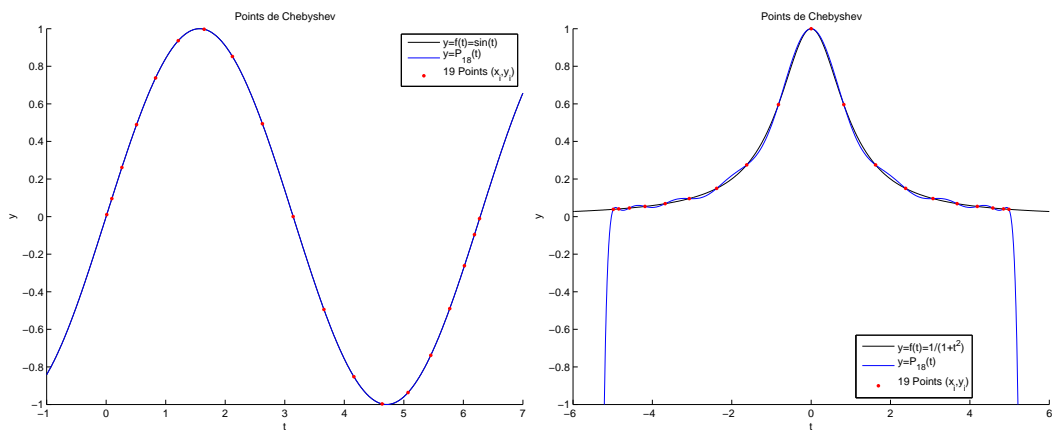


Figure 2.7: Erreurs d'interpolation avec  $n = 18$  `figPolyCheb18`


différence finie progressive :

$$f'(\bar{x}) \approx \frac{f(\bar{x} + h) - f(\bar{x})}{h} \tag{2.9} \quad \{\text{DFP}\}$$

différence finie rétrograde :

$$f'(\bar{x}) \approx \frac{f(\bar{x}) - f(\bar{x} - h)}{h} \tag{2.10} \quad \{\text{DFR}\}$$

Pour estimer l'erreur, on va utiliser le développement de Taylor :

 **Théorème 2.5: Taylor-Lagrange**

On suppose que  $f \in \mathcal{C}^{n+1}$  sur  $I$ . Alors, pour tout  $h \in \mathbb{R}$  tel que  $\bar{x} + h$  appartienne à  $I$ , il existe  $\theta_h \in ]0, 1[$  tel que l'on ait

$$f(\bar{x} + h) = \sum_{k=0}^n \frac{h^k}{k!} f^{(k)}(\bar{x}) + \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(\bar{x} + \theta_h h) \tag{2.11}$$

En effet, si  $f \in \mathcal{C}^2(]a, b[)$ , on a

$$f(\bar{x} + h) = f(\bar{x}) + hf'(\bar{x}) + \frac{h^2}{2} f^{(2)}(\xi)$$


avec  $\xi \in ]\min(\bar{x}, \bar{x} + h), \max(\bar{x}, \bar{x} + h)[$ .

On obtient alors

$$\frac{f(\bar{x} + h) - f(\bar{x})}{h} = f'(\bar{x}) + \frac{h}{2} f^{(2)}(\xi).$$

On dit que les formules (2.9) et (2.10) sont des approximations d'ordre 1 de  $f'(\bar{x})$  par rapport à  $h$ . Ici, l'ordre est donc la puissance de  $h$  dans la formule précédente.

Il est aussi possible d'obtenir ces approximations en dérivant les polynômes d'interpolation associés aux points  $\{\bar{x}, \bar{x} + h\}$  et  $\{\bar{x} - h, \bar{x}\}$ .

 **Exercice 2.2.1**

On note  $x_i = a + ih$ ,  $i \in \llbracket 0, n \rrbracket$ , une discrétisation régulière de l'intervalle  $[a, b]$ . Soit une fonction  $f : [a, b] \rightarrow \mathbb{R}$  suffisamment régulière. On suppose que les  $y_i$  sont donnés par

$$y_i = f(x_i), \quad \forall i \in \llbracket 0, n \rrbracket. \tag{2.12}$$

Ecrire une fonction DERIVE1 permettant de calculer des approximations d'ordre 1 de  $f'(x_i)$  pour  $i \in \llbracket 0, n \rrbracket$ .

Pour obtenir une formule d'approximation d'ordre 2 de  $f'(\bar{x})$ , on suppose  $f \in \mathcal{C}^3(]a, b[)$  et on peut alors développer les formules de Taylor de  $f(\bar{x} + h)$  et  $f(\bar{x} - h)$  jusqu'au troisième ordre :  $\exists \xi_+ \in ]\bar{x}, \bar{x} + h[$ ,  $\exists \xi_- \in ]\bar{x} - h, \bar{x}[$ ,

$$f(\bar{x} + h) = f(\bar{x}) + hf'(\bar{x}) + \frac{h^2}{2} f^{(2)}(\bar{x}) + \frac{h^3}{3!} f^{(3)}(\xi_+) \tag{2.13}$$

$$f(\bar{x} - h) = f(\bar{x}) - hf'(\bar{x}) + \frac{h^2}{2} f^{(2)}(\bar{x}) - \frac{h^3}{3!} f^{(3)}(\xi_-) \tag{2.14}$$

$$\tag{2.15}$$

En soustrayant ces deux équations, on obtient alors

$$f(\bar{x} + h) - f(\bar{x} - h) = 2hf'(\bar{x}) + \frac{h^3}{3!} (f^{(3)}(\xi_+) + f^{(3)}(\xi_-))$$

ce qui donne

$$f'(\bar{x}) + \frac{h^2}{12} (f^{(3)}(\xi_+) + f^{(3)}(\xi_-)) = \frac{f(\bar{x} + h) - f(\bar{x} - h)}{2h}.$$

Une approximation à l'ordre de 2 de  $f'(\bar{x})$  est donnée par

$$f'(\bar{x}) = \frac{f(\bar{x} + h) - f(\bar{x} - h)}{2h} + \mathcal{O}(h^2). \tag{2.16}$$

Cette approximation est la formule des **différences finies centrées**.


**Exercice 2.2.2**

On note  $x_i = a + ih$ ,  $i \in \llbracket 0, n \rrbracket$ , une discrétisation régulière de l'intervalle  $[a, b]$ . Soit une fonction  $f : [a, b] \rightarrow \mathbb{R}$  suffisamment régulière. On suppose que les  $y_i$  sont donnés par

$$y_i = f(x_i), \quad \forall i \in \llbracket 0, n \rrbracket. \quad (2.17)$$

Ecrire une fonction DERIVE2 permettant de calculer des approximations d'ordre 2 de  $f'(x_i)$  pour  $i \in \llbracket 0, n \rrbracket$ .

**Correction** Pour les points  $x_i$ ,  $i \in \llbracket 1, n-1 \rrbracket$ , on utilise la formule des différences finies centrées. Cette dernière n'est pas utilisable pour  $i = 0$  et  $i = n$ . Il faut donc trouver d'autres formules d'ordre 2.

- En  $\bar{x} = x_0$ , on développe les formules de Taylor de  $f(\bar{x} + h)$  et  $f(\bar{x} + 2h)$  jusqu'au troisième ordre :  $\exists \xi_1 \in ]\bar{x}, \bar{x} + h[$ ,  $\exists \xi_2 \in ]\bar{x}, \bar{x} + 2h[$ ,

$$\begin{aligned} f(\bar{x} + h) &= f(\bar{x}) + hf'(\bar{x}) + \frac{h^2}{2}f^{(2)}(\bar{x}) + \frac{h^3}{3!}f^{(3)}(\xi_1) \\ f(\bar{x} + 2h) &= f(\bar{x}) + 2hf'(\bar{x}) + \frac{(2h)^2}{2}f^{(2)}(\bar{x}) + \frac{(2h)^3}{3!}f^{(3)}(\xi_2) \end{aligned}$$

ce qui donne

$$f'(\bar{x}) + \frac{h}{2}f^{(2)}(\bar{x}) + \frac{h^2}{6}f^{(3)}(\xi_1) = \frac{f(\bar{x} + h) - f(\bar{x})}{h} \quad (2.18)$$

$$f'(\bar{x}) + hf^{(2)}(\bar{x}) + \frac{(2h)^2}{3!}f^{(3)}(\xi_2) = \frac{f(\bar{x} + 2h) - f(\bar{x})}{2h} \quad (2.19)$$

On effectue la combinaison linéaire 2(2.18)-(2.19) pour éliminer le terme en  $h^1$  et on obtient

$$\begin{aligned} f'(\bar{x}) + 2\frac{h^2}{6}f^{(3)}(\xi_1) - \frac{(2h)^2}{3!}f^{(3)}(\xi_2) &= 2\frac{f(\bar{x} + h) - f(\bar{x})}{h} - \frac{f(\bar{x} + 2h) - f(\bar{x})}{2h} \\ &= -\frac{f(\bar{x} + 2h) - 4f(\bar{x} + h) + 3f(\bar{x})}{2h} \end{aligned}$$

Une approximation à l'ordre 2 de  $f'(\bar{x})$  utilisant les valeurs de  $f$  aux points  $\{\bar{x}, \bar{x} + h, \bar{x} + 2h\}$  est donnée par

$$f'(\bar{x}) = -\frac{f(\bar{x} + 2h) - 4f(\bar{x} + h) + 3f(\bar{x})}{2h} + O(h^2). \quad (2.20)$$

- En  $\bar{x} = x_n$ , on développe les formules de Taylor de  $f(\bar{x} - h)$  et  $f(\bar{x} - 2h)$  jusqu'au troisième ordre :  $\exists \xi_1 \in ]\bar{x} - h, \bar{x}[$ ,  $\exists \xi_2 \in ]\bar{x} - 2h, \bar{x}[$ ,

$$\begin{aligned} f(\bar{x} - h) &= f(\bar{x}) - hf'(\bar{x}) + \frac{h^2}{2}f^{(2)}(\bar{x}) - \frac{h^3}{3!}f^{(3)}(\xi_1) \\ f(\bar{x} - 2h) &= f(\bar{x}) - 2hf'(\bar{x}) + \frac{(2h)^2}{2}f^{(2)}(\bar{x}) - \frac{(2h)^3}{3!}f^{(3)}(\xi_2) \end{aligned}$$

ce qui donne

$$f'(\bar{x}) - \frac{h}{2}f^{(2)}(\bar{x}) + \frac{h^2}{6}f^{(3)}(\xi_1) = \frac{f(\bar{x}) - f(\bar{x} - h)}{h} \quad (2.21)$$

$$f'(\bar{x}) - hf^{(2)}(\bar{x}) + \frac{(2h)^2}{3!}f^{(3)}(\xi_2) = \frac{f(\bar{x}) - f(\bar{x} - 2h)}{2h} \quad (2.22)$$

On effectue la combinaison linéaire 2(2.21)-(2.22) pour éliminer le terme en  $h^1$  et on obtient

$$\begin{aligned} f'(\bar{x}) + 2\frac{h^2}{6}f^{(3)}(\xi_1) - \frac{(2h)^2}{3!}f^{(3)}(\xi_2) &= 2\frac{f(\bar{x}) - f(\bar{x} - h)}{h} - \frac{f(\bar{x}) - f(\bar{x} - 2h)}{2h} \\ &= \frac{3f(\bar{x}) - 4f(\bar{x} - h) + f(\bar{x} - 2h)}{2h} \end{aligned}$$

{DerExo02: eq1}

{DerExo02: eq2}

{DerExo02: eq3}

{DerExo02: eq4}

Une approximation à l'ordre 2 de  $f'(\bar{x})$  utilisant les valeurs de  $f$  aux points  $\{\bar{x}, \bar{x} - h, \bar{x} - 2h\}$  est donnée par

$$f'(\bar{x}) = \frac{3f(\bar{x}) - 4f(\bar{x} - h) + f(\bar{x} - 2h)}{2h} + \mathcal{O}(h^2). \quad (2.23)$$

◇

### Approximations de dérivées seconde

Il est possible aussi d'obtenir des approximations de dérivées d'ordre supérieure. Par exemple, pour obtenir une approximation de  $f^{(2)}(\bar{x})$ , on suppose  $f \in \mathcal{C}^4(]a, b[)$  et on peut alors développer les formules de Taylor de  $f(\bar{x} + h)$  et  $f(\bar{x} - h)$  jusqu'au quatrième ordre :  $\exists \xi_+ \in ]\bar{x}, \bar{x} + h[$ ,  $\exists \xi_- \in ]\bar{x} - h, \bar{x}[$ .

$$f(\bar{x} + h) = f(\bar{x}) + hf'(\bar{x}) + \frac{h^2}{2}f^{(2)}(\bar{x}) + \frac{h^3}{3!}f^{(3)}(\bar{x}) + \frac{h^4}{4!}f^{(4)}(\xi_+) \quad (2.24)$$

$$f(\bar{x} - h) = f(\bar{x}) - hf'(\bar{x}) + \frac{h^2}{2}f^{(2)}(\bar{x}) - \frac{h^3}{3!}f^{(3)}(\bar{x}) + \frac{h^4}{4!}f^{(4)}(\xi_-) \quad (2.25)$$

$$(2.26)$$

En sommant ces deux équations, on obtient alors

$$f(\bar{x} + h) + f(\bar{x} - h) = 2f(\bar{x}) + h^2f^{(2)}(\bar{x}) + \frac{h^4}{4!}(f^{(4)}(\xi_+) + f^{(4)}(\xi_-))$$

ce qui donne

$$f^{(2)}(\bar{x}) + \frac{h^2}{4!}(f^{(4)}(\xi_+) + f^{(4)}(\xi_-)) = \frac{f(\bar{x} + h) - 2f(\bar{x}) + f(\bar{x} - h)}{h^2}.$$

Une approximation à l'ordre 2 de  $f^{(2)}(\bar{x})$  est donnée par

$$\frac{f(\bar{x} + h) - 2f(\bar{x}) + f(\bar{x} - h)}{h^2}. \quad (2.27)$$

## 2.3 Intégration numérique

Soit  $f$  une fonction définie et intégrable sur un intervalle  $[a, b]$  donné. On propose de chercher une approximation de

$$I = \int_a^b f(x)dx$$

dans le cas où l'on ne connaît pas de primitive de  $f$ .

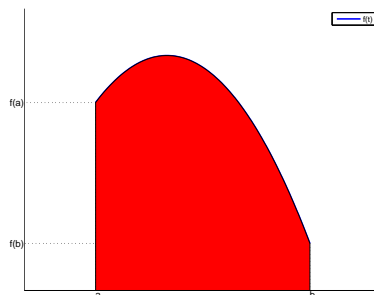


Figure 2.8: Représentation de  $\int_a^b f(x)dx$  (aire en rouge)

**Définition 2.6** On dit qu'une formule d'intégration (ou formule de quadrature) est d'ordre  $n$  ou a pour degré d'exactitude  $n$  si elle est exacte pour les polynômes de degré inférieur ou égal à  $n$ .

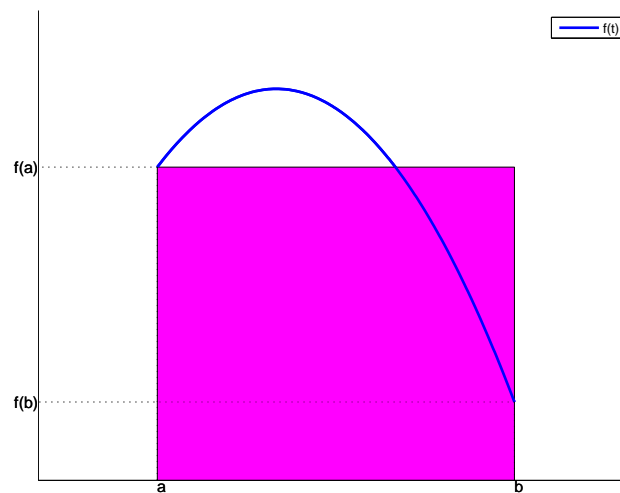


Figure 2.9: Formule du rectangle à gauche :  $\int_a^b f(x)dx \approx \int_a^b f(a)dx$ <sup>IntFigRG</sup>

### 2.3.1 Méthodes simpliste

On peut approcher  $f$  par un polynôme constant. Les trois formules usuels sont

**Méthode du rectangle à gauche :** En figure 2.9, on représente l'approximation de  $\int_a^b f(x)dx$  lorsque  $f$  est approché par le polynôme constant  $P(x) = f(a)$ . On a alors

$$\int_a^b f(x)dx \approx (b-a)f(a), \text{ formule du rectangle (à gauche)}$$

et son degré d'exactitude est 0.

**Méthode du rectangle à droite :** En figure 2.10, on représente l'approximation de  $\int_a^b f(x)dx$  lorsque  $f$  est approché par le polynôme constant  $P(x) = f(b)$ . On a alors

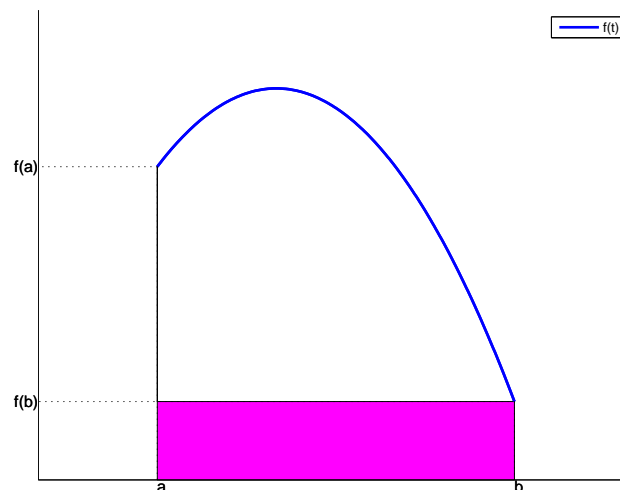


Figure 2.10: Formule du rectangle à droite :  $\int_a^b f(x)dx \approx \int_a^b f(b)dx$ <sup>IntFigRD</sup>

$$\int_a^b f(x)dx \approx (b-a)f(b), \text{ formule du rectangle (à droite)}$$

et son degré d'exactitude est 0.

**Méthode du point milieu :** En figure 2.11, on représente l'approximation de  $\int_a^b f(x)dx$  lorsque  $f$  est approché par le polynôme constant  $P(x) = f((a+b)/2)$ . On a alors

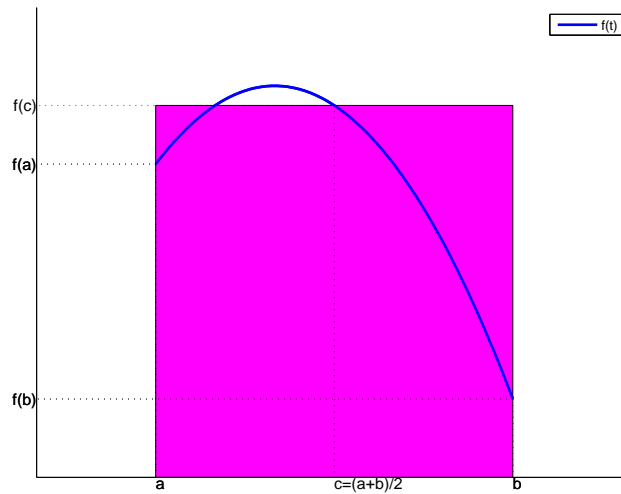


Figure 2.11: Formule du point milieu :  $\int_a^b f(x)dx \approx \int_a^b f((a+b)/2)dx$  IntFigPM

$$\int_a^b f(x)dx \approx (b-a)f\left(\frac{a+b}{2}\right), \text{ formule du point milieu}$$

et son degré d'exactitude est 1.

La précision de ces formules est toute relative!

### 2.3.2 Méthodes de Newton-Cotes

Soit  $(x_i)_{i \in [0, n]}$  une discrétisation régulière de l'intervalle  $[a, b]$ :  $x_i = a + ih$  avec  $h = (b-a)/n$ .

Sur l'intervalle  $[a, b]$ , on approche  $f$  par son polynôme d'interpolation  $\mathcal{P}_n$  aux points  $(x_i, f(x_i))_{i \in [0, n]}$ . D'après (2.1), on a

$$\mathcal{P}_n(x) = \sum_{i=0}^n L_i(x)f(x_i)$$

et, en utilisant le théorème 2.2, on obtient

$$\forall x \in [a, b], \exists \xi_x \in [a, b], f(x) - \mathcal{P}_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{i=0}^n (x - x_i) \tag{2.28}$$

On a donc

$$\int_a^b f(x)dx - \int_a^b \mathcal{P}_n(x)dx = \int_a^b \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{i=0}^n (x - x_i)dx. \tag{2.29}$$

De plus

$$\int_a^b \mathcal{P}_n(x)dx = \sum_{i=0}^n f(x_i) \int_a^b L_i(x)dx. \tag{2.30}$$

Les formules de Newton-Cotes s'écrivent sous la forme

$$\sum_{i=0}^n \alpha_i f(x_i)$$

où  $\alpha_i = \int_a^b L_i(x)dx$ . En posant  $\alpha_i = hAw_i$ , on a le tableau suivant

$n$	$A$	$w_0$	$w_1$	$w_2$	$w_3$	$w_4$	nom	ordre
1	1/2	1	1				trapèzes	1
2	1/3	1	4	1			Simpson	3
3	3/8	1	3	3	1		Simpson	3
4	2/45	7	32	12	32	7	Villarceau	5

- 1 Par exemple, la formule de Simpson ( $n = 2$ ) est

$$\int_a^b f(x)dx \approx \frac{b-a}{6} \left( f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right) \quad (2.31) \quad \{\text{Intégrati}$$

Pour un  $n$  donné, déterminer les coefficients  $\alpha_i$  de la formule de Newton-Cotes est assez simple. En effet, il suffit de remarquer que la formule doit être exacte si  $f$  est un polynôme de degré au plus  $n$ . Ensuite par linéarité de la formule, on est ramené à résoudre un système linéaire à  $n$  inconnues (les  $\alpha_i$ ) en écrivant que pour chaque monôme

$$\sum_{i=0}^n \alpha_i f(x_i) = \int_a^b f(x)dx, \quad \forall f \in \{1, X, X^2, \dots, X^n\}.$$

Par exemple, pour  $n = 2$ , on a  $b = a + 2h$  et les trois équations :

$$\begin{cases} \alpha_0 + \alpha_1 + \alpha_2 & = 2h, \quad (f(x) = 1) \\ a\alpha_0 + (a+h)\alpha_1 + (a+2h)\alpha_2 & = \int_a^b x dx = 2ah + 2h^2, \quad (f(x) = x) \\ a^2\alpha_0 + (a+h)^2\alpha_1 + (a+2h)^2\alpha_2 & = \int_a^b x^2 dx = \frac{1}{3}(6a^2h + 12ah^2 + 8h^3), \quad (f(x) = x^2). \end{cases}$$

On a alors

$$\begin{cases} \alpha_0 = -\alpha_1 - \alpha_2 + 2h, \\ a(-\alpha_1 - \alpha_2 + 2h) + (a+h)\alpha_1 + (a+2h)\alpha_2 = 2ah + 2h^2, \\ a^2(-\alpha_1 - \alpha_2 + 2h) + (a+h)^2\alpha_1 + (a+2h)^2\alpha_2 = \frac{1}{3}(6a^2h + 12ah^2 + 8h^3). \end{cases}$$

c'est à dire

$$\begin{cases} \alpha_0 = -\alpha_1 - \alpha_2 + 2h, \\ \alpha_1 + 2\alpha_2 = 2h, \\ 2ah(\alpha_1 + 2\alpha_2) + h^2(\alpha_1 + 4\alpha_2) = 4ah^2 + 8h^3/3, \end{cases}$$

Par substitution, de la deuxième équation dans la troisième, on obtient enfin

$$\begin{cases} \alpha_0 = -\alpha_1 - \alpha_2 + 2h, \\ \alpha_1 + 2\alpha_2 = 2h, \\ \alpha_1 + 4\alpha_2 = 8h/3, \end{cases}$$

- 2 ce qui donne  $\alpha_0 = \alpha_2 = h/3$  et  $\alpha_1 = 4h/3$ .

- 3 Pour les méthode de Newton-Cotes, il ne faut pas trop "monter" en ordre car le phénomène de Runge  
4 (forte oscillation possible du polynôme d'interpolation sur les bords de l'intervalle) peut conduire à de  
5 très grande erreurs.

### 2.3.3 Méthodes composites

- 7 Ces méthodes consistent en l'utilisation de la relation de Chasles pour décomposer l'intégrale en une  
8 somme d'intégrales sur des domaines plus petits puis sur ces dernières on applique une formule d'intégration  
9 numérique.

Soit  $(x_k)_{k \in \llbracket 0, n \rrbracket}$  une discrétisation régulière de l'intervalle  $[a, b]$ :  $x_k = a + kh$  avec  $h = (b - a)/n$ . On a alors

$$\int_a^b f(x)dx = \sum_{k=1}^n \int_{x_{k-1}}^{x_k} f(x)dx.$$

#### 10 Formule composite des points milieux

On note  $m_k = \frac{x_{k-1} + x_k}{2}$  le point milieu de l'intervalle  $[x_{k-1}, x_k]$ . On approche alors chacune des intégrales par la formule des points milieux

$$\int_{I_k} f(x)dx \approx hf(m_k)$$

- 11 pour obtenir

$$\int_a^b f(x)dx = h \sum_{k=1}^n f(m_k) + \mathcal{O}(h^2). \quad (2.32)$$

- 12 C'est une formule d'ordre 2 par rapport à  $h$ .

 **Exercice 2.3.1**

Soit  $f$  une fonction définie sur l'intervalle  $[a, b]$ . Ecrire la fonction QUADPM permettant de calculer une approximation de l'intégrale de  $f$  sur  $[a, b]$  par la méthode composite des points milieux.

1

**2.3.4** Formule composite des trapèzes

2

On approche chacune des intégrales par la formule des trapèzes

$$\int_{I_k} f(x)dx \approx h \left( \frac{f(x_{k-1}) + f(x_k)}{2} \right)$$


pour obtenir

3

$$\int_a^b f(x)dx = h \sum_{k=1}^n \left( \frac{f(x_{k-1}) + f(x_k)}{2} \right) + \mathcal{O}(h^2). \quad (2.33)$$

C'est une formule d'ordre 2 par rapport à  $h$ .

4

 **Exercice 2.3.2**

Soit  $f$  une fonction définie sur l'intervalle  $[a, b]$ . Ecrire la fonction QUADTRAPEZE permettant de calculer une approximation de l'intégrale de  $f$  sur  $[a, b]$  par la méthode composite des trapèzes.

5

**2.3.5** Formule composite de Simpson

6

On approche chacune des intégrales par la formule de Simpson (ordre 3)

$$\int_{I_k} f(x)dx \approx \frac{h}{6} (f(x_{k-1}) + 4f(m_k) + f(x_k))$$


pour obtenir

7

$$\int_a^b f(x)dx = \frac{h}{6} \sum_{k=1}^n (f(x_{k-1}) + 4f(m_k) + f(x_k)) + \mathcal{O}(h^4). \quad (2.34)$$

C'est une formule d'ordre 4 par rapport à  $h$ .

8

 **Exercice 2.3.3**

Soit  $f$  une fonction définie sur l'intervalle  $[a, b]$ . Ecrire la fonction QUADSIMPSON permettant de calculer une approximation de l'intégrale de  $f$  sur  $[a, b]$  par la méthode composite de Simpson.

9

Il est possible de "retrouver" numériquement l'ordre des différentes méthodes en représentant en échelle logarithmique (en abscisses et ordonnées) la fonction erreur  $h \rightarrow \text{erreur}(h)$  pour chacune des méthodes. Par exemple, pour la méthode composite de Simpson, on a vu que pour un  $h$  donné (i.e. un  $n$  donné)

$$\int_a^b f(x)dx = \frac{h}{6} \sum_{k=1}^n (f(x_{k-1}) + 4f(m_k) + f(x_k)) + Ch^4$$

où  $C$  est indépendant de  $h$ .

L'erreur  $E_{\text{Simpson}}$  de la méthode s'écrit donc

$$E_{\text{Simpson}}(h) = Ch^4.$$

De plus  $\log(E(h)) = \log(C) + 4\log(h)$ , donc en échelle logarithmique, on va représenter  $\log(h) \rightarrow \log(E_{\text{Simpson}}(h))$  qui est une droite de pente 4. En figure 2.12, on représente en échelle logarithmique les différentes erreurs ainsi que les fonctions  $h \rightarrow h^2$  et  $h \rightarrow h^4$ .

10

11

12

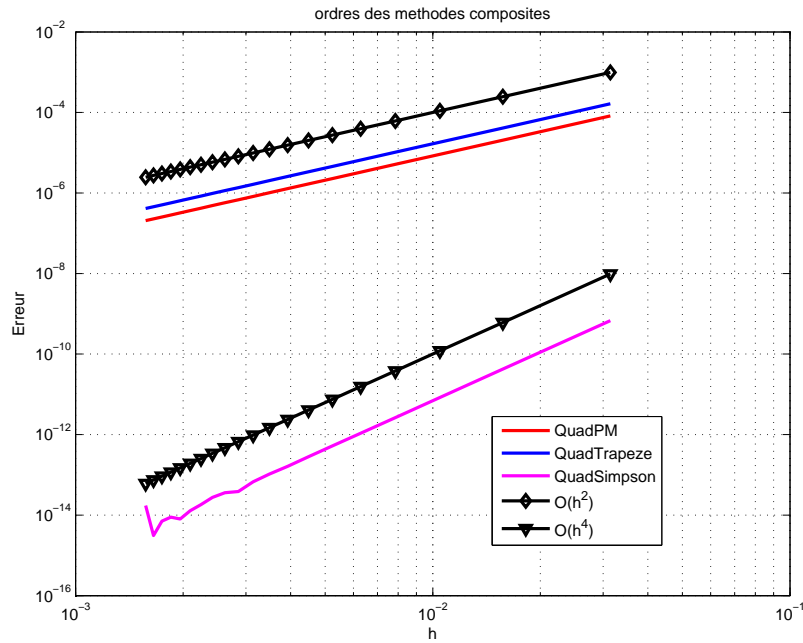


Figure 2.12: Ordre des méthodes composites pour le calcul de  $\int_0^\pi \sin(x) dx$  IntegrationOrdre

### 2.3.6 Autres méthodes

Il existe un grand nombre de méthodes d'intégration numérique :

- méthode de Gauss-Legendre,
- méthode de Gauss-Tchebychev,
- méthode de Gauss-Laguerre,
- méthode de Gauss-Hermitte,
- méthode de Gauss-Lobatto,
- méthode de Romberg...

### 2.3.7 Intégrales multiples

On veut approcher, en utilisant la formule de Simpson, l'intégrale

$$I = \int_a^b \int_c^d f(x, y) dy dx$$

Par utilisation de la formule de quadrature de Simpson (2.31) en  $y$  on a

$$g(x) = \int_c^d f(x, y) dy \approx \tilde{g}(x) = \frac{d-c}{6} \left( f(x, c) + 4f\left(x, \frac{c+d}{2}\right) + f(x, d) \right).$$

Une nouvelle utilisation de Simpson en  $x$  donne

$$\begin{aligned} I &= \int_a^b g(x) dx \approx \frac{b-a}{6} \left( g(a) + 4g\left(\frac{a+b}{2}\right) + g(b) \right) \\ &\approx \frac{b-a}{6} \left( \tilde{g}(a) + 4\tilde{g}\left(\frac{a+b}{2}\right) + \tilde{g}(b) \right) \end{aligned}$$

En posant  $\alpha = \frac{a+b}{2}$  et  $\beta = \frac{c+d}{2}$ , on obtient la formule de quadrature de Simpson "2D" :

$$I \approx \frac{b-a}{6} \frac{d-c}{6} \left( \begin{array}{l} f(a, c) + 4f(a, \beta) + f(a, d) \\ + 4(f(\alpha, c) + 4f(\alpha, \beta) + f(\alpha, d)) \\ + f(b, c) + 4f(b, \beta) + f(b, d) \end{array} \right) \quad (2.35)$$

La méthodologie pour obtenir la formule composite de Simpson "2D" est la suivante

1. Discrétisation régulière de  $[a, b]$ :  $\forall k \in \llbracket 0, n \rrbracket, x_k = a + kh_x$  avec  $h_x = \frac{b-a}{n}$ .
2. Discrétisation régulière de  $[c, d]$ :  $\forall l \in \llbracket 0, m \rrbracket, y_l = a + lh_y$  avec  $h_y = \frac{d-c}{m}$ .
3. Relation de Chasles :

$$\int_a^b \int_c^d f(x, y) dy dx = \sum_{k=1}^n \sum_{l=1}^m \int_{x_{k-1}}^{x_k} \int_{y_{l-1}}^{y_l} f(x, y) dy dx.$$

4. Formule composite de Simpson "2D" :

$$\begin{aligned} & \int_a^b \int_c^d f(x, y) dy dx \\ &= \\ & \frac{h_x h_y}{36} \sum_{k=1}^n \sum_{l=1}^m \left( \begin{aligned} & f(x_{k-1}, y_{l-1}) + 4f(x_{k-1}, \beta_l) + f(x_{k-1}, y_l) \\ & + 4(f(\alpha_k, y_{l-1}) + 4f(\alpha_k, \beta_l) + f(\alpha_k, y_l)) \\ & + f(x_k, y_{l-1}) + 4f(x_k, \beta_l) + f(x_k, y_l) \end{aligned} \right) \end{aligned}$$

avec  $\alpha_k = \frac{x_{k-1} + x_k}{2}$  et  $\beta_l = \frac{y_{l-1} + y_l}{2}$ .

## 2.4 Algèbre linéaire

Dans cette partie, on suppose le lecteur avoir quelques connaissances en algèbre linéaire. De plus, du point de vue algorithmique, on suppose l'existence des types matrices et vecteurs.

### 2.4.1 Vecteurs

#### Exercice 2.4.1

Soient  $\mathbf{x}$  et  $\mathbf{y}$  deux vecteurs de  $\mathbb{R}^n$  et  $\alpha \in \mathbb{R}$ . Ecrire la fonction VECAXPY permettant de calculer  $\mathbf{z} = \alpha\mathbf{x} + \mathbf{y}$ .

**Correction** On pose  $\mathbf{z} = \alpha\mathbf{x} + \mathbf{y}$ .  $\mathbf{z}$  est un vecteur de  $\mathbb{R}^n$  et

$$z_i = \alpha x_i + y_i, \forall i \in \llbracket 1, n \rrbracket.$$

On donne ici, à titre d'exemple, les raffinements successifs pour obtenir l'algorithme final

**Algorithme 2.2**  $\mathcal{R}_0$

1: Calculer  $\mathbf{z} \leftarrow \alpha\mathbf{x} + \mathbf{y}$ .

**Algorithme 2.2**  $\mathcal{R}_1$

1: **Pour**  $i \leftarrow 1$  à  $n$  faire  
 2:  $z_i \leftarrow \alpha x_i + y_i$   
 3: **Fin Pour**

On abouti alors à

**Algorithme 2.2** Fonction VECAXPY retournant  $\mathbf{z} = \alpha\mathbf{x} + \mathbf{y}$  avec  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$  et  $\alpha \in \mathbb{R}$


**Données :**  $\mathbf{x}, \mathbf{y}$  : deux vecteurs de  $\mathbb{R}^n$   
 $\alpha$  : un réel.

**Résultat :**  $\mathbf{z}$  : vecteur de  $\mathbb{R}^n$ .

- 1: **Fonction**  $\mathbf{z} \leftarrow \text{VECAXPY}(\alpha, \mathbf{x}, \mathbf{y})$
- 2: **Pour**  $i \leftarrow 1$  à  $n$  faire
- 3:  $z(i) \leftarrow \alpha * x(i) + y(i)$
- 4: **Fin Pour**
- 5: **Fin Fonction**

◇ 13

- 1 **Remarque 2.5** 1. On a fait le choix de ne pas mettre comme donnée explicite l'entier  $n$ . En effet, on  
 2 peut considérer que l'entier  $n$  est donné implicitement par les vecteurs  $\mathbf{x}$  et  $\mathbf{y}$ .
- 3 2. Aucun test sur les dimensions des vecteurs et le type des données n'est réalisé afin de ne pas  
 4 «alourdir» l'algorithme : on suppose donc que cette fonction sera correctement utilisée, c'est à dire  
 5 que les hypothèses sur les données seront bien vérifiées.

 **Exercice 2.5.1**

Soient  $\mathbf{x}$  et  $\mathbf{y}$  deux vecteurs de  $\mathbb{R}^n$ .

1. Ecrire la fonction VEC DOT permettant de calculer le produit scalaire entre les vecteurs  $\mathbf{x}$  et  $\mathbf{y}$ .
2. Ecrire la fonction VEC NORM1 permettant de calculer

$$\{\text{VEC NORM1}\} \quad \|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|, \quad \forall \mathbf{x} \in \mathbb{R}^n. \quad (2.36)$$

3. Ecrire la fonction VEC NORM2 permettant de calculer

$$\|\mathbf{x}\|_2 = \left( \sum_{i=1}^n |x_i|^2 \right)^{1/2}, \quad \forall \mathbf{x} \in \mathbb{R}^n.$$

**Correction**

1. On a,  $\forall \mathbf{x} \in \mathbb{R}^n, \forall \mathbf{y} \in \mathbb{R}^n$ ,

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i y_i.$$

---

**Algorithme 2.3** Fonction VEC DOT permettant de calculer le produit scalaire des vecteurs  $\mathbf{x}$  et  $\mathbf{y}$  où  $\mathbf{x} \in \mathbb{R}^n$  et  $\mathbf{y} \in \mathbb{R}^n$ .

---

**Données :**

- $\mathbf{x}$  : vecteur de  $\mathbb{R}^n$ ,  
 $\mathbf{y}$  : vecteur de  $\mathbb{R}^n$ .

**Résultat :**  $s$  : le réel tel que  $s = \langle \mathbf{x}, \mathbf{y} \rangle$ .

- 1: **Fonction**  $s \leftarrow \text{VEC DOT}(\mathbf{x}, \mathbf{y})$
  - 2:  $s \leftarrow 0$
  - 3: **Pour**  $i \leftarrow 1$  à  $n$  **faire**
  - 4:      $s \leftarrow s + x(i) * y(i)$
  - 5: **Fin Pour**
  - 6: **Fin Fonction**
- 

2. On a,  $\forall \mathbf{x} \in \mathbb{R}^n$ ,

$$\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|.$$


---

---

**Algorithme 2.4** Fonction VECNORM1 permettant de retourner  $\|\mathbf{x}\|_1$  avec  $\mathbf{x} \in \mathbb{R}^n$

---

**Données :**

$\mathbf{x}$  : vecteur de  $\mathbb{R}^n$ .

**Résultat :**

$s$  : le réel tel que  $s = \|\mathbf{x}\|_1$ .

- 1: **Fonction**  $s \leftarrow \text{VECNORM1}(\mathbf{x})$
  - 2:  $s \leftarrow 0$
  - 3: **Pour**  $i = 1$  à  $n$  **faire**
  - 4:      $s \leftarrow s + \text{ABS}(x(i))$
  - 5: **Fin Pour**
  - 6: **Fin Fonction**
- 

3. On a

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2} = \langle \mathbf{x}, \mathbf{x} \rangle^{1/2}.$$

---

**Algorithme 2.5** Fonction VECNORM2 permettant de retourner  $\|\mathbf{x}\|_2$  avec  $\mathbf{x} \in \mathbb{R}^n$

---

**Données :**

$\mathbf{x}$  : vecteur de  $\mathbb{R}^n$ .

**Résultat :**

$s$  : le réel tel que  $s = \|\mathbf{x}\|_2$ .

- 1: **Fonction**  $s \leftarrow \text{VECNORM2}(\mathbf{x})$
  - 2:  $s \leftarrow 0$
  - 3: **Pour**  $i \leftarrow 1$  à  $n$  **faire**
  - 4:      $s \leftarrow s + x(i) * x(i)$
  - 5: **Fin Pour**
  - 6:  $s \leftarrow \text{SQRT}(s)$
  - 7: **Fin Fonction**
- 

ou encore en utilisant la fonction VEC DOT :

1

---

**Algorithme 2.6** Fonction VECNORM2 permettant de retourner  $\|\mathbf{x}\|_2$  avec  $\mathbf{x} \in \mathbb{R}^n$  (utilise la fonction VEC DOT).

---

**Données :**

$\mathbf{x}$  : vecteur de  $\mathbb{R}^n$ .

**Résultat :**

$s$  : le réel tel que  $s = \|\mathbf{x}\|_2$ .

- 1: **Fonction**  $s \leftarrow \text{VECNORM2}(\mathbf{x})$
  - 2:  $s \leftarrow \text{SQRT}(\text{VECDOT}(\mathbf{x}, \mathbf{x}))$
  - 3: **Fin Fonction**
- 

◇

2

## 2.5.1 Matrices

3



### Exercice 2.5.2

Soient  $\mathbb{X}$  et  $\mathbb{Y}$  deux matrices de  $\mathcal{M}_{m,n}(\mathbb{R})$  et  $\alpha \in \mathbb{R}$  Ecrire la fonction MATAXPY permettant de retourner  $\mathbb{Z} = \alpha\mathbb{X} + \mathbb{Y}$ .

4

**Correction** On note  $Z_{i,j}$  les composantes de la matrice  $Z \in \mathcal{M}_{m,n}(\mathbb{R})$ . On a alors

$$\forall i \in \llbracket 1, m \rrbracket, \forall j \in \llbracket 1, n \rrbracket, Z_{i,j} = \alpha X_{i,j} + Y_{i,j}.$$

**Algorithme 2.7** Fonction MATAXPY permettant de retourner  $Z = \alpha X + Y$  avec  $X$  et  $Y$  dans  $\mathcal{M}_{m,n}(\mathbb{R})$ , et  $\alpha \in \mathbb{R}$

**Données :**

$\alpha$  : un réel,  
 $X$  : matrice de  $\mathcal{M}_{m,n}(\mathbb{R})$ ,  
 $Y$  : matrice de  $\mathcal{M}_{m,n}(\mathbb{R})$ .

**Résultat :**

$Z$  : matrice de  $\mathcal{M}_{m,n}(\mathbb{R})$  tel que  $Z = \alpha X + Y$ .

```

1: Fonction Z ← MATAXPY( α, X, Y )
2:   Pour i ← 1 à m faire
3:     Pour j ← 1 à n faire
4:       Z(i, j) ← α * X(i, j) + Y(i, j)
5:     Fin Pour
6:   Fin Pour
7: Fin Fonction

```

1

◇

### Exercice 2.5.3

2 Ecrire la fonction MATTRANSPOSE permettant de retourner la transposé d'une matrice.

**Correction** Soit  $X \in \mathcal{M}_{m,n}(\mathbb{R})$  et  $Z = X^t$  la transposée de  $X$ . Alors  $Z \in \mathcal{M}_{n,m}(\mathbb{R})$  et

$$\forall i \in \llbracket 1, n \rrbracket, \forall j \in \llbracket 1, m \rrbracket, Z_{i,j} = X_{j,i}.$$

**Algorithme 2.8** Fonction MATTRANSPOSE permettant de retourner  $Z = X^t$  avec  $X \in \mathcal{M}_{m,n}(\mathbb{R})$

**Données :**

$X$  : matrice de  $\mathcal{M}_{m,n}(\mathbb{R})$ .

**Résultat :**

$Z$  : matrice de  $\mathcal{M}_{n,m}(\mathbb{R})$  tel que  $Z = X^t$ .

```

1: Fonction Z ← MATTRANSPOSE( X )
2:   Pour i ← 1 à n faire
3:     Pour j ← 1 à m faire
4:       Z(i, j) ← X(j, i)
5:     Fin Pour
6:   Fin Pour
7: Fin Fonction

```

3

◇

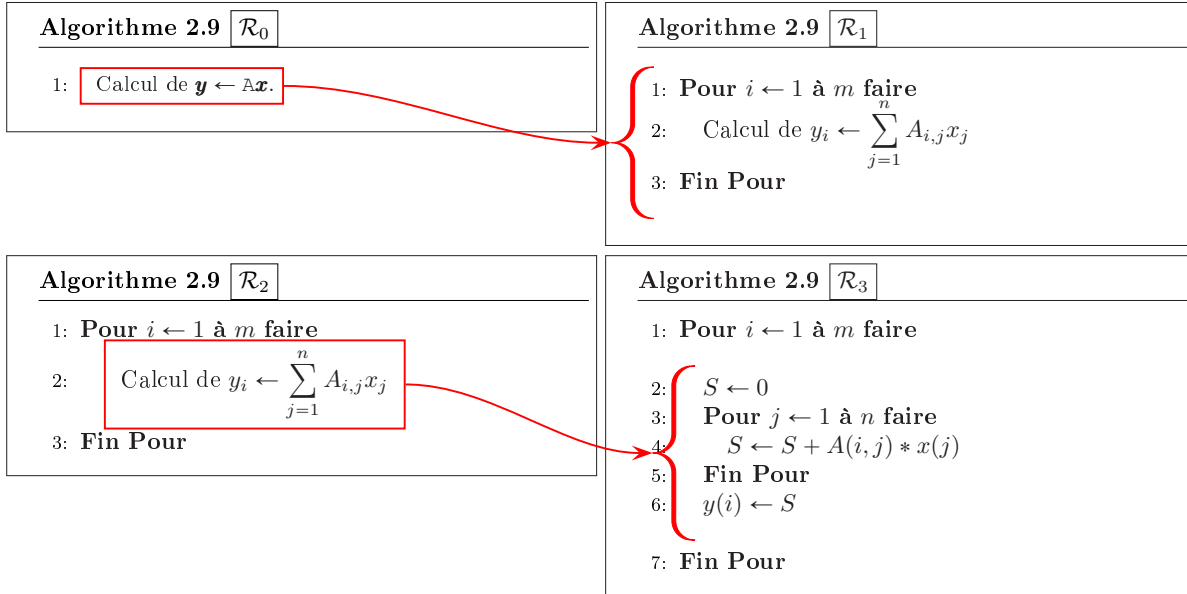
### Exercice 2.5.4

4 Ecrire la fonction MATMULT permettant de retourner le produit d'une matrice par un vecteur.

**Correction** On rappelle que le produit d'une matrice  $A \in \mathcal{M}_{m,n}(\mathbb{R})$  par un vecteur  $x \in \mathbb{R}^n$  est un vecteur de  $\mathbb{R}^m$ . On le note  $y$  et on a

$$y_i = \sum_{j=1}^n A_{i,j} x_j, \forall i \in \llbracket 1, m \rrbracket,$$

On écrit en détail les raffinements successifs permettant d'aboutir à l'algorithme final de telle manière que le passage entre deux raffinements successifs soit le plus simple possible.



On obtient alors

**Algorithme 2.9** Fonction MATMULT permettant de retourner le vecteur  $\mathbf{y} \in \mathbb{R}^m$  tel que:

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

avec  $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{R})$  et  $\mathbf{x} \in \mathbb{R}^n$

**Données :**

- $\mathbf{A}$  : matrice de  $\mathcal{M}_{m,n}(\mathbb{R})$ ,
- $\mathbf{x}$  : vecteur de  $\mathbb{R}^n$ .

**Résultat :**

- $\mathbf{y}$  : vecteur de  $\mathbb{R}^m$  tel que  $\mathbf{y} = \mathbf{A}\mathbf{x}$ .

- 1: **Fonction**  $\mathbf{x} \leftarrow \text{MATMULT}(\mathbf{A}, \mathbf{x})$
- 2: **Pour**  $i \leftarrow 1$  à  $m$  **faire**
- 3:    $S \leftarrow 0$
- 4:   **Pour**  $j \leftarrow 1$  à  $n$  **faire**
- 5:      $S \leftarrow S + A(i,j) * x(j)$
- 6:   **Fin Pour**
- 7:    $y(i) \leftarrow S$
- 8: **Fin Pour**
- 9: **Fin Fonction**

◇

**Exercice 2.5.5**

Ecrire la fonction MATMATMULT permettant de retourner le produit de deux matrices.

**Correction**

Soient  $\mathbb{X} \in \mathcal{M}_{m,n}(\mathbb{R})$  et  $\mathbb{Y} \in \mathcal{M}_{n,p}(\mathbb{R})$ . On note  $\mathbb{Z} \in \mathcal{M}_{m,p}(\mathbb{R})$  la matrice produit i.e.  $\mathbb{Z} = \mathbb{X}\mathbb{Y}$ .

On rappelle que l'on a

$$Z_{i,j} = \sum_{k=1}^n X_{i,k}Y_{k,j}, \quad \forall (i,j) \in \llbracket 1, m \rrbracket \times \llbracket 1, p \rrbracket.$$

On écrit en détail les raffinements successifs permettant d'aboutir à l'algorithme final de telle manière que le passage entre deux raffinements successifs soit le plus simple possible.

Algorithme 2.10  $\mathcal{R}_0$ 1: Calcul de  $Z = XY$ .Algorithme 2.10  $\mathcal{R}_1$ 

1: Pour  $i \leftarrow 1$  à  $m$  faire  
 2: Calcul de la ligne  $i$  de la matrice  $Z$   
 3: Fin Pour

Algorithme 2.10  $\mathcal{R}_2$ 

1: Pour  $i \leftarrow 1$  à  $m$  faire  
 2: Calcul de la ligne  $i$  de la matrice  $Z$   
 3: Fin Pour

Algorithme 2.10  $\mathcal{R}_3$ 

1: Pour  $i \leftarrow 1$  à  $m$  faire  
 2: Pour  $j \leftarrow 1$  à  $p$  faire  
 3: Calcul de  $Z_{i,j} \leftarrow \sum_{k=1}^n X_{i,k} Y_{k,j}$   
 4: Fin Pour  
 5: Fin Pour

Algorithme 2.10  $\mathcal{R}_4$ 

1: Pour  $i \leftarrow 1$  à  $m$  faire  
 2: Pour  $j \leftarrow 1$  à  $p$  faire  
 3: Calcul de  $Z_{i,j} \leftarrow \sum_{k=1}^n X_{i,k} Y_{k,j}$   
 4: Fin Pour  
 5: Fin Pour

Algorithme 2.10  $\mathcal{R}_5$ 

1: Pour  $i \leftarrow 1$  à  $m$  faire  
 2: Pour  $j \leftarrow 1$  à  $p$  faire  
 3:  $S \leftarrow 0$   
 4: Pour  $k \leftarrow 1$  à  $n$  faire  
 5:  $S \leftarrow S + X_{i,k} Y_{k,j}$   
 6: Fin Pour  
 7:  $Z_{i,j} \leftarrow S$   
 8: Fin Pour  
 9: Fin Pour

On obtient alors

**Algorithme 2.10** Fonction MATMATMULT permettant de retourner la matrice  $Z \in \mathcal{M}_{m,p}(\mathbb{R})$  tel que  $Z = XY$

avec  $X \in \mathcal{M}_{m,n}(\mathbb{R})$  et  $Y \in \mathcal{M}_{n,p}(\mathbb{R})$ .

**Données :**

$X$  : matrice de  $\mathcal{M}_{m,n}(\mathbb{R})$ ,  
 $Y$  : matrice de  $\mathcal{M}_{n,p}(\mathbb{R})$ .

**Résultat :**

$Z$  : matrice de  $\mathcal{M}_{m,p}(\mathbb{R})$  telle que  $Z = XY$ .

1: **Fonction**  $Z \leftarrow \text{MATMATMULT}(X, Y)$   
 2: **Pour**  $i \leftarrow 1$  à  $m$  faire  
 3: **Pour**  $j \leftarrow 1$  à  $p$  faire  
 4:  $S \leftarrow 0$   
 5: **Pour**  $k \leftarrow 1$  à  $n$  faire  
 6:  $S \leftarrow S + X(i, k) * Y(k, j)$   
 7: **Fin Pour**  
 8:  $Z(i, j) \leftarrow S$   
 9: **Fin Pour**  
 10: **Fin Pour**  
 11: **Fin Fonction**

◇

**Exercice 2.5.6**

Ecrire la fonction MATNORM1 permettant de retourner la norme d'une matrice  $A \in \mathcal{M}_{m,n}(\mathbb{R})$

$$\|A\|_1 = \max_{j \in \llbracket 1, n \rrbracket} \left( \sum_{i=1}^m |A_{i,j}| \right).$$

1

**Correction** Soit  $A \in \mathcal{M}_{m,n}(\mathbb{R})$ . On écrit en détail les raffinements successifs permettant d'aboutir à l'algorithme final de telle manière que le passage entre deux raffinements successifs soit le plus simple possible.

2  
3  
4

**Algorithme 2.11**  $\mathcal{R}_0$

1: Calcul de  $x = \|A\|_1$ .

**Algorithme 2.11**  $\mathcal{R}_1$

1:  $x \leftarrow 0$   
 2: **Pour**  $j \leftarrow 1$  à  $n$  faire  
 3:  $x \leftarrow \max \left( x, \sum_{i=1}^m |A_{i,j}| \right)$   
 4: **Fin Pour**

5

**Algorithme 2.11**  $\mathcal{R}_2$

1:  $x \leftarrow 0$   
 2: **Pour**  $j \leftarrow 1$  à  $n$  faire  
 3:  $x \leftarrow \max \left( x, \sum_{i=1}^m |A_{i,j}| \right)$ .  
 4: **Fin Pour**

**Algorithme 2.11**  $\mathcal{R}_3$

1:  $x \leftarrow 0$   
 2: **Pour**  $j \leftarrow 1$  à  $n$  faire  
 3:  $S \leftarrow \sum_{i=1}^m |A_{i,j}|$   
 4: **Si**  $x < S$  alors  
 5:  $x \leftarrow S$   
 6: **Fin Si**  
 7: **Fin Pour**

6

**Algorithme 2.11**  $\mathcal{R}_4$

1:  $x \leftarrow 0$   
 2: **Pour**  $j \leftarrow 1$  à  $n$  faire  
 3:  $S \leftarrow \sum_{i=1}^m |A_{i,j}|$   
 4: **Si**  $x < S$  alors  
 5:  $x \leftarrow S$   
 6: **Fin Si**  
 7: **Fin Pour**

**Algorithme 2.11**  $\mathcal{R}_5$

1:  $x \leftarrow 0$   
 2: **Pour**  $j \leftarrow 1$  à  $n$  faire  
 3:  $S \leftarrow 0$   
 4: **Pour**  $i \leftarrow 1$  à  $m$  faire  
 5:  $S \leftarrow S + \text{abs}(A_{i,j})$   
 6: **Fin Pour**  
 7: **Si**  $x < S$  alors  
 8:  $x \leftarrow S$   
 9: **Fin Si**  
 10: **Fin Pour**

7

On obtient alors

8

**Algorithme 2.11** Fonction MATNORM1 permettant de retourner la norme 1 de la matrice  $A \in \mathcal{M}_{m,n}(\mathbb{R})$  donnée par :

$$\|A\|_1 = \max_{j \in [1,n]} \left( \sum_{i=1}^m |A_{i,j}| \right).$$

**Données :**

$A$  : matrice de  $\mathcal{M}_{m,n}(\mathbb{R})$ .

**Résultat :**

$x$  : un réel tel que  $x = \|A\|_1$ .

```

1: Fonction  $x \leftarrow \text{MATNORM1}(A)$ 
2:    $M \leftarrow 0$ 
3:   Pour  $j \leftarrow 1$  à  $n$  faire
4:      $S \leftarrow 0$ 
5:     Pour  $i \leftarrow 1$  à  $m$  faire
6:        $S \leftarrow S + \text{abs}(A(i,j))$ 
7:     Fin Pour
8:     Si  $M < S$  alors
9:        $M \leftarrow S$ 
10:    Fin Si
11:  Fin Pour
12:   $x \leftarrow M$ 
13: Fin Fonction

```

1

◇