Méthodes Numériques I - Notes de cours [a]

• • • • • • •

Ingénieurs Sup'Galilée en Energétique
Formation en alternance

a. En cours de rédaction... Version du 31 mars 2014

par Cuvelier François

Université Paris Nord - Institut Galilée - LAGA
Av. J.-B. Clément 93430 Villetaneuse
email : cuvelier@math.univ-paris13.fr

Table des matières

Chapitre 1

Algoritmique Numérique

1.1 Introduction

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

Ceci est l'objet de l'algoritmique :

Définition 1 (Petit Robert 97) ***Algorithmique :*** *Enchaînement d'actions nécessaires à l'accomplissement d'une tâche.*

1.1.1 Exemple 1 : permutation de deux voitures

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

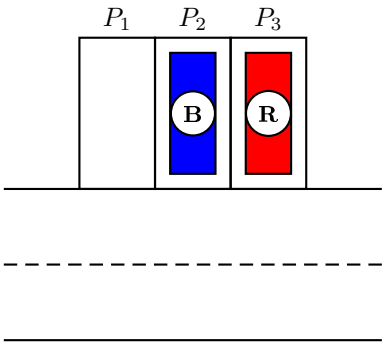

FIGURE 1.1: Avant permutation

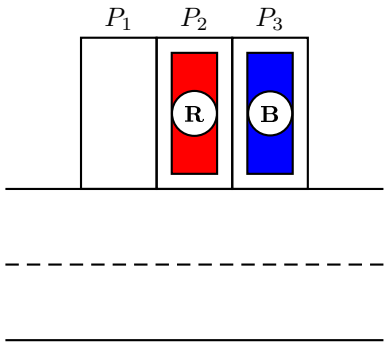

FIGURE 1.2: Après permutation

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

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.

Malheureusement, cet algorithme souffre de nombreuses lacunes et ambiguités :

- Au départ, l'emplacement P_1 est-il libre ?
- Que veut-dire l'action **déplacer** ? Si les voitures sont non-roulantes, prendre une grue ? sinon a-t'on les clés des deux véhicules ?

– Dans l'énoncé, il est dit «sans géner la circulation» ! Donc il y a de la circulation et à tout moment un véhicule extérieur au problème peut venir occuper un emplacement libre !
– ...

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

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

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

Données :
Parking de trois emplacements, numérotés de P_1 à P_3.
Deux voitures,l'une notée B et l'autre R.

Ensuite, nous précisons les **hypothèses** sur les données
Hypothèses sur les données :

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

Ensuite, nous donnons des hypothèses plus générales

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

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

1. La voiture B est sur l'emplacement P_3.
2. La voiture R est sur l'emplacement P_2.

Nous obtenons alors l'algorithme suivant :

```
1: Aller au volant du véhicule R (emplacement P3).
2: Tantque emplacement P1 est occupé faire
3:     Attendre.
4: Fin Tantque
5: Déplacer le véhicule R de l'emplacement P3 à P1
6: Aller au volant du véhicule B (emplacement P2).
7: Tantque emplacement P3 est occupé faire
8:     Attendre.
9: Fin Tantque
10: Déplacer le véhicule B de l'emplacement P2 à P3
11: Aller au volant du véhicule R (emplacement P1).
12: Tantque emplacement P2 est occupé faire
13:     Attendre.
14: Fin Tantque
15: Déplacer le véhicule R de l'emplacement P1 à P2
```

1.1.2 Exemple 2 : résolution d'une équation

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

Voici, l'«algoritme» 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 ambiguitée.

Algorithme 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 ambiguité ⇒ 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),
- expressions,
- instructions (simples et composées),
- fonctions.

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

1.2.1 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. π, la constante de gravitation,...)

1.2.2 Variables

Définition 2 *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.*

1.2.3 Opérateurs

Opérateurs arithmétiques

Nom	Symbole	Exemple
addition	$+$	$a+b$
soustraction	$-$	$a-b$
opposé	$-$	$-a$
produit	$*$	$a*b$
division	$/$	a/b
division	$\hat{}$	$a\hat{}b$

Opérateurs relationnels

Nom	Symbole	Exemple	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.

Opérateurs logiques

Nom	Symbole	Exemple	Commentaires
négation	$\sim$	$\sim a$	vrai si a est faux (ou nul), faux sinon.
ou	$\vert$	$a\vert b$	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.

Opérateur d'affectation

Nom	Symbole	Exemple	Commentaires
affectation	$\leftarrow$	$a \leftarrow b$	On affecte à la variable a le contenu de b

L'expression à gauche du symbole $\leftarrow$ doit être une variable.

1.2.4 Expressions

Définition 3 *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 – *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 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 2 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 3 Exemple de boucle «tant que»

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

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

Algorithme 4 Exemple de boucle «répéter ...jusqu'à»

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

Instructions conditionnelles «si»

Algorithme 5 Exemple d'instructions conditionnelle «si»

Données : *note* : un réel.

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

1.2.6 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 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 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** $x \leftarrow$ REPD(a, b)
2: $x \leftarrow -b/a$
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

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

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

1.4 Méthodologie

1.4.1 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

1.4.2 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.

1.4.3 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.

Exemple : algorithme pour une somme

Exercice 4 *Ecrire un algorithme permettant de calculer*

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

Correction 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 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: $\quad S \leftarrow S + k * \sin(2 * k * x)$
4: **Fin Pour**

◊

Exemple : algorithme pour un produit

Exercice 5 *Ecrire un algorithme permettant de calculer*

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

Correction 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 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: $P \leftarrow P * \sin(2 * k * z/n)\hat{}k$
4: **Fin Pour**

$\Diamond$

Exemple : série de Fourier

Exercice 6 *Soit la série de Fourier*

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

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

Correction Nous devons écrire la fonction permettant de calculer

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

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 9 En-tête de la fonction SFT retournant valeur de la série de Fourier en t tronquée au n premiers termes de l'exercice 6.

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

1: **Fonction** $x \leftarrow$ 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 10- $\mathcal{R}_0$

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 10- $\mathcal{R}_1$

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

Algorithme 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 10- $\mathcal{R}_2$

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 10 Fonction SFT retournant la valeur de la série de Fourier en t tronquée au n premiers termes de l'exercice 6.

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 7 *Reprendre 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 $P_1,\ P_2,\ P_3, \ldots$ 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 d'écrire des algorithmes !
Ceux-ci ne seront pas optimisés [a] !

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

Chapitre 2

Méthodes Numériques

2.1 Polynôme d'interpolation de Lagrange

2.1.1 Définition

Soient $n \in \mathbb{N}^*$ et $(x_i, y_i)_{i\in[\![0,n]\!]}$ 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 $(x_i, y_i)_{i\in[\![0,n]\!]}$, noté $\mathcal{P}_n$, est donné par

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

avec

$$L_i(t) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t - x_j}{x_i - x_j}, \ \forall i \in [\![0, n]\!], \ \forall t \in \mathbb{R}. \tag{2.2}$$

Théoreme 5 *Le **polynôme d'interpolation de Lagrange**, $\mathcal{P}_n$, associé aux $n+1$ points $(x_i, y_i)_{i\in[\![0,n]\!]}$, est l'unique polynôme de degré au plus n, vérifiant*

$$\mathcal{P}_n(x_i) = y_i, \ \forall i \in [\![0, n]\!]. \tag{2.3}$$

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

Exercice 8 *Ecrire la fonction* LAGRANGE *permettant de calculer $\mathcal{P}_n$ (polynôme d'interpolation de Lagrange associé aux $n+1$ points $(x_i, y_i)_{i\in[\![0,n]\!]}$) au point $t \in \mathbb{R}$.*

Correction

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

Données : $\boldsymbol{X}$: vecteur/tableau de $\mathbb{R}^{n+1}$, $X(i) = x_{i-1}$ $\forall i \in [\![1, n+1]\!]$ et $X(i) \neq X(j)$ pour $i \neq j$,

$\boldsymbol{Y}$: vecteur/tableau de $\mathbb{R}^{n+1}$, $Y(i) = y_{i-1}$ $\forall i \in [\![1, n+1]\!]$,

t : un réel.

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

Algorithme 11- $\mathcal{R}_0$

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

Algorithme 11- $\mathcal{R}_1$

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

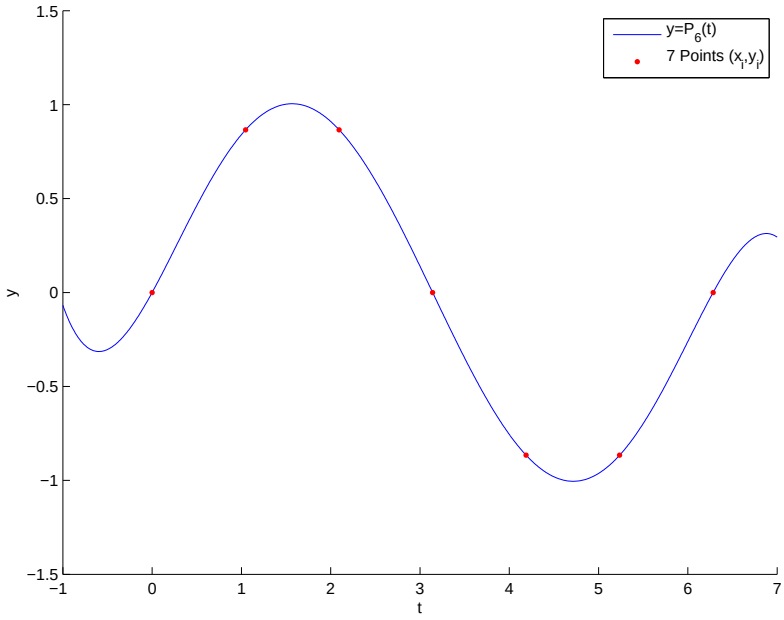

FIGURE 2.1: Polynôme d'interpolation de Lagrange avec 7 points donnés)

Algorithme 11- $\mathcal{R}_1$

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 11- $\mathcal{R}_2$

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 11- $\mathcal{R}_2$

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 11- $\mathcal{R}_3$

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

Données : $\boldsymbol{X}$: vecteur/tableau de $\mathbb{R}^{n+1}$, $X(i) = x_{i-1}$ $\forall i \in [\![1, n+1]\!]$ et $X(i) \neq X(j)$ pour $i \neq j$,
$\boldsymbol{Y}$: vecteur/tableau de $\mathbb{R}^{n+1}$, $Y(i) = y_{i-1}$ $\forall i \in [\![1, n+1]\!]$,
t : un réel.
Résultat : y : le réel $y = \mathcal{P}_n(t)$.

1: **Fonction** $y \leftarrow$ 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**

◊

2.1.2 Erreur de l'interpolation

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

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

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'etudier 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 $h = 10/n$.

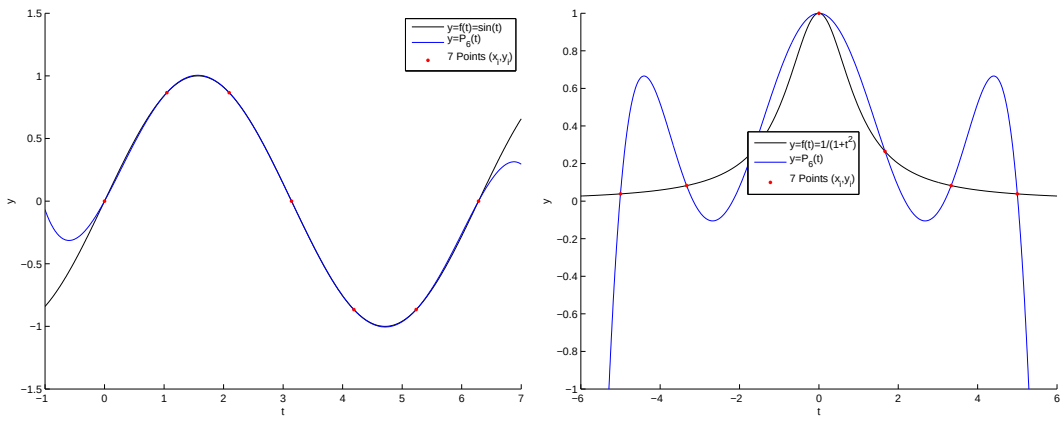

FIGURE 2.2: Erreurs d'interpolation avec $n = 6$

Le résultat suivant est du à Cauchy (1840)

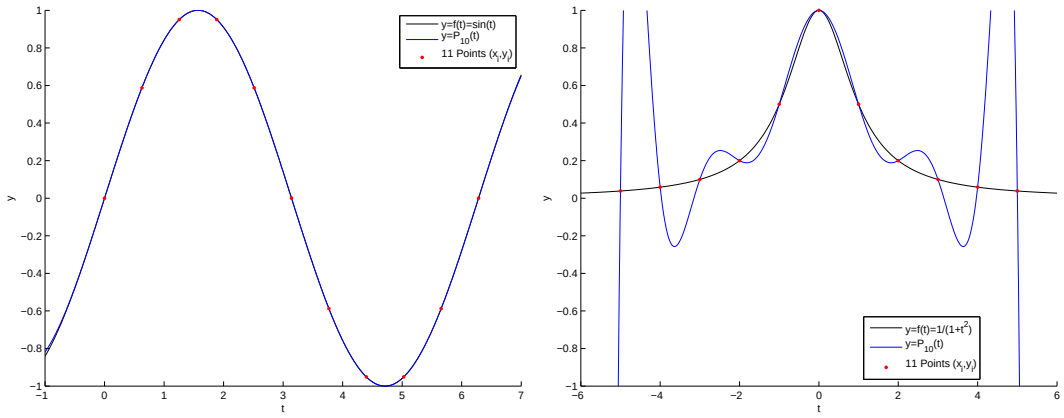

FIGURE 2.3: Erreurs d'interpolation avec $n = 10$

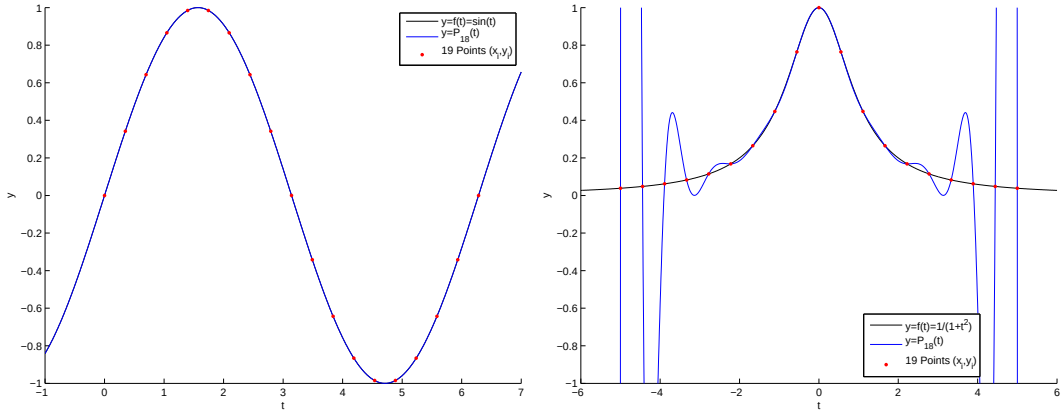

FIGURE 2.4: Erreurs d'interpolation avec $n = 18$

Théoreme 6 *Soit $f : [a,b] \longrightarrow \mathbb{R}$, une fonction $(n+1)$-fois différentiable et soit $\mathcal{P}_n(t)$ le polynôme d'interpolation de degré n passant par $(x_i, f(x_i))$, $\forall i \in [\![0,n]\!]$. Alors,*

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

2.1.3 Points de Chebyshev

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

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

On a alors le résultat suivant

Théoreme 7 *Les points réalisant (2.6) sont les points de Chebyshev donnés par*

$$\bar{x}_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{(2i+1)\pi}{2n+2}\right),\ \forall i \in [\![0,n]\!]. \tag{2.7}$$

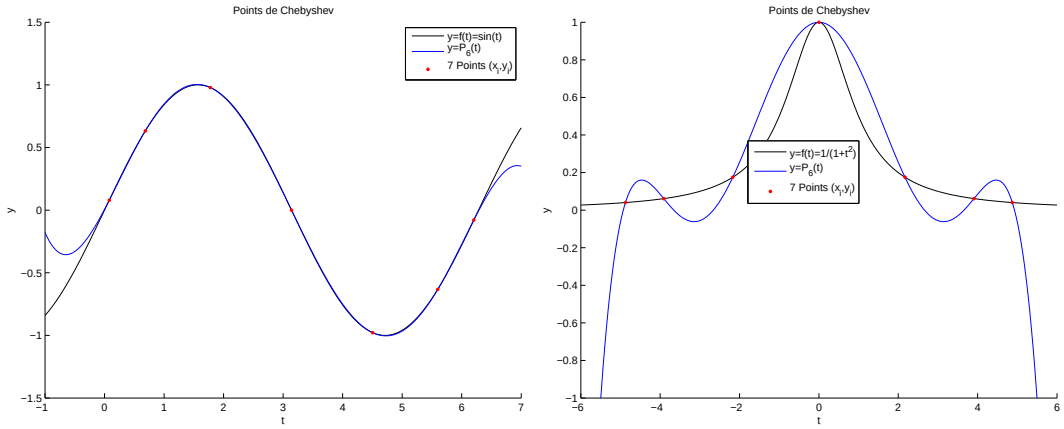

FIGURE 2.5: Erreurs d'interpolation avec $n = 6$

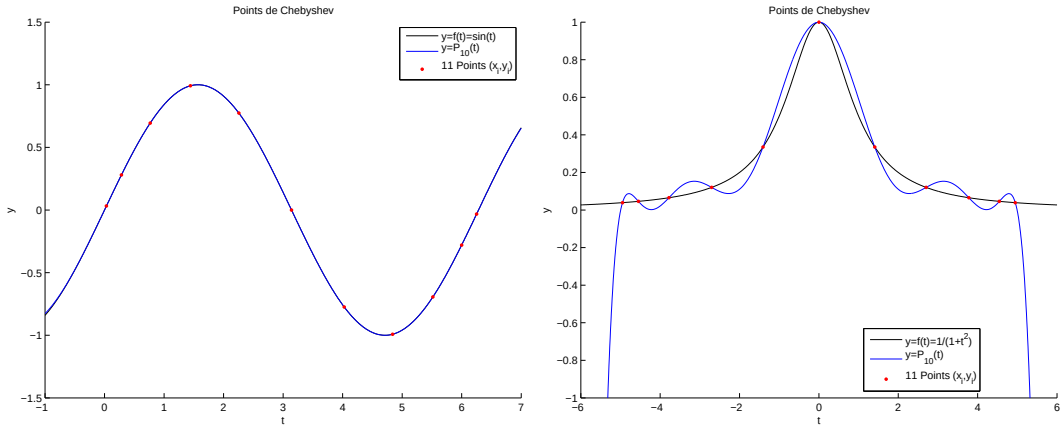

FIGURE 2.6: Erreurs d'interpolation avec $n = 10$

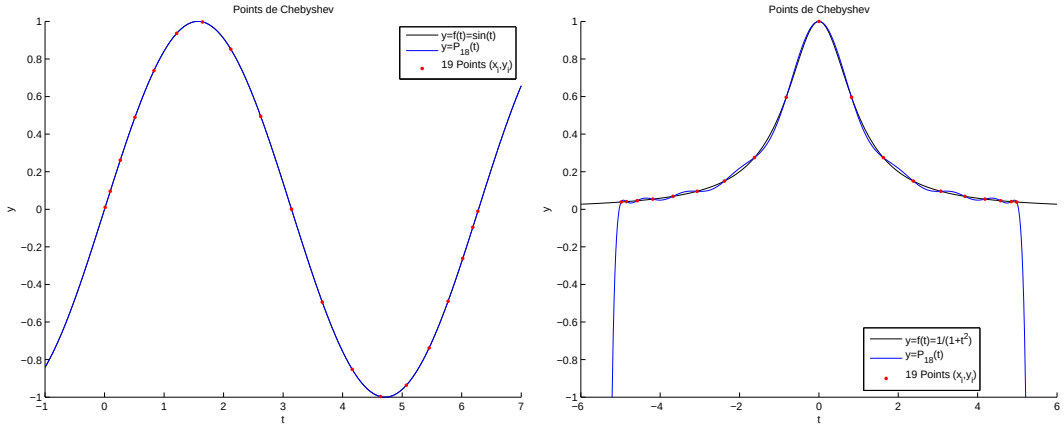

FIGURE 2.7: Erreurs d'interpolation avec $n = 18$

2.2 Dérivation numérique

Approximations de dérivées premières

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

Définition 8 *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*

$$f'(\bar{x}) = lim_{h\to 0}\frac{1}{h}(f(\bar{x}+h) - f(\bar{x}) \tag{2.8}$$

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

différence finie progressive :

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

différence finie rétrograde :

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

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

Théoreme 9 (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 (2.10) 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 9 *On note $x_i = a + ih$, $i \in [\![0,n]\!]$, une discrétisation régulière de l'intervalle $[a,b]$. Soit une fonction $f : [a,b] \longrightarrow \mathbb{R}$ suffisament régulière. On suppose que les y_i sont donnés par*

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

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

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

$$y_i = f(x_i), \quad \forall i \in [\![0, n]\!]. \tag{2.17}$$

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

Correction Pour les points x_i, $i \in [\![1, n-1]\!]$, 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 : $\exists \xi_1 \in]\bar{x}, \bar{x}+h[$, $\exists \xi_1 \in]\bar{x}, \bar{x}+2h[$,.

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

  ce qui donne

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

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

  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

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

- 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

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

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

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

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

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

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

$\Diamond$

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

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

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 connait pas de primitive de f.

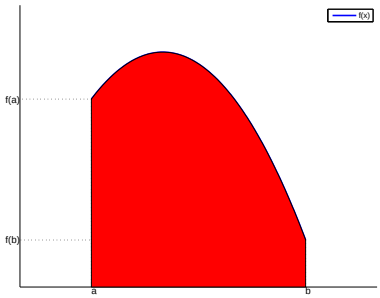

FIGURE 2.8: Représentation de $\int_a^b f(x)dx$ (aire en rouge)

Définition 10 *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.*

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

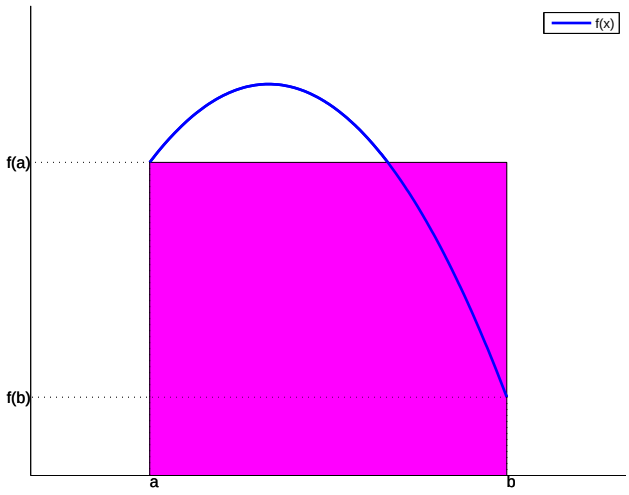

FIGURE 2.9: Formule du rectangle à gauche : $\int_a^b f(x)dx \approx \int_a^b f(a)dx$

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

et son degré d'exactitue 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

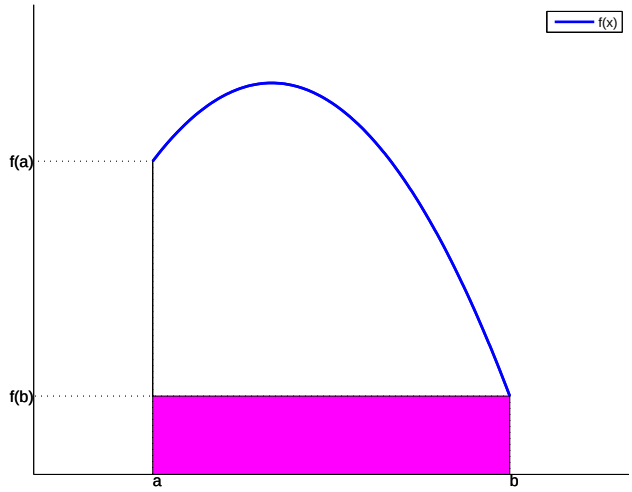

FIGURE 2.10: Formule du rectangle à droite : $\int_a^b f(x)dx \approx \int_a^b f(b)dx$

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

et son degré d'exactitue 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

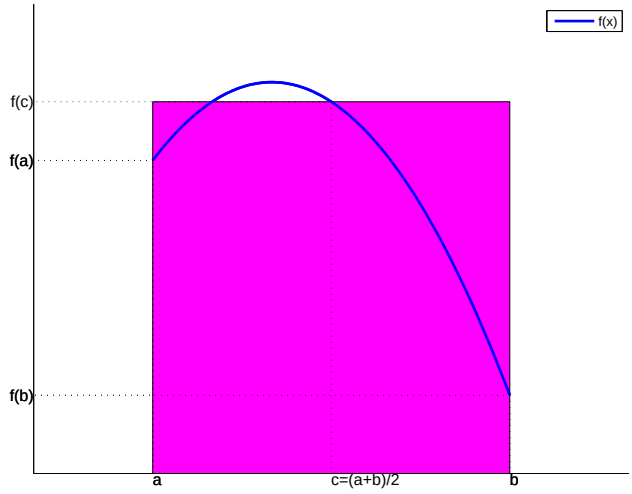

FIGURE 2.11: Formule du point milieu : $\int_a^b f(x)dx \approx \int_a^b f((a+b)/2)dx$

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

et son degré d'exactitue 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

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

et, en utilisant le théorème 6, on obtient

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

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

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

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

Pour un n donné, déterminer les coefficients α_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 une système linéaire à n inconnues (les α_i) en écrivant que pour chaque monôme

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

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

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

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

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

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

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

2.3.3 Méthodes composites

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

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

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

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

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

pour obtenir

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

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

Exercice 11 *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.*

2.3.4 Formule composite des trapèzes

On approche chacune des intégrales par la formule des trapèzes

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

pour obtenir

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

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

Exercice 12 *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.*

2.3.5 Formule composite de Simpson

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

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

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

Exercice 13 *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.*

Il est possible de "retrouver" numériquement l'ordre des différentes méthodes en représentant en échelle logarithmique (en abscisses et ordonnées) la fonction erreur $h \to erreur(h)$ 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_{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 $\log(h) \to \log(E_{\text{Simpson}}(h))$ qui est une droite de pente 4. En figure 2.12, on représente en échelle logarithmique les différentes erreurs ainsi que les fonctions $h \to h^2$ et $h \to h^4$.

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

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

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

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

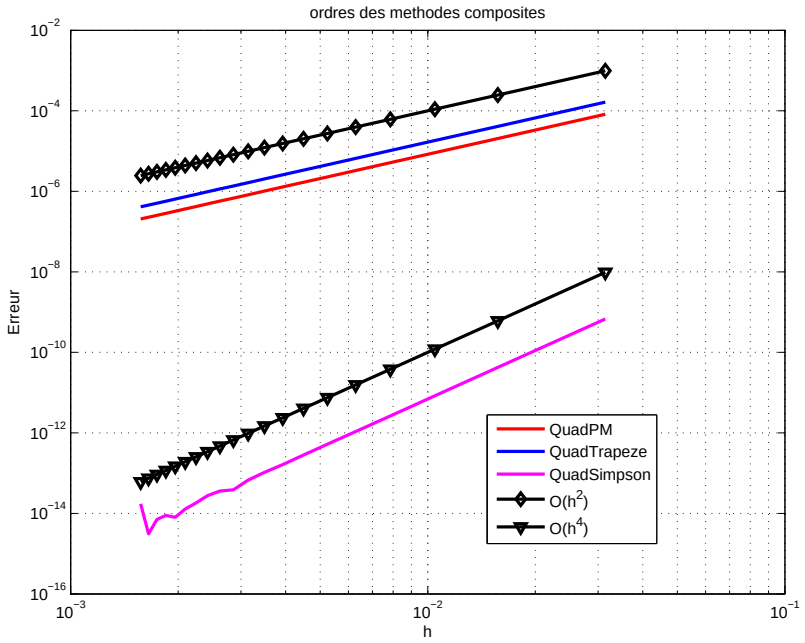

FIGURE 2.12: Ordre des méthodes composites pour le calcul de $\int_0^\pi \sin(x)dx$

Une nouvelle utilisation de Simpson en x donne

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

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

$$I \approx \frac{b-a}{6}\frac{d-c}{6}\begin{pmatrix} f(a,c) + 4f(a,\beta) + f(a,d) \\ +4(f(\alpha,c) + 4f(\alpha,\beta) + f(\alpha,d)) \\ +f(b,c) + 4f(b,\beta) + f(b,d) \end{pmatrix} \tag{2.35}$$

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

1. Discrétisation régulière de $[a,b]$: $\forall k \in [\![0,n]\!]$, $x_k = a + kh_x$ avec $h_x = \frac{b-a}{n}$.
2. Discrétisation régulière de $[c,d]$: $\forall l \in [\![0,m]\!]$, $y_l = a + lh_y$ avec $h_y = \frac{d-c}{m}$.
3. Relation de Chasles :
$$\int_a^b \int_c^d f(x,y)dydx = \sum_{k=1}^{n}\sum_{l=1}^{m}\int_{x_{k-1}}^{x_k}\int_{y_{l-1}}^{y_l} f(x,y)dydx.$$
4. Formule composite de Simpson "2D" :
$$\begin{gathered} \int_a^b \int_c^d f(x,y)dydx \\ = \\ \frac{h_x h_y}{36}\sum_{k=1}^{n}\sum_{l=1}^{m}\begin{pmatrix} f(x_{k-1},y_{l-1}) + 4f(x_{k-1},\beta_l) + f(x_{k-1},y_l) \\ +4(f(\alpha_k,y_{l-1}) + 4f(\alpha_k,\beta_l) + f(\alpha_k,y_l)) \\ +f(x_k,y_{l-1}) + 4f(x_k,\beta_l) + f(x_k,y_l) \end{pmatrix} \end{gathered}$$

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

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

$$z_i = \alpha x_i + y_i, \ \forall i \in [\![1, n]\!].$$

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

Algorithme 12- $\mathcal{R}_0$

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

Algorithme 12- $\mathcal{R}_1$

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

On abouti alors à

Algorithme 12 Fonction VecAXPY retournant $\boldsymbol{z} = \alpha\boldsymbol{x} + \boldsymbol{y}$ avec $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ et $\alpha \in \mathbb{R}$

Données : $\boldsymbol{x}, \boldsymbol{y}$: deux vecteurs de $\mathbb{R}^n$
α : un réel.
Résultat : $\boldsymbol{z}$: vecteur de $\mathbb{R}^n$.

1: **Fonction** $\boldsymbol{z} \leftarrow$ VecAXPY($\alpha, \boldsymbol{x}, \boldsymbol{y}$)
2: $\quad$ **Pour** $i \leftarrow 1$ **à** n **faire**
3: $\quad\quad z(i) \leftarrow \alpha * x(i) + y(i)$
4: $\quad$ **Fin Pour**
5: **Fin Fonction**

◊

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

Exercice 15 *Soient $\boldsymbol{x}$ et $\boldsymbol{y}$ deux vecteurs de $\mathbb{R}^n$.*

1. *Ecrire la fonction* VecDot *permettant de calculer le produit scalaire entre les vecteurs $\boldsymbol{x}$ et $\boldsymbol{y}$.*
2. *Ecrire la fonction* VecNorm1 *permettant de calculer*

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|, \ \forall \boldsymbol{x} \in \mathbb{R}^n. \tag{2.36}$$

3. Ecrire la fonction VecNorm2 *permettant de calculer*

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

Correction

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

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

Algorithme 13 Fonction VecDot permettant de calculer le produit scalaire des vecteurs $\boldsymbol{x}$ et $\boldsymbol{y}$ où $\boldsymbol{x} \in \mathbb{R}^n$ et $\boldsymbol{y} \in \mathbb{R}^n$.

Données :
$\boldsymbol{x}$: vecteur de $\mathbb{R}^n$,
$\boldsymbol{y}$: vecteur de $\mathbb{R}^n$.
Résultat : s : le réel tel que $s = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$.

```
1: Fonction s ← VecDot( x, y )
2:     s ← 0
3:     Pour i ← 1 à n faire
4:         s ← s + x(i) * y(i)
5:     Fin Pour
6: Fin Fonction
```

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

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

Algorithme 14 Fonction VecNorm1 permettant de retourner $\|\boldsymbol{x}\|_1$ avec $\boldsymbol{x} \in \mathbb{R}^n$

Données :
$\boldsymbol{x}$: vecteur de $\mathbb{R}^n$.
Résultat :
s : le réel tel que $s = \|\boldsymbol{x}\|_1$.

```
1: Fonction s ← VecNorm1( x )
2:     s ← 0
3:     Pour i = 1 à n faire
4:         s ← s+ABS(x(i))
5:     Fin Pour
6: Fin Fonction
```

3. On a

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

Algorithme 15 Fonction VecNorm2 permettant de retourner $\|\boldsymbol{x}\|_2$ avec $\boldsymbol{x} \in \mathbb{R}^n$

Données :
$\boldsymbol{x}$: vecteur de $\mathbb{R}^n$.
Résultat :
s : le réel tel que $s = \|\boldsymbol{x}\|_2$.

1: **Fonction** $s \leftarrow$ VecNorm2($\boldsymbol{x}$)
2: $s \leftarrow 0$
3: **Pour** $i \leftarrow 1$ **à** n **faire**
4: $s \leftarrow s + x(i) * x(i)$
5: **Fin Pour**
6: $s \leftarrow$sqrt(s)
7: **Fin Fonction**

ou encore en utilisant la fonction VecDot :

Algorithme 16 Fonction VecNorm2 permettant de retourner $\|\boldsymbol{x}\|_2$ avec $\boldsymbol{x} \in \mathbb{R}^n$ (utilise la fonction VecDot).

Données :
$\boldsymbol{x}$: vecteur de $\mathbb{R}^n$.
Résultat :
s : le réel tel que $s = \|\boldsymbol{x}\|_2$.

1: **Fonction** $s \leftarrow$ VecNorm2($\boldsymbol{x}$)
2: $s \leftarrow$sqrt(VecDot$(\boldsymbol{x}, \boldsymbol{x})$)
3: **Fin Fonction**

◊

2.5.1 Matrices

Exercice 16 *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* $\mathbb{Z} = \alpha\mathbb{X} + \mathbb{Y}$.

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

$$\forall i \in [\![1, m]\!],\ \forall i \in [\![1, n]\!],\quad Z_{i,j} = \alpha X_{i,j} + Y_{i,j}.$$

Algorithme 17 Fonction MATAXPY permettant de retourner $\mathbb{Z} = \alpha\mathbb{X} + \mathbb{Y}$ avec $\mathbb{X}$ et $\mathbb{Y}$ dans $\mathcal{M}_{m,n}(\mathbb{R})$, et $\alpha \in \mathbb{R}$

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

```
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
```

◊

Exercice 17 *Ecrire la fonction* MATTRANSPOSE *permettant de retourner la transposé d'une matrice.*

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

$$\forall i \in [\![1,n]\!],\ \forall i \in [\![1,m]\!],\quad Z_{i,j} = X_{j,i}.$$

Algorithme 18 Fonction MATTRANSPOSE permettant de retourner $\mathbb{Z} = \mathbb{X}^t$ avec $\mathbb{X} \in \mathcal{M}_{m,n}(\mathbb{R})$

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

```
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
```

◊

Exercice 18 *Ecrire la fonction* MATMULT *permettant de retourner le produit d'une matrice par un vecteur.*

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

$$y_i = \sum_{j=1}^{n} A_{i,j}x_j,\ \forall i \in [\![1,m]\!],$$

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 19- $\mathcal{R}_0$

1: Calcul de $\boldsymbol{y} \leftarrow \mathbb{A}\boldsymbol{x}$.

Algorithme 19- $\mathcal{R}_1$

1: **Pour** $i \leftarrow 1$ **à** m **faire**
2: Calcul de $y_i \leftarrow \sum_{j=1}^{n} A_{i,j}x_j$
3: **Fin Pour**

Algorithme 19- $\mathcal{R}_1$

1: **Pour** $i \leftarrow 1$ **à** m **faire**
2: Calcul de $y_i \leftarrow \sum_{j=1}^{n} A_{i,j}x_j$
3: **Fin Pour**

Algorithme 19- $\mathcal{R}_2$

1: **Pour** $i \leftarrow 1$ **à** m **faire**
2: $S \leftarrow 0$
3: **Pour** $j \leftarrow 1$ **à** n **faire**
4: $S \leftarrow S + A(i,j) * x(j)$
5: **Fin Pour**
6: $y(i) \leftarrow S$
7: **Fin Pour**

On obtient alors

Algorithme 19 Fonction MatMult permettant de retourner le vecteur $\boldsymbol{y} \in \mathbb{R}^m$ tel que :

$$\boldsymbol{y} = \mathbb{A}\boldsymbol{x}$$

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

Données :
$\mathbb{A}$: matrice de $\mathcal{M}_{m,n}(\mathbb{R})$,
$\boldsymbol{x}$: vecteur de $\mathbb{R}^n$.
Résultat :
$\boldsymbol{y}$: vecteur de $\mathbb{R}^m$ tel que $\boldsymbol{y} = \mathbb{A}\boldsymbol{x}$.

```
1: Fonction x ← MatMult( A, x )
2:     Pour i ← 1 à m faire
3:         S ← 0
4:         Pour j ← 1 à n faire
5:             S ← S + A(i, j) * x(j)
6:         Fin Pour
7:         y(i) ← S
8:     Fin Pour
9: Fin Fonction
```

◊

Exercice 19 *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

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

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 20- $\mathcal{R}_0$

1: Calcul de $\mathbb{Z} = \mathbb{X}\mathbb{Y}$.

Algorithme 20- $\mathcal{R}_1$

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

Algorithme 20- $\mathcal{R}_1$

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

Algorithme 20- $\mathcal{R}_2$

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 20- $\mathcal{R}_2$

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 20- $\mathcal{R}_3$

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 20 Fonction MatMatMult permettant de retourner la matrice $\mathbb{Z} \in \mathcal{M}_{m,p}(\mathbb{R})$ tel que

$$\mathbb{Z} = \mathbb{X}\mathbb{Y}$$

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

Données :
$\mathbb{X}$: matrice de $\mathcal{M}_{m,n}(\mathbb{R})$,
$\mathbb{Y}$: matrice de $\mathcal{M}_{n,p}(\mathbb{R})$.
Résultat :
$\mathbb{Z}$: matrice de $\mathcal{M}_{m,p}(\mathbb{R})$ telle que $\mathbb{Z} = \mathbb{X}\mathbb{Y}$.

```
1: Fonction Z ← MatMatMult( X, Y )
2:     Pour i ← 1 à m faire
3:         Pour j ← 1 à p faire
4:             S ← 0
5:             Pour k ← 1 à n faire
6:                 S ← S + X(i,k) * Y(k,j)
7:             Fin Pour
8:             Z(i,j) ← S
9:         Fin Pour
10:    Fin Pour
11: Fin Fonction
```

◊

Exercice 20 *Ecrire la fonction* MatNorm1 *permettant de retourner la norme d'une matrice* $\mathbb{A} \in \mathcal{M}_{m,n}(\mathbb{R})$

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

Correction Soit $\mathbb{A} \in \mathcal{M}_{m,n}(\mathbb{R})$. 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.

Algorithme 21- $\mathcal{R}_0$

```
1: Calcul de x = ||A||_1 .
```

Algorithme 21- $\mathcal{R}_1$

```
1: x ← 0
2: Pour j ← 1 à n faire
3:     x ← max( x, Σ_{i=1}^{m} |A_{i,j}| )
4: Fin Pour
```

Algorithme 21- $\mathcal{R}_1$

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

Algorithme 21- $\mathcal{R}_2$

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

Algorithme 21- $\mathcal{R}_2$

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

Algorithme 21- $\mathcal{R}_3$

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

On obtient alors

Algorithme 21 Fonction MatNorm1 permettant de retourner la norme 1 de la matrice $\mathbb{A} \in \mathcal{M}_{m,n}(\mathbb{R})$ donnée par :

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

Données :
$\mathbb{A}$: matrice de $\mathcal{M}_{m,n}(\mathbb{R})$.
Résultat :
x : un réel tel que $x = \|\mathbb{A}\|_1$.

1: **Fonction** $x \leftarrow$ MatNorm1($\mathbb{A}$)
2: $\quad M \leftarrow 0$
3: $\quad$ **Pour** $j \leftarrow 1$ **à** n **faire**
4: $\quad\quad S \leftarrow 0$
5: $\quad\quad$ **Pour** $i \leftarrow 1$ **à** m **faire**
6: $\quad\quad\quad S \leftarrow S + \text{abs}(A(i,j))$
7: $\quad\quad$ **Fin Pour**
8: $\quad\quad$ **Si** $M < S$ **alors**
9: $\quad\quad\quad M \leftarrow S$
10: $\quad\quad$ **Fin Si**
11: $\quad$ **Fin Pour**
12: $\quad x \leftarrow M$
13: **Fin Fonction**

$\Diamond$

2.5.2 Résolution d'un système linéaire par factorisation LU

Dans cet exemple, on s'intéresse à la résolution d'un système linéaire en utilisant la factorisation $\mathbb{LU}$.

On rappelle le théorème suivant

Théoreme 11 *Soit $\mathbb{A} \in \mathcal{M}_n(\mathbb{R})$ une matrice dont les sous-matrices principales sont inversibles alors il existe une unique matrice $\mathbb{L} \in \mathcal{M}_n(\mathbb{R})$ triangulaire inférieure à diagonale unité et une unique matrice $\mathbb{U} \in \mathcal{M}_n(\mathbb{R})$ triangulaire supérieure telles ques*

$$\mathbb{A} = \mathbb{LU}.$$

Principe de résolution

Soit $\mathbb{A} \in \mathcal{M}_n(\mathbb{R})$ une matrice dont les sous-matrices principales sont inversibles et $\boldsymbol{b} \in \mathbb{R}^n$. On veut résoudre le système linéaire $\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$. Pour celà, grâce au théorème 11, on obtient :

Trouver $\boldsymbol{x} \in \mathbb{R}^n$ tel que

$$\mathbb{A}\boldsymbol{x} = \boldsymbol{b}. \tag{2.37}$$

est équivalent à

Trouver $\boldsymbol{y} \in \mathbb{R}^n$ solution de

$$\mathbb{L}\boldsymbol{y} = \boldsymbol{b}. \tag{2.38}$$

puis $\boldsymbol{x} \in \mathbb{R}^n$ solution de

$$\mathbb{U}\boldsymbol{x} = \boldsymbol{y} \tag{2.39}$$

On est donc ramené à

Algorithme 22 Algorithme de base permettant de résoudre, par une factorisation $\mathbb{LU}$, le système linéaire

$$\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$$

où $\mathbb{A}$ une matrice de $\mathcal{M}_n(\mathbb{R})$ dont les sous-matrices principales sont inversibles et $\boldsymbol{b} \in \mathbb{R}^n$.

Données : $\mathbb{A}$: matrice de $\mathcal{M}_n(\mathbb{R})$ (sous-matrices principales inversibles),
$\boldsymbol{b}$: vecteur de $\mathbb{R}^n$.
Résultat : $\boldsymbol{x}$: vecteur de $\mathbb{R}^n$.

1: Trouver la factorisation $\mathbb{LU}$ de la matrice $\mathbb{A}$,
2: résoudre le système triangulaire inférieur $\mathbb{L}\boldsymbol{y} = \boldsymbol{b}$,
3: résoudre le système triangulaire supérieur $\mathbb{U}\boldsymbol{x} = \boldsymbol{y}$.

Ceci permet donc de découper le problème initial en trois sous-problèmes plus simples. De plus, ceux-ci peuvent se traiter de manière indépendante.

Algorithme 23 Fonction RSLFactLU permettant de résoudre, par une factorisation $\mathbb{LU}$, le système linéaire

$$\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$$

où $\mathbb{A}$ une matrice de $\mathcal{M}_n(\mathbb{R})$ définie positive et $\boldsymbol{b} \in \mathbb{R}^n$.

Données : $\mathbb{A}$: matrice de $\mathcal{M}_n(\mathbb{R})$ dont les sous-matrices principales sont inversibles définie positive,
$\boldsymbol{b}$: vecteur de $\mathbb{R}^n$.
Résultat : $\boldsymbol{x}$: vecteur de $\mathbb{R}^n$.

1: **Fonction** $\boldsymbol{x} \leftarrow$ RSLFactLU($\mathbb{A}, \boldsymbol{b}$)
2: $[\mathbb{L}, \mathbb{U}] \leftarrow$FactLU($\mathbb{A}$) ▷ Factorisation $\mathbb{LU}$
3: $\boldsymbol{y} \leftarrow$ResTriInf($\mathbb{L}, \boldsymbol{b}$) ▷ Résolution du système $\mathbb{L}\boldsymbol{y} = \boldsymbol{b}$
4: $\boldsymbol{x} \leftarrow$ResTriSup($\mathbb{U}, \boldsymbol{y}$) ▷ Résolution du système $\mathbb{U}\boldsymbol{x} = \boldsymbol{y}$
5: **Fin Fonction**

Il nous faut donc écrire les trois fonctions ResTriInf, ResTriSup et Cholesky (la fonction MatTranspose ayant déjà été écrite).

Résolution d'un système linéaire triangulaire inférieur

Soit $\mathbb{A}$ une matrice d'ordre n triangulaire inférieure inversible et $\boldsymbol{b} \in \mathbb{R}^n$. On veut résoudre le système linéaire $\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$.

Comme $\mathbb{A}$ est une matrice triangulaire inférieure, on a

$$A_{i,j} = 0 \quad \text{si} \quad j > i. \tag{2.40}$$

$$\begin{pmatrix} A_{1,1} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ A_{n,1} & \dots & \dots & A_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{pmatrix} \tag{2.41}$$

On remarque que l'on peut calculer successivement $x_1, x_2, \dots, x_n$, car il est possible de calculer x_i si on connait $x_1, \dots, x_{i-1}$. En effet, on a

$$(\mathbb{A}\boldsymbol{x})_i = b_i, \ \forall i \in [\![1, n]\!]$$

et donc, par définition d'un produit matrice-vecteur,

$$\sum_{j=1}^{n} A_{i,j} x_j = b_i, \ \forall i \in [\![1, n]\!].$$

En utilisant (2.40), on obtient

$$\sum_{j=1}^{i} A_{i,j} x_j = b_i, \ \forall i \in [\![1, n]\!].$$

On obtient donc x_i en fonction des $x_1, \dots, x_{i-1}$:

$$x_i = \frac{1}{A_{i,i}} \left(b_i - \sum_{j=1}^{i-1} A_{i,j} x_j \right), \ \forall i \in [\![1, n]\!]. \tag{2.42}$$

Remarque 2.6 *Comme la matrice $\mathbb{A}$ est inversible, on a $A_{i,i} \neq 0$, $\forall i \in [\![1, n]\!]$.*

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 24- $\mathcal{R}_0$

1: Résoudre $\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$ en calculant successivement $x_1,\ x_2,\ \ldots, x_n$.

Algorithme 24- $\mathcal{R}_1$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: calculer x_i connaissant $x_1, \ldots, x_{i-1}$ à l'aide de l'équation (2.42)
3: **Fin Pour**

Algorithme 24- $\mathcal{R}_1$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: Calculer x_i connaissant $x_1, \ldots, x_{i-1}$ à l'aide de l'équation (2.42)
3: **Fin Pour**

Algorithme 24- $\mathcal{R}_2$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: $S \leftarrow \sum_{j=1}^{i-1} A_{i,j} x_j$
3: $x_i \leftarrow (b_i - S)/A_{i,i}$
4: **Fin Pour**

Algorithme 24- $\mathcal{R}_2$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: $S \leftarrow \sum_{j=1}^{i-1} A_{i,j} x_j$
3: $x_i \leftarrow (b_i - S)/A_{i,i}$
4: **Fin Pour**

Algorithme 24- $\mathcal{R}_3$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: $S \leftarrow 0$
3: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
4: $S \leftarrow S + A(i,j) * x(j)$
5: **Fin Pour**
6: $x_i \leftarrow (b_i - S)/A_{i,i}$
7: **Fin Pour**

On obtient alors l'algorithme final

Algorithme 24 Fonction ResTriInf permettant de résoudre le système linéaire triangulaire inférieur inversible

$$\mathbb{A}\boldsymbol{x} = \boldsymbol{b}.$$

Données : $\mathbb{A}$: matrice triangulaire de $\mathcal{M}_n(\mathbb{R})$ inférieure inversible.
$\boldsymbol{b}$: vecteur de $\mathbb{R}^n$.
Résultat : $\boldsymbol{x}$: vecteur de $\mathbb{R}^n$.

```
1: Fonction x ← ResTriInf( A, b )
2:     Pour i ← 1 à n faire
3:         S ← 0
4:         Pour j ← 1 à i − 1 faire
5:             S ← S + A(i, j) ∗ x(j)
6:         Fin Pour
7:         x(i) ← (b(i) − S)/A(i, i)
8:     Fin Pour
9:     return x
10: Fin Fonction
```

Résolution d'un système linéaire triangulaire supérieur

Soit $\mathbb{A}$ une matrice de $\mathcal{M}_n(\mathbb{R})$ triangulaire supérieure inversible et $\boldsymbol{b} \in \mathbb{R}^n$. On veut résoudre le système linéaire $\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$.

Comme $\mathbb{A}$ est une matrice triangulaire inférieure, on a

$$A_{i,j} = 0 \quad \text{si} \quad j < i. \tag{2.43}$$

$$\begin{pmatrix} A_{1,1} & \dots & \dots & A_{1,n} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{pmatrix} \tag{2.44}$$

On remarque que l'on peut calculer successivement $x_n, x_{n-1}, \dots, x_1$, car il est possible de calculer x_i si on connait $x_{i+1}, \dots, x_n$. En effet, on a

$$(\mathbb{A}\boldsymbol{x})_i = b_i, \ \forall i \in [\![1, n]\!].$$

et donc, par définition d'un produit matrice-vecteur,

$$\sum_{j=1}^{n} A_{i,j} x_j = b_i, \ \forall i \in [\![1, n]\!].$$

En utilisant 2.43, on obtient

$$\sum_{j=i}^{n} A_{i,j} x_j = b_i, \ \forall i \in [\![1, n]\!].$$

On obtient donc x_i en fonction des $x_{i+1}, \dots, x_n$:

$$x_i = \frac{1}{A_{i,i}} \left(b_i - \sum_{j=i+1}^{n} A_{i,j} x_j \right), \ \forall i \in [\![1, n]\!]. \tag{2.45}$$

Remarque 2.7 *Comme la matrice $\mathbb{A}$ est inversible, on a $A_{i,i} \neq 0$, $\forall i \in [\![1, n]\!]$.*

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 25- $\mathcal{R}_0$

1: Résoudre $\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$ en calculant successivement $x_n, \ x_{n-1}, \ \dots, x_1$.

Algorithme 25- $\mathcal{R}_1$

1: **Pour** $i \leftarrow n$ **à** 1 **faire** (**pas de** -1)
2: calculer x_i connaissant $x_{i+1}, \dots, x_n$ à l'aide de l'équation (2.45)
3: **Fin Pour**

Algorithme 25- $\mathcal{R}_1$

1: **Pour** $i \leftarrow n$ **à** 1 **faire** (**pas de** -1)
2: Calculer x_i connaissant $x_{i+1}, \dots, x_n$ à l'aide de l'équation (2.45)
3: **Fin Pour**

Algorithme 25- $\mathcal{R}_2$

1: **Pour** $i \leftarrow n$ **à** 1 **faire** (**pas de** -1)
2: $S \leftarrow \sum_{j=i+1}^{n} A_{i,j} x_j$
3: $x_i \leftarrow (b_i - S)/A_{i,i}$
4: **Fin Pour**

Algorithme 25- $\mathcal{R}_2$

1: **Pour** $i \leftarrow n$ **à** 1 **faire** (**pas de** -1)
2: $S \leftarrow \sum_{j=i+1}^{n} A_{i,j}x_j$
3: $x_i \leftarrow (b_i - S)/A_{i,i}$
4: **Fin Pour**

Algorithme 25- $\mathcal{R}_3$

1: **Pour** $i \leftarrow n$ **à** 1 **faire** (**pas de** -1)
2: $S \leftarrow 0$
3: **Pour** $j \leftarrow i+1$ **à** n **faire**
4: $S \leftarrow S + A(i,j) * x(j)$
5: **Fin Pour**
6: $x_i \leftarrow (b_i - S)/A_{i,i}$
7: **Fin Pour**

On obtient alors l'algorithe final

Algorithme 25 Fonction ResTriSup permettant de résoudre le système linéaire triangulaire supérieur inversible

$$\mathbb{A}\boldsymbol{x} = \boldsymbol{b}.$$

Données : $\mathbb{A}$: matrice triangulaire $\mathcal{M}_n(\mathbb{R})$ supérieur inversible,
$\boldsymbol{b}$: vecteur de $\mathbb{R}^n$.
Résultat : $\boldsymbol{x}$: vecteur de $\mathbb{R}^n$.

1: **Fonction** $\boldsymbol{x} \leftarrow$ ResTriSup($\mathbb{A}, \boldsymbol{b}$)
2: **Pour** $i \leftarrow n$ **à** 1 **faire** (**pas de** -1)
3: $S \leftarrow 0$
4: **Pour** $j \leftarrow i+1$ **à** n **faire**
5: $S \leftarrow S + A(i,j) * x(j)$
6: **Fin Pour**
7: $x(i) \leftarrow (b(i) - S)/A(i,i)$
8: **Fin Pour**
9: **Fin Fonction**

Factorisation $\mathbb{LU}$

Soit $\mathbb{A} \in \mathcal{M}_n(\mathbb{R})$ une matrice dont les sous-matrices principales sont inversibles. D'après le théorème 11, il existe une unique matrice $\mathbb{L} \in \mathcal{M}_n(\mathbb{R})$ triangulaire inférieure avec $L_{i,i} = 1$, $\forall i \in [\![1,n]\!]$, et une unique matrice $\mathbb{L} \in \mathcal{M}_n(\mathbb{R})$ triangulaire supérieure telles que

$$\mathbb{A} = \mathbb{LU} \tag{2.46}$$

c'est à dire

$$\begin{pmatrix} A_{1,1} & \dots & A_{1,n} \\ \vdots & \ddots & \dots \\ A_{n,1} & \dots & A_{n,n} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ L_{2,1} & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ L_{n,1} & \dots & L_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} U_{1,1} & \dots & \dots & U_{n,1} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & U_{n,n} \end{pmatrix}. \tag{2.47}$$

Pour déterminer les matrices $\mathbb{L}$ et $\mathbb{U}$, on remarque que la 1ère ligne de $\mathbb{L}$ est déjà déterminée. On peut alors l'utiliser pour calculer la première ligne de $\mathbb{U}$: $\forall j \in [\![1,n]\!]$

$$\begin{aligned} A_{1,j} &= \sum_{k=1}^{n} L_{1,k}U_{k,j} \\ &= L_{1,1}U_{1,j} \text{ car } \mathbb{L} \text{ triangulaire inférieure} \\ &= U_{1,j} \end{aligned}$$

On a donc

$$U_{1,j} = A_{1,j}, \quad \forall j \in [\![1, n]\!]. \tag{2.48}$$

La première colonne de $\mathbb{U}$ est aussi déterminée, on peut alors l'utiliser pour calculer la première colonne de $\mathbb{L}$: $\forall i \in [\![2, n]\!]$

$$\begin{aligned} A_{i,1} &= \sum_{k=1}^{n} L_{i,k} U_{1,j} \\ &= L_{i,1} U_{1,1} \text{ car } \mathbb{U} \text{ triangulaire supérieure} \end{aligned}$$

On peut démontrer, de part les hypothèses sur la matrice $\mathbb{A}$, que $U_{1,1} \neq 0$ et alors

$$L_{1,j} = A_{i,j}/U_{1,1}, \quad \forall i \in [\![2, n]\!]. \tag{2.49}$$

La première ligne de $\mathbb{U}$ et la première colonne de $\mathbb{L}$ sont donc déterminées par les formules (2.48) et (2.49).

Par récurrence, on suppose connues les $i-1$ premières lignes de $\mathbb{U}$ et les $i-1$ premières colonnes de $\mathbb{L}$. On va montrer que l'on peux expliciter la i-ème ligne de $\mathbb{U}$ et la i-ème colonne de $\mathbb{L}$.
En effet, $\forall j \in [\![i, n]\!]$, on a

$$\begin{aligned} A_{i,j} &= \sum_{k=1}^{n} L_{i,k} U_{k,j} \\ &= \sum_{k=1}^{i-1} L_{i,k} U_{k,j} + L_{i,i} U_{i,j} + \sum_{k=i+1}^{n} L_{i,k} U_{k,j} \\ &= \sum_{k=1}^{i-1} L_{i,k} U_{k,j} + L_{i,i} U_{i,j} \text{ car } \mathbb{L} \text{ triangulaire inférieure} \end{aligned}$$

Dans l'expression $\sum_{k=1}^{i-1} L_{i,k} U_{k,j}$ tous les termes sont connus (hypothèse de récurrence) et $L_{i,i} = 1$. On en déduit,

$$\text{Pour } i \text{ allant de 1 à } n : \; U_{i,j} = \begin{cases} A_{i,j} - \sum_{k=1}^{i-1} L_{i,k} U_{k,j}, & \forall j \in [\![i, n]\!]. \\ 0, & \forall j \in [\![1, i-1]\!]. \end{cases} \tag{2.50}$$

Maintenant, on calcule, $\forall j \in [\![i+1, n]\!]$,

$$\begin{aligned} A_{j,i} &= \sum_{k=1}^{n} L_{j,k} U_{k,i} \\ &= \sum_{k=1}^{i-1} L_{j,k} U_{k,i} + L_{j,i} U_{i,i} + \sum_{k=i+1}^{n} L_{j,k} U_{k,i} \\ &= \sum_{k=1}^{i-1} L_{j,k} U_{k,i} + L_{j,i} U_{i,i} \text{ car } \mathbb{U} \text{ triangulaire supérieure} \end{aligned}$$

Dans l'expression $\sum_{k=1}^{i-1} L_{j,k} U_{k,i}$ tous les termes sont connus (hypothèse de récurrence). De plus $U_{i,i}$ est donné par (2.50) et on peut démontrer, de part les hypothèses sur la matrice $\mathbb{A}$, que $U_{i,i} \neq 0$. On a alors

$$\text{Pour } i \text{ allant de 1 à } n : \; L_{j,i} = \begin{cases} 0, & \forall j \in [\![1, i-1]\!]. \\ 1, & j = i \\ \frac{1}{U_{i,i}} \left(A_{j,i} - \sum_{k=1}^{i-1} L_{j,k} U_{k,i} \right), & \forall j \in [\![i+1, n]\!], \end{cases} \tag{2.51}$$

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 26- $\mathcal{R}_0$

1: Calculer les matrices $\mathbb{L}$ et $\mathbb{U}$

Algorithme 26- $\mathcal{R}_1$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: Calculer la ligne i de $\mathbb{U}$.
3: Calculer la colonne i de $\mathbb{L}$.
4: **Fin Pour**

Algorithme 26- $\mathcal{R}_1$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: Calculer la ligne i de $\mathbb{U}$.
3: Calculer la colonne i de $\mathbb{L}$.
4: **Fin Pour**

Algorithme 26- $\mathcal{R}_2$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
3: $U(i,j) \leftarrow 0$
4: **Fin Pour**
5: **Pour** $j \leftarrow i$ **à** n **faire**
6: $U_{i,j} \leftarrow A_{i,j} - \sum_{k=1}^{i-1} L_{i,k}U_{k,j}$
7: **Fin Pour**
8: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
9: $L_{j,i} \leftarrow 0$
10: **Fin Pour**
11: $L_{i,i} \leftarrow 1$
12: **Pour** $j \leftarrow i+1$ **à** n **faire**
13: $L_{j,i} \leftarrow \frac{1}{U_{i,i}} \left(A_{j,i} - \sum_{k=1}^{i-1} L_{j,k}U_{k,i} \right)$
14: **Fin Pour**
15: **Fin Pour**

Algorithme 26- $\mathcal{R}_2$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
3: $U(i,j) \leftarrow 0$
4: **Fin Pour**
5: **Pour** $j \leftarrow i$ **à** n **faire**
6: $U_{i,j} \leftarrow A_{i,j} - \sum_{k=1}^{i-1} L_{i,k}U_{k,j}$
7: **Fin Pour**
8: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
9: $L_{j,i} \leftarrow 0$
10: **Fin Pour**
11: $L_{i,i} \leftarrow 1$
12: **Pour** $j \leftarrow i+1$ **à** n **faire**
13: $L_{j,i} \leftarrow \frac{1}{U_{i,i}} \left(A_{j,i} - \sum_{k=1}^{i-1} L_{j,k}U_{k,i} \right)$
14: **Fin Pour**
15: **Fin Pour**

Algorithme 26- $\mathcal{R}_3$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
3: $U(i,j) \leftarrow 0$
4: **Fin Pour**
5: **Pour** $j \leftarrow i$ **à** n **faire**
6: $S_1 \leftarrow \sum_{k=1}^{i-1} L_{i,k}U_{k,j}$
7: $U_{i,j} \leftarrow A_{i,j} - S_1$
8: **Fin Pour**
9: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
10: $L_{j,i} \leftarrow 0$
11: **Fin Pour**
12: $L_{i,i} \leftarrow 1$
13: **Pour** $j \leftarrow i+1$ **à** n **faire**
14: $S_2 \leftarrow \sum_{k=1}^{i-1} L_{j,k}U_{k,i}$
15: $L_{j,i} \leftarrow \frac{1}{U_{i,i}} \left(A_{j,i} - S_2 \right).$
16: **Fin Pour**
17: **Fin Pour**

Algorithme 26- $\mathcal{R}_3$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
3: $U(i,j) \leftarrow 0$
4: **Fin Pour**
5: **Pour** $j \leftarrow i$ **à** n **faire**
6: $S_1 \leftarrow \sum_{k=1}^{i-1} L_{i,k}U_{k,j}$
7: $U_{i,j} \leftarrow A_{i,j} - S_1$
8: **Fin Pour**
9: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
10: $L_{j,i} \leftarrow 0$
11: **Fin Pour**
12: $L_{i,i} \leftarrow 1$
13: **Pour** $j \leftarrow i+1$ **à** n **faire**
14: $S_2 \leftarrow \sum_{k=1}^{i-1} L_{j,k}U_{k,i}$
15: $L_{j,i} \leftarrow \frac{1}{U_{i,i}}\left(A_{j,i} - S_2\right).$
16: **Fin Pour**
17: **Fin Pour**

Algorithme 26- $\mathcal{R}_4$

1: **Pour** $i \leftarrow 1$ **à** n **faire**
2: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
3: $U(i,j) \leftarrow 0$
4: **Fin Pour**
5: **Pour** $j \leftarrow i$ **à** n **faire**
6: $S_1 \leftarrow 0$
7: **Pour** $k \leftarrow 1$ **à** $i-1$ **faire**
8: $S_1 \leftarrow S_1 + L_{i,k} * U_{k,j}$
9: **Fin Pour**
10: $U_{i,j} \leftarrow A_{i,j} - S_1$
11: **Fin Pour**
12: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
13: $L_{j,i} \leftarrow 0$
14: **Fin Pour**
15: $L_{i,i} \leftarrow 1$
16: **Pour** $j \leftarrow i+1$ **à** n **faire**
17: $S_2 \leftarrow 0$
18: **Pour** $k \leftarrow 1$ **à** $i-1$ **faire**
19: $S_2 \leftarrow S_2 + L_{j,k} * U_{k,i}$
20: **Fin Pour**
21: $L_{j,i} \leftarrow \frac{1}{U_{i,i}}\left(A_{j,i} - S_2\right).$
22: **Fin Pour**
23: **Fin Pour**

On obtient alors l'algorithme final

Algorithme 26 Fonction FACTLU permet de calculer les matrices $\mathbb{L}$ et $\mathbb{U}$ dites matrice de factorisation $\mathbb{LU}$ associée à la matrice $\mathbb{A}$, telle que

$$\mathbb{A} = \mathbb{LU}$$

Données : $\mathbb{A}$: matrice de $\mathcal{M}_n(\mathbb{R})$ dont les sous-matrices principales sont inversibles.

Résultat : $\mathbb{L}$: matrice de $\mathcal{M}_n(\mathbb{R})$ triangulaire inférieure avec $L_{i,i} = 1,\ \forall i \in [\![1, n]\!]$

$\mathbb{U}$: matrice de $\mathcal{M}_n(\mathbb{R})$ triangulaire supérieure.

```
1: Fonction [L, U] ← FactLU( A )
2:     Pour i ← 1 à n faire
3:         Pour j ← 1 à i − 1 faire
4:             U(i, j) ← 0
5:         Fin Pour
6:         Pour j ← i à n faire
7:             S1 ← 0
8:             Pour k ← 1 à i − 1 faire
9:                 S1 ← S1 + L(i, k) ∗ U(k, j)
10:            Fin Pour
11:            U(i, j) ← A(i, j) − S1
12:        Fin Pour
13:        Pour j ← 1 à i − 1 faire
14:            L(j, i) ← 0
15:        Fin Pour
16:        L(i, i) ← 1
17:        Pour j ← i + 1 à n faire
18:            S2 ← 0
19:            Pour k ← 1 à i − 1 faire
20:                S2 ← S2 + L(j, k) ∗ U(k, i)
21:            Fin Pour
22:            L(j, i) ← (A_{j,i} − S2)/U(i, i).
23:        Fin Pour
24:    Fin Pour
25: Fin Fonction
```

Remarque 2.8 *Cet algorithme peut être amélioré, pour gagner en lisibilité... En effet, il est possible d'initialiser la matrice $\mathbb{U}$ par la matrice nulle et la matrice $\mathbb{L}$ par la matrice identitée, ce qui permet alors de supprimer les boucles $U(i, j) \leftarrow 0$ et $L(j, i) \leftarrow 0$ ainsi que la commande $L(i, i) \leftarrow 1$.*

Pour optimiser en mémoire cette fonction, il possible de stocker les matrices $\mathbb{L}$ et $\mathbb{U}$ dans une même matrice de $\mathcal{M}_n(\mathbb{R})$ ...