# Initiation à l'Algorithmique

François Cuvelier

1er février 2007

# Table des matières

# 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 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 $P_3$.*

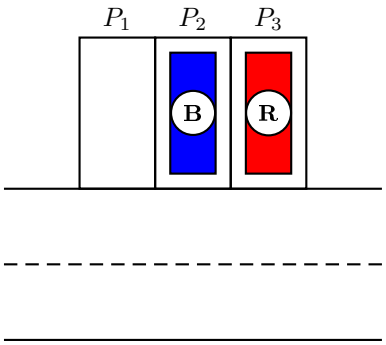


FIG. 1: Avant permutation

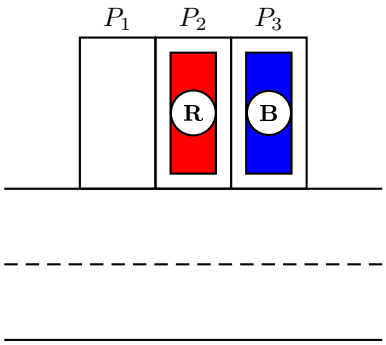


FIG. 2: Après permutation

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

## 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: $x \leftarrow b/a$

### 1.3 caractéristiques

Voici en résumé les caractéristiques d'un *bon* algorithme :
- Il ne souffre d'aucune ambiguité $\Rightarrow$ 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.

### 1.4 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*).

## 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.

### 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. $\pi$, la constante de gravitation,...)

## 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.*

## 2.3 Opérateurs

### 2.3.1 Opérateurs arithmétiques

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

### 2.3.2 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. |

### 2.3.3 Opérateurs logiques

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

### 2.3.4 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.

## 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)*

## 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.

### 2.5.1 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`.

### 2.5.2 Instructions composées

### 2.5.3 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**

#### 2.5.4 Instruction répétitive, boucle «tant que»

**Algorithme 3** Exemple de boucle «tant que»

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

#### 2.5.5 Instructions conditionnelles «si»

**Algorithme 4** Exemple d'instructions conditionnelle «si»

**Données :** *note* : un réel.

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

# 3 Méthodologie

## 3.1 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

## 3.2 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.

## 3.3 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.

**Exercice 1** *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 5** 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**

◊

**Exercice 2** *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 6** 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: $\quad P \leftarrow P * \sin(2 * k * z/n)\hat{}k$
4: **Fin Pour**

◊

**Exercice 3** *Soit la série de Fourier*

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

1. *Ecrire un algorithme permettant de calculer $x(t)$ tronquée au trois premiers termes, avec $\omega = 2\pi$ et $A = 1$.*
2. *Même question avec une troncature au n-ième terme.*

**Exercice 4** *Reprendre les trois exercices précédants en utilisant les boucles «tant que».*

# 4 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,
– ...

## 4.1 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)

## 4.2 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, ....

## 4.3 Exemples simples

### 4.3.1 Résolution d'une équation du premier degrès

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 7** Exemple de fonction : Résolution de l'équation du premier degrès $ax + b = 0$.

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

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

**Remarque 5** *Cette fonction est très simple, toutefois pour ne pas «alourdir» le code nous n'avons pas vérifié la validité des données fournies.*

**Exercice 5** *Ecrire un algorithme permettant de valider cette fonction.*

### 4.3.2 Résolution d'une équation du second degrès

Nous cherchons les solutions réelles de l'équation

$$ax^2 + bx + c = 0, \tag{1}$$

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 6**
1. *Ecrire la fonction* `discriminant` *permettant de calculer le discriminant de l'équation (1).*
2. *Ecrire la fonction* `RESD` *permettant de résoudre l'équation (1) en utilisant la fonction* `discriminant`.
3. *Ecrire un programme permettant de valider ces deux fonctions.*

**Exercice 7** *Même question que précédemment dans le cas complexe (solution et coefficients).*

### 4.3.3 Polynômes

Nous voulons écrire la fonction `polynome` permettant de calculer

$$y = \sum_{i=1}^{n} a_i x^i$$

### 4.3.4 Produit multiple

Nous voulons écrire la fonction `PM` permettant de calculer

$$y = \prod_{i=1}^{m} a_i \sin(x^i)$$

### 4.3.5 Série de Fourier

**Exercice 8** *Soit la série de Fourier*

$$x(t) = \frac{4A}{\pi}\left\{\cos\omega t - \frac{1}{3}\cos 3\omega t + \frac{1}{5}\cos 5\omega t - \frac{1}{7}\cos 7\omega t + \cdots\right\}. \qquad (2)$$

*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 8** En-tête de la fonction SFT retournant valeur de la série de Fourier en $t$ tronquée au $n$ premiers termes de l'exercice 8.

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

1: **Fonction** $x \leftarrow$ SFT( $t, n, A, \omega$ )
2: **Fin Fonction**

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

**Algorithme 9-** $\mathcal{R}_0$

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

**Algorithme 9-** $\mathcal{R}_1$

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

**Algorithme 9-** $\mathcal{R}_1$

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

2: $x_n(t) \leftarrow \frac{4A}{\pi} S$

**Algorithme 9-** $\mathcal{R}_2$

1: $S \leftarrow 0$

2: **Pour** $k = 1$ **à** $n$ **faire**

3: $\quad S \leftarrow S + (-1)^{k+1} \frac{1}{2k-1} * \cos((2k-1)\omega t)$

4: **Fin Pour**

5: $x_n(t) \leftarrow \frac{4A}{\pi} S$

Finalement la fonction est

**Algorithme 9** Fonction SFT retournant la valeur de la série de Fourier en $t$ tronquée au $n$ premiers termes de l'exercice 8.

**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:     return S ← 4*A*S/π
7: Fin Fonction
```

◊

# 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 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.

# 6 Exemples en 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.

## 6.1 Vecteurs

**Exercice 9** *Soient $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ et $\alpha \in \mathbb{R}$. Ecrire la fonction* VecAXPY *permettant de calculer $\alpha\boldsymbol{x} + \boldsymbol{y}$.*

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

**Algorithme 10-** $\mathcal{R}_0$

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

**Algorithme 10-** $\mathcal{R}_1$

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 10** 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 $\mathbb{R}^n$
$\alpha$ : 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 6**
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 10** *Soient $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.*

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

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|, \ \forall \boldsymbol{x} \in \mathbb{R}^n. \tag{3}$$

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, $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$,

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

**Algorithme 11** Fonction VECDOT permettant de calculer le produit scalaire des vecteurs $\boldsymbol{x}$ et $\boldsymbol{y}$ où $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

**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

**Algorithme 12** 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:     return s
7: Fin Fonction
```

3. On a

**Algorithme 13** 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 ← VecNorm2( x )
2:     s ← 0
3:     Pour i ← 1 à n faire
4:         s ← s + x(i) * x(i)
5:     Fin Pour
6:     s ←SQRT(s)
7: Fin Fonction
```

ou encore en utilisant la fonction VecDot :

**Algorithme 14** 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**

◊

## 6.2 Matrices

**Exercice 11** *Soient* $\mathbb{X}, \mathbb{Y} \in \mathcal{M}_{m,n}(\mathbb{R})$ *et* $\alpha \in \mathbb{R}$ *Ecrire la fonction* MatAXPY *permettant de retourner* $\alpha\mathbb{X} + \mathbb{Y}$.

**Correction**

**Algorithme 15** Fonction MatAXPY permettant de retourner $\mathbb{Z} = \alpha\mathbb{X} + \mathbb{Y}$ avec $\mathbb{X}, \mathbb{Y} \in \mathcal{M}_{m,n}(\mathbb{R})$ et $\alpha \in \mathbb{R}$

**Données :**
$\alpha$ : 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** $\mathbb{Z} \leftarrow$ MatAXPY( $\alpha, \mathbb{X}, \mathbb{Y}$ )
2: **Pour** $i \leftarrow 1$ **à** $m$ **faire**
3: **Pour** $j \leftarrow 1$ **à** $n$ **faire**
4: $Z(i,j) \leftarrow \alpha * X(i,j) + Y(i,j)$
5: **Fin Pour**
6: **Fin Pour**
7: **Fin Fonction**

◊

**Exercice 12** *Ecrire la fonction* MatTranspose *permettant de retourner la transposé d'une matrice.*

**Correction**

**Algorithme 16** Fonction MatTranspose permettant de retourner $\mathbb{Z} = \mathbb{X}^t$ avec $\mathbb{X}\mathcal{M}_{m,n}(\mathbb{R})$

**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 13** *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

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

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 17-** $\mathcal{R}_0$

```
1: Calcul de y ← Ax.
```

**Algorithme 17-** $\mathcal{R}_1$

```
1: Pour i ← 1 à m faire
2:     Calcul de y_i ← Σ_{j=1}^{n} A_{i,j} x_j
3: Fin Pour
```

**Algorithme 17-** $\mathcal{R}_1$

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

**Algorithme 17-** $\mathcal{R}_2$

On obtient alors

**Algorithme 17** 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 14** *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 $\forall i \in \{1, \ldots, m\}$, $\forall j \in \{1, \ldots, p\}$,

$$Z_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}.$$

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 18-** $\mathcal{R}_0$

```
1: Calcul de ℤ = 𝕏𝕐.
```

**Algorithme 18-** $\mathcal{R}_1$

```
1: Pour i ← 1 à m faire
2:     Calcul de la ligne i de la matrice ℤ
3: Fin Pour
```

**Algorithme 18-** $\mathcal{R}_1$

```
1: Pour i ← 1 à m faire
2:     Calcul de la ligne i de la matrice ℤ
3: Fin Pour
```

**Algorithme 18-** $\mathcal{R}_2$

```
1: Pour i ← 1 à m faire
2:     Pour j ← 1 à p faire
3:         Calcul de Z_{i,j} ← Σ_{k=1}^{n} A_{i,k} B_{k,j}
4:     Fin Pour
5: Fin Pour
```

**Algorithme 18-** $\mathcal{R}_2$

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

**Algorithme 18-** $\mathcal{R}_3$

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: $S \leftarrow S + A_{i,k}B_{k,j}$
6: **Fin Pour**
7: $Z_{i,j} \leftarrow S$
8: **Fin Pour**
9: **Fin Pour**

On obtient alors

**Algorithme 18** 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 + A(i,k) * B(k,j)
7:             Fin Pour
8:             Z(i,j) ← S
9:         Fin Pour
10:    Fin Pour
11: Fin Fonction
```

◊

**Exercice 15** *Ecrire la fonction* MatNorm1 *permettant de retourner la norme d'une matrice* $\mathbb{A} \in \mathcal{M}_{m,n}(\mathbb{R})$

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

**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 19-** $\mathcal{R}_0$

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

**Algorithme 19-** $\mathcal{R}_1$

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 19-** $\mathcal{R}_1$

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 19-** $\mathcal{R}_2$

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 19-** $\mathcal{R}_2$

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 19-** $\mathcal{R}_3$

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

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

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

```
1: Fonction x ← MatNorm1( A )
2:     M ← 0
3:     Pour j ← 1 à n faire
4:         S ← 0
5:         Pour i ← 1 à m faire
6:             S ← S+ABS(A(i, j))
7:         Fin Pour
8:         Si M < S alors
9:             M ← S
10:        Fin Si
11:    Fin Pour
12:    x ← M
13: Fin Fonction
```

◊

# 7 Exemple : résolution d'un système linéaire

Dans cet exemple, on s'intéresse à la résolution d'un système linéaire en utilisant la factorisation de Cholesky.

On rappelle le théorème suivant

**Théoreme 7** *Soit $\mathbb{A}$ une matrice symétrique définie positive alors il existe une matrice $\mathbb{L}$ triangulaire inférieure telle que*

$$\mathbb{A} = \mathbb{L}\mathbb{L}^t.$$

*Si l'on impose aux éléments diagonaux de $\mathbb{L}$ d'être strictement positifs, alors la factorisation est unique. On dit alors que $\mathbb{L}$ est la matrice de factorisation positive de Cholesky associée à la matrice $\mathbb{A}$.*

## 7.1 Principe de résolution

Soit $\mathbb{A}$ une matrice symétrique définie positive d'ordre $n$ et $\boldsymbol{b} \in \mathbb{R}^n$. On veut résoudre le système linéaire $\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$. Pour celà, grâce au théorème 7, on obtient :

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

$$\mathbb{A}\boldsymbol{x} = \boldsymbol{b}. \tag{4}$$

est équivalent à

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

$$\mathbb{L}^t \boldsymbol{x} = \boldsymbol{y} \tag{5}$$

avec $\mathbb{L}$ la factorisation positive de Cholesky de la matrice $\mathbb{A}$ et $\boldsymbol{y} \in \mathbb{R}^n$ solution de

$$\mathbb{L}\boldsymbol{y} = \boldsymbol{b}. \tag{6}$$

On est donc ramené à

**Algorithme 20** Algorithme de base permettant de résoudre, par une factorisation de Cholesky positive, le système linéaire

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

où $\mathbb{A}$ une matrice symétrique définie positive d'ordre $n$ et $\boldsymbol{b} \in \mathbb{R}^n$.

**Données :** $\mathbb{A}$ : matrice symétrique définie positive d'ordre $n$,
$\boldsymbol{b}$ : vecteur de $\mathbb{R}^n$.
**Résultat :** $\boldsymbol{x}$ : vecteur de $\mathbb{R}^n$.

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

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 21** Fonction RSLCholesky permettant de résoudre, par une factorisation de Cholesky positive, le système linéaire

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

où $\mathbb{A}$ une matrice symétrique définie positive d'ordre $n$ et $\boldsymbol{b} \in \mathbb{R}^n$.

**Données :** $\mathbb{A}$ : matrice symétrique définie positive d'ordre $n$,
$\boldsymbol{b}$ : vecteur de $\mathbb{R}^n$.
**Résultat :** $\boldsymbol{x}$ : vecteur de $\mathbb{R}^n$.

1: **Fonction** $\boldsymbol{x} \leftarrow$ RSLCholesky( $\mathbb{A}, \boldsymbol{b}$ )
2: $\mathbb{L} \leftarrow$Cholesky($\mathbb{A}$) ▷ Factorisation positive de Cholesky
3: $\boldsymbol{y} \leftarrow$ResTriInf($\mathbb{L}, \boldsymbol{b}$) ▷ Résolution du système $\mathbb{L}\boldsymbol{y} = \boldsymbol{b}$
4: $\mathbb{U} \leftarrow$MatTranspose($\mathbb{L}$) ▷ Calcul de la transposé de $\mathbb{L}$
5: $\boldsymbol{x} \leftarrow$ResTriSup($\mathbb{U}, \boldsymbol{y}$) ▷ Résolution du système $\mathbb{L}^t\boldsymbol{x} = \boldsymbol{y}$
6: **Fin Fonction**

## 7.2 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

$$A_{i,j} = 0 \quad \text{si} \quad j > i. \tag{7}$$

$$\begin{pmatrix} A_{1,1} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ A_{n,1} & \dots & \dots & A_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{pmatrix} \tag{8}$$

On a

$$\forall i \in \{1, \dots, n\} \quad (\mathbb{A}\boldsymbol{x})_i = b_i,$$

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

$$\forall i \in \{1, \dots, n\} \quad \sum_{j=1}^{n} A_{i,j} x_j = b_i.$$

En utilisant (7), on obtient

$$\forall i \in \{1, \dots, n\} \quad \sum_{j=1}^{i} A_{i,j} x_j = b_i.$$

Ce qui donne

$$\forall i \in \{1, \dots, n\} \quad x_i = \frac{1}{A_{i,i}} \left( b_i - \sum_{j=1}^{i-1} A_{i,j} x_j \right). \tag{9}$$

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 22-** $\mathcal{R}_0$

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

**Algorithme 22-** $\mathcal{R}_1$

1: **Pour** $i \leftarrow 1$ **à** $n$ **faire**
2: calculer $x_i$ connaissant $x_1, \dots, x_{i-1}$ à l'aide de l'équation (9)
3: **Fin Pour**

**Algorithme 22-** $\mathcal{R}_1$

1: **Pour** $i \leftarrow 1$ **à** $n$ **faire**
2: Calculer $x_i$ connaissant $x_1, \dots, x_{i-1}$ à l'aide de l'équation (9)
3: **Fin Pour**

**Algorithme 22-** $\mathcal{R}_2$

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 22-** $\mathcal{R}_2$

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 22-** $\mathcal{R}_3$

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'algorithe final

**Algorithme 22** Fonction ResTriInf permettant de résoudre le système linéaire triangulaire inférieur inversible

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

**Données :** $\mathbb{A}$ : matrice triangulaire inférieure inversible d'ordre $n$.
$\boldsymbol{b}$ : 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
```

## 7.3 Résolution d'un système linéaire triangulaire supérieur

Soit $\mathbb{A}$ une matrice d'ordre $n$ 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

$$A_{i,j} = 0 \quad \text{si} \quad j < i. \tag{10}$$

$$\begin{pmatrix} A_{1,1} & \dots & \dots & A_{1,n} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{pmatrix} \tag{11}$$

On a

$$\forall i \in \{1, \dots, n\} \quad (\mathbb{A}\boldsymbol{x})_i = b_i,$$

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

$$\forall i \in \{1, \dots, n\} \quad \sum_{j=1}^{n} A_{i,j} x_j = b_i.$$

En utilisant 10, on obtient

$$\forall i \in \{1, \dots, n\} \quad \sum_{j=i}^{n} A_{i,j} x_j = b_i.$$

Ce qui donne

$$\forall i \in \{1, \dots, n\} \quad x_i = \frac{1}{A_{i,i}} \left( b_i - \sum_{j=i+1}^{n} A_{i,j} x_j \right). \tag{12}$$

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 23-** $\mathcal{R}_0$

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

**Algorithme 23-** $\mathcal{R}_1$

1: **Pour** $i \leftarrow n$ **à** 1 **faire** (**pas de** $-1$)
2: calculer $x_i$ connaissant $x_{i+1}, \ldots, x_n$ à l'aide de l'équation (12)
3: **Fin Pour**

**Algorithme 23-** $\mathcal{R}_1$

1: **Pour** $i \leftarrow n$ **à** 1 **faire** (**pas de** $-1$)
2: Calculer $x_i$ connaissant $x_{i+1}, \ldots, x_n$ à l'aide de l'équation (12)
3: **Fin Pour**

**Algorithme 23-** $\mathcal{R}_2$

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 23-** $\mathcal{R}_2$

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 23-** $\mathcal{R}_3$

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 23** Fonction ResTriSup permettant de résoudre le système linéaire triangulaire supérieur inversible

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

**Données :** $\mathbb{A}$ : matrice triangulaire supérieur inversible d'ordre $n$.
$\boldsymbol{b}$ : vecteur de $\mathbb{R}^n$.
**Résultat :** $\boldsymbol{x}$ : vecteur de $\mathbb{R}^n$.

```
1: Fonction x ← ResTriSup( A, b )
2:     Pour i ← n à 1 faire (pas de −1)
3:         S ← 0
4:         Pour j ← i + 1 à n 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: Fin Fonction
```

## 7.4 Factorisation positive de Cholesky

Soit $\mathbb{A} \in \mathcal{M}_{n,n}(\mathbb{R})$ une matrice symétrique définie positive. D'après le théorème 7, il existe une unique matrice $\mathbb{L} \in \mathcal{M}_{n,n}(\mathbb{R})$ triangulaire inférieure avec $\forall i \in$

$\{1,\dots,n\}$ $L_{i,i} > 0$ telle que

$$\mathbb{A} = \mathbb{L}\mathbb{L}^t \tag{13}$$

c'est à dire

$$\begin{pmatrix} A_{1,1} & \dots & A_{1,n} \\ \vdots & \ddots & \dots \\ A_{n,1} & \dots & A_{n,n} \end{pmatrix} = \begin{pmatrix} L_{1,1} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ L_{n,1} & \dots & \dots & L_{n,n} \end{pmatrix} \begin{pmatrix} L_{1,1} & \dots & \dots & L_{1,n} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & L_{n,n} \end{pmatrix}. \tag{14}$$

Pour déterminer la matrice $\mathbb{L}$, on commence par calculer $L_{1,1}$ (la 1ère ligne de $\mathbb{L}$ est donc déterminée) ce qui nous permet de calculer la 1ère colonne de $\mathbb{L}$.
Ensuite, on calcule $L_{2,2}$ (la 2ème ligne de $\mathbb{L}$ est donc déterminée) ce qui nous permet de calculer la 2ème colonne de $\mathbb{L}$. Etc ...

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-** $\boxed{\mathcal{R}_0}$

1: Calculer la matrice $\mathbb{L}$

**Algorithme 24-** $\boxed{\mathcal{R}_1}$

1: **Pour** $i \leftarrow 1$ **à** $n$ **faire**
2: Calculer $L_{i,i}$, connaissant les $i-1$ premières colonnes de $\mathbb{L}$.
3: Calculer la $i^{\text{ème}}$ colonne de $\mathbb{L}$.
4: **Fin Pour**

Pour calculer $L_{i,i}$, connaissant les $i-1$ premières colonnes de $\mathbb{L}$. on utilise (13) :

$$A_{i,i} = \sum_{j=1}^{n} L_{i,j} L_{j,i}^t = \sum_{j=1}^{n} L_{i,j}^2$$

et comme $\mathbb{L}$ est triangulaire inférieure on obtient

$$A_{i,i} = \sum_{j=1}^{i} L_{i,j}^2 = \sum_{j=1}^{i-1} L_{i,j}^2 + L_{i,i}^2.$$

On a donc

$$L_{i,i} = \left( A_{i,i} - \sum_{j=1}^{i-1} L_{i,j}^2 \right)^{1/2} > 0. \tag{15}$$

Comme les $i-1$ premières colonnes de $\mathbb{L}$ ont déjà été calculées, $L_{i,i}$ est parfaitement déterminée par la formule précédente.

Pour calculer la $i^{\text{ème}}$ colonne de $\mathbb{L}$, il suffit de déterminer $L_{j,i}$, $\forall j \in \{i+1,\dots,n\}$. Pour celà on utilise (13)

$$\forall j \in \{i+1,\dots,n\},\ A_{j,i} = \sum_{k=1}^{n} L_{j,k} L_{k,i}^t = \sum_{k=1}^{n} L_{j,k} L_{i,k}$$

Comme $\mathbb{L}$ est triangulaire inférieure on obtient

$$\forall j \in \{i+1,\dots,n\},\ A_{j,i} = \sum_{k=1}^{i} L_{j,k} L_{i,k} = \sum_{k=1}^{i-1} L_{j,k} L_{i,k} + L_{j,i} L_{i,i}.$$

Or $L_{i,i} > 0$ vient d'être calculé et les $i-1$ premières colonnes de $\mathbb{L}$ ont déjà été calculées, ce qui donne

$$\forall j \in \{i+1,\dots,n\}, L_{j,i} = \frac{1}{L_{i,i}}\left(A_{j,i} - \sum_{k=1}^{i-1} L_{j,k}L_{i,k}\right), \quad (16)$$

$$\forall j \in \{1,\dots,i-1\}, L_{j,i} = 0. \quad (17)$$

Avec (15) et (16), on a

**Algorithme 24-** $\mathcal{R}_1$

1: **Pour** $i \leftarrow 1$ **à** $n$ **faire**
2: Calculer $L_{i,i}$, connaissant les $i-1$ premières colonnes de $\mathbb{L}$.
3: Calculer la $i^{\text{ème}}$ colonne de $\mathbb{L}$.
4: **Fin Pour**

**Algorithme 24-** $\mathcal{R}_2$

1: **Pour** $i \leftarrow 1$ **à** $n$ **faire**
2: $L_{i,i} \leftarrow \left(A_{i,i} - \sum_{j=1}^{i-1} L_{i,j}^2\right)^{1/2}$
3: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
4: $L_{j,i} \leftarrow 0$
5: **Fin Pour**
6: **Pour** $j \leftarrow i+1$ **à** $n$ **faire**
7: $L_{j,i} \leftarrow \frac{1}{L_{i,i}}\left(A_{j,i} - \sum_{k=1}^{i-1} L_{j,k}L_{i,k}\right)$.
8: **Fin Pour**
9: **Fin Pour**

**Algorithme 24-** $\mathcal{R}_2$

1: **Pour** $i \leftarrow 1$ **à** $n$ **faire**
2: $L_{i,i} \leftarrow \left(A_{i,i} - \sum_{j=1}^{i-1} L_{i,j}^2\right)^{1/2}$
3: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
4: $L_{j,i} \leftarrow 0$
5: **Fin Pour**
6: **Pour** $j \leftarrow i+1$ **à** $n$ **faire**
7: $L_{j,i} \leftarrow \frac{1}{L_{i,i}}\left(A_{j,i} - \sum_{k=1}^{i-1} L_{j,k}L_{i,k}\right)$.
8: **Fin Pour**
9: **Fin Pour**

**Algorithme 24-** $\mathcal{R}_3$

1: **Pour** $i \leftarrow 1$ **à** $n$ **faire**
2: $S_1 \leftarrow \sum_{j=1}^{i-1} L_{i,j}^2$
3: $L_{i,i} \leftarrow (A_{i,i} - S_1)^{1/2}$
4: **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
5: $L_{j,i} \leftarrow 0$
6: **Fin Pour**
7: **Pour** $j \leftarrow i+1$ **à** $n$ **faire**
8: $S_2 \leftarrow \sum_{k=1}^{i-1} L_{j,k}L_{i,k}$
9: $L_{j,i} \leftarrow \frac{1}{L_{i,i}}(A_{j,i} - S_2)$.
10: **Fin Pour**
11: **Fin Pour**

**Algorithme 24-** $\mathcal{R}_3$

1: **Pour** $i \leftarrow 1$ **à** $n$ **faire**
2: $\quad S_1 \leftarrow \sum_{j=1}^{i-1} L_{i,j}^2$
3: $\quad L_{i,i} \leftarrow (A_{i,i} - S_1)^{1/2}$
4: $\quad$ **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
5: $\quad\quad L_{j,i} \leftarrow 0$
6: $\quad$ **Fin Pour**
7: $\quad$ **Pour** $j \leftarrow i+1$ **à** $n$ **faire**
8: $\quad\quad S_2 \leftarrow \sum_{k=1}^{i-1} L_{j,k} L_{i,k}$
9: $\quad\quad L_{j,i} \leftarrow \frac{1}{L_{i,i}} (A_{j,i} - S_2)\,.$
10: $\quad$ **Fin Pour**
11: **Fin Pour**

**Algorithme 24-** $\mathcal{R}_4$

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

On obtient alors l'algorithme final

**Algorithme 24** Fonction CHOLESKY permettant de calculer la matrice $\mathbb{L}$, dites matrice de factorisation positive de Cholesky associée à la matrice $\mathbb{A}$, telle que

$$\mathbb{A} = \mathbb{L}\mathbb{L}^t.$$

**Données :** $\mathbb{A}$ : matrice de $\mathcal{M}_{n,n}(\mathbb{R})$ symétrique définie positive.
**Résultat :** $\mathbb{L}$ : matrice de $\mathcal{M}_{n,n}(\mathbb{R})$ triangulaire inférieure avec $\forall i \in \{1,\dots,n\}\ L_{i,i} > 0$.

1: **Fonction** $\mathbb{L} \leftarrow$ CHOLESKY( $\mathbb{A}$ )
2: $\quad$ **Pour** $i \leftarrow 1$ **à** $n$ **faire**
3: $\quad\quad S_1 \leftarrow 0$
4: $\quad\quad$ **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
5: $\quad\quad\quad S_1 \leftarrow S_1 + L(i,j)^2$
6: $\quad\quad$ **Fin Pour**
7: $\quad\quad L(i,i) \leftarrow$**SQRT**$(A_{i,i} - S_1)$
8: $\quad\quad$ **Pour** $j \leftarrow 1$ **à** $i-1$ **faire**
9: $\quad\quad\quad L(j,i) \leftarrow 0$
10: $\quad\quad$ **Fin Pour**
11: $\quad\quad$ **Pour** $j \leftarrow i+1$ **à** $n$ **faire**
12: $\quad\quad\quad S_2 \leftarrow 0$
13: $\quad\quad\quad$ **Pour** $k \leftarrow 1$ **à** $i-1$ **faire**
14: $\quad\quad\quad\quad S_2 \leftarrow S_2 + L(j,k) * L(i,k)$
15: $\quad\quad\quad$ **Fin Pour**
16: $\quad\quad\quad L(j,i) \leftarrow (A(j,i) - S_2)/L(i,i).$
17: $\quad\quad$ **Fin Pour**
18: $\quad$ **Fin Pour**
19: **Fin Fonction**

# Liste des algorithmes