INFOF-205 : Calcul formel et numérique

R. Petit

Année académique 2015 - 2016

Contents

1 Introduction

Définition 1.1. Le *calcul numérique* est une discipline qui traite de la définition, l'analyse, et l'implémentation d'algorithmes pour la résolution numérique des problèmes mathématiques qui proviennent de la modélisation des phénomènes réels.[1]

Le calcul numérique représente donc le triplet (modèle analytique, solution théorique, résolution algorithmique).
Remarque. Le calcul numérique étant une application informatique pour la résolution de problèmes mathématiques continus, plusieurs types de problèmes peuvent arriver (section suivante). Afin de limiter les problèmes de précision sur des fonctions continues compliquées à évaluer précisément informatiquement, plusieurs outils d'approximation peuvent être utilisés. Le théorème de Taylor permet d'approcher certaines fonctions de manière plus facilement calculatoires de manière informatique.

Théorème 1.2 (Théorème de Taylor)**.** *Soient* $f : I \to \mathbb{R}$ *où* I *est un intervalle,* $a \in I$ *et* $k \geqslant 1$*. Alors* $T(f, a, k)(x)$ *est le seul polynôme de degré inférieur ou égal à* k *tel que :*

$$\lim_{x \to a} \frac{f(x) - T(f, a, k)}{(x - a)^k} = 0,$$

où $T(f, a, k)(x)$*, appelé* polynôme de Taylor *est défini par :*

$$T(f, a, k)(x) = \sum_{i=0}^{k} \frac{f^{(i)}(a)(x - a)^i}{i!}.$$

Le calcul numérique est défini comme étant une application informatique à des problèmes relatifs à des phénomènes réels. Bien souvent, ces phénomènes sont modélisés de manière *continue*. Les ordinateurs quant à eux sont *discrets*. Cette différence majeure force des approximations par le matériel informatique. Les transformations discrètes (*discrétisations*) des phénomènes réels doivent alors être rigoureusement analysés afin de déterminer leur robustesse et leur fiabilité.

Les problèmes peuvent être classifiés de plusieurs manières selon leurs propriétés et ce qui en est attendu.

Définition 1.3. Un problème est dit :

- *qualitatif* lorsque le comportement des solutions est étudié afin d'en déterminer sa stabilité, son comportement asymptotique ;
- *quantitatif* lorsque la solution (numérique) précise est étudiée afin de tirer des conclusions spécifiques.

Un problème qualitatif cherche des conclusions globales à un problème (étude d'une famille de problèmes) et un problème quantitatif est une application précise appliquée à un problème donné.

Définition 1.4. Un problème est dit sous forme :

- *explicite* quand la solution x d'un problème est une fonction donnée des données d ;
- *implicite* quand la solution ne peut être extraite explicitement des données.

[1] Nick Trefehen, Oxford.

Exemple 1. *Trouver la racine carrée d'un nombre* $d \in \mathbb{R}$ est un problème explicite que l'on peut écrire $x = f(d) = \sqrt{d}$.

Déterminer l'ensemble des solutions x *à l'équation* $ax^2 + bx + c$ est un problème implicite car la solution n'est pas séparée explicitement des données. On peut tout de même exprimer ce problème implicite comme :

$$X = \left\{ \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \right\},$$

qui est un problème explicite.
Remarque. La forme générale d'un problème explicite est $x = F(d)$, alors que la forme générale d'un problème implicite est $F(x, d) = 0$.

Définition 1.5. Le problème explicite $x = F(d)$ est dit *bien posé* si la solution x :

1. existe ;
2. est unique ;
3. dépend **continument** de d.

Si un problème n'est pas bien posé, on dit qu'il est *mal posé*.

Définition 1.6. Un problème est dit *problème inverse* s'il correspond à la détermination des données d'entrée x d'une application G en connaissant les données de sortie $G(x) =: d$.

Remarque. Les problèmes inverses sont de manière générale mal posés, entre autre parce qu'il est fréquent que l'application G ne soit pas injective et donc que plusieurs données x peuvent donner la même donnée $d = G(x)$.
Remarque. Ce cours ne traitera que de problèmes bien posés.

Définition 1.7.

- Le *conditionnement* d'un problème $x = F(d)$ bien posé est la mesure de la *sensibilité* de x à des petits changements sur d.
- On définit par δd une faible perturbation (également dite *perturbation admissible*) sur la donnée d. On définit alors la perturbation induite sur x par $\delta x = F(d + \delta d) - F(d)$.
- Le *conditionnement relatif* du problème est la quantité :

$$\kappa(d) \coloneqq \lim_{|D| \to 0} \sup_{\delta d \in D} \frac{\|\delta x\| \|d\|}{\|x\| \|\delta d\|} = \lim_{|D| \to 0} \sup_{\delta \in D} \frac{\left\|F(d + \delta d) - F(d)\right\| \|d\|}{\left\|F(d)\right\| \|\delta d\|},$$

  où D est un voisinage de l'origine, et sup désigne la borne supérieure.

Remarque.

- si $x = 0$, alors le conditionnement relatif est forcément infini ;
- si $d = 0$, alors le conditionnement relatif est forcément nul ;
- quand $0 \in \{x, d\}$, il faut définir une autre notion, le conditionnement absolu :

$$\kappa_{abs}(d) \coloneqq \lim_{|D| \to 0} \sup_{\delta d \in D} \frac{\|\delta x\|}{\|\delta d\|}.$$

Proposition 1.8. *Si* $x = F(d)$ *est un problème explicite, et* F *est une fonction différentiable en* d*, alors, par définition :*

$$\lim_{\delta \to 0} \frac{\delta x}{\delta d}$$

ne dépend pas de la trajectoire de δd *vers* 0*. On note alors :*

$$F'(d) = \lim_{\delta d \to 0} \frac{\delta x}{\delta d},$$

et on réécrit le conditionnement par :

$$\kappa(d) = \lim_{\delta d \to 0} \frac{\|\delta x\|\|d\|}{\|\delta d\|\|x\|} = \frac{\|d\|}{\|x\|} \lim_{\delta d \to 0} \frac{\|\delta x\|}{\|\delta d\|} = \left\|F'(d)\right\| \frac{\|d\|}{\|x\|}.$$

Remarque. Si $d \in \mathbb{R}^m$, et $x \in \mathbb{R}^n$, alors $F'(d)$ est la matrice jacobienne $(\mathrm{Jac}\, F)(d)$. Et si $q = 1$, la matrice jacobienne n'a qu'une seule ligne et on parle du gradient $(\nabla F)(d)$
Exemple 2 (Conditionnement d'une différence). Soit le problème $x = F(d) = d - a$. La fonction F est différentiable, on peut donc déterminer le conditionnement :

$$\kappa(d) = \left\|F'(d)\right\| \frac{\|d\|}{\|x\|} = 1 \frac{\|d\|}{\|d - a\|}.$$

On voit alors que le conditionnement est petit pour $\|d - a\| >> 0$. Cependant, pour d proche de a, le conditionnement devient très grand. Le problème est donc *mal conditionné* pour d proche de a.
Exemple 3 (Conditionnement d'une exponentiation). Soit le problème $x = F(d) = d^r$ avec $r \in \mathbb{R}$. À nouveau, F est différentiable, et donc on peut exprimer le conditionnement comme :

$$\kappa(d) = \left\|F'(d)\right\| \frac{\|d\|}{\|x\|} = \|r\| \left\|d^{r-1}\right\| \frac{\|d\|}{\|d^r\|} = \|r\|.$$

Le conditionnement dépend donc de l'exposant r. Si r est grand, le problème est mal conditionné, peu importe la valeur de d.

Le conditionnement d'un problème est une caractéristique qui lui est intrinsèque. Si un problème est mal conditionné, peu importe l'algorithme utilisé pour le résoudre, la solution restera fortement influencé par des perturbations sur les données d. De plus, plus un problème a un grand conditionnement, plus il sera difficile à résoudre numériquement.
Remarque. les exemples ci-dessus montrent qu'un problème peut avoir un conditionnement faible pour certaines valeurs de d et un conditionnement élevé pour d'autres valeurs.

Soit $F(x, d) = g(x) - d = 0$, un problème implicite. Prenons $g : \mathbb{R} \to \mathbb{R}$ une fonction réelle différentiable. On peut alors exprimer $x = g^{-1}(d)$. Donc le conditionnement vaut :

$$\kappa(d) = \left| \frac{(g^{-1})'(d)d}{g^{-1}(d)} \right| = \left| \frac{1}{g'(g^{-1}(d))} \right| \left| \frac{d}{g^{-1}(d)} \right| = \left| \frac{1}{g'(x)} \right| \left| \frac{d}{x} \right|.$$

On remarque donc que pour $g'(x)$ petit le conditionnement devient grand. C'est intuitivement lié au fait qu'une fonction à faible dérivée (pente) en d va croitre lentement et donc pour une faible perturbation sur d (axe vertical), une grande perturbation sur x (axe horizontal) va être induite.

Définition 1.9. Soit $F(x, d) = 0$ un problème bien posé. On définit l'*algorithme numérique* pour la résolution du problème F par la suite de problèmes approchés $F_1(x^{(1)}, d^{(1)}), F_2(x^{(2)}, d^{(2)}), \ldots, F_n(x^{(n)}, d^{(n)})$ dépendant d'un paramètre n.

L'idée derrière la division en sous-problèmes est d'avoir des F_k plus simples à résoudre que F, ce qui permet une résolution numérique.

Définition 1.10. Un algorithme numérique est dit :

- *direct* si le nombre n de sous-problèmes est fixé (du moins majoré), habituellement par une fonction de la taille du problème ;
- *itératif* si n n'est pas borné et s'il existe une routine $f(u)$ admissible, indépendante de n telle que $x^{(n)} = f(x^{(n-1)})$.

Définition 1.11. Un algorithme numérique $\{F_i(x^{(i)}, d^{(i)})\}_{1 \leqslant i \leqslant n}$ est consistant si :

$$\lim_{n \to +\infty} F_n(x^{(n)}, d^{(n)}) = F(x, d^{(n)}),$$

et donc si x, la solution précise, est une solution ses sous-problèmes pour $n \to +\infty$.

Remarque. Un algorithme itératif $x^{(n)} = f(x^{(n-1)})$ est consistant si $x^{(n)} = x$ implique que $x^{(n+1)} = x$. Ce qui revient à dire qu'une fois la solution exacte atteinte, l'algorithme itératif la conserve et ne diverge pas.

Définition 1.12. Un algorithme $F_n(x^{(n)}, d^{(n)})$ est dit *fortement* consistant si :

$$\forall n \geqslant 1 : F_n(x, d^{(d)}) = 0,$$

c'est-à-dire si x est une solution admissible de tous les sous-problèmes $F_n(x^{(n)}, d^{(n)})$.

Intéressons-nous au problème $F(x, d)$ et à $F_n(x^{(n)}, d^{(n)})$, un algorithme numérique de résolution du problème. Supposons que cet algorithme numérique soit composé d'étapes telles que :

$$x^{(n)} = F_n^{(m)}\left(d_{m-1}^{(n)}\right),$$

où :

$$d_k^{(n)} = F_n^{(k)}\left(d_{k-1}^{(n)}\right) \qquad \text{et} \qquad d_0^{(n)} = d^{(n)}.$$

On considère que la résolution numérique est l'application de cet algorithme sur des valeurs perturbées :

$$\begin{cases} \bar{x}^{(n)} & = F_m^{(n)}\left(\bar{d}_{m-1}^{(n)}\right) + \delta d_m^{(n)} \\ \bar{d}_k^{(n)} & = F_k^{(n)}\left(\bar{d}_{k-1}^{(n)}\right) + \delta d_k^{(n)} \\ \bar{d}_0^{(n)} & = d^{(n)} \end{cases}$$

Remarque. On voit bien que l'algorithme à l'étape n commence avec $\bar{d}_0^{(n)} = d^{(n)}$ qui est une donnée initiale *non perturbée*.

Définition 1.13. Soit $F_n(x^{(n)}, d^{(n)})$ un algorithme numérique. On dit qu'il est *stable* (ou *bien posé*) si :

- $\forall n : \exists! x^{(n)}$, solution ;
-

$$\forall \epsilon > 0 : \exists \eta > 0, n_0 \in \mathbb{N}^* \text{ t. q. } \forall n > n_0 : \left(\forall i \in \{1, \dots, m\} : \left\|\delta d_i^{(n)}\right\| < \eta\right) \Rightarrow \left(\left\|\bar{x}^{(n)} - x^{(n)}\right\| < \epsilon\right).$$

Définition 1.14. Si l'opération $F_m^{(n)}(d)$ ne dépend ni de m, ni de n, on parle d'algorithme *itératif*.

Remarque. Un algorithme est donc dit itératif si c'est constamment la même opération qui est appliquée sur des données consécutives. On a alors :

$$x^{(n)} = f(x^{(n-1)}) = (f \circ f)(x^{(n-2)}) = \cdots = (f^n)\,(x^{(0)}) = (f^n)\,(d).$$

On peut traduire la définition de stabilité de manière plus informelle par le fait qu'un algorithme est stable s'il existe un voisinage de x, la solution tel que pour tout x_1, x_2 dans ce voisinage, on a un $R \in (0,1)$ tel que :

$$\left\|f(x_2) - f(x_1)\right\| \leqslant R\|x_2 - x_1\|\,.$$

Si f est une fonction différentiable, alors on peut dire que :

$$\left\|f'(x_2)\right\| \leqslant R.$$

2 Analyse et mesure d'erreurs

2.1 Définitions

Définition 2.1. $\mathbb{F}$ est l'ensemble des nombres représentés exactement par un ordinateur.

Remarque. On remarque aisément que $\mathbb{F} \subset \mathbb{R}$ car $\mathbb{F} \not\subset \mathbb{C} \setminus \mathbb{R}$. On peut tout de même raffiner car il semble évident que $\mathbb{F} \subset \mathbb{Q}$: un nombre irrationnel n'a pas de représentation physique finie, et ne peut donc pas être représentés par un ordinateur.

Définition 2.2. Soit x un nombre dans $\mathbb{Q}$. Une approximation de x est une valeur $\widehat{x}$ légèrement différente de x et qui remplace x dans les calculs effectués. On note également $x \simeq \widehat{x}$ pour montrer que $\widehat{x}$ est une approximation de x.

Si $\widehat{x} < x$, on parle d'approximation par défaut, et si $\widehat{x} > x$, on parle d'approximation par excès.

Définition 2.3. On définit l'*erreur absolue* entre x et son approximation $\widehat{x}$ par :

$$\delta_x = |\widehat{x} - x| .$$

On définit également l'*écart relatif* entre x et $\widehat{x}$ par :

$$\rho_x = \frac{\widehat{x} - x}{x}.$$

On définit alors l'écart absolue défini comme $\epsilon_x = |\rho_x|$.

Remarque. L'écart relatif permet d'écrire l'approximation sous la forme $\widehat{x} = x(1 + \rho_x)$ où ρ_x est censé être faible (car l'écart entre x et son approximation doit rester relativement faible). Cette notation est fortement utilisée.

Les écarts ne sont cependant définis qu'en $x \neq 0$, ce dont il faut tenir compte.

Définition 2.4. La *borne supérieure d'erreur relative* d'une approximation $\widehat{x}$, notée u est nombre positif quelconque tel que $u \geqslant \epsilon_x$.

Mis à part les erreurs de calcul (qui consistent à déterminer un résultat erroné sur base de données bien définies) et les erreurs d'implémentation (qui consiste en une implémentation erronée amenant à des erreurs de calcul), il existe cinq catégories d'erreurs que l'on peut regrouper en deux familles :

- les erreurs de modèle qui consistent en simplifier les phénomènes en omettant des éléments afin de le rendre plus facile à étudier ;
- les erreurs numériques qui consistent en l'approximation due à l'observation et non à la résolution théorique.

2.2 Les erreurs numériques[2]

Définition 2.5. Les *erreurs de troncature* sont les erreurs liées aux processus théoriques mathématiques infinis (série infinie par exemple).

[2]Les erreurs de modélisation ne sont pas traitées ici.

Définition 2.6. Les *erreurs d'arrondi* sont les erreurs liées au système numérique de la machine venant du fait qu'un ordinateur ne peut représenter qu'un sous-ensemble fini $\mathbb{F}$ des nombres.

Définition 2.7. Les *erreurs de génération et de propagation* sont les erreurs liées à l'évaluation d'une opération sur des opérandes pas exactes.

Remarque. Les erreurs de troncature sont compliquées à éviter : il est nécessaire d'avoir un procédé fini pour y échapper. Cette catégorie d'erreurs ne seront donc pas étudiées dans ce cours.

Les différents types d'erreurs concernent différentes propriétés des problèmes ou des algorithmes numériques utilisés.

Proposition 2.8.

- *Les erreurs d'approximation ou de troncature se mesurent par la* ***consistance*** *de l'algorithme ;*
- *les erreurs de propagation se mesurent par le* ***conditionnement*** *de l'algorithme ;*
- *les erreurs de génération se mesurent par la* ***stabilité de l'algorithme*** *;*
- *les erreurs d'arrondi sont liées à la représentation interne des nombres.*

2.3 Représentation interne des nombres par l'ordinateur

Il existe deux manières de représenter les nombres décimaux par un ordinateur : la *virgule fixe* et la *virgule flottante*.

2.3.1 Représentation en virgule fixe

Soit $x \in \mathbb{F}$ un nombre. Sa représentation en virgule fixe est donnée par le triplet $(\{a_i\}_{-m \leqslant i \leqslant n}, b, s)$ où :

- $b \geqslant 2$ est la base de représentation ;
- $s \in \{0, 1\}$ est le signe ;
- $\{a_{-m}, \ldots, a_n\}$ est l'ensemble de chiffres $0 \leqslant a_i \lneqq b$ pour tout i.

Remarque. m représente le nombre de chiffres suivant la virgule, et $n+1$ caractérise le nombre de chiffres précédant la virgule. On en déduit que la virgule est (implicitement) placée entre a_0 et a_{-1}.

On peut donc exprimer x par :

$$x = (-1)^s \sum_{k=-m}^{n} a_k b^k.$$

Remarque. La base choisie pour les ordinateurs est souvent 2, 8 ou 16 car 2 représente le binaire, et les bases octale et hexadécimale représentent chacune une écriture *compactée* du binaire par paquets de 3 ou 4 bits.

La base 10 est par fois également choisie.

Proposition 2.9 (Propriétés des nombres en virgule fixe)**.**

- *Les nombres représentés par virgule fixe sont équirépartis sur la droite des réels ;*
- *l'écart entre deux nombres consécutifs représentés en virgule flottante est* b^{-m}*, qui est le plus petit nombre représentable (en valeur absolue) ;*
- *et la plus grande valeur représentable est* $b^{n+1} - 1$*, où* $n+1$ *est le nombre de chiffres avant la virgule.*

Remarque. La représentation par virgule flottante limite fortement les valeurs extrêmes que peut prendre un nombre de par son équirépartition. Les nombres à virgule fixe (comme vu juste après) ont une densité de répartition différente, ce qui leur permet de prendre des valeurs maximum et minimum bien plus haute (en terme d'ordre de grandeur).

2.3.2 Représentation en virgule flottante

Soit $x \in \mathbb{F}$ un nombre. Sa représentation en virgule flottante est donnée par le quadruplet $(\{a_i\}_{1 \leqslant i \leqslant t}, b, s, e)$où :

- b et s représentent les mêmes quantités que pour la virgule fixe ;
- t est le nombre de *chiffres significatifs* ;
- $\{a_i\}_{1 \leqslant i \leqslant t}$ est l'ensemble des chiffres $0 \lneqq a_1 \lneqq b$ et $0 \leqslant a_i \lneqq b$ pour $i > 1$.

Remarque. On impose $a_1 \neq 0$ afin que la représentation d'un nombre soit unique.

Définition 2.10. La *mantisse* est la quantité définie par $\sum_{i=1}^{t} a_i b^{t-i}$. On la note $m \in \mathbb{N}$, ou encore $m[x] \in \mathbb{N}$ quand il est nécessaire de montrer que m est la mantisse de la valeur x.

Remarque. Dû à la normalisation des a_i, on sait borner m par :

$$b^{t-1} \leqslant m \leqslant b^t - 1.$$

On peut donc au final exprimer x comme :

$$x = (-1)^s b^e \sum_{i=1}^{t} a_i b^{-i} = (-1)^s b^e b^t b^{-t} \sum_{i=1}^{t} a_i b^{-i} = (-1)^s b^{e-t} \sum_{i=1}^{t} a_i b^{t-i} = (-1)^s b^{e-t} m.$$

Remarque. On note L et U les valeurs respectivement plus petite et plus grande que peut prendre la valeur e de l'exposant.

Proposition 2.11. *On paramétrise l'ensemble $\mathbb{F}$ par les quantités b, t, L, et U. Quand la paramétrisation doit être explicite, on note l'ensemble :*

$$\mathbb{F}(b, t, L, U).$$

Remarque. Dans une représentation normalisée, $0 \notin \mathbb{F}$.

Théorème 2.12. *Si x est un élément de $\mathbb{F}(b, t, L, U)$, alors on peut borner x tel que :*

$$b^{L-1} \leqslant |x| \leqslant b^U (1 - b^t).$$

Proposition 2.13. *Le cardinal de l'ensemble $\mathbb{F}(b, t, L, U)$ est donné par :*

$$\left|\mathbb{F}(b, t, L, U)\right| = 2(b-1)b^{t-1}(U - L + 1).$$

Remarque. Étant donné que l'exposant e est code sur un nombre n_e de bits (et représenté en complément à 2) dans une représentation machine de représentation flottante, on peut trouver que $L = -2^{n_e - 1} - 1 \leqslant e \leqslant 2^{n_e} = U$. On voit alors que les valeurs maximum prises par un nombre sont beaucoup plus grandes (en valeur absolue de l'ordre de grandeur) qu'en virgule fixe.

2.3.3 Distance relative dans $\mathbb{F}$

Prenons x_i et x_{i+1} dans $\mathbb{F}$. On note $\eta(x_i)$ la distance relative entre x_i et x_{i+1} que l'on définit :

$$\eta(x_i) : \left| \frac{x_{i+1} - x_i}{x_i} \right|.$$

Dans un système à virgule fixe, $\eta(x_i)$ est fixe pour tout i. Pour un système à virgule flottante, on trouve :

$$\eta(x_i) = \left| \frac{x_{i+1} - x_i}{x_i} \right| = \left| \frac{m[x_{i+1}]b^{e-t} - m[x_i]b^{e-t}}{m[x_i]b^{e-t}} \right| = \frac{1}{m[x_i]}.$$

Si x_{i+1} a un exposant plus grand que x_i, on a :

$$\eta(x_i) = \left| \frac{b^{t-1}b^{e-t+1} - (b^t - 1)b^{e-t}}{(b^t - 1)^{e-t}} \right| = \left| \frac{b^{e-t}}{(b^t - 1)b^{e-t}} \right| = \frac{1}{m[x_i]}.$$

On peut donc, en virgule flottante, également déterminer une distance générale (formule) mais qui n'est pas constante.

En sachant que :

$$b^{t-1} \leqslant m < b^t,$$

on peut trouver :

$$b^{-t} < \frac{1}{m} = \eta \leqslant b^{1-t}.$$

En divisant de part et d'autre par b^{1-t}, on trouve :

$$\frac{\epsilon}{b} < \eta(x_i) \leqslant \epsilon.$$

Définition 2.14. On appelle $\epsilon := b^{1-t}$ l'*epsilon machine* qui représente la distance maximale entre deux nombres consécutifs en virgule flottante.

Remarque. L'epsilon machine permet de déterminer $1 + \epsilon$ qui est le plus petit nombre $x \in \mathbb{F}$ tel que $x \gneqq 1$.
Remarque. Il ne faut pas confondre ϵ qui est la distance maximale entre deux nombres et b^{L-1} qui est le plus petit nombre que l'on peut représenter. En pratique, $\epsilon \simeq 10^{-16}$ alors que $b^{L-1} \simeq 10^{-308}$.

Définition 2.15. Le phénomène d'oscillation de $\eta(x_i)$ est appelé *wobbling precision* en anglais.

Remarque. Le phénomène de *wobbling precision* est d'autant plus grand que la base b est grande. C'est entre autres pour cela que les petites bases ($b = 2$) sont utilisées en pratique.

Définition 2.16. Soit $x \in \mathbb{R} \setminus \mathbb{F}$ un réel. x n'a pas de représentation exacte par l'ordinateur puisque $x \notin \mathbb{F}$.

- Si $|x| > b^U(1 - b^{-t})$, alors x est plus grand que le plus grand nombre que l'on peut représenter. On parle alors d'*overflow*. En cas d'overflow, habituellement, le système interrompt le programme.
- Si $|x| < b^{L-1}$, alors x est plus petit que le plus petit nombre possible à représenter. On parle alors d'*underflow*. En cas d'underflow, habituellement, le nombre x est arrondi à 0.
- Si x ne dépasse pas des bornes de représentation, on considère a_{t+1}, le chiffre suivant a_t dans la représentation théorique à précision infinie de x. On peut alors déterminer $\widehat{x}$ de deux manières :
  - par arrondi ;

– par troncature.

Un arrondi se fait par l'application $x \mapsto \widehat{x} = \mathrm{fl}(x)$, qui *remplace* a_t par a_t^* défini par :

$$a_t^* = \begin{cases} a_t & \text{si } a_{t+1} < \frac{b}{2}, \\ a_t + 1 & \text{si } a_{t+1} \geqslant \frac{b}{2}. \end{cases}$$

Si $a_{t+1} \geqslant \frac{b}{2}$ et $a_t = b - 1$, on réitère le processus pour a_{t-1} selon a_t, etc.

Une troncature se fait par l'application $x \mapsto \widehat{x} = \mathrm{tr}(x)$, qui conserve uniquement les t premiers chiffres de la notation théorique.

Définition 2.17. On définit ϵ_x l'erreur relative de x par :

$$\epsilon_x = \left| \frac{\mathrm{fl}(x) - x}{x} \right|.$$

On définit également la *précision machine* $u := \frac{\epsilon}{2}$.

Proposition 2.18. *Soit* $x \in \mathbb{R}$ *et* $\mathrm{fl}(x) \in \mathbb{F}$. *On a* $\epsilon_x \leqslant \frac{\epsilon}{2}$.

Preuve. $\left|\mathrm{fl}(x) - x\right|$ représente la distance entre x et son approximation. x est arrondi au nombre dans $\mathbb{F}$ le plus proche, et donc $\left|\mathrm{fl}(x) - x\right| \leqslant \frac{1}{2}|x_{i+1} - x_i|$ avec $x_i \leqslant x \leqslant x_{i+1}$. On trouve alors :

$$\epsilon_x = |\rho_x| \leqslant \frac{\epsilon}{2} = u.$$

□

Remarque. La précision machine u donne l'ordre de grandeur de la meilleure approximation possible pour un arrondi sur une machine donnée.

Remarque. On trouve alors :

$$\mathrm{fl}(x) = x(1 + \rho_x).$$

2.4 Erreurs de propagation et de génération

Les erreurs de propagation sont dues au conditionnement du problème. Les erreurs de génération quant à elles sont dues à la stabilité de l'algorithme mis en place pour la résolution du problème.

2.4.1 Erreurs de propagation

Soit F une fonction admettant une dérivée d'ordre 1 en d. Par Taylor (voir théorème 1.2), on sait que :

$$F(\widehat{d}) = F(d + d\rho_d) \simeq F(d) + F'(d)d\rho_d = F(d)\left(1 + \frac{F'(d)d}{F(d)}\rho_d\right) = x(1 + \kappa(d)\rho_d).$$

On appelle donc $\kappa(d)\rho_d$ l'*erreur de propagation*.

Il est également possible que $F(\widehat{d})$ subisse une approximation. On a alors :

$$\widehat{F(\widehat{d})} = x(1 + \kappa(d)\rho_d)(1 + \rho_{F(\widehat{d})}) = x(1 + \kappa(d)\rho_d + \rho_{F(\widehat{d})} + \kappa(d)\rho_d\rho_{F(\widehat{d})}).$$

On remarque cependant que $\rho_d \rho_{F(\widehat{d})}$ est négligeable face au reste (ordre de grandeur deux fois plus grand en valeur absolue) et donc,

$$\widehat{F(\widehat{d})} \simeq x(1 + \kappa(d)\rho_d + \rho_{F(\widehat{d})}).$$

Remarque. On dit que le conditionnement est lié aux erreurs de propagation car si $\kappa(d) > 1$, alors l'erreur d'arrondi initiale ρ_d va être amplifiée, ce qui est encore plus flagrant sur un algorithme itératif.

On ne peut pas changer le conditionnement d'un problème, et donc en cas de mauvais conditionnement, il est préférable de transformer le problème en un problème équivalent mais de plus faible conditionnement.

2.4.2 Erreurs de génération

Supposons que le problème $x = F(d)$ soit décomposé en n sous-étapes $x_i = F_i(x_{i-1})$ où $x_0 = d$. Soit $\widehat{d} = d(1 + \rho_d)$ la valeur encodée en machine de d. On détermine :

$$\widehat{x_1} = d(1 + \kappa_1(d)\rho_d + \rho_i + \kappa_1(d)\rho_d\rho_i) = x(1 + \rho_{x_1}).$$

En réitérant ceci n fois, on obtient :

$$\widehat{x} = x\left(1 + \rho_n + \sum_{i=1}^{n-1} \rho_{n-i} \prod_{k=i+1}^{n} \kappa_k(d) + \prod_{k=1}^{n} \kappa_k(d)\rho_d\right).$$

On comprend de cela que ρ_n est l'erreur d'arrondi final et ne peut donc être évité. De même, $\rho_d \prod_{k=1}^{n} \kappa_k(d)$ est l'erreur de propagation et dépend du conditionnement, qui dès lors ne peut être évitée. La somme des produits sur κ_k et ρ_{n-i} quant à elle est l'erreur de génération et est dépendante de l'algorithme choisi pour résoudre le problème. Les ρ_{n-i} dépendent des fonctions F_i choisies lors des décisions d'implémentation. Ces erreurs sont donc évitables (ou du moins contrôlables par les choix d'implémentation).
Remarque. Si l'algorithme a besoin de réaliser un calcul dangereux – tel qu'une soustraction de deux nombres proches – il est préférable que cette opération soit réalisée le plus tôt possible afin de limiter l'erreur relative.

3 Résolution de systèmes linéaires

3.1 Définitions

Définition 3.1. Un système de N équations linéaires à n inconnues est donné par $(S) : \sum_{i=1}^{n} a_{ij}x_i = b_i$ pour $j \in \{1, \cdots, N\}$.

Un tel système se note également sous forme matricielle par $Ax = b$ où $A \in \mathbb{R}^{N\times n}$, $x \in \mathbb{R}^n$ et $b \in \mathbb{R}^N$. Si $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ est un n-uple satisfaisant ces N conditions, alors x est une solutions de (S).

Remarque. Dans cette section, seuls les systèmes linéaires carrés ($N = n$) seront étudiés.

Définition 3.2. Soit A une matrice carrée. Le rang de A est la dimension de l'espace vectoriel engendré par ses vecteurs colonnes.

Proposition 3.3. *La solution d'un système linéaire carré* $(S) : Ax = b$ *existe et est unique si et seulement si une des conditions équivalentes suivantes est remplie :*

1. A *est inversible ;*
2. A *est régulière (*$\det(A) \neq 0$*) ;*
3. *le rang de* A *vaut* n *;*
4. *Le système homogène* $(\bar{S}) : Ax = 0$ *admet* $x = (0, \ldots, 0)$ *pour seule solution.*

Théorème 3.4 (Formule de Cramer)**.** *Soit* $(S) : Ax = b$ *un système linéaire carré. La solution est donnée par le vecteur* x *tel que :*

$$x_j = \frac{1}{\det(A)}\Delta_j,$$

où Δ_j *est le déterminant de la matrice obtenue en remplaçant la* j*ème colonne de* A *par le vecteur* b*.*

Remarque. Le calcul du déterminant est en $\mathcal{O}(n!)$ et est donc impraticable pour des n même relativement petits (et impensables pour $n \sim 10^5$, ce qui est l'ordre de grandeur de certains systèmes de résolutions d'équations différentielles partielles). On opte donc pour une résolution numérique du système.

Définition 3.5. Une méthode de résolution est dite *directe* si elle fournit la solution exacte en un nombre fini d'étapes. Elle est dite itérative si elle fournit une approximation convergente vers la solution.

3.2 Systèmes triangulaires

Définition 3.6. Une matrice carrée A est dite *triangulaire inférieure* si $\forall j > i : A_{ij} = 0$.

Définition 3.7. Une matrice carrée A est dite *triangulaire supérieure* si A^T est triangulaire inférieure.

Définition 3.8. Un système $(S) : Ax = b$ est dit triangulaire inférieur (respectivement supérieur) si sa matrice A est triangulaire inférieure (respectivement supérieure).

Remarque. Il est fréquent de noter les matrices triangulaires inférieures L (pour *lower*) et les matrices triangulaires supérieures U (pour *upper*).

Proposition 3.9 (Résolution d'un système triangulaire inférieur)**.** *Soit* $(S) : Lx = b$ *un système triangulaire inférieur. On trouve (dans l'ordre croissant, à savoir pour* j *de* 1 *à* n*) les solutions de la manière suivante :*

$$x_j = \frac{(b_j - \sum_{k=1}^{j-1} x_k L_{jk})}{L_{kk}}. \tag{1}$$

Corollaire 3.10. *La résolution d'un système triangulaire supérieur* $(S) : Ux = b$ *se fait de la même manière mais dans l'ordre décroissant, c'est-à-dire pour* j *de* n *à* 1*. (Cela revient, théoriquement, à dire que la solution est la transposée de la solution de la transposée.)*

Définition 3.11. On définit le *flop* comme étant une unité de mesure des opérations à virgule flottante tel qu'une opération est 1 flop.

3.2.1 Analyse du coût calculatoire de la résolution d'un système triangulaire

Par l'équation (1), on voit que pour chaque x_j, il y a à opérer $(j-1)$ multiplications, $(j-1)$ additions, et une division. La résolution d'un système carré de dimension $n \times n$ nécessite donc, au total, $n + \frac{n(n-1)}{2} + \frac{n(n-1)}{2} = n^2$ opérations. On peut donc exprimer le coût calculatoire par n^2 flops.

Définition 3.12. Soient $(S_1) : A_1x = b_1$ et $(S_2) : A_2x = b_2$ deux systèmes linéaires d'équations. On dit qu'ils sont équivalents si leur ensemble de solution respectif est identique.

Proposition 3.13. *Soit* $(S) : Ax = b$ *un système. Il existe trois opérations fondamentales permettant de modifier un système en un autre système équivalent. Ces opérations sont :*

1. *échanger deux lignes ;*
2. *multiplier une ligne par une constante ;*
3. *ajouter à une ligne une combinaison linéaire des autres.*

Théorème 3.14 (Méthode de Gauss de diagonalisation de matrice)**.** *La méthode de gauss permet de diagonaliser une matrice carrée. Une fois la matrice carrée diagonalisée, la solution est immédiate.*

La méthode de diagonalisation est la suivante : tout d'abord, on ajoute *le vecteur* n *en* $n+1$*ème colonne de* A*. Ensuite, pour chaque ligne* i *(une par une), on prend l'élément* A_{ii}*. S'il est nul, on échange la* i*ème ligne avec la première ligne* $j > i$ *telle que* $A_{ji} \neq 0$*. Si un tel* j *n'existe pas, le déterminant est nul et donc la matrice n'est pas inversible, le système n'a donc pas de solution (du moins unique).Une fois qu'il est assuré que* $A_{ii} \neq 0$*, on remplace toutes les colonnes* $j \neq i$ *par* $A_j - \frac{A_{ii}}{A_{ji}} A_i$*. Une fois toutes les lignes traitées, la matrice est diagonalisée.*

3.2.2 Analyse du coût calculatoire de la méthode de Gauss

On peut limiter l'élimination de Gauss en triangulant la matrice et pas en la diagonalisant. Dans ce cas, à chaque colonne i à normaliser, il y a $(n - i)$ divisions à opérer. Le nombre de divisions est donc $\frac{n(n-1)}{2}$.

Il y a une multiplication à chaque élément normalisé (en considérant le coefficient $\frac{A_{ii}}{A_{ji}}$ calculé par la division comptée juste au-dessus). Il y a $(n-1)$ lignes, et pour la $i$ème d'entre elles, il y a $(n-i)(n-i+1)$ normalisations. Il en est de même pour les additions et soustractions. Le coût total est donc :

$$\begin{aligned}\sum_{k=1}^{n-1} 1 + 2\sum_{k=1}^{n-1}(n-k)(n-k+1) &= \frac{n(n-1)}{2} + 2\sum_{k=1}^{n-1} k(k+1) = \frac{n(n-1)}{2} + 2\sum_{k=1}^{n-1}(k^2+k)\\ &= \frac{n(n-1)}{2} + 2\sum_{k=1}^{n-1} k^2 + 2\sum_{k=1}^{n-1} k = \frac{3n(n-1)}{2} + 2\frac{n(n-1)(2n-1)}{6}\\ &= \frac{n(n-1)}{2}\big(9 + 2(2n-1)\big) = \frac{n(n-1)}{2}(4n+7) = \frac{2n^3}{3} + \frac{7n^2}{6} - \frac{4n^2}{6} - \frac{7n}{6}\\ &= \frac{2n^3}{3} + \frac{n^2}{2} - \frac{7n}{6}.\end{aligned}$$

En rajoutant à cela les n^2 flops de résolution de la matrice triangulaire, on obtient un coût total de résolution de $\frac{2n^3}{3} + \frac{3n^2}{2} + \frac{7n}{6}$.

3.3 Factorisation de matrices

3.3.1 Factorisation LU

Définition 3.15. On définit l'application δ sur un ensemble quelconque E telle que :

$$\delta : E \times E \to E : (x, y) \mapsto \delta_{x\,y} \coloneqq \begin{cases} 1 & \text{si } x = y,\\ 0 & \text{sinon.}\end{cases}$$

Définition 3.16. Soit A une matrice triangulaire (inférieure ou supérieure). Si pour tout i, on a : $A_{i\,i} = 1$, on dit que A est triangulaire *atomique*.

Proposition 3.17. *L'inverse A^{-1} d'une matrice triangulaire atomique A est également triangulaire atomique.*

Il peut être intéressant de décomposer la matrice carrée A d'un système par un produit de deux matrices L et U respectivement triangulaire inférieure et triangulaire supérieure, où $\forall i : L_{i\,i} = 1$.

Il existe plusieurs méthodes (dont certaines directes) pour procéder à cette factorisation, mais commençons par l'adaptation de la méthode de Gauss pour créer les matrices pendant la triangulation.

À l'étape de la renormalisation de la $i$ème colonne, il y a $(n - i)$ lignes à remplacer par une combinaison linéaire des autres. Cette renormalisation de la ligne peut s'écrire :

$$\forall j > i, k \geqslant i : A_{j\,k} \leftarrow A_{j\,k} - \frac{A_{j\,i}}{A_{i\,i}} A_{i\,k}.$$

On peut noter cette opération de manière matricielle par :

$$A \leftarrow M^{(i)} A,$$

où $M^{(i)}$ est la $i$ème matrice de diagonalisation et vaut :

$$M^{(i)} = \left[M^{(i)}_{k\,l}\right]_{k\,l} = \left[\delta_{k\,l} - \delta_{i\,l}\frac{A_{k\,i}}{A_{i\,i}}\right]_{k\,l}$$

Remarque. La matrice inverse de $M^{(i)}$ est la matrice $\left(M^{(i)}\right)^{-1}$ définie par :

$$\left(M^{(i)}\right)^{-1} = \left[\delta_{kl} + \delta_{il}\frac{A_{ki}}{A_{ii}}\right]_{kl} = (2I - M^{(i)}).$$

Cela est intuitivement vrai du fait que la matrice inverse représente l'application inverse. Et l'application inverse du fait de retirer α_{ki} fois la ligne i est de l'ajouter α_{ki}, où α_{ki} représente le coefficient $\frac{A_{ki}}{A_{ii}}$.

On peut nommer les matrices intermédiaires $A^{(k)}$. On a donc $A^{(0)} = A$, $A^{(1)} = M^{(1)}A$, $A^{(2)} = M_2A^{(1)} = M_2M_1A$, etc. que l'on peut généraliser en :

$$A^{(k)} = M^{(k)}M^{(k-1)}\cdots M^{(2)}M^{(1)}A = \left(\prod_{\gamma=0}^{k} M^{(k-\gamma)}\right)A.$$

La matrice finale est une matrice triangulaire supérieure (algorithme de Gauss vu plus haut). On peut donc poser $U = A^{(n)}$ si n est la dimension de la matrice. On pose alors :

$$L \coloneqq (M^{(k)}M^{(k-1)}\cdots M^{(2)}M^{(1)})^{-1} = \left(\left(M^{(1)}\right)^{-1}\left(M^{(2)}\right)^{-1}\cdots\left(M^{(k-1)}\right)^{-1}\left(M^{(k)}\right)^{-1}\right).$$

Remarque. Étant donné que les M_i sont atomiques inférieures, leur inverse l'est également. Et la *composition* de ces matrices atomiques est également une matrice atomique inférieure. Dès lors, on sait que L est atomique inférieure et peut satisfaire la factorisation.

On a donc finalement bien $A = LU$ avec U triangulaire supérieure et L triangulaire inférieure atomique.
Remarque. L'intérêt de cette décomposition LU est qu'une fois la factorisation faite, la résolution du système $(S) : Ax = b$ se fait par la résolution de deux système triangulaires, à savoir :

$$\begin{cases} Ly &= b, \\ Ux &= y. \end{cases}$$

Supposons qu'il y ait $p \in \mathbb{N}^*$ systèmes à résoudre ayant tous la même matrice caractéristique A. Cette suite de problèmes peut donc se noter :

$$Ax^{(k)} = b^{(k)}.$$

Ainsi, en appliquant p fois Gauss, asymptotiquement, le terme n^2 de Gauss devient négligeable, et on obtient : $p\mathcal{O}(n^3)$ flops. En appliquant une fois la décomposition et en utilisant ces deux matrices pour la résolution de sous-systèmes triangulaires, on obtient : $\mathcal{O}(n^3 + pn^2)$. Par définition du grand O, c'est équivalent à $\mathcal{O}(n^3)$. Asymptotiquement, la factorisation rend le programme p fois plus rapide (du moins, moins coûteux en opérations).

3.3.2 Pivots de la méthode de Gauss

Lors d'une élimination de Gauss, si le pivot A_{kk} vaut 0 à la kème étape, l'algorithme ne peut fonctionner. Numériquement, quand A_{kk} est proche de 0, l'algorithme risque de mal se dérouler.

Changement de pivot partiel Pour le changement *partiel* de pivot, il faut trouver, dans la $k$ème colonne (en dessous de $A_{k\,k}$) un élément $A_{p\,k} \neq 0$. Et ensuite, il faut échanger les lignes p et k dans la matrice A **et** dans le vecteur $b^{(k)}$[3].

Cela évite d'avoir un pivot nul. Cependant, on a dit que numériquement, un pivot *proche* de 0 était *dangereux*. Dès lors, on cherche le plus grand $A_{p\,k}$ (en valeur absolue).

Changement de pivot total Pour le changement *total*, le plus grand élément (en valeur absolue) ne doit pas être cherché dans la $k$ème colonne mais bien dans la sous-matrice $[N_{i\,j}]_{k\leqslant i\leqslant n, k\leqslant j\leqslant m}$. Une fois le plus grand élément $A_{p\,r}$ trouvé, à nouveau, on échange les lignes p et k, mais on échange également les colonnes k et r.
Remarque. Si la recherche d'un pivot est faite à toutes les étapes (et que la recherche est supposée en $\mathcal{O}(n)$), un pivot partiel amènerait une complexité en $\mathcal{O}(n^2)$ et un pivot total impliquerait une complexité en $\mathcal{O}(n^3)$ (toutes opérations confondues, tant pour l'un que pour l'autre).

Notons $P^{(i)}$ la matrice de permutations à la $i$ème étape. La matrice de permutation[4] est celle qui permet d'échanger les lignes (et éventuellement colonnes) de A pour le $i$ème pivot. On remarque que cette matrice est *auto-inverse*. En effet, $P^{(i)}\left(P^{(i)}\right)^{-1}$ représente un échange de deux lignes (et éventuellement deux colonnes) suivi de l'opération inverse. Or pour annuler une permutation, il faut la réeffectuer.

La matrice U se remplit lors de l'exécution de l'algorithme de Gauss. Dès lors, on pouvait écrire $A = LU$ s'il n'y avait pas de permutations car tous les pivots étaient non-nuls. Avec la notation $P^{(i)}$, on peut réécrire :

$$U = \left(\prod_{i=1}^{n} M^{(n+1-i)} P^{(n+1-i)}\right) A.$$

U est toujours une matrice triangulaire supérieure (par construction). Cependant, la matrice $L' \coloneqq AU^{-1}$ n'est plus triangulaire inférieure, mais en est une permutation. Dès lors, on note $P \coloneqq P^{(n)}P^{(n-1)}P^{(n-2)}\dots P^{(2)}P^{(1)}$. On définit alors la matrice L par :

$$L = PAU^{-1}.$$

On a alors l'égalité suivante :

$$PA = LU.$$

Remarque. Même s'il est assez coûteux d'effectuer les pivots à chaque étape, il est préférable lorsque le résultat doit être assez précis de tout de même les appliquer. Les deux raisons principales sont :

1. le conditionnement (erreur de propagation) car un petit pivot peut entraîner une très grande erreur relative, or le pivot est au centre de tout l'algorithme, ce qui risquerait de fausser toutes les valeurs ;
2. la stabilité (erreur de génération) car un petit pivot pourrait amener des produits à donner des résultats très bas qui pourraient être assimilés à 0 par absorption.

On sait que le déterminant d'une matrice triangulaire est le produit des éléments de sa diagonale. Donc :

$$\det(A) = \det(L)\det(U) = 1\prod_{k=1}^{n} U_{k\,k}.$$

Si on appelle D la matrice définie par :

$$D = [D_{i\,j}] = [\delta_{i\,i}U_{i\,i}].$$

[3] Et donc dans la matrice étendue utilisée par l'algorithme de Gauss.
[4] Qui est une permutation de la matrice identité.

Il existe des méthodes directes pour la factorisation LU (contrairement à la construction par l'algorithme de gauss qui est itérative). Ces méthodes doivent fixer n éléments afin de permettre la résolution. En effet, la factorisation $A = LU$ revient à résoudre un système de $n(n+1)$ variables et n^2 équations. En fixant n inconnues, on obtient un système de n^2 équations à n^2 inconnues qui possède une solution unique si ces équations forment une partie libre.

L'algorithme de Doolittle (voir juste ci-dessous) fixe les éléments de la diagonale L, mais il est possible également de fixer les éléments de la diagonale de U par exemple, c'est ce que fait la méthode de Crout.

3.3.3 Algorithme de Doolittle

En écrivant $LU = A$ avec $L_{ii} = 1$ pour $i = 1, \dots, n$, on peut résoudre toutes les équations dans l'ordre suivante :

- la 1ère ligne de U par $U_{1i} = A_{1i}$;
- la 1ère colonne de L par $L_{i1} = \frac{A_{i1}}{U_{11}}$;
- la 2ème ligne de U ;
- la 2ème colonne de L ;
- etc.

3.3.4 Algorithme de Crout

Le principe est exactement le même que pour l'algorithme de Doolittle, sauf que ce ne sont pas les L_{ii} qui sont imposés à 1 mais bien les U_{ii}. La méthode de résolution est similaire mais se fait dans l'ordre inverse :

- la 1ère colonne de L ;
- la 1ère ligne de U ;
- la 2ème colonnes de L ;
- la 2ème ligne de U ;
- etc.

3.3.5 Factorisation de Cholesky

Prenons $(S) : Ax = b$, un système linéaire tel que A est définie positive et symétrique. Ce genre de systèmes est fréquent en statistiques de simulation.

Théorème 3.18. *Soit $A \in \mathbb{R}^{n \times n}$ symétrique et définie positive. Alors il existe une unique matrice H triangulaire supérieure telle que sa diagonale est définie positive et $A = H^T H$.*

La méthode de Cholesky permet de déterminer cette matrice H symétrique en la déterminant ligne par ligne.
Remarque. Les équations sont données explicitement en écrivant $A = H^T H$.

Tout comme pour la factorisation LU, la factorisation de Cholesky permet de scinder le système (S) en deux systèmes triangulaire qui sont beaucoup plus simples à résoudre. On a effectivement :

$$Ax = b \iff H^T Hx = b \iff \begin{cases} H^T y & = b \\ Hx & = y \end{cases}.$$

Le coût numérique de cet algorithme est de $\mathcal{O}(\frac{n^3}{6})$ produits et additions, n racines carrées et $\mathcal{O}(\frac{n^2}{2})$ divisions. On peut donc dire que Cholesky se fait en $\mathcal{O}(\frac{n^3}{3})$, comparé à la méthode de Gauss qui se fait en $\mathcal{O}(\frac{2n^3}{3})$, sans compter les pivotages.
Remarque. De plus, la place de stockage nécessaire est réduite de moitié car il n'y a qu'une seule matrice à déterminer (H) et plus deux (L et U). De plus, A est supposée symétrique, et donc A et H peuvent toutes deux être stockées dans une matrice carrée $n \times n$.

3.4 Analyse des propriétés des algorithmes de factorisation matricielle