Calcul formel et numérique (in French) (Handbook of the Numerical analysis course of ULB, Computer Science Department)

Book · December 2002

CITATIONS
0

READS
2,552

3 authors, including:

Gianluca Bontempi
Université Libre de Bruxelles
334 PUBLICATIONS **10,229** CITATIONS

SEE PROFILE

SYLLABUS
Cours de Calcul Formel et Numérique
INFO-F-205

Gianluca Bontempi
Ana da Silva Soares,
Martin De Wulf

Département d'Informatique, Faculté de Sciences
Université Libre de Bruxelles, ULB
Belgique

Table des matières

Avant-propos

Ce syllabus est destiné, en premier lieu, aux étudiants de Bachelier en Sciences Informatiques de l'Université Libre de Bruxelles. Le cours de *Calcul Formel et Numérique* vise à fournir une présentation des fondations et des outils pour le calcul scientifique. Le calcul scientifique est une discipline qui consiste à développer, analyser et appliquer des méthodes qui relèvent de l'analyse, l'algèbre linéaire et le calcul différentiel aux nombreux problèmes issus de la physique, les sciences de l'ingénieur, les sciences biologiques, l'économie et la finance.

L'objectif de ce cours est de donner une première approche des algorithmes qui permettent de calculer les objets de l'analyse mathématique de base. Le coté pratique des méthodes de résolution n'est pas négligé et chaque chapitre introduit des applications à des problèmes concrets. Pour les étudiants de candidature en informatique, les exercices permettent d'aller jusqu'à l'implémentation pratique de ces algorithmes.

La matière enseignée se répartit sur 7 sujets principaux : l'introduction au calcul numérique, la gestion des erreurs numériques, la résolution de systèmes linéaires, la résolution d'équations différentielles ordinaires, l'interpolation polynomiale, le *curve fitting* et le calcul de racines pour des fonctions non-linéaires. On trouvera dans l'annexe des rappels d'algèbre linéaire.

Le but de ce cours est aussi pratique : il vise à présenter aux étudiants de candidature les outils numériques dont ils auront besoin dans la suite de leurs études. Après une présentation théorique, l'implémentation sur ordinateur et la résolution de problèmes concrets est abordée. La mise en oeuvre pratique est proposée dans le langage MATLAB. Chaque chapitre comporte des travaux pratiques, des exercices et des programmes MATLAB.

Ce manuel ne vise ni à l'orginalité, ni à être complet. Il n'a d'autre but que de fournir un supplément à l'étudiant qui suit régulièrement le cours. De nombreuses références à des ouvrages publiés sont faites tout au long de la présentation. En particulier, la structure du cours, l'organisation du syllabus et la plupart du code MATLAB sont inspirés de l'ouvrage "Méthodes numériques pour le calcul scientifique" de A. Quarteroni, R. Sacco, F. Saleri [6].

Chapitre 1

Introduction au calcul numérique

Ce chapitre se propose d'introduire les concepts de base du calcul numérique.

D'abord, nous proposons une définition formelle du calcul numérique basée sur la définition de Trefethen [7] :

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 continus qui proviennent de la modélisation des phénomènes réels.

Cette définition s'appuie sur plusieurs notions, comme celle de modèle, de problème et d'algorithme, que nous allons introduire dans la suite.

1.1 Du modèle au problème mathématique

Nous entendons par *modèle* d'un phénomène physique l'expression dans un langage formel de tout ce que l'on connaît sur ce phénomène. La relation entre la réalité, le modèle et la personne qui utilise le modèle est à la base de la notion de modélisation.

Toutes les sciences (mathématiques, humaines et appliquées) sont caractérisées par une activité de définition, analyse et utilisation de modèles. A titre d'exemple, citons les modèles utilisés en physique, économie, finance, médecine, sociologie et toutes les sciences de l'ingénieur.

En termes généraux, l'utilisation d'un modèle pour la résolution d'un problème pratique passe à travers la résolution d'un problème mathématique (Figure 1.1). Le modèles mathématiques continus prennent typiquement la forme d'un ensemble

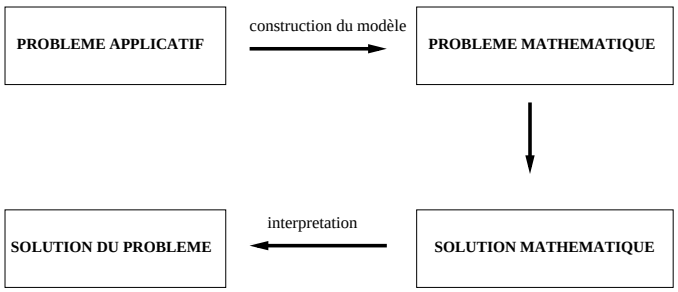


FIGURE 1.1 – Du problème réel à la solution via le problème mathématique

d'équations (algébriques ou différentielles) et/ou inéquations avec paramètres (connus et/ou inconnus). Le problème mathématique qui en découle peut être de nature qualitative ou quantitative. Plus précisément, nous entendons par

Problème qualitatif, un problème qui concerne le comportement des solutions, par exemple leur stabilité ou leur comportement asymptotique.

Problème quantitatif, le problème qui demande la calcul d'une solution (fonction ou variable).

Le cours traitera exclusivement des problèmes quantitatifs.

En particulier, nous définissons un problème mathématique quantitatif comme suit :

Définition 1 (Problème mathématique)**.** *Un problème mathématique est une relation fonctionnelle F entre un ensemble de données d et une solution x.*

On peut distinguer entre deux formes de problème :

Forme explicite :

$$x = F(d) \tag{1.1}$$

Forme implicite :

$$F(x, d) = 0 \tag{1.2}$$

Selon la nature du problème, la solution x et les données d peuvent être représentées par des matrices, des nombres réels ou des fonctions. Voici quelques exemples de problèmes mathématiques :

1. Trouver la racine carrée de d : $x = \sqrt{d}$. Ceci est un problème sous forme explicite.
2. Résoudre l'équation différentielle

   $$\varphi''(t) = -\omega^2 sin(\varphi(t)) \qquad \varphi(0) = \varphi_0, \varphi'(0) = 0 \tag{1.3}$$

   qui décrit le mouvement d'un pendule où φ est le déplacement angulaire, $\omega^2 = g/l$, et g est l'accélération de la gravité. Ceci est un problème sous forme implicite.
3. Trouver la plus grande racine x de l'équation algébrique

   $$ax^2 + bx + c = 0 \tag{1.4}$$

   où les données sont représentées par le vecteur $d = [a, b, c]$. Ceci est un problème sous forme implicite.
4. Trouver la solution $x = [x1, x2, x3]$ du système

   $$\begin{cases} 2x_1 + 4x_2 - 6x_3 = -4 \\ x_1 + 5x_2 + 3x_3 = 10 \\ x_1 + 3x_2 + 2x_3 = 5 \end{cases} \tag{1.5}$$

   où les données sont représentées par la matrice

   $$d = \begin{bmatrix} 2 & 4 & -6 & -4 \\ 1 & 5 & 3 & 10 \\ 1 & 3 & 2 & 5 \end{bmatrix}$$

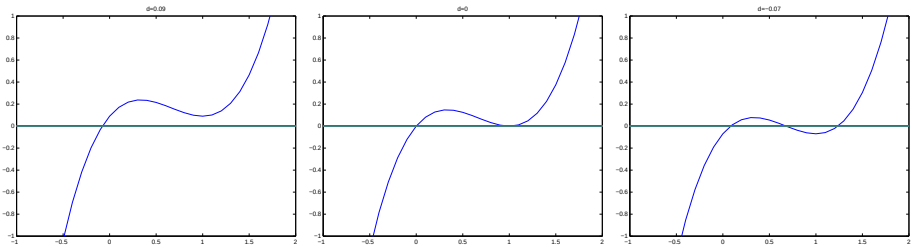


FIGURE 1.2 – Le polynôme $p(z, d)$ pour trois valeurs de d prés de zéro ($d = -0.07, 0, 0.09$).

1.1.1 Problèmes bien/mal posés

La notion de problème bien/mal posé a été introduite pour la première fois par Hadamard [3]. Afin de faciliter la présentation dans l'analyse qui suit, nous nous limiterons à considérer le cas des problèmes explicites.

Définition 2 (Problème bien posé)**.** *Le problème mathématique $x = F(d)$ est bien posé si la solution x*
- *existe,*
- *est unique,*
- *dépend* continûment *des données d.*

Autrement le problème est dit mal posé.

Nous utiliserons indifféremment les termes bien posé et *stable* et nous ne considérerons dans la suite que des problèmes bien posés.

1.1.1.1 Exemples de problèmes mal posés

Un cas simple de problème mal posé est $x = (d > 0.5)$ où d est un nombre réel et $x \in \{0, 1\}$. Ceci résulte du fait que la fonction $F(d) = (d > 0.5)$ est discontinue pour $d = 0.5$

Un autre exemple de problème mal posé est le suivant : *Trouver le nombre x de racines réelles d'un polynôme.* Considérons le polynôme

$$p(z, d) = d + z - 2z^2 + z^3$$

Le nombre de racines réelles x varie de façon discontinue pour d autour du zéro (Figure 1.2)

$$x = 1 \text{ si } d > 0$$
$$x = 2 \text{ si } d = 0$$
$$x = 3 \text{ si } d < 0$$

On retrouve des exemples de problèmes mal posés aussi dans le cas de *problèmes inverses*, qui consistent à trouver d étant donné x. Considérons un problème direct où deux valeurs différentes de d produisent le même x (Figure 1.3) : le problème inverse $d = F^{-1}(x)$ est par conséquent mal posé. Des exemples de problèmes inverses mal posés sont les suivants :
- trouver la position d'un obstacle à partir de l'information radar ;
- trouver une information sur une scène 3D à partir d'une image 2D (par exemple dans le procédé de la tomographie) ;

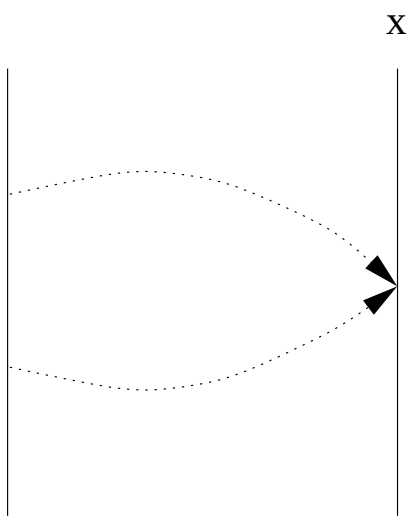


FIGURE 1.3 – Problème inverse $d = F^{-1}(x)$ mal posé

- trouver sur une barre métallique la valeur de la température en sachant la température en une autre position ;
- trouver la solution d'équations différentielles partielles.

1.1.2 Conditionnement d'un problème

Le conditionnement d'un problème de la forme (1.1) mesure la sensibilité de la solution x du problème aux changements des données d.

Définition 3 (Conditionnement Relatif)**.** *Soient δd une perturbation admissible des données et δx la modification induite sur la solution du problème $x = F(d)$. On appelle conditionnement relatif de ce problème la quantité*

$$\kappa(d) = \sup_{\delta d \in D} \frac{\|\delta x\|/\|x\|}{\|\delta d\|/\|d\|} \tag{1.6}$$

où sup *dénote la borne supérieure et D est un voisinage de l'origine.*

Si $F(\cdot)$ est différentiable en d et si nous notons par $F'(d) \approx \frac{\delta x}{\delta d}$ sa dérivée, on obtient

$$\kappa(d) \approx \frac{\|\delta x\|}{\|\delta d\|}\frac{\|d\|}{\|x\|} = \|F'(d)\|\frac{\|d\|}{\|x\|} = \|F'(d)\|\frac{\|d\|}{\|F(d)\|} \tag{1.7}$$

où le symbole $\|\cdot\|$ désigne la norme matricielle (Section A.11).

Quand $d = 0$ ou $x = 0$, il est nécessaire d'introduire le *conditionnement absolu*, défini par

$$\kappa_{abs}(d) = \sup_{\delta d \in D} \frac{\|\delta x\|}{\|\delta d\|} \tag{1.8}$$

Un problème peut avoir un petit conditionnement $\kappa(d)$ pour certaines valeurs de d et un grand conditionnement pour d'autres valeurs. Si $\kappa(d)$ est grand pour toute donnée admissible d, le problème est dit *mal conditionné*.

En outre, il est important de remarquer que

- un problème mal posé est forcément aussi un problème mal conditionné, mais que mal conditionné ne signifie pas forcement mal posé ;

– le fait d'être bien conditionné est une propriété du problème qui est indépendante de la méthode choisie pour le résoudre.

Exemples de conditionnement

– Considérons le problème $x = F(d) = d - 1$. En vertu de la formule (1.7), le conditionnement relatif du problème est

$$\kappa(d) \approx \|F'(d)\| \frac{\|d\|}{\|F(d)\|} = \frac{\|d\|}{\|d-1\|}$$

Il en résulte que $\kappa(d) > 100$ pour $0.999 < d < 1.001$. Donc, le problème est mal conditionné pour les valeurs de d autour de 1.

– Considérons le problème $x = F(d) = d^{1000}$. Alors

$$\kappa(d) \approx \|F'(d)\| \frac{\|d\|}{\|F(d)\|} = 1000 \|d^{999}\| \frac{\|d\|}{\|d^{1000}\|} = 1000$$

Le problème est mal conditionné pour toute valeur de d.

•

1.1.2.1 Le conditionnement d'un problème composé

Considérons un problème décrit par la fonction composée

$$x = F(d) = F_2(F_1(d)) \tag{1.9}$$

où $F(\cdot)$, $F_1(\cdot)$ et $F_2(\cdot)$ sont dérivables.

Calculons son conditionnement

$$\kappa(d) = \left\| \frac{F'(d)d}{F(d)} \right\|. \tag{1.10}$$

On peut réécrire (1.9) comme

$$x_1 = F_1(d), \qquad x = F_2(x_1) \tag{1.11}$$

où

$$\kappa_1(d) = \left\| \frac{F_1'(d)d}{F_1(d)} \right\|, \qquad \kappa_2(d) = \left\| \frac{F_2'(x_1)x_1}{F_2(x_1)} \right\|. \tag{1.12}$$

Des formules (1.11) et (1.12) il vient que

$$\kappa_1(d)\kappa_2(d) = \left\| \frac{F_1'(d)d}{F_1(d)} \frac{F_2'(x_1)x_1}{F_2(x_1)} \right\| = \left\| \frac{F_1'(d)F_2'(x_1)d}{F_2(x_1)} \right\| = \kappa(d). \tag{1.13}$$

Ceci montre que le conditionnement d'un modèle composé est égal au produit des conditionnements des modèles composants.

1.2 La résolution d'un problème mathématique

Les propriétés présentées jusqu'ici dépendent de la forme du problème mathématique et sont indépendantes de la méthode de résolution adoptée. Par la suite, nous allons nous pencher sur les méthodes de résolution d'un problème mathématique quantitatif.

Deux approches peuvent être envisagées :

Résolution symbolique : ceci utilise les propriétés analytiques et mathématiques du problème pour en dériver la solution x. Par exemple dans le problème élémentaire de la recherche des racines d'une équation du second ordre

$$d_1 x^2 + d_2 x + d_3 = 0, \tag{1.14}$$

où $d = [d_1, d_2, d_3]$, la solution analytique est

$$x = \frac{-d_2 \pm \sqrt{d_2^2 - 4d_1 d_3}}{2d_1}.$$

Aussi, dans le cas de l'équation différentielle ordinaire du premier ordre

$$x' = -x, \qquad x(0) = 1, \tag{1.15}$$

la solution analytique est $x(t) = e^{-t}$.

Malheureusement une solution analytique n'est pas calculable pour tous le problèmes. Par exemple dans le problème d'intégration numérique suivant

$$\int_0^\pi \sqrt{1 + \cos^2(x)} dx \tag{1.16}$$

aucune solution analytique n'est disponible.

Résolution numérique : ceci utilise une méthode numérique pour déterminer la solution x pour une valeur d donnée. Par exemple, dans le problème (1.14), pour $d = [0.1, 1.2, 0.3]$ une méthode numérique de résolution donne la solution $x = -11.74$. Dans le problème (1.15), une méthode numérique fournit comme solution une série de valeurs $x(t_i)$.

Une méthode numérique présente des bénéfices aussi bien que des inconvénients par rapport à une solution analytique. Les avantages tiennent (i) au fait qu'une solution numérique peut être obtenue aussi lorsqu'aucune solution analytique n'est disponible et (ii) au fait que la décomposition d'une méthode numérique en une longue série d'opérations arithmétiques élémentaires s'avère être facilement gérable par un ordinateur. A son détriment, il faut mentionner que l'analyse et l'étude d'une solution numérique sont typiquement plus coûteuses.

1.3 Algorithmes et problèmes numériques

Avant de présenter une notion formelle, nous jugeons utile d' introduire d'abord une notion intuitive d'algorithme.

1.3.1 Définition qualitative d'algorithme

Définition 4. *Un algorithme est un texte fini qui spécifie l'exécution d'une série finie d'opérations élémentaires, conçue pour résoudre des problèmes d'une classe ou d'un type particulier.*

Les propriétés d'un algorithme sont :
- un algorithme décrit la résolution d'un problème en utilisant un niveau d'abstraction qui dépend du type et du nombre d'opérations élémentaires ;
- un algorithme est destiné à résoudre une classe de problèmes et pas simplement une instance isolée d'un problème ;
- un algorithme est exécuté par un processeur. Le processeur peut être un être humain, une mécanique ou plus généralement un ordinateur ;
- l'exécution d'un algorithme procède par étapes ;

Ordre	Classe de complexité	exemple de $C(p)$
$O(1)$	constante	$c \in \mathbb{R}^+$
$O(\log p)$	logarithmique	$c \log p$
$O(p)$	linéaire	$c_1 p + c_0$
$O(p^2)$	quadratique	$c_2 p^2 + c_1 p + c_0$
$O(p^3)$	cubique	$c_3 p^3 + c_2 p^2 + \ldots$
$\vdots$	$\vdots$	$\vdots$
$O(p^m)$, $m \in \mathbb{N}$	polynomiale	$c_m p^m + c_{m-1} p^{m-1} + \ldots$
$O(c^p)$	exponentielle	c^{dp}+polynôme(p)
$O(p!)$	factorielle	$cp!$

TABLE 1.1 – Classes de complexité

- un algorithme peut être décomposé en plusieurs sous-algorithmes ;
- la description d'un algorithme est nécessairement de longueur finie ;
- on dit qu'un algorithme se *termine* s'il s'achève après un nombre fini d'étapes et s'il renvoie un résultat à chaque exécution ;
- un algorithme est dit *déterministe* si à chaque moment de l'exécution, la prochaine étape est déterminée de façon unique. Si plusieurs alternatives existent, l'algorithme est dit *non-déterministe*. Un algorithme non-déterministe est dit *stochastique* si la probabilité des différentes alternatives est décrite par une distribution de probabilité.

Pour une introduction plus complète et rigoureuse de la notion d'algorithme nous renvoyons le lecteur aux cours d'algorithmique et programmation.

1.3.2 Complexité d'un algorithme

Plusieurs mesures peuvent être adoptées pour caractériser la complexité d'un algorithme : le nombre d'étapes, le temps d'exécution, l'occupation de la mémoire ou des mesures dépendantes de l'architecture du processeur.

Toutefois, dans certains cas l'utilisateur de l'algorithme est intéressé à avoir une idée qualitative du comportement de l'algorithme plutôt que le nombre exact d'opérations. Pour ce faire, la notion d'ordre de complexité (ou complexité asymptotique) à été introduite.

Définition 5 (Ordre de complexité)**.** *Si la complexité d'un algorithme dépend d'un paramètre p (représentant la taille du problème), la complexité $C(p)$ d'un algorithme est dite d'ordre $f(p)$ s'il existe deux constantes a et b telles que*

$$C(p) \leq bf(p) \qquad \textit{pour tout } p \geq a. \tag{1.17}$$

Ceci est indiqué par la notation conventionnelle

$$C(p) = O(f(p)). \tag{1.18}$$

Les algorithmes peuvent être répartis en classes de complexité. Les classes de complexité les plus importantes sont mentionnées à la Table 1.1.

Les éléments à la Table 1.1 suivent un ordre de complexité asymptotique croissante : par exemple, un algorithme ayant une complexité cubique, pour un grand p, demande un effort de calcul plus important qu'un algorithme de complexité logarithmique.

En général, si deux algorithmes ont la même complexité asymptotique, ils demandent le même effort de calcul pour une grande taille du problème.

Exemple Nous verrons dans la suite que l'algorithme de Gauss pour la résolution d'un système linéaire d'ordre n demande un total de $2n^3/3 + 3n^2/2 - 7n/6$ opérations élémentaires. Par conséquent, nous pouvons dire que cet algorithme a une complexité cubique, ou d'ordre 3.

•

1.3.2.1 Complexité d'un problème

Notons que même si nous pouvons définir la complexité d'un algorithme, la notion de complexité d'un problème reste floue, car plusieurs algorithmes de complexité différente peuvent être utilisés pour résoudre le même problème.

En général, nous définissons la *complexité d'un problème* comme étant la complexité de l'algorithme qui a la complexité la plus petite parmi ceux qui résolvent le problème.

1.3.3 Algorithmes numériques

Cette section traitera les algorithmes numériques et leurs propriétés théoriques.

Définition 6 (Algorithme numérique)**.** *Étant donné un problème bien posé $F(x, d) = 0$, nous définissons l'algorithme numérique pour la résolution du problème F par la suite de problèmes approchés*

$$F_1(x^{(1)}, d^{(1)}), F_2(x^{(2)}, d^{(2)}), \ldots, F_n(x^{(n)}, d^{(n)}) \tag{1.19}$$

dépendant d'un paramètre n, où $x^{(n)}$ est la solution du sous-problème F_n.

L'idée sous-jacente à la décomposition en sous-problèmes est que la résolution des F_n est plus simple que la résolution de F. Ceci rend possible l'exécution d'un algorithme par une machine.

Différents algorithmes peuvent résoudre le même problème numérique. Il est alors nécessaire de choisir l'algorithme qui présente les meilleurs propriétés numériques et de complexité.

Les propriétés souhaitées d'un algorithme numérique sont
- la consistance,
- la stabilité,
- la convergence

Voyons ces propriétés en détail.

1.3.4 Consistance

Définition 7. *Un algorithme $F_n(x^{(n)}, d^{(n)})$ est dit consistant si*

$$\lim_{n\to\infty} F_n(x, d^{(n)}) = F(x, d) = 0,$$

c.-à-d., si la solution exacte x du problème est une solution de $F_n(x^{(n)}, d^{(n)}) = 0$ pour $n \to \infty$.

En d'autres termes, un algorithme est consistant si à partir d'une certaine étape le sous-problème approché F_n a la solution x parmi ses solutions.

Définition 8. *Un algorithme $F_n(x^{(n)}, d^{(n)})$ est dit fortement consistant si*

$$F_n(x, d^{(n)}) = 0$$

pour tout n.

Exemple Considérons le problème mathématique $F(x, d) = 0$ qui correspond à la résolution d'un système linéaire d'ordre 3 :

$$\begin{cases} 2x_1 + 4x_2 - 6x_3 = -4 \\ x_1 + 5x_2 + 3x_3 = 10 \\ x_1 + 3x_2 + 2x_3 = 5 \end{cases} \tag{1.20}$$

Les données sont représentées par

$$d = \begin{bmatrix} 2 & 4 & -6 & -4 \\ 1 & 5 & 3 & 10 \\ 1 & 3 & 2 & 5 \end{bmatrix}$$

et la solution est $x = [-3, 2, 1]$.

Considérons un algorithme de résolution numérique en 2 étapes où la première étape est représentée par

$$F_1(x^{((1))}, d^{(1)}) = 0 \Leftrightarrow \begin{cases} 2x_1 + 4x_2 - 6x_3 = -4 \\ 3x_2 + 6x_3 = 12 \\ x_2 + 5x_3 = 7 \end{cases}$$

où

$$d_1 = \begin{bmatrix} 2 & 4 & -6 & -4 \\ 0 & 3 & 6 & 12 \\ 0 & 1 & 5 & 7 \end{bmatrix}$$

Soit $x^{(1)} = x = [-3, 2, 1]$ la solution de la première étape. Notons que, dans cet exemple, nous avons utilisé la notion $x^{((n))}$ afin de ne pas confondre la solution (vecteur) de la $n$ème étape de l'algorithme et le $n$ème composant (scalaire) de la solution x du problème.

Soit la deuxième étape représentée par

$$F_2(x^{(2)}, d^{(2)}) = 0 \Leftrightarrow \begin{cases} 2x_1 + 4x_2 - 6x_3 = -4 \\ 3x_2 + 6x_3 = 12 \\ 3x_3 = 3 \end{cases}$$

avec

$$d^{(2)} = \begin{bmatrix} 2 & 4 & -6 & -4 \\ 0 & 3 & 6 & 12 \\ 0 & 0 & 3 & 3 \end{bmatrix}$$

et $x^{(2)} = x = [-3, 2, 1]$ sa solution.

Pour toute valeur de n $(n = 1, 2)$, $F_n(x, d^{(n)}) = 0$. Donc cet algorithme est fortement consistant.

•

1.3.5 Stabilité d'un algorithme

Soit $x^{(n)}$ la suite des solutions approchées produite par l'algorithme pour les données $d^{(n)}$ et $\bar{x}^{(n)} = x^{(n)} + \delta x^{(n)}$ la suite des solutions approchées produite par l'algorithme suite aux perturbations des données $\{\delta d^{(i)}\}$, $i = 1, \ldots, n$.

Définition 9 (Stabilité). *Un algorithme numérique est dit stable (ou bien posé) si*

1. *il existe pour tout n une solution unique $x^{(n)}$*

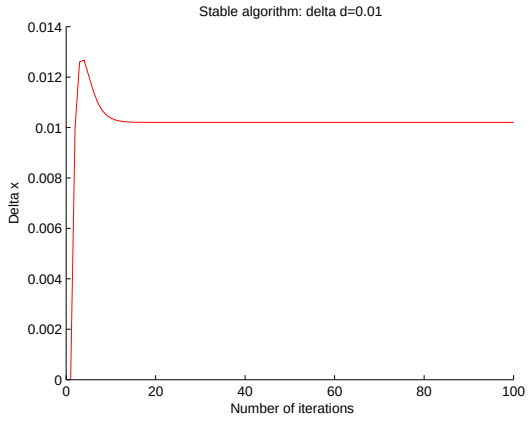


FIGURE 1.4 – Algorithme stable : perturbation de la solution numérique $x^{(n)}$ vs. nombre d'itérations pour une perturbation $\delta d^{(n)} = 1e-2$ de la donnée d. Fichier MATLAB `s_stable2.m`

2. *il existe un* $\eta > 0$ *et un* $\epsilon > 0$ *tel que*

$$\|\delta d^{(i)}\| \leq \eta \Rightarrow \|\bar{x}^{(n)} - x^{(n)}\| \leq \epsilon$$

En d'autres termes, un algorithme est stable si des petites perturbations des données du sous-problème F_n entraînent des petites perturbations de sa solution $x^{(n)}$.

Définition 10 (Conditionnement asymptotique relatif)**.** *Nous appelons conditionnement asymptotique relatif de la méthode numérique la quantité*

$$\kappa^{num}(d^{(n)}) = \lim_{k\to\infty} \sup_{n\geq k} \kappa_n(d^{(n)}), \tag{1.21}$$

où

$$\kappa_n(d^{(n)}) = \sup_{\delta d^{(n)} \in D_n} \frac{\|\delta x^{(n)}\|/\|x^{(n)}\|}{\|\delta d^{(n)}\|/\|d^{(n)}\|}. \tag{1.22}$$

L'algorithme numérique est dit bien conditionné si κ^{num} est *petit* pour toutes données $d^{(n)}$ et mal conditionné sinon.

Exemple Considérons le problème mathématique

$$2x = 1$$

et les deux algorithmes itératives suivants

1.
$$x^{(n)} = d^{(n)}x^{(n-1)} + \frac{1}{4}, \qquad d^{(n)} = \frac{1}{2}$$

2.
$$x^{(n)} = d^{(n)}x^{(n-1)} - \frac{1}{2}, \qquad d^{(n)} = 2$$

Tous les deux sont consistants mais uniquement le premier algorithme est stable. Dans le premier cas uniquement, un petit changement $\delta d^{(n)}$ des données entraîne un changement borné de la solution $x^{(n)}$ pour tous n (Figure 1.4).

•

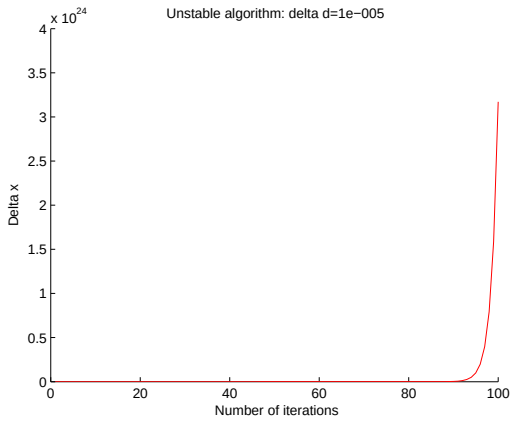


FIGURE 1.5 – Algorithme instable : perturbation de la solution numérique $x^{(n)}$ vs. nombre d'itérations pour une perturbation $\delta d^{(n)} = 1e-5$ de la donnée d. (Script MATLAB `s_unstable2.m`).

1.3.6 Convergence d'un algorithme

Soit $\bar{x}^{(n)} = x^{(n)} + \delta x^{(n)}$ la suite des solutions approchées produite par l'algorithme suite aux perturbations des données $\{\delta d^{(i)}\}$, $i = 1, \ldots, n$.

Définition 11 (Convergence)**.** *Un algorithme est convergent si*

$$\lim_{n\to\infty, \delta d^{(i)}\to 0} \bar{x}^{(n)} = \lim_{n\to\infty, \delta d^{(i)}\to 0} x^{(n)} + \delta x^{(n)} = x$$

ou, en d'autres termes, si pour chaque $\epsilon > 0$ il existe $\eta > 0$ et un n_0 tels que pour tous $n > n_0$

$$\|\delta d^{(i)}\| \leq \eta \Rightarrow \|\bar{x}^{(n)} - x\| \leq \epsilon$$

Les concepts de stabilité et convergence sont fortement liés dans le cas d'algorithmes consistants. Les méthodes numériques que nous allons considérer dans ce livre satisfont les conditions du théorème de Lax-Richtmyer (ou théorème d'équivalence) selon lequel *pour un algorithme numérique consistant, la stabilité est équivalente à la convergence.*

Remarque Les notions présentées jusqu'ici peuvent paraître au lecteur comme purement théoriques et sans lien évident avec les problèmes pratiques. Le chapitre prochain illustrera pourquoi une analyse théorique est nécessaire et comment les propriétés énoncées ici justifient le comportement, parfois inattendu, des algorithmes numériques.

•

1.4 Exercices

1. Montrer que l'évaluation de $F(d) = \cos(d)$ est bien conditionnée près de $d = 0$.
2. Que peut on dire du conditionnement de la fonction $F(d) = d-1$ pour $0.999 < d < 1.001$? Est-il grand ou petit ?
3. Calculer le conditionnement $\kappa(d)$ des expressions suivantes :
   – $x - a^d = 0, a > 0$;

– $d - x + 1 = 0$.

4. Etudier le conditionnement de la formule $x_\pm = -p \pm \sqrt{p^2 + q}$ donnant la solution d'une équation du second degré $x^2 + 2px - q$ par rapport aux perturbations des paramètres p et q séparément.
   Rappel : $(f^m)' = mf^{m-1}f'$
5. Que peut-on dire du conditionnement de la fonction $F(d) = (d-1)^2$ pour d proche de 1 ?

Chapitre 2

Analyse d'erreurs

Chaque analyse numérique doit se confronter avec une certaine dose d'erreurs. Dans ce chapitre nous répondrons aux questions suivantes

- Qu'est-ce qu'une erreur ?
- D'où viennent les erreurs ?
- Quelles conséquences ont-elles ?
- Comment analyser leurs effets ?

2.1 Erreurs absolue et relative

Un *nombre approché* $\hat{x}$ est un nombre légèrement différent du nombre exact x et qui dans le calcul remplace ce dernier. Si l'on sait que $\hat{x} < x$, $\hat{x}$ est dit valeur approchée du nombre x *par défaut* ; si $\hat{x} > x$, $\hat{x}$ est une valeur approchée *par excès*.

Exemple Soit $x = \sqrt{2}$. Le nombre $\hat{x} = 1.41$ est une valeur approchée par défaut, alors que le nombre $\hat{x} = 1.42$ est une valeur approchée par excès.

•

Si $\hat{x}$ est une valeur approchée de x on note $\hat{x} \approx x$.

Définition 12 (Erreur absolue)**.** *On appelle* erreur absolue δ_x *d'un nombre approché* $\hat{x}$ *la valeur absolue de la différence entre le nombre exact* x *correspondant et le nombre approché donné*

$$\delta_x = |\hat{x} - x|. \tag{2.1}$$

Définition 13 (Écart relatif)**.** *L'écart relatif d'un nombre approché* $\hat{x}$ *est le rapport*

$$\rho_x = \frac{\hat{x} - x}{x}. \tag{2.2}$$

Cette relation peut aussi être écrite sous la forme

$$\hat{x} = x(1 + \rho_x). \tag{2.3}$$

Définition 14 (Erreur relative)**.** *L'erreur relative* ε_x *d'un nombre approché* $\hat{x}$ *est la valeur absolue de l'écart relatif, c.-à-d. le rapport de l'erreur absolue* δ_x *de ce nombre et du module du nombre exact correspondant (si* $x \neq 0$*)*

$$\varepsilon_x = |\rho_x| = \left|\frac{\hat{x} - x}{x}\right| = \frac{\delta_x}{|x|} \tag{2.4}$$

L'erreur relative fournit une information plus pertinente sur la grandeur réelle de l'erreur. Cependant, elle n'est définie que pour $x \neq 0$.

Définition 15 (Borne supérieure relative.)**.** *La borne supérieure d'erreur relative u d'un nombre approché $\hat{x}$ donné est un nombre quelconque supérieur ou égal à l'erreur relative de ce nombre*

$$\varepsilon_x = |\rho_x| \leq u \tag{2.5}$$

2.2 Sources d'erreurs

Les erreurs commises dans les problèmes mathématiques peuvent être en principe classées en cinq catégories :

Erreurs de modélisation : ces erreurs sont dues à la façon même dont est posé le problème. Puisque les modèles mathématiques sont plus ou moins idéalisés, ceci donne lieu à plusieurs erreurs. Un exemple est l'erreur du modèle (1.3) du pendule qui ne tient pas en considération la force de friction.

Erreurs de mesure : ces erreurs sont dues à la présence dans le modèle mathématique de paramètres numériques dont les valeurs ne peuvent être déterminées qu'approximativement suite à des mesures expérimentales. Telles sont toutes les constantes physiques.

Erreurs d'approximation ou de troncature : ces sont les erreurs associées aux processus infinis en analyse mathématique (par exemple les séries numériques [2]). Un processus infini ne se terminant pas en général en un nombre fini de pas

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

on est obligé d'y mettre fin à un certain terme de la suite en le considérant comme une approximation de la solution cherchée.

Un autre exemple d'erreurs de troncature sont les erreurs dues à la discrétisation de processus continus, comme dans les séries temporelles continues.

Erreurs d'arrondi : ce sont les erreurs associées au système de numération. Elles sont dues au fait qu'un ordinateur ne peut prendre en considération qu'un nombre fini de chiffres. Notons que même des nombres rationnels notés dans le système décimal peuvent comporter à droite de la virgule une infinité de chiffres (un nombre décimal périodique, par exemple). L'*erreur d'arrondi* sera le sujet principal du reste du chapitre.

Erreurs de propagation et génération : ces sont les erreurs qui apparaissent dans le résultat d'une opération comme conséquence des erreurs des opérandes. Ces erreurs tiennent au fait qu'en effectuant des calculs sur des nombres approchés, les erreurs des données de départ sont propagées sur le résultat.

Les deux premiers types d'erreur sont regroupés sous le nom d' **erreurs de modélisation** tandis que les trois derniers sont appelés **erreurs numériques**.

Dans ce qui suit nous allons étudier en détail les erreurs numériques.

2.3 Représentation des nombres en machine

Un ordinateur ne peut représenter qu'un sous-ensemble fini de l'ensemble des nombres réels. Par conséquent, toute opération d'un ordinateur est entachée par des erreurs d'arrondi.

Les notations machine les plus adoptées pour représenter les nombres réels sur un ordinateur sont la notation *à virgule fixe* et la notation *à virgule flottante*.

2.3.1 Notation à virgule fixe

Soit x un nombre réel non nul. Sa représentation en *virgule fixe* est

$$\{[x_n x_{n-1} \dots x_1 x_0, x_{-1} x_{-2} \dots x_{-m}], b, s\} \tag{2.6}$$

où

- $b \in \mathbb{N}$, $b \geq 2$ est appelée la *base*,
- $s \in \{0, 1\}$ est appelé le *signe*,
- $x_i \in \mathbb{N}$, $0 \leq x_i < b$, $i = -m, \dots, n$ sont les *symboles*,
- m désigne le nombre de chiffres après la virgule,
- $n + 1$ est le nombre de chiffres avant la virgule,

et la valeur $x \in \mathbb{R}$ codée par la notation (2.6) est

$$x = (-1)^s \left(\sum_{k=-m}^{n} x_k b^k \right). \tag{2.7}$$

Il s'ensuit que pour $s = 0$, x est un nombre positif, alors que pour $s = 1$, x est un nombre négatif.

Exemples

Soient $b = 10$, $n = 3$, $m = 6$, $s = 0$. Alors

- l'écriture à virgule fixe $[0030, 421000]$ désigne le réel

$$x = 3 \cdot 10^1 + 0 \cdot 10^0 + 4 \cdot 10^{-1} + 2 \cdot 10^{-2} + 1 \cdot 10^{-3} = 30.421$$

- l'écriture $[0000, 043700]$ désigne le réel $x = 0.0437$.

Soient $b = 16$, $n = 3$, $m = 6$, $s = 0$. Alors

- l'écriture $[0030, 421000]$ désigne le réel

$$3 \cdot 16^1 + 0 \cdot 16^0 + 4 \cdot 16^{-1} + 2 \cdot 16^{-2} + 1 \cdot 16^{-3} = 48.258$$

- l'écriture $[0000, 043700]$ désigne le réel $x = 0.0165$.

•

Il est important de remarquer la différence qui existe entre symboles et nombres. Par exemple en utilisant la notation à virgule fixe (formule (2.6)) l'écriture $[111, 101]$ correspond au nombre 111.101 si et seulement si $b = 10$, alors que elle correspond au nombre 7.625 pour $b = 2$ et au nombre 13.3704 pour $b = 3$.

Une autre particularité concerne le nombre de chiffres significatifs nécessaire pour représenter un nombre x. Par exemple, le nombre réel $x = 0.1$ a une représentation finie en base $b = 10$ tandis qu'il demande un nombre infini de chiffres significatifs pour la base $b = 2$.

Bien que, d'un point de vue théorique, toutes les bases soient équivalentes, les ordinateurs emploient souvent trois bases :

- $b = 10$: ceci est la base du système décimal. Les 10 symboles sont $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. Cette base est normalement utilisée pour l'affichage des résultats.
- $b = 2$: ceci est la base du système binaire. Les symboles sont $0, 1$ (bits). Cette base est habituellement utilisée pour stocker les nombres et effectuer les calculs.
- $b = 16$: ceci est la base du système hexadécimal. Les 16 symboles sont

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$$

Cette base permet une représentation compacte des nombres binaires.

Les propriétés de la notation en virgule fixe peuvent être résumées ainsi :
- Les nombres en virgule fixe sont équirépartis le long de la droite réelle.
- L'écart entre deux nombres consécutifs réels qui peuvent être représentés en notation à virgule fixe est égal à b^{-m}.
- L'utilisation de la virgule fixe limite considérablement les valeurs maximales et minimales des nombres représentés par l'ordinateur.
- La chaîne de caractères nécessaire à la représentation d'un même nombre est autant plus longue que la base est petite.

2.3.2 Notation à virgule flottante

Étant donné un nombre réel non nul x, sa représentation à *virgule flottante* est

$$\{[a_1 a_2 \dots a_t], e, b, s\} \tag{2.8}$$

où
- $b \in \mathbb{N}$, $b \geq 2$ est appelée la base,
- $e \in \mathbb{Z}$, $L \leq e \leq U$ est appelé l'exposant,
- $s \in \{0, 1\}$ est appelé le signe,
- t est le nombre de chiffres significatifs,
- $a_i \in \mathbb{N}$, $0 < a_1 < b$, $0 \leq a_i < b$, $i = 2, \dots, t$
- la quantité

$$m = m(x) = \sum_{i=1}^{t} a_i b^{t-i} \tag{2.9}$$

est appelée *mantisse*.

Nous ne considérons que le cas normalisé, c.-à-d.

$$a_1 > 0 \tag{2.10}$$

qui implique la relation suivante

$$b^{t-1} \leq m \leq b^t - 1 \tag{2.11}$$

La notation (2.8) est utilisée pour encoder le nombre réel

$$x = (-1)^s b^e \sum_{i=1}^{t} a_i b^{-i} = (-1)^s m b^{e-t} \tag{2.12}$$

Exemples
- La notation à virgule flottante $\{[3, 4], e = 1, b = 10, s = 0\}$ désigne le réel

$$x = 10^1 (3 \cdot 10^{-1} + 4 \cdot 10^{-2}) = 3.4$$

- La notation $\{[3, 4], e = -1, b = 10, s = 1\}$ désigne le réel $x = -0.034$
- La notation $\{[3, 4], e = -1, b = 16, s = 1\}$ désigne le réel

$$x = (-1)^1 16^{-1} (3 \cdot 16^{-1} + 4 \cdot 16^{-2}) = -0.0127$$

•

$[a_1]$	e	x
1	-1	0.01
2	-1	0.02
3	-1	0.03
...	-1	...
9	-1	0.09
1	0	0.1
2	0	0.2
...	0	...
9	0	0.9
1	1	1
2	1	2
3	1	3
...	1	...
9	1	9

TABLE 2.1 – Ensemble des nombres réels positifs $x \in \mathbb{F}(10, 1, -1, 1)$

2.3.3 Répartition des nombres à virgule flottante

Notons par $\mathbb{F}(b, t, L, U)$ l'ensemble des nombres réels qui sont représentés par une notation à virgule flottante en base b, comportant t chiffres significatifs et dont l'exposant varie dans l'intervalle $[L, U]$. L'ensemble $\mathbb{F}$ ne contient pas le zéro si la représentation est normalisée.

Il est évident que l'ensemble des nombres à virgule flottante est un sous-ensemble fini de $\mathbb{R}$

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

Une relation intéressante est la suivante

$$x \in \mathbb{F}(b, t, L, U) \Leftrightarrow b^{(L-1)} \leq |x| \leq b^U(1 - b^{-t}) \tag{2.13}$$

On peut montrer que

$$\text{card}(\mathbb{F}) = 2(b-1)b^{t-1}(U - L + 1)$$

où $\text{card}(E)$ dénote le cardinal d'un ensemble E.

A la différence des nombres à virgule fixe, les nombres réels qui appartienent à l'ensemble $\mathbb{F}$ ne sont pas équirépartis le long de la droite réelle. Donc, il est important d'évaluer l'écart entre deux nombres consécutifs.

Exemple Considérons un système de représentation à virgule flottante où la base $b = 10$, les bornes de l'exposant sont $L = -1$ et $U = 1$, le signe $s = 0$ et le nombre de chiffres significatifs est $t = 1$. L'ensemble de nombres réels qui peuvent être représentés par ce système est indiqué dans la Table 2.1. On trouve que 27 nombres réels positifs sont représentés par cette notation. Au total $\text{card}(\mathbb{F}) = 54$.

•

Exemple Nous représentons l'ensemble des $x \in \mathbb{F}$ pour $b = 10$, $t = 1$, $L = -1$, $U = 1$ dans la Figure (2.1).

•

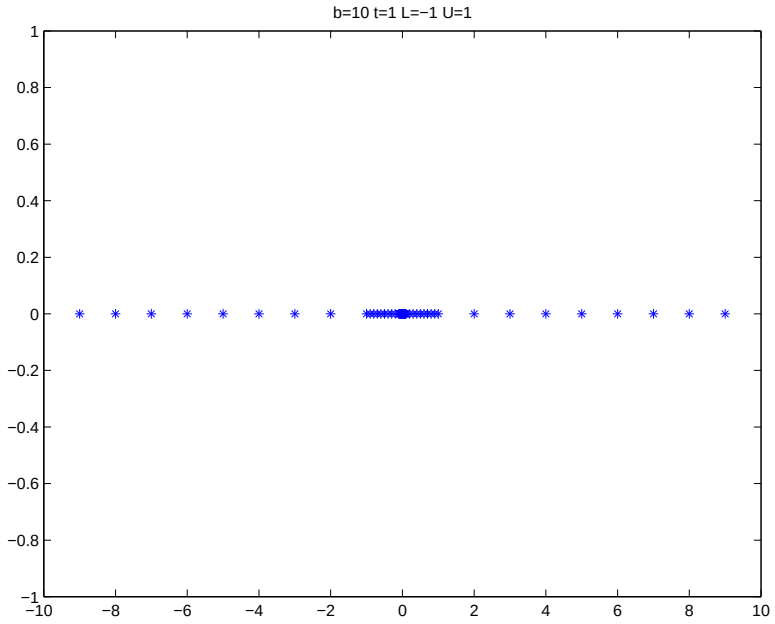


FIGURE 2.1 – Ensemble des nombres réels $x \in \mathbb{F}(10, 1, -1, 1)$. Script MATLAB `s_floset.m`

Nous définissons la *distance relative* entre 2 nombres consécutifs $x_i \in \mathbb{F}$ et $x_{i+1} \in \mathbb{F}$ par

$$\eta(x_i) = \frac{(x_{i+1} - x_i)}{x_i}.$$

Cette distance peut être dérivée en fonction des paramètres du système de notation

$$\eta(x_i) = \left| \frac{x_{i+1} - x_i}{x_i} \right| = \frac{b^{e-t}}{m(x_i)b^{e-t}} = \frac{1}{m(x_i)}, \tag{2.14}$$

où $m(x_i)$ est la mantisse du nombre réel x_i (formule (2.9)).

Puisque

$$b^t \geq m \geq b^{t-1} \Rightarrow b^{-t} \leq \frac{1}{m} \leq b^{1-t}, \tag{2.15}$$

on obtient

$$\frac{1}{b}\epsilon \leq \left| \frac{x_{i+1} - x_i}{x_i} \right| \leq \epsilon, \tag{2.16}$$

où $\epsilon = b^{1-t}$ est dit l'**epsilon machine**

Exemple Prenons le cas où $b = 10$, $L = -1$,$U = 1$, $s = 0$ et $t = 1$. L'ensemble des nombres qui appartiennent à $\mathbb{F}(10, 1, -1, 1)$ et les distances relatives associées sont indiquées dans la Table (2.2).

•

Phénomène du *wobbling* Considérons l'ensemble $\mathbb{F}(10, 1, -1, 1)$. Si nous traçons les distances relatives $\eta(x_i)$ en fonction des réels $x_i \in \mathbb{F}$, nous avons le graphique de la Figure 2.2. Notons que, conformément à la formule (2.15) les bornes inférieure et supérieure sont respectivement $b^{-t} = 0.1$ et $b^{1-t} = 1$.

$[a_1] = m(x_i)$	e	x_i	$\eta(x_i) = (x_{i+1} - x_i)/x_i$
1	-1	0.01	(0.02-0.01)/0.01=1
2	-1	0.02	(0.03-0.02)/0.02=1/2
...	-1	...	...
8	-1	0.08	(0.09-0.08)/0.08=1/8
9	-1	0.09	(0.1-0.09)/0.09=1/9
1	0	0.1	(0.2-0.1)/0.1=1
...	0	...	...
8	0	0.8	(0.9-0.8)/0.8=1/8
9	0	0.9	(1-0.9)/0.9=1/9
1	1	1	(2-1)/1=1
2	1	2	(3-2)/2=1/2
...	1	...	...
8	1	8	(9-8)/8=1/8
9	1	9	

TABLE 2.2 – Ensemble de nombres $x \in \mathbb{F}(10, 1, -1, 1)$ et distances relatives

Le phénomène d'oscillation des distances relatives $\eta(x_i)$ est connu sous le nom de *wobbling precision*. Il est d'autant plus prononcé que la base b est grande. C'est une raison pour laquelle on préfère employer de petites bases dans les ordinateurs.

•

Une autre différence notable entre la représentation à virgule flottante et la représentation à virgule fixe est la suivante : pour un espace mémoire fixé à l'avance, le système à virgule flottante permet la représentation d'un plus grand intervalle des nombres.

Exemple
Considérons une notation binaire à virgule flottante qui utilise $t = 23$ bits pour la mantisse et 8 bits (incluant le signe) pour l'exposant. La valeur absolue la plus grande parmi les $x \in \mathbb{F}$ est $|x_{\text{maxfl}}| \approx 2^U = 2^{(2^7-1)}$.

Pour avoir une valeur maximale comparable, le système à virgule fixe devrait avoir un nombre de cases n (seulement pour la partie entière) tel que

$$|x_{\text{maxfix}}| \approx 2^{n+1} - 1 = 2^{(2^7-1)}.$$

Donc, au moins $(n + 1) \approx 2^7 - 1 = 127$ bits sont nécessaires pour obtenir un intervalle de valeurs comparable à celui du système en virgule flottante.

•

2.4 Représentation machine des nombres réels

Considérons un ordinateur qui utilise la notation à virgule flottante avec base b, t chiffres significatifs et $L \leq e \leq U$. Dans un tel ordinateur, seul un sous-ensemble $\mathbb{F}(b, t, L, U) \subset \mathbb{R}$ de nombres réels peut être représenté et manipulé. Par conséquent, nous sommes confrontés au problème de représentation d'un nombre réel quelconque $x \in \mathbb{R}$ qui n'appartient pas à $\mathbb{F}$.

L'approche typique consiste à arrondir x de façon à ce que le nombre arrondi appartienne à $\mathbb{F}$.

En vertu de la relation (2.13), trois situations peuvent se produire

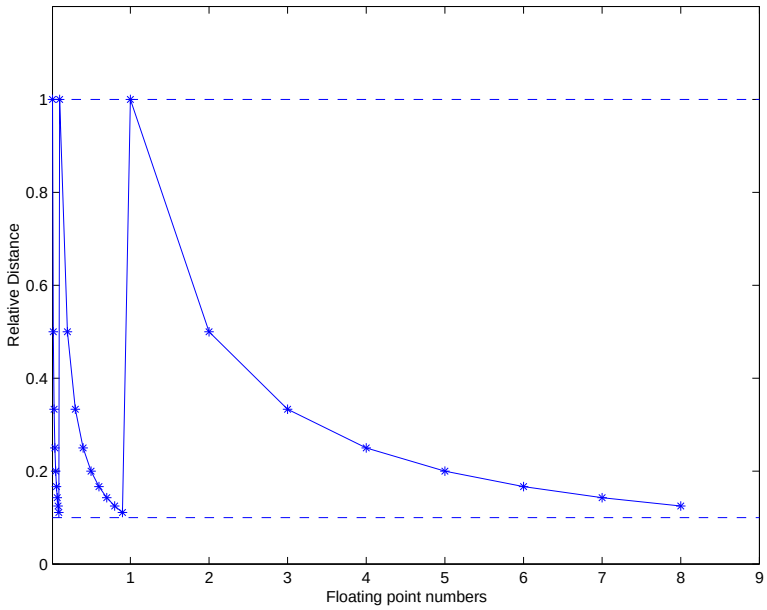


FIGURE 2.2 – *Wobbling* : variation de la distance relative pour l'ensemble des nombres $\mathbb{F}(10, 1, -1, 1)$. (Script MATLAB `s_dist_flo.m`)

1. $|x| > b^U(1 - b^{-t})$: le nombre x ne peut être représenté par le système de notation et on dit que nous sommes dans une situation d'*overflow*. L'overflow provoque normalement une interruption du programme par le système.
2. $|x| < b^{L-1}$: dans ce cas on parle d'*underflow*. D'habitude l'underflow est géré en remplaçant x par 0.
3. $b^{L-1} \leq |x| \leq b^U(1 - b^{-t})$: dans ce cas, en prenant un nombre des chiffres significatifs infini, le nombre $x \notin \mathbb{F}$ peut être représenté de la manière suivante

$$\{[a_1 a_2 \dots a_t a_{t+1} a_{t+2} \dots], e, b, s\}$$

où $e \in [L, U]$. On en déduit que deux transformations de x sont possibles :

Transformation d'arrondi :

$$x \to \mathrm{fl}(x) \tag{2.17}$$

où $\mathrm{fl}(x) \in \mathbb{F}(b, t, L, U)$ a comme notation en virgule flottante

$$\mathrm{fl}(x) = \{[a_1 a_2 \dots a_t^*], e, b, s\} \tag{2.18}$$

et

$$a_t^* = \begin{cases} a_t & \text{si } a_{t+1} < b/2 \\ a_t + 1 & \text{si } a_{t+1} \geq b/2 \end{cases} \tag{2.19}$$

Transformation de troncature :

$$x \to \mathrm{tr}(x) \tag{2.20}$$

où $\mathrm{tr}(x) \in \mathbb{F}(b, t, L, U)$ a comme notation en virgule flottante

$$\mathrm{tr}(x) = \{[a_1 a_2 \dots a_t], e, b, s\} \tag{2.21}$$

Dans ce qui suit, nous nous bornerons à l'analyse des erreurs qui dérivent de la transformation d'arrondi.

2.4.1 L'erreur d'arrondi

Considérons un ordinateur utilisant une notation $\mathbb{F}(b,t,L,U)$. Soit $x \in \mathbb{R}$ un nombre réel quelconque et $\mathrm{fl}(x) \in \mathbb{F}$ la représentation machine à virgule flottante de x. Désignons par ε_x son erreur relative

$$\varepsilon_x = \frac{|\mathrm{fl}(x) - x|}{|x|}. \tag{2.22}$$

Nous allons évaluer la borne supérieure de l'erreur relative d'arrondi. Il est évident que cette quantité ne peut dépasser la moitié de la distance relative entre les 2 nombres consécutifs à virgule flottante $x_i \in \mathbb{F}(b,t,L,U)$ et $x_{i+1} \in \mathbb{F}(b,t,L,U)$ tels que

$$x_i \leq x \leq x_{i+1}. \tag{2.23}$$

Il vient alors que

$$\varepsilon_x \equiv |\rho_x| \leq \frac{1}{2}\left|\frac{x_{i+1} - x_i}{x}\right| \leq \frac{1}{2}b^{1-t} = \frac{1}{2}\epsilon = u \tag{2.24}$$

où u est dite la **précision machine**.

En vertu de la formule (2.3) on obtient

$$\hat{x} = \mathrm{fl}(x) = x(1 + \rho_x) \tag{2.25}$$

où $\varepsilon_x = |\rho_x| \leq u$ est dit l'erreur d'arrondi.

Remarquons que la précision machine donne l'ordre de grandeur de la meilleure précision atteignable sur un ordinateur !

2.5 Erreurs d'arrondi et résolution numérique

Dans les sections qui précèdent, nous avons vu comme chaque représentation machine d'un nombre réel x implique une approximation et introduit par conséquence, une erreur ε dite d'arrondi.

Évidemment, ceci est valable aussi pour les données d'un problème numérique $x = F(d)$ (Section 1.1). En appliquant la formule (2.25) on obtient

$$\mathrm{fl}(d) = d(1 + \rho_d) \qquad \text{avec } |\rho_d| \leq u = \frac{1}{2}b^{1-t}. \tag{2.26}$$

La propagation des erreurs d'arrondi des données pendant la résolution d'un problème numérique engendre deux types d'erreur : les *erreurs de propagation* (dues au modèle) et les *erreurs de génération* (dues à l'algorithme).

2.5.1 L'erreur de propagation

Considérons le problème mathématique bien posé

$$x = F(d) \tag{2.27}$$

où d représente les données du problème.

Supposons qu'un ordinateur avec notation à virgule flottante $\mathbb{F}(b,t,L,U)$ est utilisé pour résoudre le problème. Pour ce faire, les données d doivent être introduites et codées par la machine. Ceci engendre une approximation des données par

$$\hat{d} = \mathrm{fl}(d) = d(1 + \rho_d). \tag{2.28}$$

En appliquant la fonction $F(\cdot)$ aux données $\hat{d}$, grâce aux relations (2.27) et (1.7), on obtient

$$F(\hat{d}) = F(d + d\rho_d) \approx F(d) + F'(d)d\rho_d = x\left(1 + \frac{F'(d)d\rho_d}{F(d)}\right) = x(1 + \kappa\rho_d) \quad (2.29)$$

où ρ_d est l'erreur d'arrondi des données et κ est le conditionnement du problème (2.27).

Notons que même si $F(\hat{d}) \in \mathbb{R}$ il pourrait se produire que $F(\hat{d}) \notin \mathbb{F}(b, t, L, U)$. Dans ce cas l'ordinateur renvoie comme résultat final de l'évaluation

$$\hat{x} = \mathrm{fl}(F(\hat{d})) = x(1 + \kappa\rho_d)(1 + \rho_{F(\hat{d})}) \quad (2.30)$$

où $\rho_{F(\hat{d})}$ désigne l'erreur d'arrondi de $F(\hat{d})$ et $\kappa\rho_d$ est défini comme l'*erreur de propagation*.

Il est important de remarquer l'écart existant entre la solution exacte du problème x et la solution approchée $x(1 + \kappa\rho_d)(1 + \rho_{F(\hat{d})})$ fournie par la résolution numérique.

En particulier si le conditionnement $\kappa > 1$, le calcul de la solution par le biais d'une machine a comme conséquence l'agrandissement de l'erreur initiale ρ_d.

Un cas particulier d'erreur de propagation est l'*erreur d'annulation*. Ceci se vérifie pendant la soustraction de deux termes très rapprochés.

Exemples d'erreur d'annulation Considérons une représentation en virgule flottante avec base $b = 10$ et $t = 4$ chiffres significatifs. Soit $x = F(d) = d - 1$ le problème à résoudre et $d = 1.00098$ la donnée.

La solution exacte est $x = 0.98 \cdot 10^{-3}$.

Après le codage du nombre réel d en notation machine, on obtient $\hat{d} = 0.1001 \cdot 10^1$.

L'erreur absolue relative de la donnée approchée est

$$\varepsilon_d = \left|\frac{\hat{d} - d}{d}\right| \approx \frac{1}{50000}. \quad (2.31)$$

Cette quantité est petite et peut être considérée comme négligeable.

En résolvant le problème à l'aide de la machine, on obtient la solution approchée $\hat{x} = 1 \cdot 10^{-3}$.

L'erreur absolue relative de la solution approchée est

$$\varepsilon_x = \left|\frac{F(\hat{d}) - x}{x}\right| \approx \frac{1}{50}. \quad (2.32)$$

Remarquons que

- contrairement à la quantité (2.31) cette quantité n'est pas négligeable ;
- l'erreur d'annulation est égale au produit du conditionnement du problème par l'erreur relative d'entrée

$$\kappa(d)\varepsilon_d \approx |F'(d)|\frac{|d|}{|F(d)|}\varepsilon_d = \frac{|d|}{|d-1|}\varepsilon_d \approx 1000 \cdot \frac{1}{50000} = \frac{1}{50}. \quad (2.33)$$

•

Rappelons que le fait qu'un problème soit bien/mal conditionné est une propriété indépendante de l'algorithme numérique choisi pour résoudre le problème.

Il en découle que l'erreur de propagation ne peut pas être évitée en changeant la méthode numérique. La prochaine section introduira l'erreur de génération qui, au contraire de l'erreur de propagation, dépend de la méthode numérique.

2.5.2 L'erreur de génération

Supposons que le problème $x = F(d)$ soit résolu par un algorithme décomposable en une série d'étapes :

$$x_1 = F_1(d), x_2 = F_2(x_1), x_3 = F_3(x_2), \ldots, x = F_n(x_{n-1}). \tag{2.34}$$

Soit $d \in \mathbb{R}$ la donnée codée par la machine en $\hat{d} = d(1 + \rho_d)$.

Après la première étape $x_1 = F_1(d)$, l'algorithme génère (formule (2.30))

$$\hat{x}_1 = x_1(1 + \kappa_1\rho_d)(1 + \rho_1) = x_1(1 + \rho^c_{x_1}) \tag{2.35}$$

où κ_1 est le conditionnement de F_1, ρ_1 est l'erreur d'arrondi commise pour stocker le résultat de la première étape et

$$\rho^c_{x_1} \approx (\kappa_1\rho_d + \rho_{x_1}). \tag{2.36}$$

Notons qu'à chaque étape la machine exécute le calcul et transforme le résultat en virgule flottante. Ceci engendre à chaque étape une erreur additionnelle d'arrondi qui va se propager dans les étapes successives.

En fait, à la deuxième étape on obtient

$$\hat{x}_2 = x_2(1 + \kappa_2\rho^c_{x_1})(1 + \rho_2) \approx x_2(1 + \rho_2 + \kappa_2\rho_1 + \kappa_2\kappa_1\rho_d) \tag{2.37}$$

D'une manière récursive (figure (2.3)), on déduit que

$$\hat{x} \approx x(1 + \overbrace{\rho_n}^{\text{err. arr}} + \underbrace{\kappa_n\kappa_{n-1}\rho_{n-2} + \cdots + \kappa_n \ldots \kappa_2\rho_1}_{\text{erreur génération}} + \overbrace{\kappa_n \ldots \kappa_1\rho_d}^{\text{err. prop}}) \tag{2.38}$$

Il résulte que l'erreur finale est composée de trois parties : l'erreur d'arrondi, l'erreur de propagation (indépendante de l'algorithme) et l'erreur de génération (liée à la forme (2.34) de la méthode).

Exemple Considérons un système de représentation en virgule flottante avec base $b = 10$ et $t = 3$ chiffres significatifs.

Soit $x = F(d) = e^d - 1$ le problème à résoudre et $d = 0.123 \cdot 10^{-1}$ la donnée.

La solution exacte est $x = 0.124 \cdot 10^{-1}$.

La donnée garde la même valeur après le codage puisque $d = \hat{d} \in \mathbb{F}$.

Considérons deux algorithmes différents pour résoudre ce même problème : le premier algorithme effectue le calcul en une seule étape

$$x = F_1(d) = F(d),$$

alors que le deuxième utilise deux étapes

$$x_1 = F_1(d) = e^d, \qquad x_2 = F_2(x_1) = x_1 - 1.$$

Nous allons vérifier que bien que les deux algorithmes soient identiques d'un point de vue mathématique, ils produisent deux résultats numériques différents.

Le premier algorithme fournit la solution exacte $\hat{x} = x$.

Pour ce qui concerne le deuxième algorithme nous obtenons

$$\hat{x}_1 = F_1(\hat{d}) = e^{\hat{d}} = 0.101 \cdot 10^1,$$
$$\hat{x} = F_2(\hat{x}_1) = \hat{x}_1 - 1 = 1 \cdot 10^{-2}.$$

L'erreur relative du deuxième algorithme (due à la composante de génération) est

$$\varepsilon_x = \left|\frac{\hat{x} - x}{x}\right| = 0.19.$$

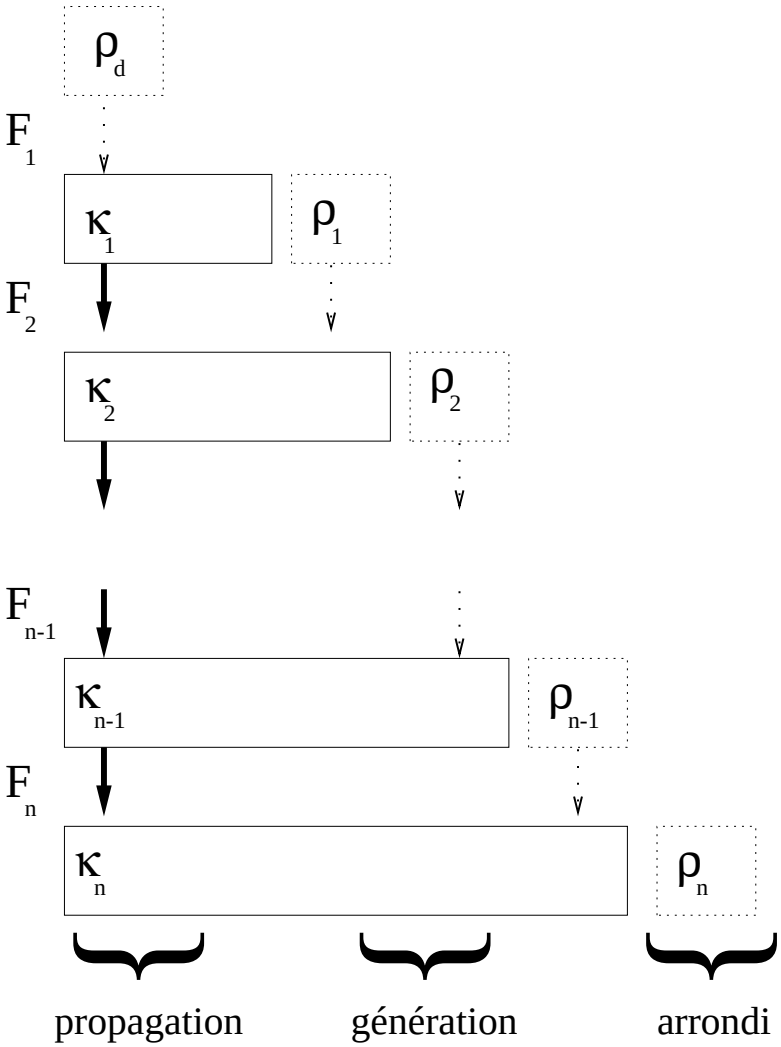


FIGURE 2.3 – Les trois composantes de l'erreur numérique

•

Un exemple d'erreur de génération est l'*erreur d'absorption* qui se vérifie pendant l'addition de deux opérandes d_1 et d_2 avec $d_1 \gg d_2$.

Exemple d'erreur d'absorption Considérons une représentation en virgule flottante avec base $b = 10$ et $t = 2$ chiffres significatifs. Soit $x = F(d) = d_1 + d_2$ le problème à résoudre et $d = [d_1, d_2] = [0.12 \cdot 10^{-4}, 0.1 \cdot 10^{1}]$ le vecteur des données.

Puisque $d_1 = \hat{d}_1$ et $d_2 = \hat{d}_2$, on obtient

$$\hat{x} = \text{fl}(0.100012 \cdot 10^{1}) = 0.10 \cdot 10^{1} = d_2$$

Les chiffres significatifs de d_1 sont perdus. Ceci implique une erreur relative de petite taille mais qui peut se avérer gênante dans la poursuite du calcul.

Pour le montrer, considérons un autre exemple. Supposons $b = 10$, $t = 6$ et $x = F(d) = (d_1 + d_2) - 1$ avec $d = [0.123456 \cdot 10^{-4}, 0.1 \cdot 10^{1}]$. La solution exacte est $x = d_1$.

Puisque

$$\hat{x} = \text{fl}(\text{fl}(\hat{d}_1 + \hat{d}_2) - 1) = \text{fl}(0.1 \cdot 10^{-4}) = 0.1 \cdot 10^{-4},$$

l'erreur relative absolue s'élève à

$$\varepsilon_x = \left| \frac{\hat{x} - x}{x} \right| = \left| \frac{\hat{x} - x}{x} \right| = \left| \frac{0.23456 \cdot 10^{-5}}{0.123456 \cdot 10^{-4}} \right| \approx 0.19.$$

L'erreur finale est donc non négligeable.

•

Comme nous l'avons souligné précédemment, plusieurs algorithmes peuvent être proposés pour résoudre le même problème mathématique. Afin de réduire l'erreur de génération, qui est le seul à dépendre de l'algorithme même, une règle empirique générale doit être tenue en considération : *si on ne peut pas éliminer les calculs avec un grand conditionnement, alors il vaut mieux les insérer le plus tôt possible dans l'algorithme.*

Comme illustré en Figure 2.3, à chaque étape de l'algorithme l' erreur d'entrée est amplifiée d'une façon proportionnelle au conditionnement de l'étape. En déplaçant, si possible, les étapes qui affichent le plus grand conditionnement au début, celles-ci causeront une amplification de l'erreur plus petite que dans le cas d'une autre localisation dans l'algorithme.

2.6 Opérations machines en virgule flottante

Jusqu'ici, nous avons considéré comme étapes d'un algorithme, des transformations génériques F_i. Il est évident que dans la plupart des cas, ces transformations prennent la forme d'opérations arithmétiques. Nous analyserons maintenant les effets des erreurs d'arrondi sur les opérations arithmétiques effectuées par un ordinateur à virgule flottante.

Soit

$$\circ : \mathbb{R} \times \mathbb{R} \to \mathbb{R} \tag{2.39}$$

une opération arithmétique quelconque entre deux opérandes réelles. Nous noterons

$$\circledcirc : \mathbb{R} \times \mathbb{R} \to \mathbb{F} \tag{2.40}$$

l'opération machine correspondante

$$d_1 \circledcirc d_2 = \text{fl}(\text{fl}(d_1) \circ \text{fl}(d_2)). \tag{2.41}$$

Cette opération est connue aussi sous le nom anglais *flop (floating operation)*.

Elle peut être décomposée en trois sous-opérations de la machine

1. l'arrondi des opérandes $d_1 \to \mathrm{fl}(d_1)$ et $d_2 \to \mathrm{fl}(d_2)$,
2. le calcul $\mathrm{fl}(d_1) \circ \mathrm{fl}(d_2)$ (qui pourrait ne pas être effectué en arithmétique exacte),
3. l'arrondi du résultat du calcul.

Par exemple dans le cas de l'addition

$$d_1 \oplus d_2 = \mathrm{fl}(\mathrm{fl}(d_1) + \mathrm{fl}(d_2)) = \mathrm{fl}(\hat{d}_1 + \hat{d}_2). \tag{2.42}$$

Il est important de remarquer que certaines propriétés de l'arithmétique exacte sont conservées tandis que d'autres sont perdues. Voici les propriétés conservées :

1. la propriété de *monotonie (relation d'ordre)*

$$\forall d \in \mathbb{F}, \quad \mathrm{fl}(d) = d; \tag{2.43}$$

$$\forall d_1, d_2 \in \mathbb{R}, \quad d_1 \leq d_2; \Rightarrow \mathrm{fl}(d_1) \leq \mathrm{fl}(d_2) \tag{2.44}$$

2. la propriété de *commutativité de la somme*

$$d_1 \oplus d_2 = d_2 \oplus d_1. \tag{2.45}$$

En revanche, la propriété d'*associativité* de la somme n'est en général pas conservée

$$d_1 \oplus (d_2 \oplus d_3) \neq (d_1 \oplus d_2) \oplus d_3 \tag{2.46}$$

Une propriété souhaitée (mais pas toujours respectée) est que l'opération entre deux opérandes $d_1 \in \mathbb{F}$ et $d_2 \in \mathbb{F}$ soit toujours bien définie, c.-à-d.

$$d_1 \circledcirc d_2 = (d_1 \circ d_2)(1 + \rho), \qquad |\rho| < u = \frac{1}{2} b^{1-t}. \tag{2.47}$$

Cette propriété n'est pas toujours satisfaite pour l'addition et la soustraction. Supposons $b = 10$ et $t = 2$.

En soustrayant $d_1 = 1$ et $d_2 = 0.99$ en utilisant seulement deux chiffres significatifs nous avons

$$\begin{array}{rcl} 0.10 \cdot 10^1\text{-} & & 0.10 \cdot 10^1\text{-} \\ 0.99 \cdot 10^0 & \Rightarrow & 0.09 \cdot 10^1 \\ & & 0.01 \cdot 10^1 \end{array}$$

Il s'ensuit que l'erreur relative est égale à $|\rho| = 9$ pour $u = 0.05$. Cette situation est appelée *aberrante*.

Afin d'éviter des résultats aberrants, on utilise un bit supplémentaire dit *chiffre de garde*. Le tableau suivant montre comment l'adoption d'un chiffre de garde dans le calcul résout le problème.

$$\begin{array}{rcl} 0.10 \cdot 10^1\text{-} & & 0.10 \cdot 10^1\text{-} \\ 0.99 \cdot 10^0 & \Rightarrow & 0.09\mathbf{9} \cdot 10^1 \\ & & 0.01 \cdot 10^0 \end{array}$$

Malgré les erreurs dues aux opérations aberrantes, plusieurs ordinateurs dans l'histoire de l'informatique ont renoncé au chiffre de garde à l'avantage de la vitesse :

- IBM'360 de 1964 à 1967 ;
- Cray 1,2,X-MP,Y-MP ;
- C90 avant 1992 ;
- HP-80.

Il est important de mentionner qu'en revanche le standard IEEE (Section 2.8) utilise bien 2 chiffres de garde.

2.6.1 La propagation d'erreurs dans les opération arithmétiques en virgule flottante

Nous allons maintenant dériver d'une manière analytique la valeur de l'erreur de propagation dans le cas des 4 opérations arithmétiques effectuées en virgule flottante.

Considérons l'opération de somme de deux opérandes $d_1 \in \mathbb{R}$ et $d_2 \in \mathbb{R}$. Le résultat exact est $x = d_1 + d_2$, $x \in \mathbb{R}$.

Si nous effectuons l'opération en virgule flottante, le résultat $\hat{x}$ est entaché par la propagation des erreurs d'arrondi ρ_{d_1} et ρ_{d_2} et par son erreur d'arrondi. Si nous posons $s = \hat{d_1} + \hat{d_2} = d_1(1+\rho_{d_1}) + d_2(1+\rho_{d_2})$, où s est un nombre réel, nous avons

$$\hat{x} = d_1 \oplus d_2 = x(1+\rho_x) = \mathrm{fl}(s) = s(1+\rho_s) = (d_1(1+\rho_{d_1})+d_2(1+\rho_{d_2}))(1+\rho_s). \quad (2.48)$$

Nous dérivons ainsi une borne sur la valeur $\varepsilon_x = |\rho_x|$:

$$\begin{aligned}
|\rho_x| &= \left|\frac{x-\hat{x}}{x}\right| = \\
&= \left|\frac{x-(\hat{d_1}+\hat{d_2})(1+\rho_s)}{x}\right| = \left|\frac{d_1+d_2-(d_1+d_1\rho_{d_1}+d_2+\rho_{d_2})(1+\rho_s)}{d_1+d_2}\right| = \\
&= \left|\frac{d_1(\rho_{d_1}+\rho_s)+d_2(\rho_{d_2}+\rho_s)+(d_1\rho_{d_1}+d_2\rho_{d_2})\rho_s}{d_1+d_2}\right| \\
&= \left|\frac{d_1\rho_{d_1}+d_2\rho_{d_2}+(d_1(1+\rho_{d_1})+d_2(1+\rho_{d_2}))\rho_s}{d_1+d_2}\right| = \\
&= \left|\frac{(d_1+d_2)\rho_s+d_1\rho_{d_1}(1+\rho_s)+d_2\rho_{d_2}(1+\rho_s)}{d_1+d_2}\right| \leq \\
&\leq |\rho_s| + |\rho_{d_1}(1+\rho_s)|\left|\frac{d_1}{d_1+d_2}\right| + |\rho_{d_2}(1+\rho_s)|\left|\frac{d_2}{d_1+d_2}\right| \\
&\leq u + u(1+u)\left|\frac{|d_1|+|d_2|}{|d_1+d_2|}\right|. \qquad (2.49)
\end{aligned}$$

Cette relation illustre la relation entre ε_x et les valeurs des opérandes. En plus, elle explique d'une manière formelle le phénomène de l'erreur d'annulation (Section 2.5.1). On en déduit que si $|d_1 + d_2| \approx 0$ l'erreur ε_x devient grande.

En négligeant les termes de deuxième ordre, on peut réécrire la formule (2.49) avec une notation moins lourde

$$\varepsilon_x = \left|\frac{(d_1+d_2)\rho_s+d_1\rho_{d_1}+d_2\rho_{d_2}}{d_1+d_2}\right| \leq u + u\frac{|d_1|+|d_2|}{|d_1+d_2|} \qquad (2.50)$$

D'une façon analogue, nous obtenons les formules de l'erreur relative pour les autres opérations :

– Soustraction : $x = d_1 - d_2$

$$\varepsilon_x = \left|\frac{(d_1-d_2)\rho_s+d_1\rho_{d_1}-d_2\rho_{d_2}}{d_1-d_2}\right|. \qquad (2.51)$$

– Multiplication : si $x = d_1 \cdot d_2$ et $p = \mathrm{fl}(d_1)\mathrm{fl}(d_2)$

$$\hat{x} = \mathrm{fl}(p) = (d_1(1+\rho_{d_1})d_2(1+\rho_{d_2}))\,(1+\rho_p) \approx d_1d_2(1+\rho_{d_1}+\rho_{d_2}+\rho_p)$$

et donc

$$\begin{aligned}
\varepsilon_x &= \left|\frac{x-\hat{x}}{x}\right| = \\
&= \left|\frac{x-d_1d_2-(d_1d_2\rho_{d_1}+d_1d_2\rho_{d_2}+d_1d_2\rho_p)}{d_1d_2}\right| = |\rho_{d_1}+\rho_{d_2}+\rho_p|. \qquad (2.52)
\end{aligned}$$

L'erreur croît donc de manière additive.

– Division : si $x = d_1/d_2$ et $p = \mathrm{fl}(d_1)/\mathrm{fl}(d_2)$

$$\varepsilon_x = \left| \frac{x - d_1/d_2 - ((d_1/d_2\rho_{d_1}) + (d_1/d_2\rho_{1/d_2}) + (d_1/d_2)\rho_p)}{d_1/d_2} \right| = \\ = |(\rho_{d_1} + \rho_{1/d_2}) + \rho_p|. \quad (2.53)$$

Remarque Les effets les plus macroscopiques de la propagation des erreurs due aux opérations arithmétiques apparaissent dans l'analyse numérique des suites. Quelques situations pathologiques peuvent se produire

– Une suite ne convergeant pas en réalité, converge en machine. Un exemple en est la série harmonique.
– Une suite convergente en réalité, ne converge pas en machine.
– Une suite converge en réalité et en machine mais vers des valeurs différentes.

•

2.7 Quelques conseils pour concevoir un algorithme stable

Les programmeurs d'algorithmes numériques doivent se confronter chaque jour aux problèmes de stabilité des algorithmes. Voici quelques conseils basés sur les résultats précédents pour réduire ces problèmes

– Éviter, si possible, de soustraire des quantités entachées d'erreur.
– Minimiser la magnitude des résultats intermédiaires pour éviter les erreurs d'absorption (Section 2.5.2).
– Évaluer les formulations alternatives qui sont mathématiquement équivalentes mais numériquement différentes.
– Utiliser seulement des transformations bien conditionnées.
– Être attentif aux problèmes de *overflow/underflow*.
– Regarder attentivement les résultats intermédiaires.

2.8 Standard IEC/IEEE

L'effort de standardisation concernant la représentation de nombres réels sur ordinateur dérive de la nécessité d'éviter une prolifération de systèmes de représentation, qui différent en base, nombre de chiffres significatifs et exposants. A présent, le standard le plus répandu est le standard connu sous le nom IEC 559. Ceci a été développé par le IEEE (*Institute of Electrical and Electronic Engineers*) en 1985 et approuvé en 1989 par le IEC (*International Electronic Commission*).

Le standard spécifie, entre autres, les opérations arithmétiques de base, la racine carrée, le reste, la conversion entre représentation décimale et binaire, le comportement suite à un *overflow/underflow* (voir Section (2.4)). Le but est d'éviter les incompatibilités entre ordinateurs différents.

Il définit aussi les codages spéciaux pour les situations exceptionnelles qui ne sont pas couvertes par le codage standard (Table 2.3). Par exemple les non-nombres (en abrégé NaN pour *Not a Number*) qui correspondent entre autres au résultat de la division 0/0.

Valeur	Exposant	Mantisse
± 0	$L-1$	0
$\pm\infty$	$U+1$	0
NaN	$U+1$	$\neq 0$

TABLE 2.3 – Codages IEC559 des valeurs spéciales

2.9 Gestion de la mémoire

Chaque ordinateur a à sa disposition un nombre fini d'espace mémoire pour stocker les nombres réels. Le concepteur de l'architecture hardware est donc tenu de concilier la précision et la performance avec les contraintes techniques. En particulier, nous supposerons ici qu'un nombre N de cases mémoires binaires ($b = 2$) est disponible pour stocker un nombre réel. Nous nous focaliserons alors sur la répartition des cases mémoire entre les chiffres significatifs et l'exposant.

Puisque la ressource est finie (nombre maximum N de cases), il est évident que le concepteur est tenu de trancher entre deux exigences différentes : la précision de la représentation (fonction de l'erreur d'arrondi) et le domaine de valeurs qui peuvent être représentées.

En vertu des formules (2.24) et (2.13) nous remarquons que

- l'ampleur du domaine $\mathbb{F}$ est déterminé par le nombre de bits assigné à l'exposant et à la base ;
- la précision est déterminée par le nombre de bits assigné aux chiffres significatifs et à la base.

Ceci signifie que pour une base fixée, en échangeant bits de la mantisse contre bits de l'exposant nous sacrifions la précision (l'erreur d'arrondi est autant plus grande que t est petit) au bénéfice de l'ampleur du domaine (l'intervalle des valeurs de $\mathbb{F}$ est autant plus ample que e est grand).

Il s'ensuit aussi qu'en augmentant la base, nous élargissons le domaine, ayant pour conséquence une plus large précision pour les nombres plus petits et une précision plus réduite pour les grands.

En réalité la plupart de ces choix architecturaux est fait en suivant des standards consolidés. A part le cas historique du système IBM 370 ($b = 16$) la base normalement utilisée est $b = 2$. En plus, il y a habituellement deux manières standards pour répartir les N bit de codage :

- Représentation en *simple précision* : les nombres sont codés avec $N = 32$ bits, où 1 bit est utilisé pour le signe, 8 bits pour l'exposant et 23 bits pour la mantisse.
- Représentation en *double précision* : les nombres sont codés avec $N = 64$ bits, où 1 bit est utilisé pour le signe, 11 bits pour l'exposant et 52 bits pour la mantisse.

2.10 L'analyse de stabilité

Soit P un problème mathématique $x = F(d)$ et $\hat{x}$ la solution approchée fournie par un algorithme A.

L'analyse de la stabilité de l' algorithme numérique A peut être menée en suivant deux stratégies différentes :

- l'analyse *a priori* qui vise à établir la relation entre les perturbations δ_d des données d et les perturbations δ_x de la solution x ;
- l'analyse *a posteriori* qui vise à établir la relation entre l'erreur de la solution approchée $\hat{x}$ et une quantité qui peut être calculée par l'algorithme numérique

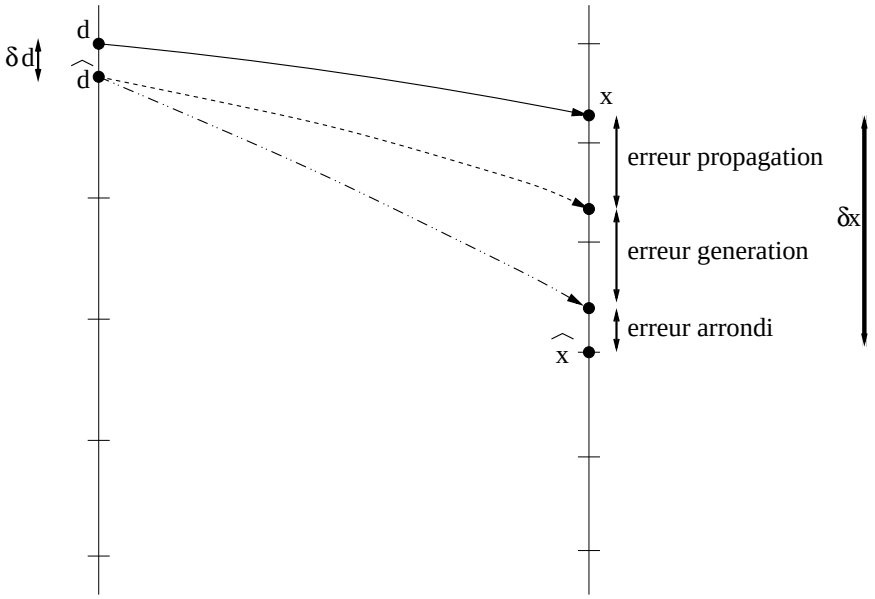


FIGURE 2.4 – Analyse a priori directe : on analyse la relation entre la perturbation δd et la perturbation δx de la solution x.

(par exemple le résidu $F(\hat{x}, d)$ de la solution).
Il y a deux types d'analyse a priori :

Analyse a priori directe : on impose des perturbations δ_d aux données d et on analyse les variations δ_x de la solution numérique. Cette analyse demande la connaissance de la forme analytique $F(\cdot)$ et de l'algorithme numérique.

Analyse a priori rétrograde ou rétrospective : on suppose que la solution $\hat{x}$ approchée du problème P est la solution correcte d'un problème ayant des données différentes $d+\delta_b d$. Ensuite, on estime la valeur $\delta_b d$ qui en arithmétique exacte donnerait la solution $\hat{x} = F(d+\delta_b d)$. Cette analyse ne tient pas compte de l'algorithme numérique qui a été utilisé pour obtenir $\hat{x}$.

2.11 Exercices

2.11.1 Représentation en virgule fixe

Dans cette section, b désigne la base, $n+1$ le nombre de chiffres avant la virgule, m le nombre de chiffres après la virgule et s le signe.

1. Si $b = 10$, $n = 3$, $m = 6$ et $s = 0$ que représente le nombre [0012,200000] ?
2. Idem si $b = 3$.
3. Si $b = 3$, $n = 3$, $m = 6$ et $s = 0$ que représente le nombre [3012,200000] ?

2.11.2 Représentation en virgule flottante normalisée

Dans cette section b désigne la base, e l'exposant, s le signe, t le nombre de chiffres significatifs et U et L respectivement les bornes supérieures et inférieures sur l'exposant. .

1. Que désigne la notation $\{[3, 4], e = 2, b = 10, s = 0\}$?

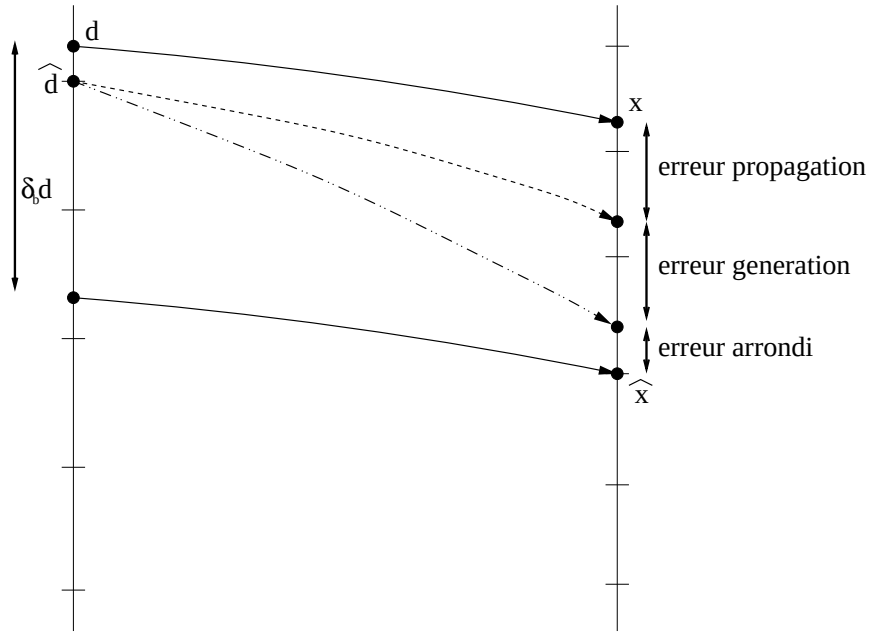


FIGURE 2.5 – Analyse a priori rétrograde : on calcul quel changement $\delta_b d$ des données cause comme solution exacte $\hat{x} = F(d + \delta_b d)$.

2. Que désigne la notation $\{[3, 4], e = 1, b = 16, s = 1\}$?
3. Que désigne la notation $\{[1, 0, 0, 0, 1], e = 6, b = 2, s = 0\}$?
4. Représenter le réel $234,5$ en virgule flottante, avec $b = 2$, les autres paramètres étant suffisamment grands.
5. Combien de nombres réels peut-on représenter en virgule flottante avec les paramètres suivants : $b = 4, L = -2, U = 2, t = 2$?
6. Expliquer comment est trouvée la formule de la cardinalité de $\mathbb{F}(b, t, L, U)$.
7. Le système

$$\begin{cases} 0.461x_1 + 0.311x_2 = 0.150 \\ 0.209x_1 + 0.141x_2 = 0.068 \end{cases}$$

a pour solution $x_1 = 1$ et $x_2 = -1$. Montrer que la résolution du système en utilisant la représentation machine à virgule flottante ($b = 10, t = 3$) manque de précision (utiliser l'arithmétique tronquée).

8. Supposons que $z = 0.180 \times 10^2$ est une solution approchée de $Ax = B$ pour $a = 0.111 \times 10^0$, $b = 0.200 \times 10^1$. Utiliser l'arithmétique décimale tronquée à 3 chiffres pour calculer le reste $r = b - a \times z$. Comparer avec le résultat en arithmétique exacte.

2.11.3 Erreur d'annulation

1. Vérifier que pour $d_1 = 0.1656, d_2 = 7.4409, d_3 = 2.9168$ dans un système à virgule flottante où $b = 10$ et $t = 2$:

$$d_1 \oplus (d_2 \oplus d_3) \neq (d_1 \oplus d_2) \oplus d_3$$

(utiliser l'arithmétique arrondie).

2. Trouver un exemple où $(x \oplus y) \oplus z \neq x \oplus (y \oplus z)$.
3. Pour un système en virgule flottante où $b = 10$ et $t = 4$ montrer que l'évaluation de la fonction $F(d) = d - 1$ est sujette à une importante erreur d'annulation dans le cas où $d = 1.00098$.

2.11.4 Erreur de génération

1. Pour $\alpha = 0.8717$ et $\beta = 0.8719$, calculer le point milieu de l'intervalle $(\alpha+\beta)/2$. D'abord, utiliser l'arithmétique décimale tronquée à 4 chiffres, puis l'arithmétique décimale arrondie à 4 chiffres. Enfin, proposer une autre séquence d'évaluation qui génère moins d'erreur.
2. Dans un système en virgule flottante où $b = 10$ et $t = 3$, comparer les deux algorithmes suivant d'évaluation de la fonction $F(d) = d^2 - 1$ où $d = 0.107 \times 10^1$:
   - Algorithme 1 $\begin{array}{l} x_1^* = d * d \\ x_2 = x_1^* - 1 \end{array}$
   - Algorithme 2 $\begin{array}{l} x_1^* = d - 1 \\ x_2 = x_1^*(2 + x_1^*) \end{array}$

Chapitre 3

Résolution des systèmes linéaires

3.1 Exemples et motivations

La résolution d'un système linéaire est un problème rencontré couramment dans la physique et les sciences de l'ingénieur. Les modèles linéaires sont souvent utilisés dans la représentation de phénomène physiques, autant pour leur interprétation intuitive que pour la disponibilité d'un grand nombre de techniques de résolution.

Les systèmes linéaires apparaissent souvent aussi dans la résolution d'autres problèmes mathématiques comme le calcul des valeurs propres d'une matrice ou la résolution d'équations aux dérivées partielles (utilisées pour modéliser plusieurs phénomènes, comme la diffusion de la chaleur, la propagation des ondes..).

Dans ce qui suit, nous présentons quelques exemples simples de problèmes réels qui peuvent être abordés par le biais de systèmes linéaires.

Réseau électrique : on considère un réseau électrique purement résistif comportant 3 résistances et un générateur de tension (Figure 3.1). Supposons que les valeurs V, R_1, R_2 et R_3 sont connues et que nous voulons trouver le courant i_1 passant par R_3. La loi de Kirchoff conduit à un système linéaire

$$\begin{cases} R_3 i_1 = V - (R_1(i_1 + i_2)) \\ R_2 i_2 = V - (R_1(i_1 + i_2)) \end{cases} \tag{3.1}$$

d'ordre 3. Sa résolution permet le calcul de la quantité inconnue i_1.

Système mécanique : Considérons un système mécanique composé de 5 ressorts en série (Figure 3.2), où L dénote la distance entre les deux points fixes du

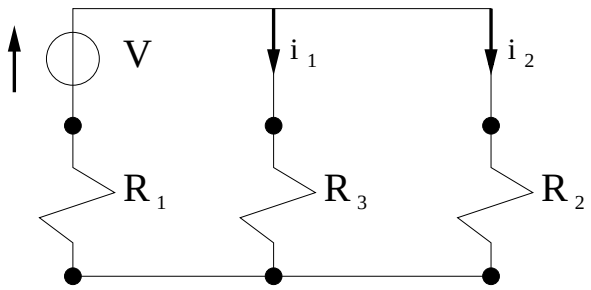


FIGURE 3.1 – Réseau électrique

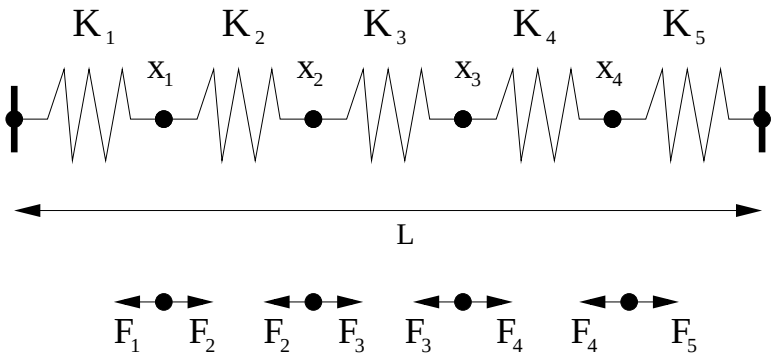


FIGURE 3.2 – Système mécanique

système. Soient K_i et l_i la constante élastique et la longueur au repos du $i$ème ressort, respectivement. Nous sommes intéressés par le vecteur x_i, $i = 1, \ldots, 4$ qui représente la position d'équilibre du système.
Nous écrivons les équations d'équilibre des deux forces élastiques F_i et F_{i+1} dans chaque noeud du système mécanique, d'où le système d'équations linéaires :

$$\begin{cases} (K_1 + K_2)x_1 - K_2x_2 = K_1l_1 - K_2l_2 \\ -K_2x_1 + (K_2 + K_3)x_2 - K_3x_3 = K_2l_2 - K_3l_3 \\ -K_3x_2 + (K_3 + K_4)x_3 - K_4x_4 = K_3l_3 - K_4l_4 \\ -K_4x_3 + (K_4 + K_5)x_4 = K_4l_4 + K_5L - K_5l_5 \end{cases} \tag{3.2}$$

La résolution du système linéaire (3.2) fournit la position d'équilibre du système mécanique.

3.2 Systèmes linéaires

Un système linéaire de N équations à n inconnues est un ensemble de relations algébriques de la forme

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1, \ldots, N, \tag{3.3}$$

où les termes $a_{ij} \in \mathbb{R}$ sont les *coefficients* du système, $b_i \in \mathbb{R}$ sont les composants du deuxième membre et x_j sont les *inconnues*.

On appelle *solution du système linéaire* tout n-tuple de valeurs $[x_1, x_2, \ldots, x_n]$ qui satisfait les N relations (3.3).

Il est commode d'écrire le système (3.3) sous la forme matricielle

$$Ax = b \tag{3.4}$$

où $A_{(N\times n)}$ est la matrice des coefficients, $b_{(N\times 1)}$ est le vecteur du second membre et $x_{(n\times 1)}$ est le vecteur des inconnues du système. Nous renvoyons le lecteur à l'appendice pour un rappel sur le calcul matriciel.

Dans ce chapitre, nous nous bornerons à traiter des *systèmes carrés*, c-à-d. des systèmes où le nombre N d'équations est égal au nombre n d'inconnues. Dans ce cas, voici les conditions qui assurent l'existence et l'unicité de la solution.

Théorème 3.1 (Solution d'un système linéaire)**.** *La solution d'un système linéaire carré existe et est unique si une des conditions équivalentes suivantes est remplie*

– *A est inversible,*
– *A est régulière (non singulière), c.-à-d.* $\det(A) \neq 0$*,*
– *le rang* $rg(A) = n$*,*
– *le système homogène* $Ax = 0$ *admet seulement la solution nulle.*

La solution du système carré (3.4) est donnée, d'un point de vue théorique, par les formules de Cramer [2]

$$x_j = \frac{\Delta_j}{\det(A)}, \qquad j = 1, \ldots, n, \tag{3.5}$$

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

Néanmoins, l'algorithme de résolution basé sur les formules de Cramer affiche une complexité factorielle $O((n+1)!)$ (Section 1.3.2) ; il est donc inutilisable pour une matrice de grande taille.

Par conséquent, plusieurs méthodes numériques moins onéreuses ont été proposées comme alternative. On peut repartir ces méthodes en deux classes :
– *méthodes directes* : celles-ci fournissent la solution en un nombre fini d'étapes ;
– *méthodes itératives* : celles-ci nécessitent théoriquement un nombre infini d'étapes. Elles sont appliquées principalement dans le cas de matrices creuses et de matrices de taille très grande (par exemple dans le cadre de la résolution numérique d'équations aux dérivées partielles).

Le choix entre ces deux options est typiquement faite sur la base des considérations théoriques et architecturales (mémoire de l'ordinateur, temps d'exécution requis, contraintes temps-réel).

3.3 Systèmes triangulaires

Nous commencons par analyser les méthodes de résolution du cas plus simple de système linéaire carré : le système triangulaire. Suivant la forme de la matrice A, nous distinguons deux types de systèmes triangulaires.

Définition 16 (Matrices triangulaires)**.** *La matrices des coefficients A est dite* triangulaire supérieure *si*

$$a_{ij} = 0, \qquad \forall i, j : 1 \leq j < i \leq n. \tag{3.6}$$

La matrices des coefficients A est triangulaire inférieure *si*

$$a_{ij} = 0, \qquad \forall i, j : 1 \leq i < j \leq n. \tag{3.7}$$

Définition 17 (Système triangulaire)**.** *Un système linéaire dont la matrice des coefficients est triangulaire supérieure (resp. inférieure) est dit un* système linéaire triangulaire supérieur (resp. inférieur)*.*

Puisque le déterminant d'une matrice triangulaire est

$$\det A = \prod_{i=1}^{n} a_{ii}, \tag{3.8}$$

afin de garantir qu'elle soit non singulière, nous devons imposer la condition supplémentaire $a_{ii} \neq 0$ pour tout $i = 1, \ldots, n$.

Un système triangulaire inférieur d'ordre 3 a la forme suivante :

$$\begin{cases} a_{11}x_1 = b_1 \\ a_{21}x_1 + a_{22}x_2 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases} \tag{3.9}$$

ou, sous forme matricielle

$$\begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{3.10}$$

On peut donc déterminer successivement les valeurs inconnues x_i pour $i = 1, 2, 3$

$$x_1 = \frac{b_1}{a_{11}}, \tag{3.11}$$

$$x_2 = \frac{b_2 - a_{21}x_1}{a_{22}}, \tag{3.12}$$

$$x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}}. \tag{3.13}$$

Pour le cas général d'un système $Ax = b$, où A est une matrice triangulaire inférieure $(n \times n)$, $n > 2$ et $a_{ii} \neq 0$ on a alors

$$x_1 = \frac{b_1}{a_{11}}, \tag{3.14}$$

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j}{a_{ii}}, \qquad i = 2, \ldots, n. \tag{3.15}$$

Cet algorithme est appelé *méthode de substitution progressive*.

Dans le cas d'un système triangulaire supérieur d'ordre 3

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{3.16}$$

la solution est

$$x_3 = \frac{b_3}{a_{33}} \tag{3.17}$$

$$x_2 = \frac{b_2 - a_{23}x_3}{a_{22}} \tag{3.18}$$

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \tag{3.19}$$

Dans la cas général d'un système triangulaire supérieur d'ordre $n \geq 2$, on a alors

$$x_n = \frac{b_n}{a_{nn}} \tag{3.20}$$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}} \qquad i = n-1, \ldots, 1 \tag{3.21}$$

Cet algorithme est appelé *méthode de substitution rétrograde*.

3.3.1 Analyse de complexité

Nous analysons maintenant la complexité (Section 1.3.2) d'un algorithme de substitution progressive (3.14) en termes d'opérations arithmétiques. Puisque

$$\begin{array}{ll} x_1 = b_1/a_{11} & \rightarrow 1 \text{ div} \\ x_2 = (b_2 - a_{21}x_1)/a_{22} & \rightarrow 1 \text{ div}+1 \text{ add}+1 \text{ mul} \\ x_3 = (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} & \rightarrow 1 \text{ div}+2 \text{ add}+2 \text{ mul} \\ x_4 = (b_4 - a_{41}x_1 - a_{42}x_2 - a_{43}x_3)/a_{44} & \rightarrow 1 \text{ div}+3 \text{ add}+3 \text{ mul} \\ \ldots & \\ x_n = (b_n - \sum_{j=1}^{n-1} a_{nj}x_j)/a_{nn} & \rightarrow 1 \text{ div}+(n-1) \text{ add}+(n-1) \text{ mul} \end{array} \tag{3.22}$$

en utilisant la relation

$$\sum_{r=1}^{n-1}(n-r) = \frac{n(n-1)}{2}, \tag{3.23}$$

on trouve que la résolution du système triangulaire demande

- n divisions,
- $\frac{n(n-1)}{2}$ multiplications et
- $\frac{n(n-1)}{2}$ additions (ou soustractions).

Au total, l'algorithme nécessite n^2 opérations en virgule flottante (flops). La complexité des algorithmes de résolution d'un système linéaire triangulaire est donc quadratique.

Par la suite, nous allons considérer les méthodes pour la résolution d'un système linéaire non triangulaire.

3.4 Élimination de Gauss

La méthode d'élimination de Gauss s'appuie sur la notion de systèmes équivalents.

Définition 18 (Systèmes équivalents)**.** *Deux systèmes linéaires $A_1x_1 = b_1$ et $A_2x_2 = b_2$ sont équivalents s'ils ont la même solution, c.-à-d. si $x_1 = x_2$.*

Soit un système linéaire $Ax = b$. On obtient un système équivalent si on effectue une ou plusieurs de ces opérations :

- on inverse les deux mêmes lignes dans A et b,
- on multiplie la même ligne de A et b par une constante,
- on ajoute à une ligne donnée de A et b une combinaison linéaire des autres lignes.

Exemples Soit

$$\begin{bmatrix} 4 & 8 & 12 \\ 3 & 8 & 13 \\ 2 & 9 & 18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 11 \end{bmatrix} \tag{3.24}$$

un système linéaire ayant pour solution $x = [1, -3, 2]$.

Les systèmes linéaires suivants

$$\begin{bmatrix} 3 & 8 & 13 \\ 4 & 8 & 12 \\ 2 & 9 & 18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 11 \end{bmatrix} \tag{3.25}$$

où la première et la deuxième ligne de (3.24) ont été inversées,

$$\begin{bmatrix} 8 & 16 & 24 \\ 3 & 8 & 13 \\ 2 & 9 & 18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 5 \\ 11 \end{bmatrix} \tag{3.26}$$

où la première ligne de (3.24) a été multipliée par 2, et

$$\begin{bmatrix} 4 & 8 & 12 \\ 3 & 8 & 13 \\ 9 & 25 & 43 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 20 \end{bmatrix} \tag{3.27}$$

où on a ajouté à la troisième ligne de (3.24) la somme des deux premières, sont équivalents au système (3.24).

•

La méthode d'élimination de Gauss a pour but la transformation du système $Ax = B$ en un système équivalent sous forme triangulaire supérieure, pour lequel le calcul de la solution est simple.

Ceci est effectué à l'aide de combinaisons linéaires des diverses équations.

3.4.1 Description de l'algorithme

L'algorithme de Gauss pour un système générique $Ax = b$ de taille n est composé de $n-1$ étapes. Chaque étape transforme les matrices $\{A^{(k)}, b^{(k)}\}$ en les matrices $\{A^{(k+1)}, b^{(k+1)}\}$ où $k = 1, \ldots, n-1$ et $A^{(1)} = A$, $b^{(1)} = b$.

3.4.1.1 Résolution d'un système d'ordre $n = 3$

Afin de faciliter la description de l'algorithme nous commençons par un système d'ordre $n = 3$. Considérons le système à résoudre $A^{(1)}x = b^{(1)}$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix} \tag{3.28}$$

Dorénavant, par ligne $l_i^{(k)}$ on fait référence à la concaténation du vecteur ligne $A_i^{(k)}$ et du terme $b_i^{(k)}$

$$l_i^{(k)} = [a_{i1}^{(k)}, a_{i2}^{(k)}, a_{i3}^{(k)}, b_i^{(k)}] \tag{3.29}$$

De plus, nous supposons que pour toutes les opérations de division le dénominateur diffère de zéro. Cette hypothèse sera discutée dans la Section (3.9).

La première étape de l'algorithme consiste à

1. calculer les deux *multiplicateurs*

$$m_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}}, \qquad m_{31} = \frac{a_{31}^{(1)}}{a_{11}^{(1)}}; \tag{3.30}$$

2. remplacer la deuxième ligne par la différence entre elle-même et la première ligne multipliée par m_{21}

$$l_2^{(2)} \leftarrow l_2^{(1)} - l_1^{(1)} m_{21}; \tag{3.31}$$

3. remplacer la troisième ligne par la différence entre elle-même et la première ligne multipliée par m_{31}

$$l_3^{(2)} \leftarrow l_3^{(1)} - l_1^{(1)} m_{31}. \tag{3.32}$$

Au bout de la première itération, le système $A^{(1)}x = b^{(1)}$ est transformé dans le système équivalent $A^{(2)}x = b^{(2)}$ où

$$A^{(2)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} - m_{21}a_{11}^{(1)} & a_{22}^{(1)} - m_{21}a_{12}^{(1)} & a_{23}^{(1)} - m_{21}a_{13}^{(1)} \\ a_{31}^{(1)} - m_{31}a_{11}^{(1)} & a_{32}^{(1)} - m_{31}a_{12}^{(1)} & a_{33}^{(1)} - m_{31}a_{13}^{(1)} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(1)} - m_{21}a_{12}^{(1)} & a_{23}^{(1)} - m_{21}a_{13}^{(1)} \\ 0 & a_{32}^{(1)} - m_{31}a_{12}^{(1)} & a_{33}^{(1)} - m_{31}a_{13}^{(1)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix} \tag{3.33}$$

et

$$b^{(2)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} - m_{21} b_1^{(1)} \\ b_3^{(1)} - m_{31} b_1^{(1)} \end{bmatrix} = \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix} \tag{3.34}$$

La deuxième étape de l'algorithme consiste à

1. calculer le multiplicateur
$$m_{32}^{(2)} = \frac{a_{32}^{(2)}}{a_{22}^{(2)}}; \tag{3.35}$$
2. remplacer la troisième ligne par la différence entre elle-même et la deuxième ligne multipliée par m_{32}
$$l_3^{(3)} \leftarrow l_3^{(2)} - m_{32} l_2^{(2)}. \tag{3.36}$$

Au bout de la deuxième itération le système $A^{(2)}x = b^{(2)}$ est transformé dans le système équivalent $A^{(3)}x = b^{(3)}$ où

$$A^{(3)} = \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} - m_{32} a_{22}^{(2)} & a_{33}^{(2)} - m_{32} a_{23}^{(2)} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(2)} - m_{32} a_{23}^{(2)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(3)} & a_{12}^{(3)} & a_{13}^{(3)} \\ 0 & a_{22}^{(3)} & a_{23}^{(3)} \\ 0 & 0 & a_{33}^{(3)} \end{bmatrix} \tag{3.37}$$

et

$$b^{(3)} = \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} - m_{32} b_2^{(2)} \end{bmatrix} = \begin{bmatrix} b_1^{(3)} \\ b_2^{(3)} \\ b_3^{(3)} \end{bmatrix} \tag{3.38}$$

Le système $A^{(3)}x = b^{(3)}$ est triangulaire supérieur et peut être facilement résolu par l'algorithme de substitution rétrograde (3.20).

Exemple.
Considérons la résolution par élimination de Gauss du système linéaire (3.24). Nous calculons à la première étape

$$m_{21} = \frac{3}{4} \quad \text{et} \quad m_{31} = \frac{1}{2}. \tag{3.39}$$

Il s'ensuit que

$$A^{(2)} = \begin{bmatrix} 4 & 8 & 12 \\ 0 & 2 & 4 \\ 0 & 5 & 12 \end{bmatrix} \quad \text{et} \quad b^{(2)} = \begin{bmatrix} 4 \\ 2 \\ 9 \end{bmatrix}. \tag{3.40}$$

Puisque à la deuxième étape $m_{32}^{(2)} = \frac{5}{2}$, on obtient

$$U = A^{(3)} = \begin{bmatrix} 4 & 8 & 12 \\ 0 & 2 & 4 \\ 0 & 0 & 2 \end{bmatrix} \quad b^{(3)} = \begin{bmatrix} 4 \\ 2 \\ 4 \end{bmatrix} \tag{3.41}$$

On trouve ainsi la solution $x = [1, -3, 2]$.

•

3.4.1.2 Cas général

En général, pour un système de taille n, les opérations effectuées à l'étape k, $k = 1, \ldots, n-1$ sont :

- calcul des multiplicateurs

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \qquad i = k+1, \ldots, n; \tag{3.42}$$

- modification de la matrice $A^{(k)}$

$$a_{ij}^{(k+1)} \leftarrow a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}, \qquad i = k+1, \ldots, n, \quad j = k, \ldots, n; \tag{3.43}$$

- modification du seconde membre $b^{(k)}$

$$b_i^{(k+1)} \leftarrow b_i^{(k)} - m_{ik} b_k^{(k)}, \qquad i = k+1, \ldots, n. \tag{3.44}$$

On obtient ainsi un système équivalent $A^{(k+1)}x = b^{(k+1)}$. Au bout de l'étape $k = n-1$, la matrice $A^{(n)} = U$ est triangulaire supérieure.

3.4.2 Analyse de complexité

Afin de faciliter la présentation, considérons le cas d'un système d'ordre 3.

Considérons la première étape $k = 1$. Cette étape transforme les matrices $A^{(k)}, b^{(k)}$ en $A^{(k+1)}, b^{(k+1)}$ en effectuant

- $2 = (n-k)$ divisions (calcul des 2 multiplicateurs (3.30)) ;
- $3 \cdot 2 = (n-k+1) \cdot (n-k)$ multiplications (produit de la première ligne par m_{21} et m_{31}) ;
- $3 \cdot 2 = (n-k+1) \cdot (n-k)$ soustractions (soustraction de la première ligne pré multipliée).

En utilisant les relations suivantes

$$\sum_{k=1}^{n-1} (n-k) = \frac{n(n-1)}{2}, \qquad \sum_{k=1}^{n-1} (n-k)(n-k+1) = \frac{n^3}{3} - \frac{n}{3} \tag{3.45}$$

on obtient que l'élimination de Gauss demande

- $n(n-1)/2$ divisions,
- $n^3/3 - n/3$ multiplications,
- $n^3/3 - n/3$ additions (ou soustractions)

pour un système d'ordre n.

En ajoutant le coût (n^2) de la méthode de substitution, la complexité globale de la méthode de Gauss s'élève à

$$2\frac{n^3}{3} + 3\frac{n^2}{2} - 7\frac{n}{6} \tag{3.46}$$

opérations en virgule flottante.

3.5 La factorisation LU

La méthode de Gauss transforme le système initial $Ax = b$ en un système triangulaire équivalent $Ux = b^{(n)}$.

Dans cette section nous allons montrer que la méthode de Gauss peut aussi être utilisée pour transformer la matrice des coefficients A en le produit d'une matrice triangulaire inférieure L et une matrice triangulaire supérieure U

$$A = LU. \tag{3.47}$$

On parle dans ce cas de *factorisation* de la matrice A.

Considérons le cas simple d'une matrice $A_{(3\times 3)}$ de la forme (3.28). Après la première étape de la méthode de Gauss la matrice des coefficients prend la forme $A^{(2)}$ donnée par (3.33).

On peut aisément vérifier que la matrice $A^{(2)}$ peut être écrite comme le produit

$$A^{(2)} = M_1 A^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ -m_{31} & 0 & 1 \end{bmatrix} A^{(1)} \tag{3.48}$$

où les termes m_{21} et m_{31} sont les multiplicateurs définis par (3.30).

De façon similaire, on peut montrer la relation

$$U = A^{(3)} = M_2 A^{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -m_{32} & 1 \end{bmatrix} A^{(2)} = M_2 M_1 A. \tag{3.49}$$

Dans le cas général,

$$U = M_{n-1} M_{n-2} \dots M_1 A. \tag{3.50}$$

Si nous posons

$$L = (M_{n-1} M_{n-2} \dots M_1)^{-1} = M_1^{-1} \dots M_{n-2}^{-1} M_{n-1}^{-1} \tag{3.51}$$

on a alors la factorisation $A = LU$.

Les matrices M_k, $k = 1, \dots, n-1$, ont la forme d'une matrice identité d'ordre n où la $k$ème colonne est remplacée par

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ -m_{k+1,k} \\ \vdots \\ -m_{n,k} \end{bmatrix} \tag{3.52}$$

et satisfont la propriété suivante

$$M_k^{-1} = 2I_n - M_k. \tag{3.53}$$

Ceci signifie que les matrices M_k^{-1} ont la forme d'une matrice identité d'ordre n où la $k$ème colonne est remplacée par

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ m_{k+1,k} \\ \vdots \\ m_{n,k} \end{bmatrix} \tag{3.54}$$

Il s'ensuit que la matrice L prend la forme triangulaire inférieure

$$L = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ m_{21} & 1 & 0 & 0 & 0 \\ \dots & m_{32} & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{n1} & m_{n2} & \dots & m_{n,n-1} & 1 \end{bmatrix} \tag{3.55}$$

La factorisation de la matrice des coefficients A en le produit d'une matrice triangulaire inférieure et une matrice triangulaire supérieure permet la résolution du système linéaire $Ax = b$. En fait, résoudre $Ax = b = LUx$ équivaut à résoudre 2 systèmes triangulaires

$$Ly = b \tag{3.56}$$
$$Ux = y \tag{3.57}$$

On peut facilement remarquer que le coût de la résolution du système linéaire en utilisant la factorisation LU reste d'ordre cubique $O(n^3)$.

3.5.1 La méthode de Gauss vs. la factorisation LU

Nous avons mis en évidence dans la section précédente que le coût de la méthode d'élimination de Gauss et celui de la méthode de factorisation LU sont équivalents.

Alors, quel est l'avantage d'utiliser la factorisation LU ?

Supposons qu'on doit résoudre p systèmes linéaires gouvernés par la même matrice A mais avec des seconds membres différents :

$$Ax_l = b_l, \qquad l = 1, \ldots, p. \tag{3.58}$$

La résolution des p systèmes peut être faite de deux façons différentes :

1. appliquer p fois l'algorithme de Gauss,
2. calculer *une seule fois* la factorisation LU et résoudre p systèmes de la forme

$$Ly_l = b_l \tag{3.59}$$
$$Ux_l = y_l \tag{3.60}$$

Les deux alternatives sont équivalentes en termes de solution numérique mais affichent une complexité de calcul différente.

Dans le premier cas le coût s'élève à p fois un coût $O(n^3 + n^2)$.

Dans le second cas nous devons répéter p fois la partie moins lourde du calcul, c.-à-d. la résolution des 2 système triangulaires. Ceci entraîne un coût de l'ordre $O(n^3 + pn^2)$.

3.5.2 Le calcul du déterminant

Soit A une matrice carrée d'ordre n qui a été factorisée en le produit $A = LU$ où L est triangulaire inférieure avec $l_{ii} = 1$ et U est triangulaire supérieure.

Nous nous intéressons au calcul du déterminant de la matrice A. En vertu des propriétés des déterminants (Section A.4),

$$\det(A) = \det(L)\det(U) = \det(U) = u_{11}u_{22}\ldots u_{nn}. \tag{3.61}$$

Donc, pour calculer le déterminant d'une matrice A on recourt généralement à la factorisation LU.

Remarque `MATLAB`© La fonction `det` en `MATLAB`©calcule le déterminant d'une matrice carrée à partir des déterminants des matrices L et U.

•

3.5.3 Existence et unicité de la factorisation

Le théorème suivant énonce la condition nécessaire et suffisante afin que la matrice A soit factorisable et que la factorisation soit unique.

Théorème 3.2. *La factorisation LU de $A_{(n\times n)}$ avec $l_{ii} = 1$ existe et est unique si et seulement si les sous-matrices principales de A sont inversibles, c.-à-d.*

$$\det(A(1:k, 1:k)) \neq 0 \text{ pour } k = 1, \ldots, n-1. \tag{3.62}$$

Corollaire 3.1. *Si A est une matrice à diagonale dominante par ligne ou par colonne, alors la factorisation LU de A existe et est unique.*

La définition de matrice à diagonale dominante est présentée dans la Section (A.12).

Exemple. Notons que le théorème (3.2) mentionne la non-singularité des sous-matrices principales mais pas la singularité de la matrice A. Dans l'exemple qui suit, nous verrons 3 matrices singulières, dont seulement une n'admet pas de factorisation LU.

1. la matrice singulière
$$A = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} = LU \tag{3.63}$$
admet la fatorisation LU.
2. la matrice singulière
$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{3.64}$$
n'admet pas la factorisation LU.
3. la matrice singulière
$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.65}$$
admet la factorisation LU.

•

3.6 Les méthodes directes pour la factorisation

Les méthodes directes sont des techniques alternatives pour factoriser une matrice A. Ces méthodes nécessitent moins de résultats intermédiaires que la méthode présentée dans la Section (3.5).

Le principe sous-jacent aux méthodes directes est qu'une factorisation LU est formellement équivalente à résoudre le produit matriciel $A = LU$, c.-à-d. le système linéaire

$$a_{ij} = \sum_{r=1}^{n} l_{ir} u_{rj}. \tag{3.66}$$

Ce système a n^2 équations et $n^2 + n$ inconnues. Afin de pouvoir le résoudre, nous devons fixer arbitrairement la valeur de n coefficients.

On adopte deux strategies qui mènent à deux méthodes directes de factorisation :

Méthode de Doolittle où les n termes diagonaux de L sont fixés à 1.

Méthode de Crout où les n termes diagonaux de U sont fixés à 1.

Voyons comment l'algorithme de Doolittle fonctionne. Considérons le produit matriciel

$$\begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ l_{21} & 1 & 0 & 0 & \dots \\ l_{31} & l_{32} & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{n,n-1} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1,n-1} & u_{1n} \\ 0 & u_{22} & \dots & u_{2,n-1} & u_{2n} \\ \dots & 0 & \ddots & \dots & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & u_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1,n-1} & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2,n-1} & \dots \\ \dots & \dots & \ddots & \dots & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & a_{nn} \end{bmatrix} \tag{3.67}$$

où le deuxième membre est donné et le nombre d'inconnues au premier membre est égal à n^2.

Nous allons maintenant écrire les produits ligne-colonne qui fournissent les éléments de la première ligne de A, c-à-d. $A(1, 1:n)$.

La première relation que nous pouvons écrire en effectuant le produit de la première ligne de L par la première colonne de U est

$$u_{11} = a_{11}. \tag{3.68}$$

En effectuant le produit de la première ligne de L par la deuxième colonne de U on a

$$u_{12} = a_{12}. \tag{3.69}$$

En continuant de cette manière (première ligne de L par les différentes colonnes de U), nous obtenons

$$u_{13} = a_{13} \tag{3.70}$$

$$\dots\dots \tag{3.71}$$

$$u_{1n} = a_{1n} \tag{3.72}$$

Nous passons maintenant aux termes restants de la première colonne de A, c.-à-d. $A(2:n; 1)$. Nous pouvons donc écrire les relations suivantes

$$l_{21}u_{11} = a_{21} \Rightarrow l_{21} = \frac{a_{21}}{u_{11}} \tag{3.73}$$

$$,\dots\dots \tag{3.74}$$

$$l_{n1}u_{11} = a_{n1} \Rightarrow l_{n1} = \frac{a_{n1}}{u_{11}}. \tag{3.75}$$

Il est important de remarquer que la résolution des équations (3.73),(3.74),(3.75) est possible en vertu des étapes précédentes (3.68), (3.69), (3.70).

La méthode continue de la même manière avec la sous-matrice $A(2:n, 2:n)$.

La méthode de Doolitte peut dès lors être résumée par le pseudo-code qui suit :

```
for k=1:n
    for j=k:n
```

$$u_{kj} = a_{kj} - \sum_{r=1}^{k-1} l_{kr} u_{rj}$$

```
    end
    for i=k+1:n
```

$$l_{ik} = \frac{1}{u_{kk}} \left(a_{ik} - \sum_{r=1}^{k-1} l_{ir} u_{rk} \right)$$

```
    end
end
```

Notons que la méthode de Doolittle fournit d'abord la $k$ème ligne de U, puis la $k$ème colonne de L.

La méthode de Crout s'obtient de façon similaire et peut être résumée par le pseudo-code qui suit :

```
for k=1:n
    for i=k:n
```

$$l_{ik} = a_{ik} - \sum_{r=1}^{k-1} l_{ir} u_{rk}$$

```
    end
    for j=k+1:n
```

$$u_{kj} = \frac{1}{l_{kk}} \left(a_{kj} - \sum_{r=1}^{k-1} l_{kr} u_{rj} \right)$$

```
    end
end
```

Notons que la méthode de Crout fournit d'abord la $k$ème colonne de L, puis la $k$ème ligne de U.

3.7 La factorisation de Choleski

Cette factorisation s'applique aux matrices symétriques définies positives (Section A.12). D'abord, nous présentons le théorème suivant :

Théorème 3.3. *Soit $A_{(n\times n)}$ une matrice symétrique définie positive. Alors il existe une unique matrice triangulaire supérieure H dont les termes diagonaux sont positifs telle que*

$$A = H^T H \tag{3.76}$$

Une fois la matrice A factorisée sous la forme $H^T H$, la résolution du système $Ax = b$ peut être effectuée en résolvant les 2 systèmes triangulaires

$$H^T y = b \tag{3.77}$$

$$Hx = y \tag{3.78}$$

Nous allons présenter maintenant l'algorithme de factorisation de Choleski.

La factorisation de Choleski peut être représentée par la multiplication

$$\begin{bmatrix} h_{11} & 0 & 0 & \dots & 0 \\ h_{12} & h_{22} & 0 & \dots & 0 \\ h_{13} & h_{23} & h_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{1n} & h_{2n} & h_{3n} & \dots & h_{nn} \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1n} \\ 0 & h_{22} & h_{23} & \dots & h_{2n} \\ 0 & 0 & h_{33} & \dots & h_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & h_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{12} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{13} & a_{23} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & a_{3n} & \dots & a_{nn} \end{bmatrix} \tag{3.79}$$

Nous allons écrire les produits ligne-colonne qui fournissent les éléments de la première colonne de A ($A(1, 1 : n)$) :

$$h_{11}^2 = a_{11} \Rightarrow h_{11} = \sqrt{a_{11}} \tag{3.80}$$

$$h_{12}h_{11} = a_{12} \Rightarrow h_{12} = \frac{a_{12}}{h_{11}} \tag{3.81}$$

$$\dots\dots \tag{3.82}$$

En passant à la deuxième colonne (en fait on ne considère que le sous-vecteur $A(2 : n, 2)$ pour des raisons de symétrie) on obtient

$$h_{12}^2 + h_{22}^2 = a_{22} \Rightarrow h_{22} = \sqrt{a_{22} - h_{12}^2} \tag{3.83}$$

$$h_{13}h_{12} + h_{23}h_{22} = a_{23} \Rightarrow h_{23} = \frac{a_{23} - h_{13}h_{12}}{h_{22}} \dots \qquad \dots \tag{3.84}$$

et ainsi de suite.

La méthode de factorisation de Choleski peut être alors résumé par le pseudo-code qui suit :

```
for i=1:n
```

$$h_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} h_{ki}^2} \tag{3.85}$$

```
    for j=i+1:n
```

$$h_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} h_{kj}h_{ki}}{h_{ii}}$$

```
    end
end
```

Notons que, puisque la matrice A est définie positive, les termes sous la racine (3.85) sont toujours positifs.

En termes de complexité on peut montrer que l'algorithme demande $O(n^3/6)$ multiplications, $O(n^3/6)$ additions, $O(n^2/2)$ divisions et n racines carrées. Donc, même si encore d'ordre cubique, la complexité affichée est $O(n^3/3)$ par rapport au $O(2n^3/3)$ des autres méthodes de factorisation .

Toutefois, le bénéfice le plus important de la méthode de Cholesky tient a son occupation réduite de mémoire. En fait il est possible de stocker à la fois la matrice A et la factorisation H dans la même location mémoire de A : par exemple on peut enregistrer H dans la partie triangulaire inférieure et conserver A (symétrique) dans la partie triangulaire supérieure. Les termes diagonaux de A peuvent être aisément récupérés par la relation

$$a_{ii} = h_{ii}^2 + \sum_{k=1}^{i-1} h_{ki}^2. \tag{3.86}$$

Exemple. Voici un exemple de factorisation de Cholesky d'une matrice symétrique définie positive

$$A = \begin{bmatrix} 4 & 2 & 2 \\ 2 & 5 & 3 \\ 2 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.87}$$

•

3.8 Le calcul de l'inverse

Le calcul de l'inverse X d'une matrice non singulière $A_{(n\times n)}$ peut être effectué en utilisant la factorisation LU.

Puisque $AX = I$, les vecteurs colonnes x_i de X doivent satisfaire les n systèmes linéaires

$$Ax_i = e_i \quad i = 1, \ldots, n \tag{3.88}$$

où e_i est le $i$ème vecteur colonne de la matrice unité de taille n.

En utilisant la factorisation LU le coût du calcul d'une matrice inverse s'élève à $O(n^3) + nO(n^2)$. On en déduit que le calcul de l'inverse d'une matrice est une opération coûteuse.

3.9 Les méthodes de changement de pivot

Les méthodes de Gauss et la factorisation LU sont basées sur le calcul des multiplicateurs. A la $k$ème ($k = 1, \ldots, n-1$) itération, les deux méthodes demandent le calcul de $n - k$ multiplicateurs

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \qquad i = k+1, \ldots, n \tag{3.89}$$

où les termes diagonaux $a_{kk}^{(k)}$ (qui correspondent aussi aux termes diagonaux de U) sont appelés *pivots*.

Les méthodes échouent si $a_{kk}^{(k)} = 0$, c.-à-d. qu'elles s'arrêtent en cas d'annulation des pivots.

Afin d'éviter cette situation, deux stratégies ont été proposées pour la méthode de Gauss :

Changement de pivot partiel : si à la $k$ème étape $a_{kk}^{(k)} = 0$, cette technique

1. cherche le plus grand terme (en valeur absolue) non nul dans le vecteur $A(k+1:n, k)$

$$p^* = \arg \max_{p=k+1,\ldots,n} |a_{pk}^{(k)}| \tag{3.90}$$

2. échange la $k$ème ligne et la p^*ème ($k \leftrightarrow p^*$) dans la matrice $A^{(k)}$ et le vecteur $b^{(k)}$.

Changement de pivot total : si à la $k$ème étape $a_{kk}^{(k)} = 0$, cette technique

1. recherche le plus grand terme (en valeur absolue) non nul dans la matrice $A(k:n, k:n)$
$$[p^*, q^*] = \arg \max_{\substack{p=k+1,\ldots,n \\ q=k+1,\ldots,n}} |a_{pq}^{(k)}| \tag{3.91}$$
2. échange la $k$ème ligne et la p^*ème ($k \leftrightarrow p^*$) dans la matrice $A^{(k)}$ et le vecteur $b^{(k)}$,
3. échange la $k$ème colonne et la q^*ème colonne ($k \leftrightarrow q^*$) dans la matrice $A^{(k)}$.

Il faut remarquer que si on échange la $k$ème colonne et la q^*ème dans la matrice $A^{(k)}$, alors la position des variables inconnues x_k et x_{q^*} dans la solution finale sera aussi échangée.

Exemple. Considérons le système linéaire $Ax = b$ où

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 7 & 8 & 9 \end{bmatrix}, \qquad b = \begin{bmatrix} 6 \\ 11 \\ 24 \end{bmatrix} \tag{3.92}$$

Suite à la première itération

$$A^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & -1 \\ 0 & -6 & -12 \end{bmatrix} \qquad b^{(2)} = \begin{bmatrix} 6 \\ -1 \\ -18 \end{bmatrix} \tag{3.93}$$

le pivot $a_{22}^{(2)}$ est nul.

Si nous appliquons la stratégie de changement de pivot partiel les matrices deviennent

$$A^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -6 & -12 \\ 0 & 0 & -1 \end{bmatrix} \qquad b^{(2)} = \begin{bmatrix} 6 \\ -18 \\ -1 \end{bmatrix} \tag{3.94}$$

Dans ce cas la matrice $A^{(2)}$ est déjà triangulaire et la méthode de substitution fournit la solution.

Si nous appliquons la stratégie de changement de pivot total les matrices deviennent

$$A^{(2)} = \begin{bmatrix} 1 & 3 & 2 \\ 0 & -12 & -6 \\ 0 & -1 & 0 \end{bmatrix} \qquad b^{(2)} = \begin{bmatrix} 6 \\ -18 \\ -1 \end{bmatrix} \tag{3.95}$$

Dans ce cas une troisième étape est nécessaire avant d'obtenir la forme triangulaire.

•

Le stratégies de changement de pivot partiel et total peuvent être appliquées aussi à la méthode de factorisation. La recherche du pivot s'effectue de la même manière mais son utilisation est différente.

Pour introduire le sujet, nous définissons la notion de *matrice de permutation*, comme étant une matrice identité où soit deux lignes soit deux colonnes ont été permutées.

Soit A une matrice carrée. La permutation de deux lignes i et j dans A fournit une matrice $A_{(i\leftrightarrow j)}$ qui peut être écrite sous la forme

$$A_{(i\leftrightarrow j)} = P_{(i\leftrightarrow j)}A \tag{3.96}$$

où $P_{(i\leftrightarrow j)}$ est obtenue à partir de la matrice identité en permutant la ligne i et la ligne j.

Exemple Soit

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 7 & 8 & 9 \end{bmatrix} \tag{3.97}$$

et $A_{(2\leftrightarrow 3)}$ la matrice obtenue en permutant les lignes 2 et 3. On voit aisément que

$$A_{(2\leftrightarrow 3)} = \begin{bmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \\ 2 & 4 & 5 \end{bmatrix} = P_{(2\leftrightarrow 3)}A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 7 & 8 & 9 \end{bmatrix} \tag{3.98}$$

•

Soit A une matrice carrée. La permutation de deux lignes i et j et de deux colonnes k et l dans A fournit une matrice $A_{(i\leftrightarrow j,k\leftrightarrow l)}$ qui peut être écrite sous la forme

$$A_{(i\leftrightarrow j,k\leftrightarrow l)} = P_{(i\leftrightarrow j)}AQ_{(k\leftrightarrow l)} \tag{3.99}$$

où $P_{(i\leftrightarrow j)}$ est obtenue à partir de la matrice identité en permutant la ligne i et la ligne j et $Q_{(k\leftrightarrow l)}$ est obtenue à partir de la matrice identité en permutant la colonne k et la colonne l.

Exemple Soit

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 7 & 8 & 9 \end{bmatrix} \tag{3.100}$$

et $A_{(2\leftrightarrow 3,1\leftrightarrow 2)}$ la matrice obtenue en permutant les lignes 2 et 3 et les colonnes 1 et 2. On voit que

$$A_{(2\leftrightarrow 3,1\leftrightarrow 2)} = \begin{bmatrix} 2 & 1 & 3 \\ 8 & 7 & 9 \\ 4 & 2 & 5 \end{bmatrix} = P_{(2\leftrightarrow 3)}AQ_{(1\leftrightarrow 2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.101}$$

•

Voyons en détail les 2 types de pivoting pour la factorisation LU.

LU et changement de pivot partiel : si à la $k$ème étape $a_{kk}^{(k)} = 0$, cette technique

1. cherche le plus grand terme (en valeur absolue) non nul dans le vecteur $A(k+1:n,k)$ (formule (3.90)),
2. remplace la matrice $A^{(k)}$ par $P_kA^{(k)}$ où $P_k = P_{(k\leftrightarrow p^*)}$.

Au bout de la factorisation, on obtient

$$M_{n-1}P_{n-1}M_{n-2}P_{n-2}\dots M_1P_1A = U \tag{3.102}$$

Si on pose

$$M = M_{n-1}P_{n-1}M_{n-2}P_{n-2}\dots M_1P_1 \tag{3.103}$$

$$P = P_{n-1}P_{n-2}\dots P_1 \tag{3.104}$$

alors

$$MP^{-1}PA = U \tag{3.105}$$

c.à.d. la factorisation résultante est $PA = LU$ où $L = PM^{-1}$.

Considérons donc un système linéaire $Ax = b$ et supposons que le pivoting partiel ait été effectué pendant la résolution. Donc nous avons à disposition les trois matrices P, L et U telles que $PA = LU$. Afin de résoudre le système initial nous passons par le système équivalent

$$PAx = LUx = Pb \tag{3.106}$$

Pour obtenir la solution x il nous suffit donc de résoudre les deux systèmes triangulaires $Ly = Pb$ et $Ux = y$.

LU et changement de pivot total : si à la $k$ème étape $a_{kk}^{(k)} = 0$,cette technique

1. cherche le plus grand terme (en valeur absolue) non nul dans la matrice $A(k : n, k : n)$ (formule 3.91),
2. remplace la matrice $A^{(k)}$ par $P_k A^{(k)} Q_k$ où $P_k = P_{(k \leftrightarrow p^*)}$ et $Q_k = Q_{(k \leftrightarrow q^*)}$.

Au bout de la factorisation, on obtient

$$M_{n-1}P_{n-1}M_{n-2}P_{n-2}\dots M_1P_1AQ_1\dots Q_{n-2}Q_{n-1} = U \tag{3.107}$$

Si on pose

$$M = M_{n-1}P_{n-1}M_{n-2}P_{n-2}\dots M_1P_1 \tag{3.108}$$

$$P = P_{n-1}P_{n-2}\dots P_1 \tag{3.109}$$

$$Q = Q_1Q_2\dots Q_{n-1} \tag{3.110}$$

alors

$$MP^{-1}PAQ = U \tag{3.111}$$

c.-à-d. la factorisation est $PAQ = LU$ où $L = PM^{-1}$.

Considérons le système $Ax = b$ et supposons que le pivoting totale ait été effectué pendant la résolution. Donc nous avons à disposition les matrices P, Q, L et U telles que $PAQ = LU$. Or, nous définissons la variable z telle que $x = Qz$. Le système $Ax = b$ peut être écrit sous la forme $AQz = b$ et il est équivalent au système

$$PAQz = LUz = Pb \tag{3.112}$$

ayant z pour solution. Afin d'obtenir z, nous allons résoudre les deux systèmes triangulaires $Ly = Pb$ et $Uz = y$. Finalement, il suffit de calculer $x = Qz$ pour obtenir x.

Remarque La discussion sur les méthodes de changement de pivot suppose que la matrice A est non singulière. Dans le cas singulier, les itérations de la méthode de Gauss transforment nécessairement la matrice des coefficients en une matrice avec tous les pivots égaux à zéro. Par exemple si A est singulière

$$A^{(1)} = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 4 & -2 \\ 3 & 6 & -3 \end{bmatrix} \tag{3.113}$$

alors après la première étape

$$A^{(2)} = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.114}$$

où on voit que tous les pivots sont égaux à zéro.

•

Jusqu'ici nous avons proposé les méthodes de changement de pivot comme des techniques pour éviter l'arrêt des méthodes de résolution. En réalité, on peut voir qu'une petite valeur non nulle de $a_{kk}^{(k)}$ peut amplifier les erreurs d'arrondi affectant ainsi la précision de la solution finale.

En fait, il se trouve que les méthodes de changement de pivot sont adoptées par défaut dans les algorithmes numériques de résolution linéaire, même en absence de pivots nuls, afin d'améliorer la précision de calcul.

Voyons un exemple de leur utilité.

Exemple Considérons le système linéaire

$$\begin{cases} 0.000100x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases} \tag{3.115}$$

La solution en arithmétique exacte est

$$\begin{cases} x_1 = 1.00010 \\ x_2 = 0.99990 \end{cases} \tag{3.116}$$

Toutefois, si nous effectuons les calculs en virgule flottante avec $b = 10$ et $t = 3$ le résultat obtenu

$$\begin{cases} \hat{x}_1 = 0 \\ \hat{x}_2 = 1 \end{cases} \tag{3.117}$$

est entaché par une grande erreur.

Au contraire, en inversant les deux équations selon le principe du changement de pivot partiel, le calcul en virgule flottante fournit une bien meilleure approximation

$$\begin{cases} \hat{x}_1 = 1 \\ \hat{x}_2 = 1 \end{cases} \tag{3.118}$$

•

Remarque `MATLAB`© L'opérateur \ (backslash en anglais) permet une approche unifiée à la solution d'un système linéaire. La résolution du système $Ax = b$ est représentée en `MATLAB`©par la commande

```
>> x=A\b
```

L'outil `MATLAB`©sélectionne automatiquement la procédure de résolution la plus indiquée sur la base du type de matrice. Par exemple, si A est triangulaire la solution est calculée par substitution, si A est symétrique et définie positive la solution est obtenue via la décomposition de Cholesky.

Notons aussi que l'inverse X d'une matrice A peut être obtenue par la commande

```
>> X=A\eye(size(A))
```

•

3.10 L'analyse de stabilité

Comme montré dans l'exemple précedent, la résolution d'un système linéaire par une méthode numérique sur ordinateur conduit inévitablement à l'introduction, la propagation et la génération d'erreurs.

A cause des erreurs d'arrondi, une méthode numérique de résolution d'un système linéaire ne fournit jamais la solution exacte x mais toujours une solution approchée $\hat{x} = x + \delta x$.

Afin d'éviter la détérioration de la solution, il est donc nécessaire de considérer des problèmes biens conditionnés autant que des algorithmes stables.

Cette section présentera quelques résultats théoriques sur le conditionnement d'un problème linéaire et sur la stabilité des méthodes discutées auparavant. Voyons d'abord la notion de conditionnement d'une matrice.

3.10.1 Le conditionnement d'une matrice

Définition 19 (Conditionnement d'une matrice)**.** *Le* conditionnement *d'une matrice est défini par*

$$\kappa_p(A) = \|A\|_p \|A^{-1}\|_p \tag{3.119}$$

où $\|\cdot\|_p$ *est une norme matricielle (Section (A.11)) .*

Le conditionnement d'une matrice A exprime la sensibilité de la solution du système $Ax = b$ aux perturbations des données. En d'autres termes, la solution est autant plus sensible que $\kappa_p(A)$ est grand. Notons aussi que cette sensibilité est indépendante de la méthode choisie pour résoudre le problème.

Le conditionnement d'une matrice A satisfait les propriétés suivantes :

- le conditionnement de A est toujours plus grand ou égal à 1 :

$$1 = \|A \cdot A^{-1}\| \leq \|A\| \cdot \|A^{-1}\| = \kappa(A); \tag{3.120}$$

- $\kappa(aA) = \kappa(A)$, $\forall a \in \mathbb{R}$;
- $\kappa(A) = \kappa(A^{-1})$;
- si A est une matrice orthogonale (Section A.3) alors $\kappa_2(A) = 1$;
- pour $p = 2$

$$\kappa_2(A) = \frac{\sigma_1(A)}{\sigma_n(A)} \tag{3.121}$$

  où $\sigma_1(A)$ est la plus grande valeur singulière de A (Section A.9) et $\sigma_n(A)$ est la plus petite ;
- si A est singulière alors $\kappa_2(A) = \infty$;
- si A est symétrique définie positive

$$\kappa_2(A) = \frac{\lambda_{\max}}{\lambda_{\min}} = \rho(A)\rho(A^{-1}) \tag{3.122}$$

  où $\lambda_{\max}$ ($\lambda_{\min}$) est la plus grande (resp. la plus petite) valeur propre de A et ρ est le rayon spectral de A (Section A.6).

Exemple. Un exemple connu de matrice mal conditionnée est la matrice de *Hilbert*

$$H_{(n\times n)} = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{bmatrix} \tag{3.123}$$

Voici la valeur du conditionnement $\kappa_2(H_{(n\times n)})$ pour différentes valeurs de n

n	2	3	4	5	6
$\kappa_2(H_n)$	$1.2 \cdot 10^1$	$1.9 \cdot 10^2$	$6.5 \cdot 10^3$	$1.8 \cdot 10^5$	$4.4 \cdot 10^6$

•

Le calcul du conditionnement $\kappa(A)$ requiert le calcul de la norme de l'inverse de A. Comme vu dans la Section 3.8, le calcul de l'inverse d'une matrice est extrêmement coûteux et peu stable. Pour éviter cela, des algorithmes approchés pour estimer le conditionnement ont été proposés. Un exemple est l'algorithme implémenté dans la librairie LAPACK et l'algorithme `rcond` qui estime l'inverse du conditionnement en `MATLAB`©.

3.10.2 L'analyse a priori

Nous allons présenter les résultats théoriques de l'analyse de stabilité a priori des méthodes de résolution d'un système linéaire.

Comme discuté en Section 2.10, l'analyse a priori se répartit en analyse directe et rétrograde.

3.10.2.1 L'analyse directe à priori

L'analyse a priori traite de la sensibilité de la solution aux perturbations des données.

Dans ce qui suit, nous négligeons l'erreur de génération dû à l'algorithme et nous nous bornerons à considérer que l'erreur de propagation (Section 2.5.1).

Supposons que les perturbations des données δd introduisent des modifications des matrices A et b

$$A \to A + \delta A, \quad b \to b + \delta b$$

et par conséquence de la solution x

$$(A + \delta A)(x + \delta x) = b + \delta b$$

On peut montrer les théorèmes suivants.

Théorème 3.4. *Soit $A_{(n\times n)}$ une matrice inversible et $\|A^{-1}\|\|\delta A\| < 1$, alors*

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|} \right) \tag{3.124}$$

où $\kappa(A)$ est le conditionnement de A.

Théorème 3.5. *Si les conditions du Théorème 3.4 sont remplies et $\delta A = 0$, on peut montrer que*

$$\frac{1}{\kappa(A)} \frac{\|\delta b\|}{\|b\|} \leq \frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\delta b\|}{\|b\|} \tag{3.125}$$

Théorème 3.6. *Supposons que $\|\delta A\| \leq \gamma\|A\|$, $\|\delta b\| \leq \gamma\|b\|$, avec $\gamma \in \mathbb{R}^+$. Alors si $\gamma\kappa(A) < 1$ on a les inégalités suivantes :*

$$\frac{\|x + \delta x\|}{\|x\|} \leq \frac{1 + \gamma\kappa(A)}{1 - \gamma\kappa(A)}, \qquad \frac{\|\delta x\|}{\|x\|} \leq \frac{2\gamma}{1 - \gamma\kappa(A)}\kappa(A) \tag{3.126}$$

Remarque. Notons que la borne superieure en (3.125) peut être facilemenet dérivée de la manière suivante. Soit

$$A(x + \delta x) = b + \delta b$$

Grace à la relation $Ax = b$ on obtient

$$A\delta x = \delta b$$

et

$$\|\delta x\| = \|A^{-1}\delta b\| \leq \|A^{-1}\|\|\delta b\| \tag{3.127}$$

Puisque

$$Ax = b \Rightarrow \|A\|\|x\| \geq \|b\| \Rightarrow \frac{1}{\|x\|} \leq \frac{\|A\|}{\|b\|} \tag{3.128}$$

il s'ensuit de (3.127) et (3.128) la borne superieure en (3.125)

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A\|\|A^{-1}\|\|\delta b\|}{\|b\|} = \kappa(A)\frac{\|\delta b\|}{\|b\|} \tag{3.129}$$

•

Les théorèmes précédents indiquent qu'un grand conditionnement pourrait causer une forte imprécision de la solution. Voici un exemple pratique.

Exemple Considérons le système $Ax = b$ où

$$A = \begin{bmatrix} 7 & 10 \\ 5 & 7 \end{bmatrix}$$

Puisque

$$A^{-1} = \begin{bmatrix} -7 & 10 \\ 5 & -7 \end{bmatrix}$$

le conditionnement $\kappa_1(A) = \|A\|_1\|A^{-1}\|_1 = 17 \cdot 17 = 289 \gg 1$.

Pour $b = [1, 0.7]^T$, la solution est $x = [0, 0.1]^T$. On peut montrer que si le vecteur b change légèrement (par exemple $b = [1.01, 0.69]^T$), la solution subit une forte perturbation ($x = [-0.17, 0.22]$).

•

Remarque. En vertu des théorèmes énoncés, une matrice de coefficients avec un grand conditionnement n'empêche pas nécessairement le système linéaire d'être bien conditionné pour des choix particuliers du second membre b.

En revanche, le fait qu'un système linéaire soit bien conditionné n'implique pas nécessairement que la solution soit calculée avec précision. Comme discuté dans la Section 2.5.2, l'erreur finale dépend aussi de la stabilité de l'algorithme.

•

3.10.2.2 L'analyse rétrograde a priori

Dans cette section nous allons discuter la stabilité de quelques algorithmes en utilisant la technique rétrograde.

Nous commençons par un théorème sur les méthode de substitution (Section 3.3).

Théorème 3.7. *Soit $A_{(n\times n)}$ une matrice triangulaire non singulière. Soit $\hat{x}$ la solution approchée du système linéaire $Ax = b$ obtenue en virgule flottante (base b et nombre t de chiffres significatifs). La solution approchée $\hat{x}$ peut être considérée comme la solution exacte d'un système $(A + \delta A)x = b$ où*

$$|\delta A| \leq \gamma_n|A| = \frac{nu}{1 - nu}|A| \tag{3.130}$$

et où u est la précision machine (formule (2.24)).

D'après ce théorème, si la valeur u est petite, la précision de l'algorithme de substitution est très élevée. En plus, le résultat ne dépend pas de l'ordre des opérations.

En ce qui concerne l'algorithme de factorisation LU (Section 3.5) on peut montrer les deux théorèmes suivants :

Théorème 3.8. *Soit $A_{(n\times n)}$ une matrice non singulière. Supposons que A est factorisée en le produit LU par un algorithme numérique en virgule flottante (base b et nombre t de chiffres significatifs).*

Soient $\hat{L}$ et $\hat{U}$ les deux matrices résultant de la factorisation telles que

$$\hat{L}\hat{U} = A + \Delta$$

Alors on peut montrer que

1. *la perturbation Δ est bornée superieurement de la manière suivante*

$$|\Delta| \leq \gamma_n |\hat{L}||\hat{U}| \tag{3.131}$$

   où $\gamma_n = \frac{nu}{1-nu}$

2. *la solution approchée $\hat{x}$ peut être considérée comme la solution exacte d'un système $(A + \delta A)x = b$ où*

$$|\delta A| \leq 2\gamma_n |\hat{L}||\hat{U}| = 2\frac{nu}{1-nu}|\hat{L}||\hat{U}| \tag{3.132}$$

Notons que la présence de petits pivots pendant la factorisation peut rendre très grandes les valeurs des multiplicateurs en (3.55) et par conséquent rendre très grands les secondes membres des inégalités (3.131) et (3.132).

Théorème 3.9. *Soit $A_{(n\times n)}$ une matrice non singulière. Supposons que A est factorisée en le produit LU par un algorithme numérique en virgule flottante (base b et nombre t de chiffres significatifs).*

Soient $\hat{L}$ et $\hat{U}$ les deux matrices résultant de la factorisation telles que telles que

$$\hat{L}\hat{U} = A + \Delta$$

Supposons que $|\hat{L}||\hat{U}| = |\hat{L}\hat{U}|$ (par exemple $\hat{L}$ et U n'ont que des termes positifs). Alors on peut montrer que

1. *la perturbation Δ est bornée superieurement de la manière suivante*

$$|\Delta| \leq \frac{nu}{1-2nu}|A| \tag{3.133}$$

2. *la solution approchée $\hat{x}$ du système linéaire peut être considérée comme la solution exacte d'un système $(A + \delta A)x = b$ où*

$$|\delta A| \leq \frac{2\gamma_n}{1-\gamma_n}|A| = \frac{2nu}{1-2nu}|A| \tag{3.134}$$

et où u est la précision machine.

Notons comme, dans ce deuxième cas, les valeurs des bornes supérieures ne sont pas liées à la présence des petits pivots.

3.10.3 L'analyse a posteriori

Cette analyse a pour but de relier le résidu et la précision de la solution (Section 2.10).

La solution rendue par la méthode de Gauss a typiquement un petit résidu

$$r = b - A\hat{x}. \tag{3.135}$$

Néanmoins, cela ne garantit pas que l'erreur

$$e = x - \hat{x} \tag{3.136}$$

soit petite.

Le résidu peut être relié à l'erreur absolue grâce à la relation suivante :

$$\begin{aligned}\delta_{\hat{x}} = \|e\| = \|x - \hat{x}\| = \|A^{(-1)}b - A^{(-1)}(b - r)\| = \\ = \|A^{(-1)}r\| \leq \|A^{(-1)}\|\|A\|\frac{\|r\|}{\|A\|} = \kappa(A)\frac{\|r\|}{\|A\|}.\end{aligned} \tag{3.137}$$

Ceci conduit à la relation

$$\varepsilon_{\hat{x}} = \frac{\|x - \hat{x}\|}{\|x\|} \leq \kappa(A)\|r\|\frac{1}{\|A\|\|x\|}\frac{\|r\|}{\|b\|} \tag{3.138}$$

Les relations précédentes nous montrent que pour un grand conditionnement ($\kappa(A) \gg 1$), un petit résidu ne garantit pas une petite erreur.

Exemple. Considérons le système linéaire $Ax = b$ où

$$A = \begin{bmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{bmatrix} \quad b = \begin{bmatrix} 0.8642 \\ 0.1440 \end{bmatrix} \tag{3.139}$$

La solution exacte est $x = [2, -2]^T$.

Considérons la solution approchée $\hat{x} = [0.9911, -0.4870]^T$. Cette solution présente une large erreur mais son résidu

$$r = b - A\hat{x} = \begin{bmatrix} -10^{-8} \\ 10^{-8} \end{bmatrix}$$

est très petit.

Ceci est dû au grand conditionnement de la matrice A. Puisque

$$A^{(-1)} = 10^8 \begin{bmatrix} 0.1441 & -0.8648 \\ -0.2161 & 1.2969 \end{bmatrix}$$

le conditionnement

$$\kappa_\infty(A) = \|A\|_\infty\|A^{(-1)}\|_\infty \approx 2.16 \cdot 1.51 \cdot 10^8 \approx 3.26 \cdot 10^8$$

•

3.11 Raffinement itératif

La résolution d'un système linéaire en utilisant une machine introduit inévitablement des erreurs dans la solution. Par conséquent, l'algorithme au lieu de produire

la solution exacte x, fournit une solution approchée $\hat{x}$. La procédure de *raffinement itératif* a pour but la réduction de l'erreur de $\hat{x}$. La procédure se base sur les relations suivantes

$$e = x - \hat{x} \tag{3.140}$$
$$r = b - A\hat{x} = b - A(x - e) = Ae \tag{3.141}$$

où la quantité r dénote le *résidu* de la solution. Notons que les quantités A et r sont connues, alors que l'erreur e est inconnue.

Afin de réduire la quantité e on peut effectuer la procédure itérative suivante

`While` $\frac{\|e\|}{\|x^{(i)}\|} <$ `tol`

$$r^{(i)} = b - Ax^{(i)}$$
$$Ae = r^{(i)}$$
$$x^{(i+1)} = x^{(i)} + e$$

`end`

où $x^{(0)} = \hat{x}$ et `tol` est une seuil de précision fixé.

On peut montrer que si le système linéaire n'est pas trop mal conditionné, le raffinement itératif converge très rapidement.

Exemple Considérons le système linéaire

$$\begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 23 \\ 32 \\ 33 \\ 31 \end{bmatrix}$$

qui a pour solution exacte $x = [1, 1, 1, 1]^T$.

Si nous résolvons le système en notation à virgule flottante $b = 2, t = 15$, nous obtenons la solution approchée

$$\hat{x} = \begin{bmatrix} 0.9881 \\ 1.0073 \\ 1.0028 \\ 0.9983 \end{bmatrix}$$

On peut vérifier (script Matlab `s_refine.m`) qu'une étape de raffinement itératif ($x^{(0)} = \hat{x}$) réduit sensiblement l'erreur

$$\hat{x} = \begin{bmatrix} 1.0001 \\ 0.9999 \\ 1.0000 \\ 1.0000 \end{bmatrix}$$

•

3.12 Les méthodes itératives

Les méthodes itératives visent à construire une suite de vecteurs $x^{(k)}$ qui converge vers la solution du système $Ax = b$ pour $k \to \infty$

$$\lim_{k \to \infty} x^{(k)} = x \tag{3.142}$$

En pratique le nombre d'itérations d'une méthode itérative est fini : la méthode s'arrête à la première itération K telle que

$$\|x^{(K)} - x\| \leq \text{tol} \tag{3.143}$$

où tol est un seuil pré-fixé.

La forme générale d'une méthode itérative d'*ordre* m est la suivante

$$x^{(0)} = f_0(A, b) \tag{3.144}$$

$$x^{(k+1)} = f_{k+1}(x^{(k)}, x^{(k-1)}, \ldots x^{(k-m+1)}, A, b) \quad \text{pour } k \geq m-1 \tag{3.145}$$

où les $f_k(\cdot)$ sont des fonctions et les $x^{(m-1)}, \ldots, x^{(1)}$ sont des vecteurs donnés.

Si les fonctions $f_k(\cdot)$ sont indépendantes de k, la méthode est dite *stationnaire*, autrement elle est dite *non stationnaire*.

Si $f_k(\cdot)$ est une fonction linéaire, la méthode est dite *linéaire*, autrement elle est dite *non linéaire*.

Comme nous avons vu dans les sections précédentes, un algorithme de résolution directe demande un coût de calcul d'ordre $O(n^3)$. Puisque les algorithmes itératifs requièrent typiquement le calcul du résidu à chaque étape, le coût d'une simple itération s'élève à $O(n^2)$. Une méthode itérative est alors avantageuse si le nombre maximal d'itérations K est indépendant de n ou bien s'il croît sous-linéairement avec n.

Une autre situation où les méthodes itératives apparaissent comme avantageuses est la résolution d'un système linéaire avec une matrice de coefficients qui est *creuse*.

Définition 20 (Matrice creuse)**.** *Une matrice $A_{(n\times n)}$ est dite* creuse *si le nombre d'éléments non nuls est d'ordre n.*

Bien qu'une matrice creuse puisse être stockée dans un espace de mémoire réduit, la résolution directe du système linéaire entraîne un phénomène de remplissage (*fill-in*). Ceci signifie que tout le long de la résolution directe, le nombre d'éléments non nuls peut devenir d'ordre n^2.

Exemple Considérons une matrice de coefficients $A_{(n\times n)}$ creuse avec $n = 100$. Nous représentons graphiquement les éléments non nuls par des points à la Figure 3.3.

Supposons qu'on factorise la matrice en utilisant la méthode LU. Les éléments non nuls à l'itération $k = 15$ et $k = 16$ sont représentés dans les Figures 3.4 et 3.5, respectivement. Ces figures rendent compte d'une manière évidente du phénomène de remplissage.

•

3.12.1 Les méthodes itératives linéaires

Soit

$$x^{(k+1)} = Bx^{(k)} + f \tag{3.146}$$

une méthode itérative linéaire du premier ordre, où $x^{(0)}$ est la condition initiale, $B_{(n\times n)}$ est dite la *matrice d'itération* et $f_{(n\times 1)}$ est un vecteur dépendant de b (second membre du système $Ax = b$).

Définition 21 (Consistance)**.** *Une méthode itérative de la forme (3.146) est dite* consistante *(Section 1.3.4) avec le problème $Ax = b$ si f et B sont tels que*

$$x = Bx + f \tag{3.147}$$

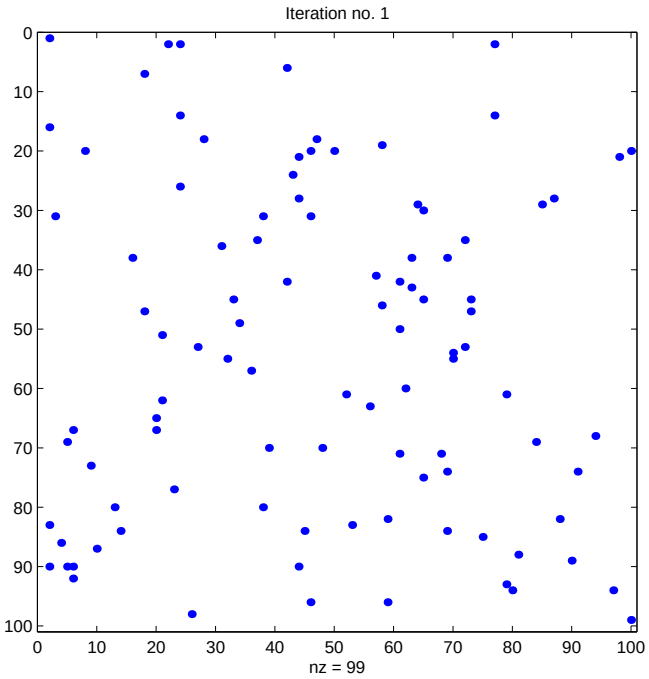


FIGURE 3.3 – Remplissage de la matrice creuse $A_{(100\times100)}$. Les points (en quantité nz) représentent les éléments non nuls. (Script MATLAB `s_fillin.m`)

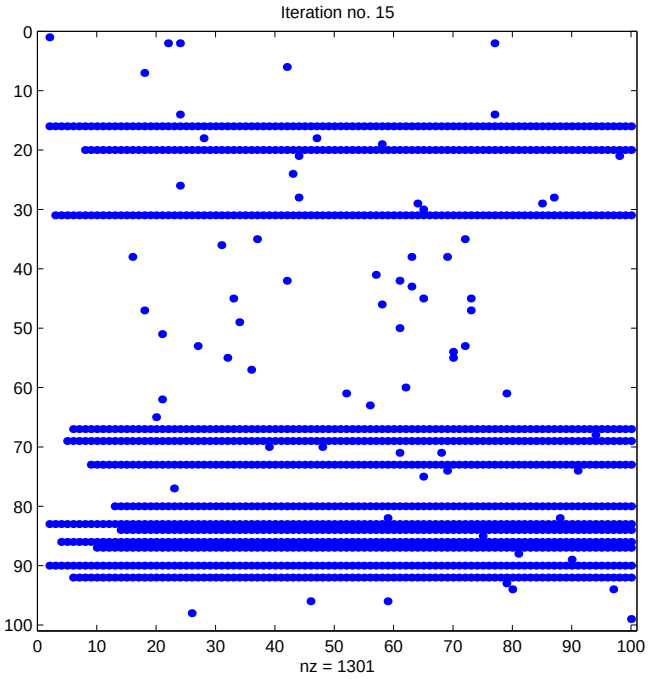


FIGURE 3.4 – Remplissage de la matrice $A^{(15)}$ à la 15ème itération de l'algorithme direct LU. Les points (en quantité nz) représentent les éléments non nuls. (Script MATLAB `s_fillin.m`)

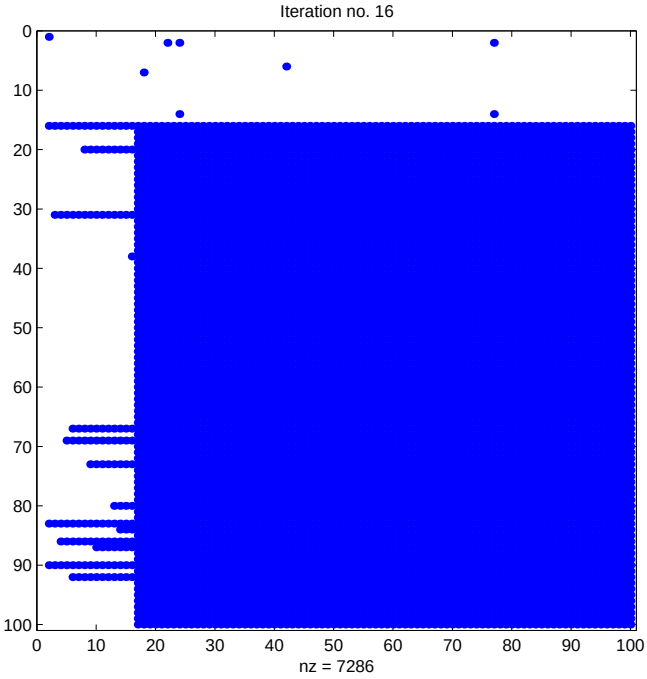


FIGURE 3.5 – Remplissage de la matrice $A^{(16)}$ à la 16ème itération de l'algorithme direct LU. Les points (en quantité nz) représentent les éléments non nuls. (Script MATLAB `s_fillin.m`)

ou, de manière équivalente, si f et B satisfont la relation

$$f = (I - B)A^{(-1)}b. \tag{3.148}$$

La seule propriété de consistance ne suffit pas à assurer la convergence d'une méthode itérative, comme le montre l'exemple suivant.

Exemple On veut résoudre le système linéaire

$$2Ix = b$$

avec I matrice identité. La méthode itérative

$$x^{(k+1)} = -x^{(k)} + b$$

est consistante mais elle n'est pas convergente. En effet, pour $x^{(0)} = 0$ la solution approchée oscille entre 0 et b.

•

Soit $e^{(k)} = x^{(k)} - x$ l'erreur à la $k$ème itération. Si la méthode itérative (3.146) est consistante, alors on obtient la relation de récurrence

$$e^{(k+1)} = x^{(k+1)} - x = (Bx^{(k)} + f) - (Bx + f) = B(x^{(k)} - x) = Be^{(k)} \tag{3.149}$$

qui entraîne

$$e^{(k)} = B^k e^{(0)}. \tag{3.150}$$

Définition 22 (Convergence)**.** *Une méthode itérative de la forme (3.146) est dite* convergente *(Section 1.3.6) si*

$$\lim_{k\to\infty} e^{(k)} = 0. \tag{3.151}$$

Théorème 3.10. *Si la méthode itérative (3.146) est consistante, alors elle est aussi convergente si et seulement si*

$$\rho(B) < 1 \tag{3.152}$$

où $\rho(\cdot)$ dénote le rayon spectral (Section A.6) et $\rho(B)$ définit le facteur de convergence asymptotique.

Il est raisonnable de penser que la convergence est d'autant plus rapide que le facteur $\rho(B)$ est petit. Une estimation de $\rho(B)$ peut donc fournir une bonne indication sur la convergence d'un algorithme itératif de la forme (3.146).

En vertu du théorème A.5 on peut énoncer la condition suffisante suivante.

Corollaire 3.2. *Si la méthode itérative (3.146) est consistante, alors elle est aussi convergente si*

$$\|B\| < 1 \tag{3.153}$$

pour une norme matricielle arbitraire consistante.

3.12.2 La forme générale linéaire

Une forme générale sous laquelle écrire les méthodes itératives linéaires est

$$Px^{(k+1)} = Nx^{(k)} + b \tag{3.154}$$

où P et N sont des matrices carrées de taille n et P (dite *matrice de préconditionnement*) est inversible.

La matrice d'itération (formule (3.146)) d'une méthode de la forme (3.154) est $B = P^{-1}N$.

Remarquez que si $P = A$ et $N = 0$ on retrouve la méthode directe.

Une méthode de la forme (3.154) est consistante si

$$Px = Nx + b \Leftrightarrow P - N = A \tag{3.155}$$

Considérons une méthode itérative consistante de la forme (3.154). Soit

$$r^{(k)} = b - Ax^{(k)} \tag{3.156}$$

le résidu de la solution $x^{(k)}$. Puisque

$$P^{-1}r^{(k)} = P^{-1}b - P^{-1}Ax^{(k)} \tag{3.157}$$

et $A = P - N$ on en tire la relation

$$\begin{aligned} Px^{(k+1)} = Nx^{(k)} + b \Leftrightarrow x^{(k+1)} = P^{-1}Nx^{(k)} + P^{-1}b \Leftrightarrow \\ \Leftrightarrow x^{(k+1)} = P^{-1}Nx^{(k)} + P^{-1}r^{(k)} + P^{-1}Ax^{(k)} \Leftrightarrow \\ \Leftrightarrow x^{(k+1)} = x^{(k)} + P^{-1}r^{(k)} \end{aligned} \tag{3.158}$$

Par la suite, nous allons présenter en détail deux algorithmes itératifs linéaires de la forme (3.154) et discuter leur propriétés.

3.12.3 La méthode de Jacobi

Afin d'introduire la méthode de Jacobi, nous commençons par analyser un système linéaire d'ordre 3

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases} \tag{3.159}$$

Ce système peut être réécrit sous la forme

$$\begin{cases} x_1 = \frac{1}{a_{11}}\left(b_1 - a_{12}x_2 - a_{13}x_3\right) \\ x_2 = \frac{1}{a_{22}}\left(b_2 - a_{21}x_1 - a_{23}x_3\right) \\ x_3 = \frac{1}{a_{33}}\left(b_3 - a_{31}x_1 - a_{32}x_2\right) \end{cases} \tag{3.160}$$

Supposons que $x^{(k)}$ soit une approximation de la solution x du système. L'idée de la méthode de Jacobi est de calculer $x^{(k+1)}$ sur la base de la relation (3.160)

$$\begin{cases} x_1^{(k+1)} = \frac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)}\right) \\ x_2^{(k+1)} = \frac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)}\right) \\ x_3^{(k+1)} = \frac{1}{a_{33}}\left(b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)}\right) \end{cases} \tag{3.161}$$

En passant au cas général, pour une donnée initiale $x^{(0)}$ choisie, la méthode de Jacobi calcule $x^{(k+1)}$ par

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, \ldots, n \tag{3.162}$$

ce qui est équivalent à la forme matricielle

$$x^{(k+1)} = D^{-1}\Big((D - A)x^{(k)} + b\Big) \tag{3.163}$$

où $D = \text{diag}(A)$.

Puisque l'itération (3.163) peut être mise sous la forme

$$Dx^{(k+1)} = \Big((D - A)x^{(k)} + b\Big)$$

la méthode de Jacobi est un cas particulier de la forme (3.154) où

$$P = D, \quad N = D - A. \tag{3.164}$$

Puisque $P = N + A$ la méthode de Jacobi est consistante.

Par conséquent la matrice d'itération de la méthode de Jacobi est

$$B_J = D^{-1}(D - A) = I - D^{-1}A. \tag{3.165}$$

3.12.4 La méthode de Gauss-Seidel

La méthode de Gauss-Seidel est basée sur une idée semblable à la méthode de Jacobi. Elle part aussi de la relation (3.160) avec la différence que les valeurs $x_i^{(k+1)}$ déjà calculées sont utilisées pour mettre à jour la solution.

Ceci signifie que pour le système (3.159) d'ordre 3 l'itération est la suivante

$$\begin{cases} x_1^{(k+1)} = \frac{1}{a_{11}} \left(b_1 - a_{12} x_2^{(k)} - a_{13} x_3^{(k)} \right) \\ x_2^{(k+1)} = \frac{1}{a_{22}} \left(b_2 - a_{21} \boxed{x_1^{(k+1)}} - a_{23} x_3^{(k)} \right) \\ x_3^{(k+1)} = \frac{1}{a_{33}} \left(b_3 - a_{31} \boxed{x_1^{(k+1)}} - a_{32} \boxed{x_2^{(k+1)}} \right) \end{cases} \tag{3.166}$$

où les termes encadrés sont ceux qui différencient la méthode de Gauss-Seidel de la méthode de Jacobi (formule (3.161)).

Dans le cas général

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right), \qquad i = 1, \ldots, n \tag{3.167}$$

et sous forme matricielle

$$x^{(k+1)} = (D - E)^{-1} (F x^{(k)} + b) \tag{3.168}$$

où

$$D - A = \begin{bmatrix} 0 & -a_{12} & \ldots & \ldots & -a_{1n} \\ -a_{21} & 0 & \ldots & \ldots & -a_{2n} \\ -a_{31} & -a_{32} & 0 & \ldots & -a_{3n} \\ \vdots & \vdots & \vdots & \ddots & -a_{n-1,n} \\ -a_{n1} & -a_{n2} & \ldots & -a_{n,n-1} & 0 \end{bmatrix} =$$

$$= \begin{bmatrix} 0 & 0 & \ldots & \ldots & 0 \\ -a_{21} & 0 & \ldots & \ldots & 0 \\ -a_{31} & -a_{32} & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ -a_{n1} & -a_{n2} & \ldots & -a_{n,n-1} & 0 \end{bmatrix} + \begin{bmatrix} 0 & -a_{12} & \ldots & \ldots & -a_{1n} \\ 0 & 0 & \ldots & \ldots & -a_{2n} \\ 0 & 0 & 0 & \ldots & -a_{3n} \\ 0 & \vdots & \vdots & \ddots & -a_{n-1,n} \\ 0 & 0 & \ldots & \ldots & 0 \end{bmatrix} = E + F, \tag{3.169}$$

E est la matrice triangulaire inférieure

$$\begin{cases} e_{ij} = -a_{ij} & \text{si } i > j \\ e_{ij} = 0 & \text{si } i \leq j \end{cases} \tag{3.170}$$

et F est la matrice triangulaire supérieure

$$\begin{cases} f_{ij} = -a_{ij} & \text{si } j > i \\ f_{ij} = 0 & \text{si } j \leq i \end{cases} \tag{3.171}$$

Exemple. Considérons le système d'ordre $n = 2$

$$\begin{cases} a_{11} x_1 + a_{12} x_2 = b_1 \\ a_{21} x_1 + a_{22} x_2 = b_2 \end{cases}$$

La méthode itérative de Gauss-Seidel pour ce système est basée sur l'itération

$$\begin{cases} x_1^{(k+1)} = \frac{1}{a_{11}} \left(b_1 - a_{12} x_2^{(k)} \right) \\ x_2^{(k+1)} = \frac{1}{a_{22}} \left(b_2 - a_{21} x_1^{(k+1)} \right) = \frac{1}{a_{22}} \left(b_2 - \frac{a_{21}}{a_{11}} \left(b_1 - a_{12} x_2^{(k)} \right) \right) = \frac{a_{21} a_{12}}{a_{22} a_{11}} x_2^{(k)} + \frac{b_2}{a_{22}} - \frac{b_1 a_{21}}{a_{11} a_{22}} \end{cases}$$

On retrouve la forme matricielle (3.168) car

$$(D-E)^{-1} = \begin{bmatrix} \frac{1}{a_{11}} & 0 \\ -\frac{a_{21}}{a_{11}a_{22}} & \frac{1}{a_{22}} \end{bmatrix}$$
$$(D-E)^{-1}F = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} \\ 0 & \frac{a_{12}a_{21}}{a_{11}a_{22}} \end{bmatrix}$$

•

La méthode de Gauss-Seidel est un cas particulier de la forme (3.154) où

$$P = D - E, \quad N = F \tag{3.172}$$

Puisque

$$N + A = F + A = D - A - E + A = D - E$$

alors $P = N + A$ et la méthode de Gauss-Seidel est consistante.

La matrice d'itération de la méthode de Gauss-Seidel est

$$B_{GS} = (D-E)^{-1}F \tag{3.173}$$

3.12.5 Résultats de convergence

Il existe deux cas où on peut établir des propriétés de convergence pour les deux méthodes examinées auparavant.

Théorème 3.11. *Si A est une matrice à diagonale dominante stricte (Appendix A.12), les méthodes de Jacobi et Gauss-Seidel sont convergentes.*

Démonstration : Nous démontrons ce théorème dans le cas de la méthode de Jacobi. D'après l'Equation (3.165)

$$B_J = D^{-1}(D-A) = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & \dots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \dots & \dots & -\frac{a_{2n}}{a_{22}} \\ -\frac{a_{31}}{a_{33}} & -\frac{a_{32}}{a_{33}} & \ddots & & \vdots \\ \vdots & & & 0 & -\frac{a_{n-1,n}}{a_{n-1,n-1}} \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & \dots & 0 \end{bmatrix}$$

Si A est une matrice à diagonal dominante stricte par lignes alors pour tout i

$$|a_{ii}| > \sum_{j=1}^{n} |a_{ij}|,$$

ce qui implique

$$\|B_J\|_\infty = \max_{i=1,\dots,n} \sum_{j=1, j\neq i}^{n} \frac{|a_{ij}|}{|a_{ii}|} < 1$$

et donc la convergence de la méthode de Jacobi par le corollaire (3.2).

Théorème 3.12. *Si A est une matrice symétrique définie positive (Appendix A.12), la méthode de Gauss-Seidel est convergente.*

Deux considérations peuvent être faites :
- la convergence de la méthode de Gauss-Seidel n'implique pas la convergence de le méthode de Jacobi et vice versa ;
- si les deux méthodes convergent, le taux de convergence de la méthode de Gauss-Seidel est *en général* supérieur.

3.12.6 Analyse du coût dans le cas des matrices creuses

Analysons le coût des algorithmes de Jacobi et Gauss-Seidel dans la résolution itérative d'un système linéaire $Ax = b$ où la matrice A est creuse, c.-à-d. elle a $p \approx O(n)$ éléments non nuls.

A chaque itération les algorithmes effectuent
- n divisions,
- p multiplications,
- p additions,

donc, si le nombre d'itérations total est égal à K, une méthode itérative est avantageuse par rapport à une méthode directe si $O(K) < n^2$.

3.12.7 La méthode du gradient

Considérons une méthode itérative consistante de la forme (3.154).

Soit

$$r^{(k)} = b - Ax^{(k)} \tag{3.174}$$

le résidu à la $k$ème itération.

Puisque $N = P - A$ (formule (3.155)) alors on peut réécrire (3.154) sous la forme

$$Px^{(k+1)} = Px^{(k)} - Ax^{(k)} + b \Leftrightarrow P(x^{(k+1)} - x^{(k)}) = r^{(k)}. \tag{3.175}$$

La méthode du gradient est une méthode modifiée où

$$P(x^{(k+1)} - x^{(k)}) = \alpha_k r^{(k)} \tag{3.176}$$

où α_k est un paramètre défini par la méthode afin d'accélérer la convergence.

L'idée de l'algorithme est que si A est une matrice symétrique et définie positive, résoudre le système linéaire $Ax = b$ est équivalent à la détermination de la valeur qui minimise la forme quadratique (*énergie du système*)

$$\Phi(y) = \frac{1}{2} y^T A y - y^T b \tag{3.177}$$

En utilisant les formules (A.5) on trouve que le gradient $\Phi(\cdot)$ est

$$\Delta\Phi(y) = \frac{1}{2}\left(A^T + A\right) y - b = Ay - b \tag{3.178}$$

et donc

$$\Delta\Phi(x) = 0 \Leftrightarrow Ax = b \tag{3.179}$$

où

$$x = \arg\min_y \Phi(y). \tag{3.180}$$

Exemple. Considérons le système $Ax = b$ où

$$A = \begin{bmatrix} 5 & 0.1 \\ 0.1 & 5 \end{bmatrix} \tag{3.181}$$

est une matrice symétrique et définie positive et $b = [1, 3]^T$. La solution est $x = [0.1881, 0.5962]^T$.

L'énergie du système est une fonction des deux arguments x_1 et x_2 qui est visualisée en Figure 3.6, dont les courbes de niveau sont représentées en Figure 3.7. Les deux figures illustrent aisément que la solution x du système est localisée dans le minimum de la fonction énergie.

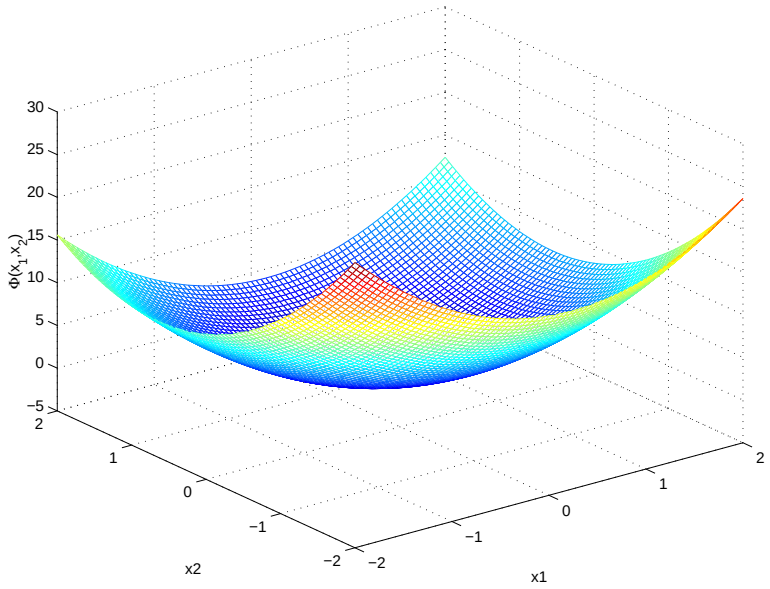


FIGURE 3.6 – Fonction énergie. (Script MATLAB `s_gradient.m`)

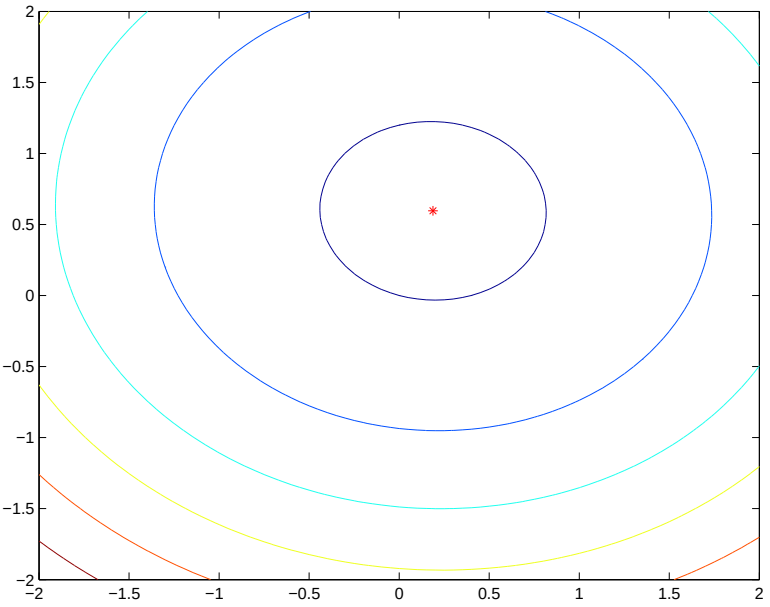


FIGURE 3.7 – Courbes de niveau de la fonction énergie. (Script MATLAB `s_gradient.m`)

•

Voyons en détail l'algorithme du gradient.

Soit $x^{(k)}$ une solution approchée à la $k$ème itération. La solution approchée $x^{(k)}$ et la solution exacte x correspondent à deux points différents dans l'espace des solutions.

L'algorithme essaie de déterminer la direction $p^{(k)}$ de la solution x par rapport à $x^{(k)}$ de manière à améliorer l'approximation $x^{(k)}$ en utilisant la mise à jour

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \tag{3.182}$$

où $p^{(k)}$ est un vecteur $(n \times 1)$.

La question est alors de déterminer la direction de déplacement $p^{(k)}$ qui permet de se rapprocher le plus de x.

Deux méthodes alternatives existent

1. méthode de la plus profonde descente,
2. méthode du gradient conjugué.

Les deux section suivantes analyseront en détail les deux variantes de l'algorithme du gradient.

3.12.7.1 La méthode de la plus profonde descente

La méthode de la plus profonde descente est une méthode du gradient où la direction $p^{(k)}$ en (3.182) est égale à la direction de descente de pente maximale de la fonction $\Phi(\cdot)$.

Puisque le gradient de $\Phi(\cdot)$ en $x^{(k)}$ est

$$\nabla\Phi(x^{(k)}) = Ax^{(k)} - b = -r^{(k)} \tag{3.183}$$

où $r^{(k)}$ est le résidu de la solution approchée à la $k$ème itération, la méthode pose en (3.182)

$$p^{(k)} = r^{(k)} \tag{3.184}$$

Le terme α_k est choisi afin d'atteindre le minimum local de l'expression

$$\Phi(x^{(k+1)}) = \frac{1}{2}\left(x^{(k)} + \alpha p^{(k)}\right)^T A\left(x^{(k)} + \alpha p^{(k)}\right) - \left(x^{(k)} + \alpha p^{(k)}\right) b \tag{3.185}$$

c.à.d.

$$\alpha_k = \arg\min_{\alpha} \frac{1}{2}\left(x^{(k)} + \alpha p^{(k)}\right)^T A\left(x^{(k)} + \alpha p^{(k)}\right) - \left(x^{(k)} + \alpha p^{(k)}\right) b \tag{3.186}$$

Puisque

$$\begin{aligned}\frac{\partial\Phi(x^{(k+1)})}{\partial\alpha} = 0 \Leftrightarrow \frac{1}{2}(p^{(k)})^T(A + A^T)\left(x^{(k)} + \alpha p^{(k)}\right) - (p^{(k)})^T b = 0 \Leftrightarrow \\ (p^{(k)})^T A\left(x^{(k)} + \alpha p^{(k)}\right) = (p^{(k)})^T b \Leftrightarrow \alpha(p^{(k)})^T A p^{(k)} = (p^{(k)})^T r^{(k)}\end{aligned} \tag{3.187}$$

on obtient

$$\alpha_k = \frac{(p^{(k)})^T r^{(k)}}{(p^{(k)})^T A p^{(k)}} = \frac{(r^{(k)})^T r^{(k)}}{(r^{(k)})^T A r^{(k)}} \tag{3.188}$$

Il en résulte que la quantité α_k est une fonction du résidu est qu'elle s'adapte donc *dynamiquement* à chaque étape de la méthode.

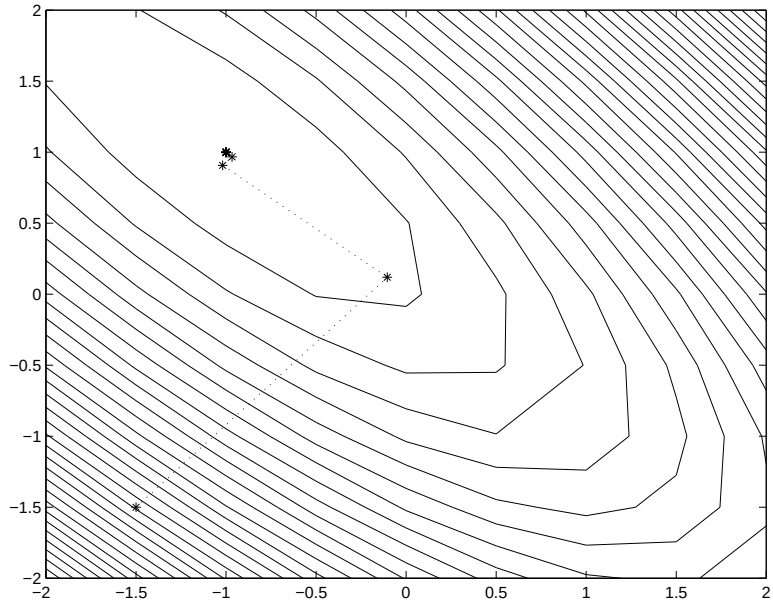


FIGURE 3.8 – Convergence de la méthode de la plus profonde descente. (Script MATLAB `s_grad_al.m`)

Sur base des formules (3.135),(3.182) et (3.188), on obtient que l'algorithme de la plus profonde descente est basé sur l'itération suivante

$$\begin{cases} r^{(k)} = b - Ax^{(k)} \\ \alpha_k = \frac{(r^{(k)})^T r^{(k)}}{(r^{(k)})^T A r^{(k)}} \\ x^{(k+1)} = x^{(k)} + \alpha_k r^{(k)} \end{cases} \tag{3.189}$$

où $x^{(0)}$ est donné.

Exemple. Considérons le système linéaire $Ax = b$ où

$$A = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \tag{3.190}$$

La solution exacte est $x = [-1, 1]^T$. La figure (3.8) montre les différentes étapes de l'algorithme jusqu'à la convergence vers x.

•

En ce qui concerne la convergence de la méthode de la plus profonde descente, on peut montrer le théorème suivant.

Théorème 3.13. *Soit A une matrice symétrique définie positive ; alors*

1. *la méthode du gradient est convergente pour n'importe quelle donnée initiale $x^{(0)}$,*
2. *l'inéquation*

$$\|e^{(k+1)}\|_A \leq \frac{\kappa_2(A) - 1}{\kappa_2(A) + 1} \|e^{(k)}\|_A \tag{3.191}$$

est satisfaite, où $e^{(k)}$ est l'erreur à l'étape k et $\|\cdot\|_A$ est la norme de l'énergie (Section A.12).

Il en résulte que la convergence peut être assez lente si la quantité $\kappa_2(A) = \frac{\lambda_1}{\lambda_2}$ est grande.

3.12.7.2 La méthode du gradient conjugué

La méthode du gradient conjugué effectue à chaque itération les deux opérations suivantes :

1. le choix de la direction $p^{(k)}$ (qui n'est plus parallèle au résidu),
2. le choix du point optimal $x^{(k+1)}$ le long de la direction $p^{(k)}$.

Voyons d'abord comment le point optimal $x^{(k+1)}$ est choisi dans la direction $p^{(k)}$. La définition de point optimal signifie que

$$\Phi(x^{(k+1)}) \leq \Phi(x^{(k+1)} + \lambda p^{(k)}) \tag{3.192}$$

pour tout $\lambda \in \mathbb{R}$. En d'autres termes, la dérivée partielle par rapport à λ

$$\begin{aligned}\frac{\partial \Phi}{\partial \lambda}\left(x^{(k+1)} + \lambda p^{(k)}\right) = \\ = (p^{(k)})^T\left(Ax^{(k+1)} - b\right) + \lambda (p^{(k)})^T A p^{(k)} = (p^{(k)})^T r^{(k+1)} + \lambda (p^{(k)})^T A p^{(k)}\end{aligned}$$

s'annule pour $\lambda = 0$.

Puisque

$$\frac{\partial \Phi}{\partial \lambda}\left(x^{(k+1)} + \lambda p^{(k)}\right)\Big|_{\lambda=0} = 0 \Leftrightarrow (p^{(k)})^T r^{(k+1)} = 0,$$

un point le long d'une direction $p^{(k)}$ est optimal si

$$(p^{(k)})^T r^{(k+1)} = 0. \tag{3.193}$$

Puisque le minimum de $\Phi(\cdot)$ est par définition optimal le long de toutes les directions qui passent par x, l'idée de la méthode du gradient conjugué est d'imposer à chaque étape k

$$(p^{(j)})^T r^{(k+1)} = 0, \qquad j = 0, \ldots, k. \tag{3.194}$$

Ceci est équivalent à imposer

$$\left(Ap^{(j)}\right)^T p^{(k)} = 0, \qquad j = 0, \ldots, k-1. \tag{3.195}$$

Les directions de descente qui satisfont la relation précédente sont dites mutuellement *A-orthogonales* ou *A-conjuguées*.

On peut montrer que, pour ce faire, il faut imposer

$$p^{(k+1)} = r^{(k+1)} - \beta_k p^{(k)} \tag{3.196}$$

où

$$\beta_k = \frac{\left(Ap^{(k)}\right)^T r^{(k+1)}}{\left(Ap^{(k)}\right)^T p^{(k)}} \tag{3.197}$$

Des formules (3.187),(3.182), (3.196) il vient que l'algorithme du gradient conjugué est basé sur l'itération

$$\begin{cases} \alpha_k = \frac{(p^{(k)})^T r^{(k)}}{(p^{(k)})^T A p^{(k)}} \\ x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \\ r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)} \\ \beta_k = \frac{\left(Ap^{(k)}\right)^T r^{(k+1)}}{\left(Ap^{(k)}\right)^T p^{(k)}} \\ p^{(k+1)} = r^{(k+1)} - \beta_k p^{(k)} \end{cases} \tag{3.198}$$

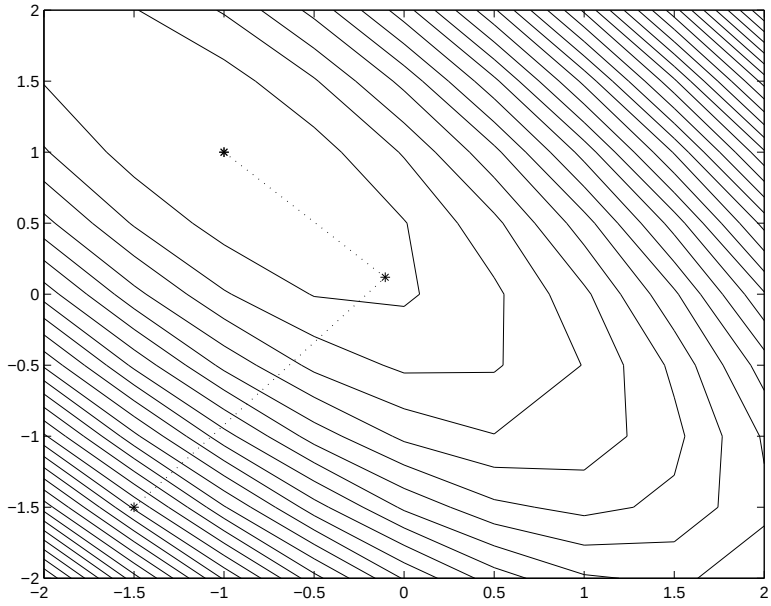


FIGURE 3.9 – Convergence de la méthode du gradient conjugué. (Script MATLAB `s_coniug.m`)

où $r^{(0)} = b - Ax^{(0)}$ et $p^{(0)} = r^{(0)}$. Notons que l'algorithme demande à chaque étape le calcul de deux paramètres α_k et β_K

Exemple. Considérons le système linéaire $Ax = b$ défini en (3.190). La solution exacte est $x = [-1, 1]^T$.

La figure (3.9) montre comment la méthode du gradient conjugué converge vers x en $n = 2$ étapes.

•

Pour ce qui concerne la méthode du gradient conjugué, on peut établir les résultats de convergence suivants.

Théorème 3.14. *Soit A une matrice symétrique définie positive d'ordre n. La méthode du gradient conjugué conduit à la solution exacte en au plus n itérations.*

Théorème 3.15. *Soit A une matrice symétrique définie positive d'ordre n et soit λ_1 (resp. λ_n) la valeur propre de A de plus grand (resp. petit) module, alors,*

1. *l'erreur $e^{(k)}$ est orthogonale à $p^{(j)}$, $j = 0, \ldots, k-1$,*
2. *l'inéquation suivante est satisfaite*

$$\|e^{(k)}\|_A \leq \frac{2c^k}{1 + c^{2k}} \|e^{(0)}\|_A \tag{3.199}$$

où

$$c = \frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \tag{3.200}$$

et $\|\cdot\|_A$ est la norme de l'énergie (Section A.12).

Le premier résultat montre que, en absence d'erreur d'arrondi, la méthode du gradient conjugué peut être considérée comme une méthode directe puisqu'elle converge en un nombre fini d'étapes. En pratique, pour une taille n très grande, la méthode est arrêtée à une itération $K < n$, donc la méthode est utilisée comme méthode itérative.

Le deuxième résultat de convergence énonce que la dépendance de l'erreur par rapport au conditionnement de la matrice est plus favorable que pour la méthode du gradient.

3.12.8 Les tests d'arrêt

D'une manière idéale, une méthode itérative poursuit les itérations jusqu'à ce que une approximation $x^{(K)}$ de la solution exacte x est obtenue. En pratique, x étant inconnu, les méthodes itératives s'arrêtent quand un test d'arrêt basé sur quelconque information disponible est satisfait.

Les deux tests d'arrêt les plus communs sont basés sur

1. l'incrément de la solution approchée, c.-à-d.
$$\|x^{(k+1)} - x^{(k)}\| < \epsilon, \tag{3.201}$$
2. le résidu de la solution approchée, c.-à-d.
$$\|r^{(k)}\| = \|Ax^{(k)} - b\| < \epsilon, \tag{3.202}$$

où ϵ est un seuil pré-fixé. Dans ce qui suit, nous allons analyser la relation qui existe entre le seuil de tolérance ϵ et la vraie erreur commise par la méthode dans les deux cas.

3.12.8.1 Arrêt sur l'incrément

Cette section vise à estimer l'ordre de grandeur de l'erreur commise par une méthode itérative qui s'achève quand la condition (3.201) est satisfaite.

Considérons une méthode itérative de la forme (3.146). En vertu de la formule (3.149), si $\|B\| = \gamma$, on obtient
$$\|e^{(k+1)}\| \leq \gamma \|e^{(k)}\| \tag{3.203}$$

Puisque
$$\|x^{(k+1)} - x^{(k)}\| = \|(x - x^{(k)}) - (x - x^{(k+1)})\| \geq \|e^{(k)}\| - \|e^{(k+1)}\| \geq (1 - \gamma)\|e^{(k)}\| \tag{3.204}$$
on obtient de (3.201),(3.203) et (3.204)
$$\|e^{(k+1)}\| \leq \gamma \|e^{(k)}\| \leq \frac{\gamma}{1-\gamma}\|x^{(k+1)} - x^{(k)}\| \leq \frac{\gamma}{1-\gamma}\epsilon \tag{3.205}$$

3.12.8.2 Arrêt sur le résidu

Cette section vise à estimer l'ordre de grandeur de l'erreur commise par une méthode itérative qui s'achève quand la condition (3.202) est satisfaite.

On obtient aisément la relation
$$\|e^{(k)}\| = \|x - x^{(k)}\| = \|A^{-1}b - A^{-1}(b - r^{(k)})\| = \|A^{-1}r^{(k)}\| \leq \kappa(A)\frac{\|r^{(k)}\|}{\|A\|} \leq$$
$$\leq \kappa(A)\frac{\|\epsilon\|}{\|A\|} \tag{3.206}$$

où $\kappa(A)$ est le conditionnement de la matrice A (Section 3.10.1).

Il s'ensuit que le test d'arrêt sur le résidu garantit une erreur de taille comparable à ϵ seulement si la matrice A n'est pas mal conditionnée.

3.13 Exercices

3.13.1 Méthode de Cramer

1. Résoudre le système suivant

$$\begin{cases} 4x_1 + 8x_2 + 12x_3 &= 4 \\ 3x_1 + 8x_2 + 13x_3 &= 5 \\ 2x_1 + 9x_2 + 18x_3 &= 11 \end{cases} \tag{3.207}$$

par la méthode de Cramer.

3.13.2 Élimination de Gauss

1. Résoudre le système (3.207) par la méthode de Gauss.
2. Considérons le système suivant :

$$\begin{cases} 0.000100x_1 + 1.00x_2 &= 1.00 \\ 1.00x_1 + 1.00x_2 &= 2.00 \end{cases}$$

dont la solution exacte est $x_1 = 1.00010,\ x_2 = 0.99990$.

(a) Résoudre ce système par la méthode de Gauss en utilisant la représentation machine à virgule flottante avec $b = 10$ et $t = 3$ (utiliser l'arithmétique arrondie).

(b) Résoudre de nouveau le système comme au point (a) mais en inversant les deux équations. Comparer les résultats et tirer des conclusions.

3.13.3 Élimination de Gauss avec recherche de pivot partiel

Résoudre les systèmes suivants en utilisant la méthode de Gauss avec recherche de pivot partiel :

1. $\begin{cases} -x_1 + 2x_2 + x_3 = 5 \\ x_1 + 4x_2 - 3x_3 = -8 \\ -2x_1 + x_3 = 5 \end{cases}$

2. $\begin{cases} x_1 + 2x_2 - x_3 = 2 \\ 2x_1 + 4x_2 + x_3 = 7 \\ 3x_1 + 6x_2 - 2x_3 = 7 \end{cases}$

3. $\begin{cases} x_1 - x_2 + x_3 = 0 \\ 2x_1 + x_2 - x_3 = -3 \\ x_1 + 2x_2 - 2x_3 = -2 \end{cases}$

4. $\begin{cases} 2x_1 - 3x_2 + 2x_3 + 5x_4 = 3 \\ x_1 - x_2 + x_3 + 2x_4 = 1 \\ 3x_1 + 2x_2 + 2x_3 + x_4 = 0 \\ x_1 + x_2 - 3x_3 - x_4 = 0 \end{cases}$

3.13.4 Décomposition LU

Soit A une matrice $n \times n$ admettant la décomposition LU. Montrer que la résolution du système $A\boldsymbol{x} = \boldsymbol{b}$, où $\boldsymbol{x}$ et $\boldsymbol{b}$ sont des vecteurs colonne dans $\mathbb{R}^n$, est équivalente à la résolution de deux systèmes triangulaires, l'un inférieur et l'autre supérieur.

3.13.5 Décomposition LU et élimination de Gauss

1. Le système (3.207) peut s'écrire sous forme matricielle $A\boldsymbol{x} = \boldsymbol{b}$ avec

$$A = \begin{bmatrix} 4 & 8 & 12 \\ 3 & 8 & 13 \\ 2 & 9 & 18 \end{bmatrix} \quad \text{et} \quad \boldsymbol{b} = \begin{bmatrix} 4 \\ 5 \\ 11 \end{bmatrix}.$$

   Déterminer la décomposition LU de la matrice A en vous basant sur la méthode de Gauss de l'exercice 3.13.2.

2. Déterminer la décomposition LU avec pivot de la matrice correspondant au système de l'exercice 3.13.3.4.

3.13.6 Méthode de Doolittle

1. Donner les équations permettant de calculer les matrices L et U par la méthode de Doolittle.

2. Résoudre le système $A\boldsymbol{x} = \boldsymbol{b}$, avec

$$A = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix} \quad \text{et} \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}.$$

   en utilisant la méthode de Doolittle.

3.13.7 Factorisation de Cholesky

1. Soit A une matrice $n \times n$ symétrique définie positive. Notons par $A = H^t H$ sa factorisation de Cholesky, où H est une matrice $n \times n$ triangulaire supérieure telle que $H_{ii} > 0,\ \forall\ 1 \leq i \leq n$. Donner les équations permettant de calculer les composantes de la matrice H.

2. Résoudre par la méthode de Cholesky le système $A\boldsymbol{x} = \boldsymbol{b}$ où

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 1 & 3 \end{bmatrix} \quad \text{et} \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

3.13.8 Conditionnement d'un système

1. Lorsqu'une matrice A est mal conditionnée, de petits changements de données induisent de larges changements de la solution. Pour le système suivant :

$$\begin{cases} 7x_1 & + & 10x_2 & = & b_1 \\ 5x_1 & + & 7x_2 & = & b_2 \end{cases}$$

   Trouver les solutions pour

$$b = \begin{bmatrix} 1 \\ 0.7 \end{bmatrix} \quad \text{et} \quad b = \begin{bmatrix} 1.01 \\ 0.69 \end{bmatrix}$$

   Que peut-on en conclure à propos du conditionnement du système ? Quelle est la valeur exacte du conditionnement κ_1 ? Et celle de κ_∞ ?

3.13.9 Méthodes itératives de Jacobi et de Gauss-Seidel

1. Considérons le système $A\boldsymbol{x} = \boldsymbol{b}$ où

$$A = \begin{bmatrix} 10 & -2 & 1 \\ -2 & 10 & -2 \\ -2 & -5 & 10 \end{bmatrix} \quad \text{et} \quad \boldsymbol{b} = \begin{bmatrix} 9 \\ 12 \\ 18 \end{bmatrix}$$

dont la solution exacte est $\boldsymbol{x} = (1, 2, 3)^t$.

(a) Déterminer une solution approchée du système par la méthode de Jacobi ; calculer trois itérations en partant du vecteur initial $\boldsymbol{x}^{(0)} = (0, 0, 0)^t$.

(b) Idem mais avec la méthode de Gauss-Seidel.

Que peut-on conclure ?

Chapitre 4

Résolution d'équations différentielles ordinaires

Une équation différentielle est une équation qui contient une ou plusieurs dérivées d'une fonction inconnue. Si toutes les derivées sont ordinaires l'équation prend le nom d'*équation différentielle ordinaire (EDO)* ; si au moins une des dérivées est partielle l'équation est dite aux *dérivées partielles*. L'ordre d'une équation différentielle est celui de la dérivée d'ordre le plus élevé. Ce chapitre traitera de la résolution numérique d'équations différentielles ordinaires.

Les équations différentielles ordinaires (EDO) sont parmi les outils de modélisation mathématique les plus puissants. On retrouve des modèles basés sur les EDO dans la plupart des sciences et sciences appliquées. Dans ce chapitre nous allons présenter les méthodes de résolution numérique des EDO. Nous nous bornerons à considérer le problème aux valeurs initiales pour équations différentielles ordinaires du premier ordre (c.à.d. où la plus grand dérivée est d'ordre 1) et scalaires.

4.1 Exemples et motivations

Le succès des équations différentielles ordinaires dans la science est dû au fait que la plupart des lois scientifiques peuvent être facilement représentées sous forme de taux de changement. Un exemple est le modèle qui décrit en physique le taux de changement de la température u d'un corps qui perd de l'énergie par convection

$$\frac{du}{dt} = -0.27(u - a) \tag{4.1}$$

où a est la température de l'environnement. En biologie, un des modèles les plus connus est le modèle de *Lotka-Volterra* [5, 8]. Celui-ci décrit l'interaction entre deux espèces animales (par exemple proie-prédateur) dans un écosystème et, dans sa forme la plus simple, peut être représenté par l'EDO

$$\begin{cases} dL/dt &= aL - bLR \\ dR/dt &= -cR + dLR \end{cases} \tag{4.2}$$

où

- $L(t)$ et $R(t)$ représentent la densité à l'instant t de lapins (espèce proie) et de renards (espèce prédatrice), respectivement,
- a est le taux de croissance des lapins en absence de prédateurs,
- c est le taux de décès des renards en absence de nourriture,
- d est le taux de reproduction d'un renard par proie capturée,

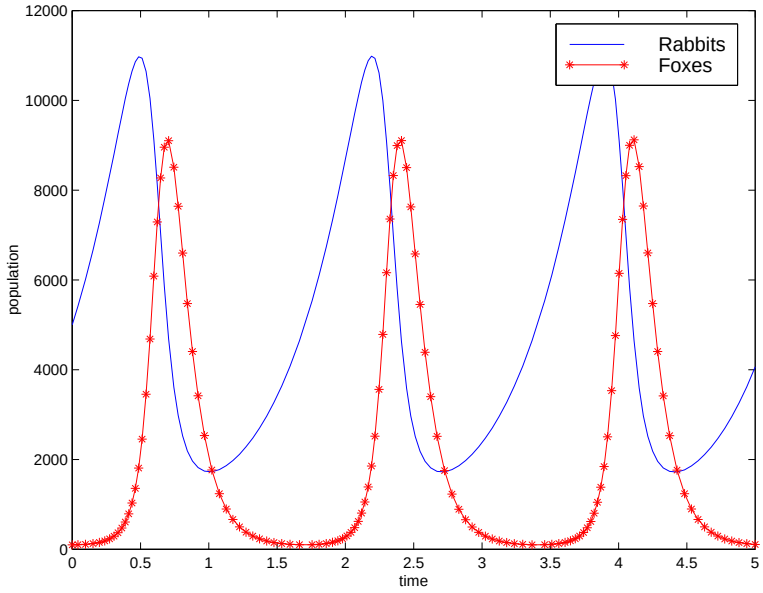


FIGURE 4.1 – Modèle Lotka-Volterra pour deux populations : solution numérique. (Script MATLAB `s_lotkavolterra.m`)

– b est le taux de capture.

La résolution de cette équation fournit l'évolution des deux populations au cours du temps. Supposons que $a = 2$, $b = 0.001$, $c = 10$, $d = 0.002$, et que la densité des deux populations à l'instant $t = 0$ s'élève à $L(0) = 5000$, $R(0) = 100$. En résolvant numériquement le système, on trouve que l'évolution des deux populations suit le comportement montré sur la Figure 4.1.

Les graphiques mettent bien en évidence une variation stable et cyclique des deux populations.

4.2 Problème aux valeurs initiales

Le problème aux valeurs initiales (aussi appelé *problème de Cauchy*) consiste à trouver la solution $y(t)$, $t \in I$, d'une équation différentielle ordinaire (EDO)

$$y'(t) = \frac{dy}{dt} = f(t, y(t)) \tag{4.3}$$

satisfaisant la condition initiale

$$y(t_0) = y_0 \tag{4.4}$$

où $f(\cdot, \cdot)$ est une fonction à valeurs réelles définie sur le produit $\{I \times]-\infty, \infty[\}$ et continue par rapport aux deux variables.

Si f ne dépend pas explicitement de t, l'EDO est dite *autonome*.

Définition 23 (Condition de Lipschitz)**.** *La fonction $f(t, y)$ satisfait la* condition de Lipschitz *par rapport à y si pour tout $t \in I$ et pour tout $y_1 \in \mathbb{R}$, $y_2 \in \mathbb{R}$,*

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2| \tag{4.5}$$

où $L > 0$ est appelée la constante de Lipschitz.

Si la dérivée partielle de f par rapport à y est continue, alors, d'après le théoreme de Lagrange des accroissements finis

$$|f(t,y_1) - f(t,y_2)| = \left|\frac{\partial f}{\partial y}(t,\xi)\right| |y_1 - y_2| \tag{4.6}$$

pour $y_1 \leq \xi \leq y_2$. Il s'ensuit que puisque $\left|\frac{\partial f}{\partial y}\right|$ est bornée supérieurement, alors f satisfait la condition de Lipschitz où $L = \max\left|\frac{\partial f}{\partial y}\right|$. Notons que cette condition est suffisante mais pas nécessaire.

Rappelons maintenant les résultats d'existence et d'unicité pour le problème (4.3).

Théorème 4.1. *Soit $f(t,y)$ une fonction continue pour tout $t \in I$ et pour tout $y \in \mathbb{R}$, qui satisfait la condition de Lipschitz. Alors, pour chaque valeur y_0, le problème aux valeurs initiales*

$$\begin{cases} y' = f(t,y) \\ y(t_0) = y_0 \end{cases} \tag{4.7}$$

a une solution unique $y(t) = y(t,y_0) \in \mathcal{C}^1$ pour tout $t \in I$.

4.2.1 Solution analytique et numérique

La solution analytique d'une équation différentielle ordinaire de la forme (4.3) est fournie par la formule

$$y(t) = y_0 + \int_{t_0}^{t} f(\tau, y(\tau))d\tau \tag{4.8}$$

qui demande la résolution d'une intégrale.

Malheureusement, on ne sait intégrer qu'un très petit nombre d'EDO où la fonction f est non linéaire. Par exemple, l'équation

$$y'(t) = e^{-t^2} \tag{4.9}$$

peut être résolue seulement par un développement en série.

De plus, quand cela est possible, il n'est pas toujours facile d'exprimer la solution sous forme explicite. Par exemple l'équation

$$y' = (y-t)/(y+t) \tag{4.10}$$

admet des solutions qui vérifient la relation implicite

$$\frac{1}{2}\ln(t^2+y^2) + \operatorname{arctg}\frac{y}{t} = C \tag{4.11}$$

Pour cette raison on fait appel aux méthodes de résolution numérique.

Une méthode numérique ne renvoie pas une fonction $y(\cdot)$ mais produit un tableau de valeurs $\hat{y}(t)$ qui approchent la solution $y(t)$ pour un ensemble de valeurs $\{t_0, t_1, \ldots, t_N\}$ choisies à l'avance. L'ensemble des valeurs

$$\{y(t_0) = \hat{y}(t_0), \hat{y}(t_1), \ldots, \hat{y}(t_n)\}$$

représente la *solution numérique* du problème.

Exemple Dans la présentation qui suit, nous utiliserons l'EDO

$$\frac{dy}{dt} = -2t - y, \quad y(0) = -1 \tag{4.12}$$

dont la solution analytique est

$$y(t) = -3\exp^{-t} -2t + 2, \tag{4.13}$$

comme un exemple pratique pour comparer la précision de différentes solutions numériques.

•

4.3 La méthode du développement de Taylor

La méthode de Taylor est une méthode de résolution qui s'appuie sur le développement en série de la fonction $y(t)$ autour du point $t = t_0$

$$y(t) \approx y(t_0) + y'(t_0)(t - t_0) + \frac{y''(t_0)}{2!}(t - t_0)^2 + \frac{y'''(t_0)}{3!}(t - t_0)^3. \tag{4.14}$$

En posant $t - t_0 = h$, la solution $y(\cdot)$ peut être écrite sous la forme

$$y(t_0 + h) \approx y(t_0) + y'(t_0)h + \frac{y''(t_0)}{2!}h^2 + \frac{y'''(t_0)}{3!}h^3, \tag{4.15}$$

où $y(t_0)$ est la condition initiale (connue) du problème de Cauchy, $y'(t_0) = f(t_0, y(t_0))$ et les termes $y''(t_0)$, $y'''(t_0)$ peuvent être calculés en dérivant la fonction f.

Exemple Nous appliquons la méthode de Taylor à la résolution du problème (4.12). Puisque

$$\begin{aligned} y(t_0) &= y(0) = -1 \\ y'(t_0) &= f(t_0, y(t_0)) = -2(0) - (-1) = 1 \\ y''(t_0) &= -2 - y'(t_0) = -2 - 1 = -3 \\ y'''(t_0) &= -y''(t_0) = 3 \end{aligned}$$

la solution de Taylor est

$$y(h) = -1 + 1.0h - 1.5h^2 + 0.5h^3 + \text{erreur} \tag{4.16}$$

Le graphique de la solution est donné à la Figure 4.2.

•

La méthode de Taylor est simple et intuitive mais présente quelques limitations importantes :

- le calcul des dérivées n'est pas toujours simple ;
- la forme analytique de l'erreur est connue mais son calcul n'est pas possible ;
- il est difficile de déterminer le nombre de termes du développement ;
- la méthode nécessite un ordre élevé afin de fournir une solution précise pour un grand h.

Par conséquent, la méthode n'est pas utilisée normalement dans les problèmes pratiques, où on fait souvent appel à des méthodes itératives.

4.4 Méthodes itératives

Soit $I =]t_0, t_0+T[$ l'intervalle d'intégration du problème (4.3) et $t_n = t_0+nh$, $n = 0, 1, \ldots, N_h$, une suite de noeuds de I où $h > 0$ est appelé le *pas de discrétisation*.

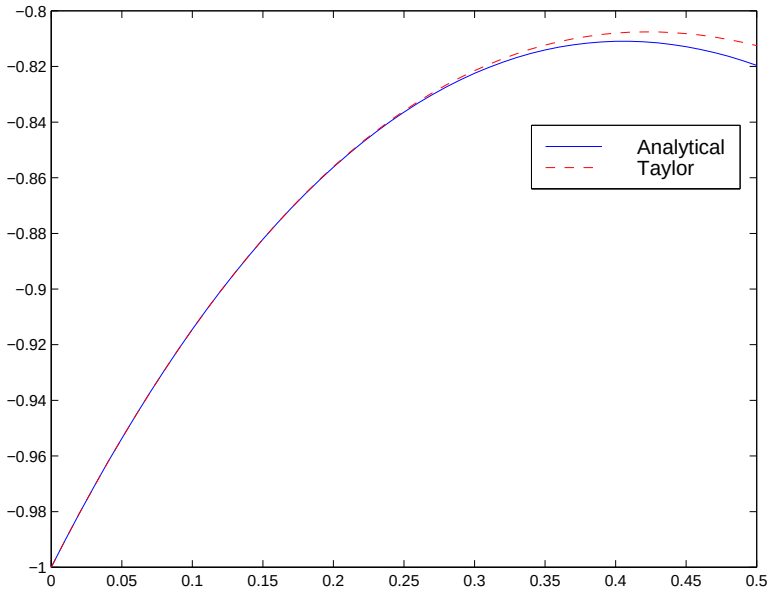


FIGURE 4.2 – Solution analytique et solution numérique du problème (4.12) en utilisant la méthode de Taylor numérique. (Script MATLAB `s_taylor.m`)

Nous entendons par

$$\hat{y}_n = \hat{y}(t_n) \tag{4.17}$$

la valeur approchée de la solution $y(t_n)$ au noeud t_n calculée par la méthode itérative au bout de la $n$ème étape.

On peut classifier les méthodes itératives sur la base de la procédure utilisée pour obtenir la solution $\hat{y}_n$.

Définition 24 (Méthodes à un pas/multi-pas)**.** *Une méthode numérique pour la résolution d'une EDO est dite à* un pas *si* $\forall n \geq 0$, $\hat{y}_{n+1}$ *ne dépend que de* $\hat{y}_n$. *Autrement le schéma est dit* multi-pas *(ou à pas multiples).*

Définition 25 (Méthodes explicites/implicites)**.** *Une méthode itérative pour la résolution d'une EDO est dite* explicite *si la valeur* $\hat{y}_{n+1}$ *peut être calculée directement à l'aide des valeurs précédentes* $\hat{y}_k$, $k \leq n$ *(ou une partie d'entre elles). Une méthode est dite* implicite *si* $\hat{y}_{n+1}$ *n'est définie que par une relation implicite faisant intervenir la fonction* f.

4.5 La méthode d'Euler progressive

Les méthodes d'Euler sont des méthodes itératives où à chaque étape l'approximation $\hat{y}(t_{n+1})$ de la vraie solution $y(t_{n+1})$ est calculée sur base des deux premiers termes du développement de Taylor autour de $\hat{y}(t_n)$. Ces méthodes sont justifiées par le fait que l'erreur du développement de Taylor et la quantité h sont des grandeurs à accroissements proportionnels.

Nous commencons par traiter l'instance la plus simple des méthodes d'Euler : la *méthode progréssive*. Les autres instances de la méthode d'Euler seront présentées dans la Section 4.7.

La méthode d'Euler progressive est une méthode itérative *à un pas* et *explicite*,

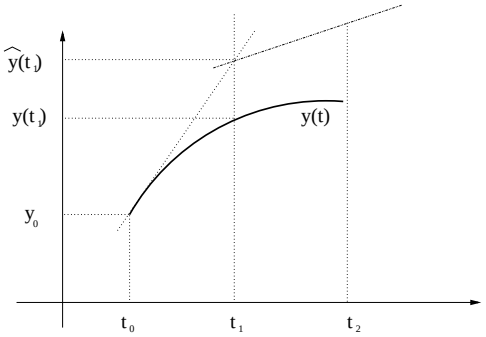


FIGURE 4.3 – Interprétation géométrique de la méthode de Euler progressive

dont la $n$ème étape peut être écrite sous la forme

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + hf(t_n, \hat{y}(t_n)) \tag{4.18}$$

où $\hat{y}(t_0) = y(t_0)$.

En interprétant la formule (4.18) en termes géométriques, on voit que la pente de la solution approchée $\hat{y}$ au début de l'intervalle $I_n = [t_n, t_{n+1}]$ est utilisée pour calculer l'incrément de la solution (Figure 4.3).

Exemple. Cet exemple montre comment la solution de la méthode d'Euler dépend de la taille du pas de discrétisation. Considérons deux solutions numériques du problème (4.12) obtenues par la méthode de Euler progressive avec deux pas de discrétisation différents ($h = 0.05$ et $h = 0.1$) La solution analytique (exacte) du problème (4.12) et les deux solutions numériques (approchées) sont calculées par le script `MATLAB`©`s_euler.m`) et tracées en Figure 4.4.

Il est intéressant de remarquer que, en divisant en deux le pas de discrétisation, la solution numérique se rapproche de la solution exacte.

•

4.6 Analyse des méthodes à un pas

L'analyse d'une méthode numérique de résolution d'une EDO s'intéresse à quatre propriétés :

1. la consistance,
2. la zéro-stabilité,
3. la convergence,
4. la stabilité absolue.

Afin d'introduire les propriétés, quelques définitions préliminaires s'imposent.

Soit $\hat{y}_n$ la solution approchée au noeud t_n d'une méthode à un pas. Nous définissons

$$e_{n+1} = y(t_{n+1}) - \hat{y}(t_{n+1}), \quad n = 0, 1, \ldots \tag{4.19}$$

l'*erreur globale* de la méthode au noeud t_{n+1}, qui peut aussi être écrite sous la forme

$$e_{n+1} = y(t_{n+1}) - \hat{y}^*(t_{n+1}) + \hat{y}^*(t_{n+1}) - \hat{y}(t_{n+1}) \tag{4.20}$$

où $\hat{y}^*(t_{n+1})$ est la solution obtenue par la méthode à l'étape $n+1$ si $\hat{y}(t_n) = y(t_n)$.

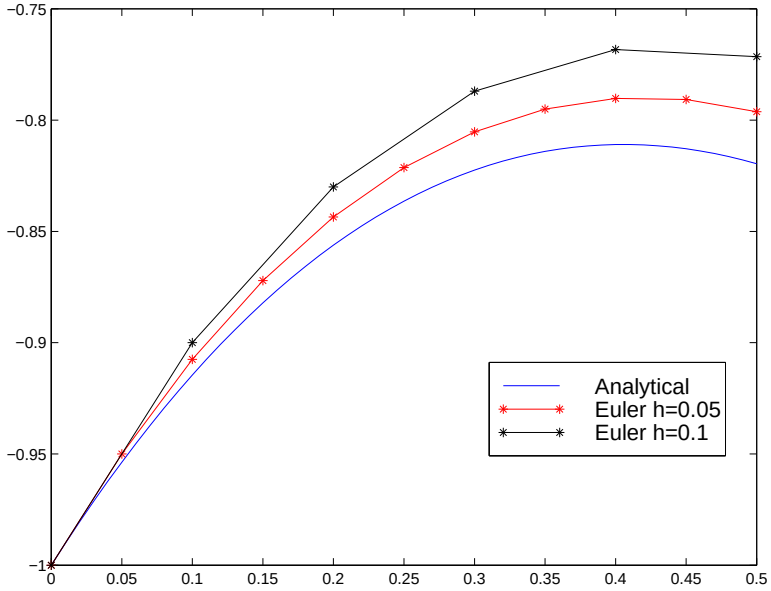


FIGURE 4.4 – Solution analytique et solution numérique du problème (4.12) en utilisant la méthode de Euler progressive avec deux pas d'intégration différents. (Script MATLAB `s_euler.m`)

On appelle *erreur de troncature locale* la quantité τ_{n+1} telle que

$$\epsilon_{n+1} = h\tau_{n+1} \tag{4.21}$$

où

$$\epsilon_{n+1} = y(t_{n+1}) - \hat{y}^*(t_{n+1}). \tag{4.22}$$

Nous désignons par

$$\tau(h) = \max_{0 \leq n \leq N_h - 1} |\tau_{n+1}(h)| \tag{4.23}$$

l'*erreur de troncature globale*.

L'erreur globale e_{n+1} et la quantité ϵ_{n+1} dans le cas de la méthode d'Euler progressive sont représentées sur la Figure 4.5.

4.6.1 Consistance

Sur la base des définitions précédentes, nous allons introduire la notion de consistance d'une méthode de résolution d'EDO. La propriété de consistance garantit que la méthode introduit à chaque étape une erreur qui tend vers zéro pour $h \to 0$.

Définition 26 (Consistance)**.** *Une méthode numérique de résolution des équations différentielles ordinaires est dite* consistante *si son erreur de troncature locale est un infiniment petit en h, ou $\mathcal{O}(h)$, c.à.d.*

$$\lim_{h \to 0} \tau(h) = 0 \tag{4.24}$$

Définition 27 (Consistance d'ordre p)**.** *Une méthode numérique est dite consistante d'ordre p pour un p entier positif, si $\forall t \in I$, la relation*

$$\tau(h) = \mathcal{O}(h^p) \quad \textit{pour } h \to 0 \tag{4.25}$$

est satisfaite.

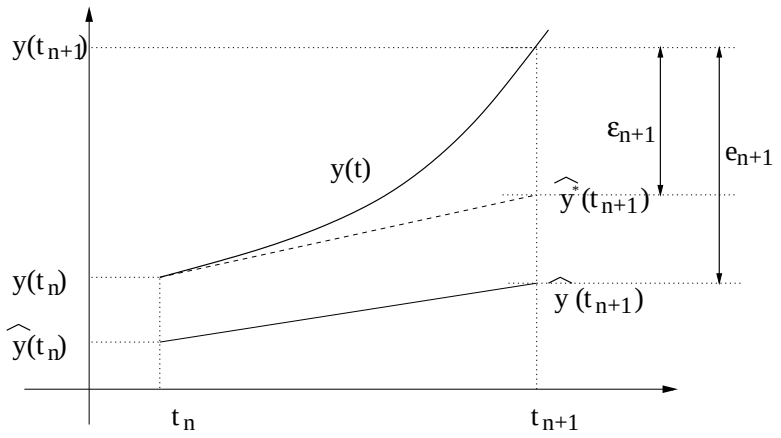


FIGURE 4.5 – Erreur globale et locale dans la méthode de Euler

4.6.1.1 Consistance de la méthode d'Euler

En ce qui concerne la méthode d'Euler progressive, puisque

$$\hat{y}^*(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) \tag{4.26}$$

et

$$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) + y''(\xi)h^2/2 \tag{4.27}$$

pour $\xi \in [t_n, t_{n+1}]$, l'erreur de troncature locale est

$$\tau_{n+1} = \frac{y(t_{n+1}) - \hat{y}^*(t_{n+1})}{h} = y''(\xi)h/2 \tag{4.28}$$

Il s'ensuit de la définition de consistance que la méthode d'Euler progressive est consistante d'ordre 1.

4.6.2 Zéro-stabilité

La propriété de stabilité d'une méthode numérique garantit la robustesse de la méthode par rapport aux inévitables erreurs introduites par l'arithmétique finie des ordinateurs. En particulier, deux types de stabilité sont étudiés dans la résolution d'EDO : la *zéro-stabilité* et la *stabilité absolue*. La propriété de zéro-stabilité concerne le comportement de la méthode numérique dans le cas limite $h \to 0$, alors que la propriété de stabilité absolue concerne le comportement pour $t_n \to \infty$ (Section 4.6.5).

Cette section abordera la notion de zéro-stabilité alors que la stabilité absolue sera traité en Section 4.6.5.

4.6.2.1 Zéro-stabilité de la méthode d'Euler

Considérons deux solutions $\hat{y}_1$ et $\hat{y}_2$ fournies par la méthode d'Euler progressive telles que

$$\begin{cases} \hat{y}_1(t_{n+1}) = \hat{y}_1(t_n) + hf(t_n, \hat{y}_1(t_n)) \\ \hat{y}_1(t_0) = y_0 \end{cases} \tag{4.29}$$

et

$$\begin{cases} \hat{y}_2(t_{n+1}) = \hat{y}_2(t_n) + h\left[f(t_n, \hat{y}_2(t_n)) + \delta(t_{n+1})\right] \\ \hat{y}_2(t_0) = y_0 + \delta(t_0) \end{cases} \tag{4.30}$$

Définition 28 (Zéro-stabilité). *Si pour toute perturbation $\delta(t_k) \leq \epsilon$, telle que $0 \leq k \leq N_h$ et $h < h_0$, il existe un $C > 0$ indépendant de ϵ tel que $|\hat{y}_1(t) - \hat{y}_2(t)| < C\epsilon$, alors la méthode est* zéro-stable.

Ce type de stabilité est une propriété caractéristique de la méthode numérique et pas du problème du Cauchy (garantie déjà par le théorème (4.1)).

En général, pour les méthodes à un pas, on peut énoncer le théorème suivant.

Théorème 4.2. *Considérons une méthode générique explicite et à un pas*

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + h\Phi(t_n, \hat{y}(t_n)) \tag{4.31}$$

Si la fonction d'incrément Φ est lipschitzienne par rapport à sa seconde variable, avec une constante de Lipschitz indépendante de h et des noeuds $t_j \in [t_0, t_0 + T]$ alors la méthode est zéro-stable.

Il en découle que la méthode d'Euler est zéro-stable si f est lipschitzienne.

4.6.3 Convergence

Définition 29 (Convergence). *Une méthode est dite convergente si $\forall n = 0, \ldots, N_h$*

$$|e(t_n)| = |y(t_n) - \hat{y}(t_n)| \leq C(h) \tag{4.32}$$

où $C(h)$ est un infiniment petit en h.

Si $\exists K > 0$ tel que

$$C(h) = Kh^p \tag{4.33}$$

la méthode est dite convergente avec un ordre p.

Selon le *théorème d'équivalence* ou théorème de *Lax-Richtmyer*, une méthode consistante et zéro-stable est aussi convergente. En plus, on peut montrer que l'ordre de convergence est égal à l'ordre de consistance.

Il en découle que la méthode de Euler progressive est convergente d'ordre $p = 1$.

4.6.3.1 Estimation de la convergence

Dans les études expérimentales de convergence, nous retrouvons souvent des graphes qui tracent l'erreur e en fonction de h en échelle logarithmique. Ceci signifie que le graphe porte en abscisse les valeurs de $\log(h)$ et en ordonnées les valeurs de $\log(e)$. L'avantage de cette représentation est évidente : si $e = Ch^p$ alors

$$\log(e) = \log(C) + p\log(h)$$

Par conséquent, p en représentation logarithmique correspond à la pente de la droite $\log(e)$. En d'autres termes, si deux méthodes diffèrent pour leur ordre de convergence, la méthode dont la pente en échelle logarithmique est la plus grande sera la méthode avec l'ordre le plus élevé.

Une méthode alternative à la méthode graphique pour établir l'ordre de convergence d'une méthode est la suivante. Soient e_i les erreurs pour certaines valeurs h_i, $i = 1, \ldots, H$. Nous formulons l'hypothèse que $e_i = Ch_i^p$ pour C indépendant de i. Il s'ensuit que p peut être estimé par les valeurs

$$p_i = \frac{\log\left(\frac{e_i}{e_{i-1}}\right)}{\log\left(\frac{h_i}{h_{i-1}}\right)} \qquad i = 2, \ldots, H$$

4.6.3.2 Convergence de la méthode d'Euler progressive

Dans cette section nous allons vérifier la convergence de la méthode d'Euler progressive.

On sait que d'après le développement de Taylor la solution $y(\cdot)$ satisfait la relation

$$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2} f'(\xi_n, y(\xi_n)) \tag{4.34}$$

Puisque la solution approchée d'Euler repose sur l'itération

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + hf(t_n, \hat{y}(t_n)) \tag{4.35}$$

on obtient que

$$y(t_{n+1}) - \hat{y}(t_{n+1}) = y(t_n) - \hat{y}(t_n) + h\left[f(t_n, y(t_n)) - f(t_n, \hat{y}(t_n))\right] + \frac{h^2}{2} f'(\xi_n, y(\xi_n))$$

Soit $e_n = y(t_n) - \hat{y}(t_n)$ l'erreur globale de la méthode. En vertu de la condition de Lipschitz on obtient

$$|e_{n+1}| \leq |e_n| + hL|y(t_n) - \hat{y}(t_n)| + \frac{h^2}{2}|f''(\xi_n, y(\xi_n))| \tag{4.36}$$

Si nous définissons

$$M_2 = \max_{t_0 \leq t \leq t_0 + T} |f'(t)| \tag{4.37}$$

il s'ensuit alors que

$$|e_{n+1}| \leq |e_n|(1 + hL) + \frac{h^2}{2} M_2 \tag{4.38}$$

On peut donc appliquer le théorème suivant :

Théorème 4.3. *Soit d_n une suite de réels positifs telle que*

$$d_{n+1} \leq (1 + \delta)d_n + \mu \qquad n = 0, 1, \ldots \tag{4.39}$$

Si $\delta > 0$ et $\mu > 0$, alors pour tout $n \geq 0$

$$d_n \leq \exp^{n\delta} d_0 + \mu \frac{\exp^{n\delta} - 1}{\delta} \tag{4.40}$$

Grâce au théorème, en posant $\delta = hL$ et $\mu = h^2 M_2/2$, on obtient

$$|e_n| \leq \exp^{nhL} |e_0| + hM_2 \frac{\exp^{nhL} - 1}{2L} = \frac{\exp^{L(t_n - t_0)} - 1}{L} \frac{M_2}{2} h \tag{4.41}$$

puisque $e_0 = 0$. Étant donné que

$$t_n - t_0 \leq T \tag{4.42}$$

on obtient

$$\max_{0 \leq n \leq N_h} |e_n| = \max_{0 \leq n \leq N_h} |y(t_n) - \hat{y}(t_n)| \leq \frac{\exp^{LT} - 1}{L} \frac{M_2}{2} h \tag{4.43}$$

L'erreur globale de la méthode d'Euler est bornée par une constante fois h. L'erreur est alors infiniment petite en h.

En plus, puisque l'erreur de troncature locale $\tau(h) \leq (M_2/2)h$, on déduit que l'erreur globale tend vers zéro avec le même ordre que l'erreur de troncature locale.

4.6.4 Erreur d'arrondi et méthode d'Euler

Nous avons montré que l'erreur globale de la méthode d'Euler est infiniment petite en h. Une question surgit alors naturellement : *Faut-il en pratique prendre toujours le plus petit h pour résoudre numériquement une EDO par la méthode d'Euler ?*

La réponse est non.

La justification est que les résultats théoriques découlent de l'hypothèse que tous les calculs sont faits en arithmétique exacte. En réalité, en utilisant la méthode d'Euler sur un ordinateur on obtient à chaque étape

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + h(f(t_n, \hat{y}(t_n)) + \epsilon_n) + \rho_n \tag{4.44}$$

où ϵ_n est l'erreur d'arrondi qui suit au calcul de $f(t_n, \hat{y}(t_n))$ et ρ_n est l'erreur d'arrondi du résultat de l'itération.

En supposant $\epsilon_n \leq \epsilon$ et $\rho_n \leq \rho$, pour tout n et $h \leq h_0$, on obtient que

$$\max_{0 \leq n \leq N_h} |y(t_n) - \hat{y}(t_n)| \leq \frac{\exp^{LT} - 1}{L} \left(\frac{M_2}{2} h + \epsilon + \frac{\rho}{h} \right) \tag{4.45}$$

Il s'ensuit que la composante de l'erreur finale due à l'erreur de génération (Section 2.5.2) est autant plus importante que le pas de discrétisation est petit.

La présence d'erreurs d'arrondi ne permet donc pas de conclure que l'erreur tend vers zéro quand $h \to 0$. En revanche, il existe une valeur optimale (non nulle) h_{opt} de h qui minimise l'erreur : pour $h < h_{\text{opt}}$, l'erreur d'arrondi domine l'erreur de troncature et l'erreur globale augmente.

Exemple Dans cet exemple nous montrons les effets des erreurs d'arrondi dans la solution numérique fournie par la méthode de Euler progressive. Considérons une notation à virgule flottante avec une base $b = 10$ et $t = 2$ chiffres significatifs. Les solutions numériques du problème (4.12) pour une série de pas de discrétisation décroissante (de $h = 0.1$ à $h = 0.005$) sont illustrées sur la Figure 4.6.

L'exécution du script `MATLAB`$^{©}$`s_euler_arr.m` montre qu'en diminuant h on obtient d'abord une amélioration de la solution ($h > h_{\text{opt}}$) et puis une dégradation ($h < h_{\text{opt}}$).

•

4.6.5 Stabilité absolue

Jusqu'ici nous avons considéré un intervalle d'intégration I fixé et borné supérieurement. La notion de *stabilité absolue* concerne les propriétés de la méthode numérique pour $t_n \to \infty$.

L'analyse de stabilité absolue se limitera à considérer la solution d'un problème de Cauchy linéaire (aussi appelé *problème test*) de la forme

$$\begin{cases} y'(t) = \lambda y(t) \\ y(0) = 1 \end{cases} \tag{4.46}$$

où $\lambda \in \mathbb{R}$, $\lambda < 0$.

La solution analytique du problème test

$$y(t) = \exp^{\lambda t} \tag{4.47}$$

converge vers zéro pour $t \to \infty$.

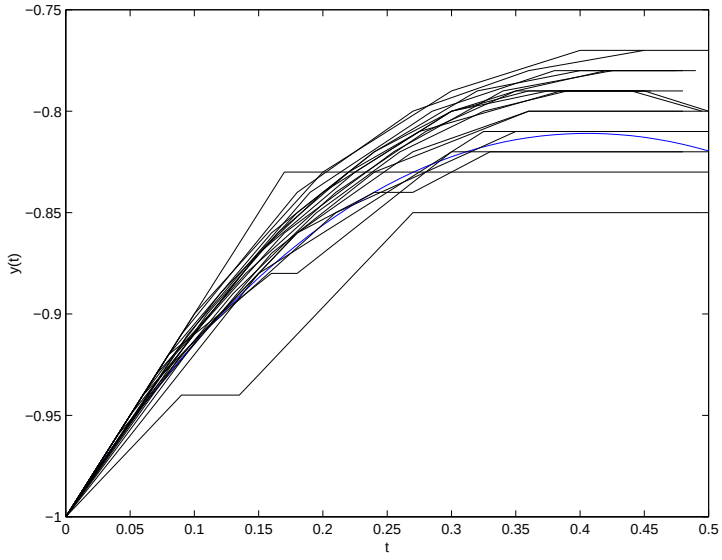


FIGURE 4.6 – Solutions du problème (4.12) en utilisant la méthode de Euler progressive pour différents h et avec une notation à virgule flottante ($b = 10$ et $t = 2$). (Script MATLAB `s_euler_arr.m`)

Définition 30 (Méthode absolument stable)**.** *Une méthode numérique pour l'approximation du problème test est* absolument stable *si*

$$\lim_{t_n \to \infty} |\hat{y}(t_n)| = 0 \tag{4.48}$$

Définition 31 (Région de stabilité absolue)**.** *La région de stabilité absolue de la méthode numérique pour l'approximation du problème test $y' = \lambda y$ est le sous-ensemble*

$$\mathcal{A} = \{z = h\lambda \in \mathbb{R} : \lim_{t_n \to \infty} |\hat{y}(t_n)| = 0\} \tag{4.49}$$

4.6.5.1 Stabilité absolue et Euler progressive

La méthode d'Euler progressive appliquée au problème test repose sur le schéma itératif

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + h\lambda\hat{y}(t_n) = \hat{y}(t_n)(1 + h\lambda) \tag{4.50}$$

où $y(t_0) = \hat{y}(t_0) = 1$.

Il s'ensuit que

$$\hat{y}(t_1) = (1 + h\lambda), \quad \hat{y}(t_2) = (1 + h\lambda)\hat{y}(t_1) = (1 + h\lambda)^2, \ldots$$

Donc, en procédant par récurrence sur n on a

$$\hat{y}(t_n) = (1 + h\lambda)^n \tag{4.51}$$

Alors la méthode est absolument stable si et seulement si $|1 + h\lambda| < 1$, ce qui revient à

$$0 < h < -\frac{2}{\lambda} \text{ où } \lambda < 0 \tag{4.52}$$

Autrement, la méthode d'Euler progressive est instable.

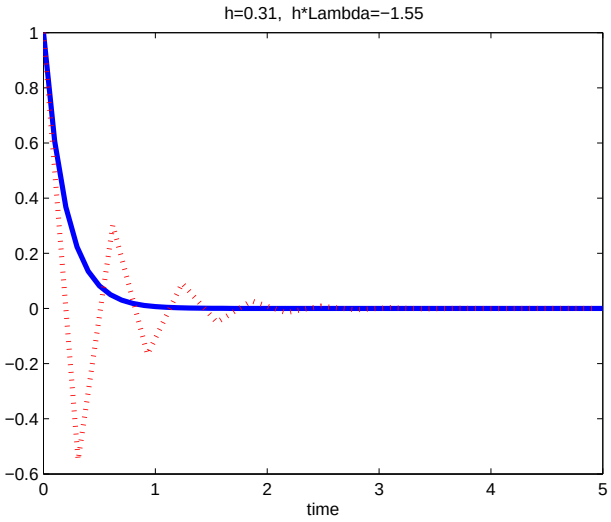


FIGURE 4.7 – Analyse de stabilité absolue. La ligne continue représente la solution analytique alors que la ligne pointillée représente la solution d'Euler progressive pour $h = 0.31$. (Script MATLAB `s_eul_stababs.m`)

Exemple Cet exemple nous permet de visualiser la notion de stabilité absolue dans le cas d'une résolution basée sur la méthode d'Euler progressive.

Considérons la méthode d'Euler progressive appliquée au problème de Cauchy

$$y'(t) = -5y(t), \qquad y(0) = 1 \tag{4.53}$$

Les deux solutions correspondantes à deux différentes valeurs de h ($h = 0.41$ et $h = 0.31$) sont mises en graphique en Figure 4.7 et 4.8. On remarque que la solution en Figure 4.8 n'est pas stable à cause de la valeur trop grande du pas h.

•

4.7 Les autres méthodes d'Euler

La version implicite de la méthode d'Euler à un pas est connue sous le nom de *méthode d'Euler rétrograde*. Cette méthode se base sur l'itération

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + hf(t_{n+1}, \hat{y}(t_{n+1})) \tag{4.54}$$

où le terme inconnu $\hat{y}(t_{n+1})$ apparait dans les deux membres de l'équation. Donc, chaque itération demande la résolution d'une équation non linéaire afin de calculer la valeur $\hat{y}(t_{n+1})$. Par conséquent, la méthode d'Euler implicite s'avère être plus onéreuse que la méthode progressive. En retour, la méthode est absolument stable pour tout h. De plus, on peut montrer que la méthode est d'ordre 1.

Un autre exemple de méthode d'Euler implicite est la *méthode d'Euler modifiée ou du trapèze* qui à chaque itération résout l'équation

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + \frac{h}{2}\left[f(t_n, \hat{y}(t_n)) + f(t_{n+1}, \hat{y}(t_{n+1}))\right] \tag{4.55}$$

On peut montrer que cette méthode est d'ordre 2.

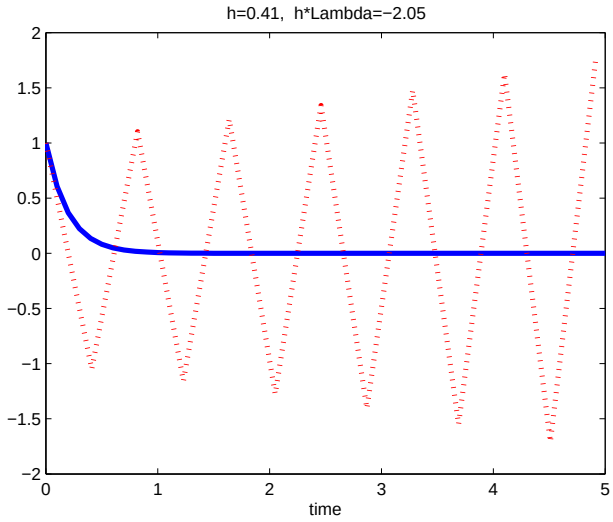


FIGURE 4.8 – Analyse de stabilité absolue. La ligne continue représente la solution analytique alors que la ligne pointillée représente la solution d'Euler progressive pour $h = 0.41$. (Script MATLAB `s_eul_stababs.m`)

La méthode d'Euler *prédicteur-correcteur* est une méthode qui approche la méthode implicite en utilisant deux étapes explicites

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + hf(t_{n+1}, \hat{y}_n + hf(t_n, \hat{y}_n)) \tag{4.56}$$

L'itération précedente peut être decomposée en deux étapes :

1. Prédiction : $\hat{y}_p(t_{n+1}) = \hat{y}(t_n) + hf(t_n, \hat{y}_n)$
2. Correction : $\hat{y}(t_{n+1}) = \hat{y}(t_n) + hf(t_{n+1}, \hat{y}_p(t_{n+1}))$

On peut montrer que si la méthode pour la prédiction est d'ordre $p-1$ et la méthode pour la correction est d'ordre p, alors l'ordre de la méthode prédicteur-correcteur est égal à p.

Exemple La solution numérique du problème (4.12) en utilisant la méthode d'Euler prédicteur-correcteur est tracée en Figure 4.9.

•

4.8 Méthodes de Runge-Kutta

Les méthodes de Runge-Kutta (RK) sont des méthodes à un pas qui augmentent leur précision en augmentant le nombre d'évaluations de la fonction f à chaque pas de temps.

La méthode classique pour construire une méthode RK explicite consiste à faire en sorte que le plus grand nombre de termes du développement de Taylor de la solution exacte $y(t_{n+1})$ coïncide avec ceux de la solution approchée $\hat{y}(t_{n+1})$ (en supposant de connaître la solution exacte $y(t_n)$ pour $t = t_n$).

Dans sa forme la plus générale, une méthode RK peut s'écrire

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + hF(t_n, \hat{y}(t_n), h, f) \tag{4.57}$$

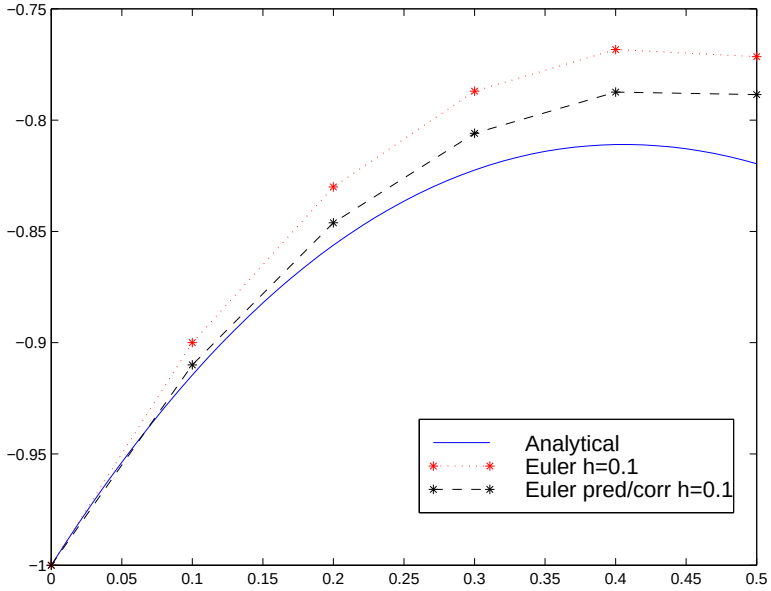


FIGURE 4.9 – Solution analytique (ligne en continu) et solutions numériques du problème (4.12) en utilisant la méthode d'Euler progressive (ligne en pointillé) et la méthode de Euler prédicteur-correcteur (ligne en trait) avec le même pas $h = 0.1$. (Script MATLAB `s_euler_pc.m`)

où F est définie par

$$F(t_n, \hat{y}(t_n), h, f) = \sum_{i=1}^{s} b_i K_i$$

$$K_i = f(t_n + c_i h, \hat{y}(t_n) + h \sum_{j=1}^{s} a_{ij} K_j), \qquad i = 1, 2, \ldots, s$$

et s désigne le nombre d'*étapes* de la méthode. Les coefficients $\{a_{ij}\}$, $\{c_i\}$ et $\{b_i\}$ caractérisent complètement une méthode RK. Nous supposerons que la condition suivante est satisfaite

$$c_i = \sum_{j=1}^{s} a_{ij} \qquad i = 1, \ldots, s \tag{4.58}$$

On peut montrer que si $a_{ij} = 0$ pour $j \geq i$, la méthode est explicite.

4.8.1 Méthode de RK explicite d'ordre 2

Considérons une méthode explicite RK d'ordre $s = 2$. En vertu de la formule (4.58)

$$c_1 = \sum_{j=1}^{2} a_{1j} = a_{11} + a_{12} = 0$$

$$c_2 = \sum_{j=1}^{2} a_{2j} = a_{21} + a_{22} = a_{21}$$

Supposons que à l'instant t_n on dispose de la solution exacte $y(t_n)$. Donc la méthode RK2 peut être écrite sous la forme

$$\begin{aligned}\hat{y}(t_{n+1}) &= y(t_n) + h(b_1K_1 + b_2K_2)\\ K_1 &= f(t_n, y(t_n))\\ K_2 &= f(t_n + hc_2, y(t_n) + hc_2K_1)\end{aligned}$$

En écrivant le développement de Taylor de K_2 en t_n au second ordre, on obtient

$$K_2 = f(t_n) + \frac{\partial f}{\partial t}(t_n)hc_2 + \frac{\partial f}{\partial y}(t_n)hc_2K_1 = \\ = f(t_n) + hc_2(f_{n,t} + K_1 f_{n,y}) + O(h^2)$$

et

$$\hat{y}(t_{n+1}) = y(t_n) + h(b_1K_1 + b_2K_2) = \\ = y(t_n) + hb_1f(t_n) + hb_2f(t_n) + h^2b_2c_2(f_{n,t} + K_1f_{n,y}) + O(h^3)$$

où par définition

$$f_{n,t} \equiv \frac{\partial f}{\partial t}(t_n), \qquad f_{n,y} \equiv \frac{\partial f}{\partial y}(t_n)$$

Si on effectue le même développement sur la solution exacte, on obtient

$$y(t_{n+1}) = y(t_n) + hf(t_n) + \frac{h^2}{2}(f_{n,t} + f(t_n)f_{n,y}) + O(h^3) \tag{4.59}$$

En identifiant les termes des deux développements jusqu'à l'ordre le plus élevé, on trouve que les coefficients de la méthode RK doivent vérifier

$$b_1 + b_2 = 1, \qquad c_2b_2 = 1/2 \tag{4.60}$$

où on peut choisir de façon arbitraire un des coefficients.

Si cette relation est satisfaite, la méthode est consistante d'ordre 2.

Exemple La solution numérique du problème (4.12) obtenue en utilisant la méthode Runge-Kutta d'ordre 2 avec $h = 0.1$ est tracée en Figure 4.10 avec la solution donnée par la méthode d'Euler pour le même h.

•

4.8.2 La méthode de Runge Kutta d'ordre 4

Un exemple de méthode RK du quatrième ordre ($s = 4$) est donné par le schéma explicite à 4 étapes suivant :

$$\begin{aligned}K_1 &= f(t_n, \hat{y}(t_n))\\ K_2 &= f(t_n + \frac{1}{2}h, \hat{y}(t_n) + \frac{1}{2}K_1)\\ K_3 &= f(t_n + \frac{1}{2}h, \hat{y}(t_n) + \frac{1}{2}K_2)\\ K_4 &= f(t_n + h, \hat{y}(t_n) + K_3)\\ \hat{y}(t_{n+1}) &= \hat{y}(t_n) + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)\end{aligned}$$

On peut montrer que la méthode est convergente d'ordre 4.

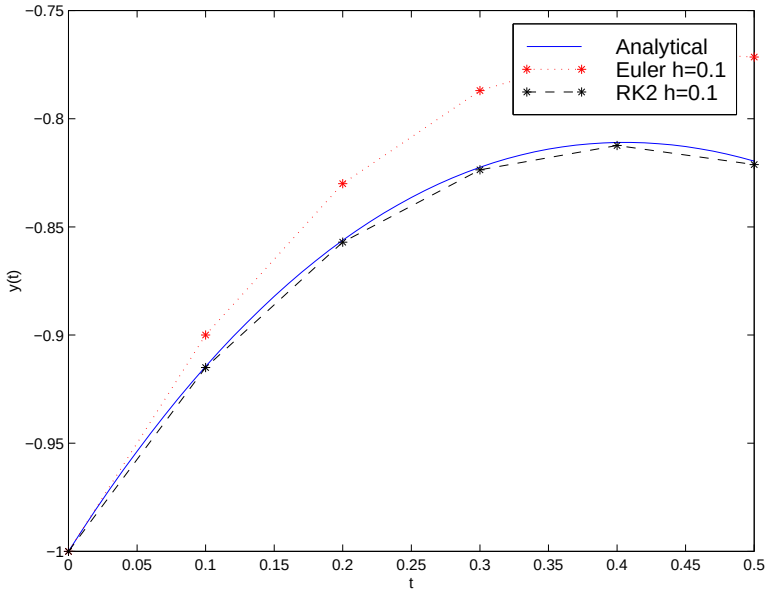


FIGURE 4.10 – Solution analytique et solutions numériques du problème (4.12) en utilisant la méthode de Euler progressive (ligne en pointillé) et la méthode de Runge-Kutta (ligne en trait) d'ordre 2 avec le même pas $h = 0.1$. (Script MATLAB `s_rk2.m`)

4.9 Analyse des méthodes RK

Nous mentionons les théorèmes suivants sur les propriétés des méthodes RK :

Théorème 4.4. *Une méthode RK est consistante si et seulement si*

$$\sum_{i=1}^{s} b_i = 1 \tag{4.61}$$

En plus, puisque les méthodes RK sont des méthodes à un pas, la consistance implique la zéro-stabilité et donc la convergence. La relation (4.60) montre donc que la méthode RK2 est convergente d'ordre 2.

Le résultat suivant établit une relation entre l'ordre et le nombre d'étapes des méthodes RK explicites.

Propriété 4.1. *L'ordre d'une méthode RK explicite à s étapes ne peut pas être plus grand que s. De plus, il n'existe pas de méthode à s étapes d'ordre s si $s \geq 5$.*

En ce qui concerne la stabilité, nous rappelons les résultats suivants.

Théorème 4.5 (Stabilité absolue des méthodes RK)**.** *Considérons une méthode RK explicite d'ordre $s = 1, \ldots, 4$. La méthode est absolument stable ssi $|R(h\lambda)| < 1$ où*

$$R(h\lambda) = 1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \cdots + \frac{1}{s!}(h\lambda)^s \tag{4.62}$$

On vérifie que la taille des régions de stabilité absolue augmente quand l'ordre augmente.

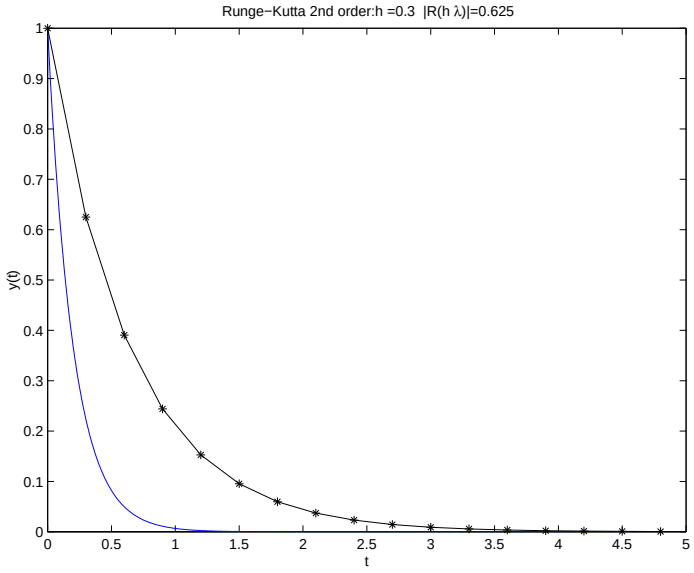


FIGURE 4.11 – Solution analytique (ligne continue) et solution RK2 du problème (4.65) pour $h = 0.3$. La solution est absolument stable (Script MATLAB `s_rk2_stab.m`).

Exemple Dans cet exercice nous allons retrouver le résultat théorique énoncé dans le théorème précédent pour le cas d'une méthode RK2, c-à-d. $s = 2$.

En posant $b_1 = b_2 = 0.5$, $c_2 = 1$, puisque

$$\begin{aligned} \hat{y}_{n+1} &= \hat{y}_n + \frac{h}{2}(K_1 + K_2) \\ K_1 &= f_n = \lambda \hat{y}_n \\ K_2 &= f(t_n + h, \hat{y}_n + hK_1) = \lambda(\hat{y}_n + h\lambda \hat{y}_n) \end{aligned}$$

on obtient

$$\hat{y}_{n+1} = \hat{y}_n + (h\lambda + \frac{1}{2}h^2\lambda^2)\hat{y}_n = (1 + h\lambda + \frac{1}{2}h^2\lambda^2)\hat{y}_n \tag{4.63}$$

Semblablement à la relation (4.51), on retrouve par récurrence sur n la condition de stabilité absolue pour $s = 2$

$$\left|1 + h\lambda + \frac{1}{2}(h\lambda)^2\right| < 1 \tag{4.64}$$

•

Exemple Considérons la méthode RK2 appliquée au problème de Cauchy

$$y'(t) = -5y(t), \qquad y(0) = 1 \tag{4.65}$$

avec un pas d'integration $h = 0.3$. Puisque $|R(h\lambda)| = 0.625$, la solution est stable. La solution est mise en graphique sur la Figure 4.11.

•

4.10 Les méthodes multi-pas

Nous rappelons la definition suivante

Définition 32. *Une méthode numérique pour la résolution d'une EDO est dite à* un pas *si* $\forall n \geq 0$, $\hat{y}_{n+1}$ *ne dépend que de* $\hat{y}_n$. *Autrement le schéma est une méthode* multi-pas *(où à pas multiples).*

Voici des exemples de méthodes à *deux* pas :

Méthode du point milieu

$$\hat{y}(t_{n+1}) = \hat{y}(t_{n-1}) + 2hf(t_n, \hat{y}(t_n)) \qquad n \geq 1 \tag{4.66}$$

qui est une méthode explicite.

Méthode de Simpson

$$\begin{aligned}\hat{y}(t_{n+1}) &= \hat{y}(t_{n-1}) + \\ &+ \frac{h}{3}\left[f(t_{n-1}, \hat{y}(t_{n-1})) + 4f(t_n, \hat{y}(t_n)) + f(t_{n+1}, \hat{y}(t_{n+1}))\right] \qquad n \geq 1\end{aligned}$$

qui est un exemple de schéma implicite.

La formule générique d'une méthode multi-pas linéaire à $p+1$ pas, $p \in \mathbb{N}$ est

$$\hat{y}(t_{n+1}) = \sum_{j=0}^{p} a_j \hat{y}(t_{n-j}) + h \sum_{j=-1}^{p} b_j f(t_{n-j}, \hat{y}(t_{n-j})) \qquad n = p, p+1, \ldots \tag{4.67}$$

où $a_j, b_j \in \mathbb{R}$ et $a_p^2 + b_p^2 \neq 0$, et où on suppose connues les valeurs initiales

$$\{y_0 = \hat{y}_0, \hat{y}_1, \ldots, \hat{y}_p\}$$

Si $b_{-1} = 0$ la méthode est explicite, autrement elle est implicite. Notons que

- si $p = 0$, $a_0 = 1$, $b_0 = 1$, $b_{-1} = 0$ en (4.67), alors on retrouve la méthode d'Euler explicite
- si $p = 0$, $a_0 = 1$, $b_0 = 0$, $b_{-1} = 1$ en (4.67), on retrouve la méthode d'Euler implicite.

Parmi les methodes multi-pas linéaires, nous nous attarderons sur les méthodes d'Adams.

4.10.1 Méthodes d'Adams

A partir de la relation

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(\tau, y(\tau))d\tau \tag{4.68}$$

les méthodes multi-pas d'Adams proposent comme solution numérique

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + \int_{t_n}^{t_{n+1}} \Pi_{p+\gamma}(\tau)d\tau \tag{4.69}$$

où $\Pi_{p+\gamma}$ est le polynôme d'ordre $p+\gamma$ interpolant (Chapitre 5) les $p+\gamma+1$ couples $(t_j, f(t_j, \hat{y}(t_k)))$ pour $j = n+\gamma, \ldots, n+\gamma-p$.

Si nous posons $\gamma = 0$, nous obtenons les méthodes explicites connues sous le nom de méthodes d'Adams-Bashforth. Pour $\gamma = 1$ les méthodes sont implicites et connues sous le nom de méthodes d'Adams-Moulton.

4.10.1.1 Méthodes d'Adams-Bashforth

On peut répartir les méthodes d'Adams-Bashforth selon la valeur de p. Voyons quelques examples.

- **p=0**

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + hf(t_n, \hat{y}(t_n)) \tag{4.70}$$

  On retrouve la méthode d'Euler progressive !

- **p=1**

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + \frac{h}{2}(3f(t_n, \hat{y}(t_n)) - f(t_{n-1}, \hat{y}(t_{n-1}))) \tag{4.71}$$

- **p=2**

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + \frac{h}{12}(23f(t_n, \hat{y}(t_n)) - 16f(t_{n-1}, \hat{y}(t_{n-1})) + \\ + 5f(t_{n-2}, \hat{y}(t_{n-2})))$$

4.10.1.2 Méthodes d'Adams-Moulton

Ce sont les méthodes d'Adams pour $\gamma = 1$. Voyons quelques exemples pour des valeurs de p différentes.

- **p=0**

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + hf(t_{n+1}, \hat{y}(t_{n+1})) \tag{4.72}$$

  On retrouve la méthode d'Euler rétrograde et implicite !

- **p=0**

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + \frac{h}{2}(f(t_n, \hat{y}(t_n)) + f(t_{n+1}, \hat{y}(t_{n+1}))) \tag{4.73}$$

- **p=1**

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + \frac{h}{12}(5f(t_{n+1}, \hat{y}(t_{n+1})) + 8f(t_n, \hat{y}(t_n)) + \\ - f(t_{n-1}, \hat{y}(t_{n-1})))$$

4.11 Exercices

4.11.1 Méthode de Taylor

Considérons le problème de Cauchy

$$\begin{cases} y'(t) = t + y(t) \\ y(0) = 2 \end{cases}$$

Donner une solution par la méthode de Taylor de degré 3.

4.11.2 Méthodes d'Euler

1. Soit le problème de Cauchy :

$$\begin{cases} y'(t) = t + y(t) \\ y(0) = 2 \end{cases}$$

En posant $h = 0.1$, calculer les quatre premières approximations $\hat{y}(t_n)$ des solutions $y(t_n)$ par la méthode d'Euler progressive. Sachant que la solution analytique est $y = 3e^t - t - 1$, calculer l'erreur absolue pour chacune des quatre étapes. Que remarque-t-on ?

2. Résoudre

$$\begin{cases} y'(t) = t^2 - y(t) \\ y(0) = 1 \end{cases}$$

par la méthode d'Euler progressive.

   (a) Utiliser une première fois la méthode avec $h = 0.2$ et faire deux itérations.

   (b) Ensuite, utiliser la méthode une deuxième fois avec $h = 0.1$ et faire quatre itérations.

   (c) Comparer les résultats avec la solution exacte $y(0.4)$ sachant que la solution est $y(t) = -e^{-t} + t^2 - 2t + 2$

3. Résoudre

$$\begin{cases} y'(t) = ty(t) + t \\ y(0) = 0 \end{cases}$$

sur $[0, 0.5]$ avec $h = 0.1$ par la méthode d'Euler rétrograde.

4.11.3 Condition de Lipschitz

1. Montrer que

   (a) la fonction

$$f(t, y) = t^2 cos^2(y) + t sin^2(y)$$

définie pour $|t| < 1$ et $y \in \mathbb{R}$ satisfait la condition de Lipschitz avec une constante $L = 4$;

   (b) la fonction

$$f(t, y) = |y|$$

satisfait la condition de Lipschitz même si elle ne satisfait pas la condition suffisante.

4.11.4 Stabilité absolue

On considère la famille de méthodes suivante (dites θ-méthodes) :

$$\hat{y}(t_{n+1}) = \hat{y}(t_n) + h\theta f(t_{n+1}, \hat{y}(t_{n+1})) + h(1-\theta) f(t_n, \hat{y}(t_n))$$

h étant le pas de temps et $\theta \in [0,1]$, pour approcher le problème

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(0) = y_0 \end{cases}$$

Noter que $\hat{y}(t_n)$ est ici considéré comme une approximation de $y(t_n)$.

1. Vérifier que pour $\theta = 0$, on retrouve le schéma d'Euler progressif tandis que pour $\theta = 1$, on retrouve le schéma d'Euler rétrograde.
2. Soit $\lambda < 0$ un nombre réel négatif donné. On considère le problème test

   $$\begin{cases} y'(t) = \lambda y(t) \\ y(0) = y_0 \end{cases}$$

   Trouver en fonction de θ, les valeurs de h pour lesquelles les θ-méthodes sont stables (c'est-à-dire $\lim_{n\to\infty} \hat{y}(t_n) = 0$).
3. Retrouver, à partir du résultat du point précédent, les conditions de stabilité des schémas d'Euler progressif et rétrograde.

4.11.5 Méthodes de Runge-Kutta

Une méthode RK explicite d'ordre $s = 1, \ldots, 4$ est absolument stable ssi $|R(h\lambda)| < 1$ où

$$R(h\lambda) = 1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \cdots + \frac{1}{s!}(h\lambda)^s$$

Vérifier ce résultat sur la méthode RK explicite où $s = 2$ avec $b_1 = b_2 = 0.5, c_2 = 1$.

Chapitre 5

Interpolation

5.1 Exemples et motivations

Dans plusieurs problèmes de calcul numérique il est important de reconstruire le comportement d'une fonction à partir de ses valeurs en certains points.

Considérons par exemple une fonction $f(\cdot)$, connue sous forme analytique mais difficile à évaluer, différentier ou intégrer par ordinateur. Pour des raisons numériques, il est envisageable de remplacer cette fonction par une approximation qui soit la plus fidèle possible mais avec une expression algébrique plus abordable. Ceci est le cas de la résolution numérique d'équations différentielles (Section 4.10) .

D'autres applications apparaissent dans la modélisation des données et dans l'estimation des modèles de prédiction. Considérons par exemple une étude démographique [4] qui collectionne des données sur les prix des maisons dans différents quartiers d'une ville en fonction des caractéristiques du quartier (taux de criminalité, pollution, distance du centre ville, accessibilité de transport en commun,...) (Table 5.1). Il est envisageable, par exemple pour un agent immobilier, de construire sur la base des observations collectées un modèle prédictif qui puisse donner une bonne estimation du prix d'un nouvel établissement sur la base des variables caractéristiques du quartier. Dans ce cas, il est intéressant d'avoir une méthode qui puisse donner une représentation synthétique de données expérimentales.

PRICE	CRIM	ZN	NOX	DIS	RAD
296.0	0.00632	18.00	0.5380	4.0900	1
242.0	0.02731	0.00	0.4690	4.9671	2
242.0	0.02729	0.00	0.4690	4.9671	2
222.0	0.03237	0.00	0.4580	6.0622	3
...	...	...	...	...	...

TABLE 5.1 – Un sous-ensemble des données du dataset *Boston house-price data* (1978)

Les deux exemples sont représentatifs des deux cas qui peuvent se présenter dans le traitement des données :

- **déterministe** : les valeurs observées sont considérées comme exactes. Le problème de la prédiction est abordé par les méthodes d'interpolation. Ce problème sera traité dans ce chapitre.
- **stochastique** : les valeurs sont entachées d'erreur. Le problème de la prédiction est abordé par les méthodes de lissage (*fitting*) des données. Ces méthodes seront traitées dans le chapitre suivant.

5.2 Le problème de l'interpolation

Soient donnés une fonction $f : \mathbb{R} \Rightarrow \mathbb{R}$ et $n+1$ couples (x_i, y_i), $i = 0, \ldots, n$, tels que $x_i \in \mathbb{R}$, $y_i \in \mathbb{R}$ et

$$f(x_i) = y_i \tag{5.1}$$

Les méthodes d'interpolation ont pour but de trouver une fonction $\phi(\cdot)$ telle que $\phi(x_i) = y_i$ pour $i = 0, \ldots, n$. Les valeurs x_i sont aussi connues sous le nom de *noeuds d'interpolation.*

Parmi les méthodes d'interpolation nous retrouvons les méthodes suivantes :

Interpolation polynomiale où la fonction interpolante $\phi(\cdot)$ est un polynôme.

Approximation trigonométrique où la fonction interpolante $\phi(\cdot)$ est un polynôme trigonométrique.

Interpolation polynomiale par morceaux où la fonction interpolante $\phi(\cdot)$ est une fonction polynomiale par morceaux (*splines*).

Dans ce qui suit, nous nous attarderons sur l'interpolation polynomiale dans la Section 5.3 et sur l'interpolation par morceaux dans la Section 5.4.

5.3 L'interpolation polynomiale

Considérons $n+1$ couples (x_i, y_i), $i = 0, \ldots, n$. Soit P^n est l'ensemble des polynômes d'ordre n Le problème de l'interpolation polynomiale consiste à trouver un polynôme $\Pi_n \in P^n$

$$\Pi_n(x) = a_n x^n + \cdots + a_1 x + a_0, \qquad a_i \in \mathbb{R}, \quad i = 0, \ldots, n \tag{5.2}$$

tel que $\Pi_n(x_i) = y_i$, pour $i = 0, \ldots, n$.

La recherche de ce polynôme est justifiée par un important théorème d'existence et unicité.

Théorème 5.1. *[Existence et unicité] Étant donnés $n+1$ points distincts $x_0, \ldots, x_n$ et $n+1$ valeurs correspondantes $y_0, \ldots, y_n$, il existe un unique polynôme $\Pi_n \in P_n$ tel que $\Pi_n(x_i) = y_i$ pour $i = 0, \ldots, n$.*

Cependant, la preuve théorique de l'existence du polynôme ne nous fournit pas les valeurs des paramètres a_i, $i = 0, \ldots, n$ en (5.2). Il existe deux approches possibles pour le calcul des valeurs des paramètres a_i à partir des $n+1$ couples (x_i, y_i).

Méthode générale : cette méthode consiste à remplacer les coordonnées des couples dans l'expression du polynôme (5.2) et à résoudre le système linéaire résultant (Chapitre 3). Malheureusement cette méthode, même si elle est intuitive, présente les problèmes suivants : elle s'appuie sur une procédure de calcul très coûteuse ($O(n^3)$) et surtout elle peut mener à des systèmes linéaires mal conditionnés (Section 3.10.1).

Méthodes ad hoc : ces approches ont été développées expressément pour résoudre numériquement le problème de l'interpolation polynomiale. Parmi elles, nous comptons les méthodes de Lagrange et de Newton, qui ont complexité $O(n^2)$ et qui seront développées dans les Sections 5.3.1 et 5.3.2, respectivement.

5.3.1 La méthode de Lagrange

Soient donnés les $n+1$ couples (x_i, y_i), $i = 0, \ldots, n$. Nous définissons *polynôme de Lagrange* un polynôme interpolant de la forme

$$\Pi_n(x) = \sum_{i=0}^{n} y_i l_i(x) \tag{5.3}$$

où les polynômes

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \qquad i = 0, \ldots, n \tag{5.4}$$

sont dits les *polynômes caractéristiques de Lagrange*. Les polynômes caractéristiques de Lagrange ne dépendent pas des ordonnées y_i et jouissent des propriétés suivantes :

$$l_i(x_i) = 1 \tag{5.5}$$

$$l_i(x_j) = 0, \qquad \forall j \neq i \tag{5.6}$$

Exemple Étant donné les 4 couples ($n = 3$)

x	x_0	x_1	x_2	x_3
y	y_0	y_1	y_2	y_3

le polynôme interpolant de Lagrange est le suivant

$$\begin{aligned}\Pi_3(x) = \\ = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}y_0 + \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}y_1 + \\ + \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}y_3\end{aligned}$$

Notons que le polynôme est d'ordre 3 et que tous les termes x_i et y_i, $i = 0, \ldots, 3$, sont des scalaires.

•

Exemple La figure 5.1 montre l'allure de 7 polynômes caractéristiques de Lagrange ($n = 6$) pour les 7 valeurs x_i ($i = 0, \ldots, 6$) suivantes : $-3, -2, -1, 0, 1, 2, 3$.

Notons que les propriétés (5.5) et (5.6) sont satisfaites par tous ces polynômes. Par exemple, $l_0(-3) = 1$ et $l_0(-2) = l_0(-1) = l_0(0) = l_0(1) = l_0(2) = 0$.

•

Exemple La figure 5.2 montre le polynôme interpolant de Lagrange passant par les 7 couples (x_i, y_i) où $y_i = x_i^3$ et x_i prend ses valeurs dans l'ensemble $\{-3, -2, -1, 0, 1, 2, 3\}$. Notons que les abscisses des 7 couples coïncident avec les valeurs x_i prises en considération dans l'exemple précèdent. Donc, les polynômes caractéristiques employés dans l'interpolation de Lagrange sont ceux affichés en Figure 5.1.

•

Le polynôme interpolant (5.3) peut être écrit sous la forme

$$\Pi_n(x) = \sum_{i=0}^{n} \frac{\omega_{n+1}(x)}{(x - x_i)\omega'_{n+1}(x_i)} y_i \tag{5.7}$$

où $\omega_{n+1}(x)$ est appelé le *polynôme nodal* de degré $n + 1$ et est défini par

$$\omega_{n+1}(x) = \prod_{j=0}^{n} (x - x_j) \tag{5.8}$$

et

$$\omega'_{n+1}(x_i) = \prod_{j=0, j\neq i}^{n} (x_i - x_j) \tag{5.9}$$

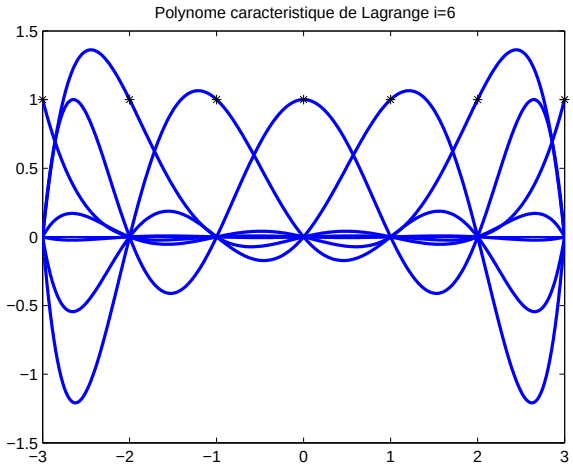


FIGURE 5.1 – Les sept polynômes caractéristiques de Lagrange pour les valeurs d'abscisse suivantes : -3,-2,-1,0,1,2,3. (Script MATLAB `s_pol_lag.m`)

.

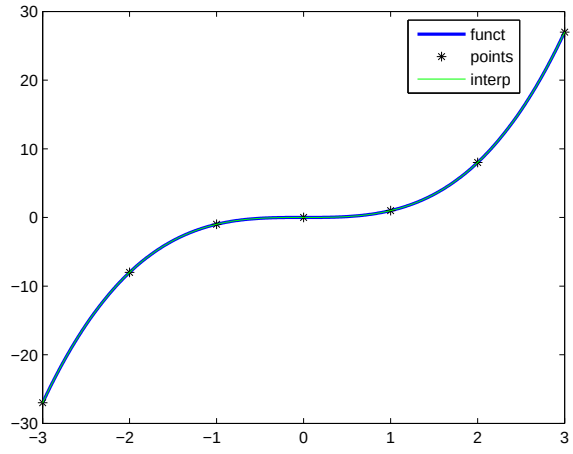


FIGURE 5.2 – Polynôme interpolant de Lagrange. (Script MATLAB `s_lag.m`)

5.3.2 La méthode de Newton

L'approche de Newton de l'interpolation polynomiale se base sur l'idée suivante. Étant données $n+1$ paires $(x_i, y_i), i = 0, \ldots, n$, nous pouvons représenter le polynôme $\Pi_n(\cdot)$, interpolant les points (x_i, y_i) pour $i = 0, \ldots, n$, comme la somme du polynôme $\Pi_{n-1}(\cdot)$, interpolant les points (x_i, y_i) pour $i = 0, \ldots, n-1$, et d'un polynôme $q_n(\cdot)$ de degré n qui dépend d'un seul coefficient inconnu. En forme analytique ceci signifie :

$$\Pi_n(x) = \Pi_{n-1}(x) + q_n(x) \tag{5.10}$$

Puisque le terme $q_n(\cdot)$ satisfait la relation suivante

$$q_n(x_i) = \Pi_n(x_i) - \Pi_{n-1}(x_i) = 0 \tag{5.11}$$

pour $i = 0, \ldots, n-1$, on obtient

$$q_n(x) = a_n(x - x_0)\ldots(x - x_{n-1}) = a_n\omega_n(x) \tag{5.12}$$

où $\omega_n(x)$ est un polynôme nodal (Équation 5.8). Puisque $f(x_n) = \Pi_n(x_n)$ (définition de polynôme interpolant), il découle des relations (5.10) et (5.12) que

$$a_n = \frac{f(x_n) - \Pi_{n-1}(x_n)}{\omega_n(x_n)} \tag{5.13}$$

Appelons le coefficient a_n la n-ème *différence divisée de Newton* et notons le de la manière suivante

$$a_n = \frac{f(x_n) - \Pi_{n-1}(x_n)}{\prod_{i=0}^{n-1}(x_n - x_i)} = f[x_0, x_1, \ldots, x_n] = D^n f(x_0) \tag{5.14}$$

En vertu des formules (5.10) et (5.14) on obtient

$$\Pi_n(x) = \Pi_{n-1}(x) + \omega_n(x) f[x_0, x_1, \ldots, x_n] \tag{5.15}$$

Puisque $\omega_{n+1}(x) = \prod_{i=0}^{n}(x - x_i)$, il s'ensuit que

$$\begin{aligned}
&\pi_0(x) = \omega_0(x) f[x_0] \\
&\pi_1(x) = \pi_0(x) + \omega_1(x) f[x_0, x_1] = \pi_0(x) + (x - x_0) f[x_0, x_1] \\
&\pi_2(x) = \pi_1(x) + \omega_2(x) f[x_0, x_1, x_2] = \pi_1(x) + (x - x_0)(x - x_1) f[x_0, x_1, x_2] \\
&\ldots
\end{aligned}$$

Puisque $\pi_0(x) = f(x_0)$, en posant $y_0 = f(x_0) = f[x_0]$ et $\omega_0 = 1$ on obtient par récurrence

$$\begin{aligned}
\Pi_n(x) = \sum_{k=0}^{n} \omega_k(x) f[x_0, \ldots, x_k] = \\
= f[x_0] + (x - x_0) f[x_0, x_1] + (x - x_0)(x - x_1) f[x_0, x_1, x_2] + \ldots \\
+ (x - x_0)(x - x_1)\ldots(x - x_{n-1}) f[x_0, \ldots, x_n]
\end{aligned} \tag{5.16}$$

où pour un x générique, la notation $f[\cdot, \ldots, \cdot]$ doit être interpretée de la manière suivante

$k = 0$

$$f[x] = f(x) \tag{5.17}$$

$k = 1$

$$f[x_0, x] = \frac{f[x] - f[x_0]}{x - x_0} = f[x, x_0] \tag{5.18}$$

$k \geq 2$

$$f[x_0, x_1, \ldots, x_{k-1}, x] = \frac{f[x_1, \ldots, x_{k-1}, x] - f[x_0, x_1, \ldots, x_{k-2}, x_{k-1}]}{x - x_0} \tag{5.19}$$

La forme (5.16) est communément appelée la *formule des différences divisées* de Newton du polynôme d'interpolation. Puisque le polynôme (5.14) est un polynôme interpolant, en vertu du théorème (5.1) il coïncidera avec le polynôme obtenu par la formule (5.3) de Lagrange.

Les differences divisées peuvent être stockées dans une matrice triangulaire inférieure de taille $n + 1$

x_i	$f(x_i)$	$Df(x_i)$	$D^2 f(x_i)$		$D^n f(x_i)$
x_0	$\mathbf{f}[x_0]$				
x_1	$f[x_1]$	$\mathbf{f}[x_0, x_1]$			
x_2	$f[x_2]$	$f[x_1, x_2]$	$\mathbf{f}[x_0, x_1 x_2]$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$		
x_n	$f[x_n]$	$f[x_{n-1}, x_n]$	$f[x_{n-2}, x_{n-1}, x_n]$	$\ldots$	$\mathbf{f}[x_0, \ldots, x_n]$

où la première colonne contient les valeurs $f(x_i)$ et les valeurs de la $j$ème colonne ($j > 1$) peuvent être calculées à partir de la $(j-1)$ème colonne grâce aux formules (5.18) et (5.19).

Notons, qu'une fois construite la matrice des différences divisées, le polynôme interpolant de Newton peut être aisément obtenu à partir des élements diagonaux (en gras)

$$\begin{aligned}\Pi_n(x) = \sum_{k=0}^{n} \omega_k(x) f[x_0, \ldots, x_k] = \\ = \mathbf{f}[x_0] + (x - x_0)\mathbf{f}[x_0, x_1] + (x - x_0)(x - x_1)\mathbf{f}[x_0, x_1, x_2] + \ldots \\ + (x - x_0)(x - x_1)\ldots(x - x_{n-1})\mathbf{f}[x_0, \ldots, x_n]\end{aligned}$$

Exemple Considérons la fonction $f(x) = \sin(x)$ et l'ensemble de couples (x_i, y_i), $i = 0, \ldots, 4$, où $x_i \in \{0, 0.5, 1, 1.5, 2\}$.

La forme tabulaire de la méthode de Newton pour les 5 couples considérés est la suivante :

0	**0**	0	0	0	0
0.5	0.4794	**0.9589**	0	0	0
1	0.8415	0.7241	**-0.2348**	0	0
1.5	0.9975	0.3120	-0.4120	**-0.1182**	0
2	0.9093	-0.1764	-0.4884	-0.0509	**0.0336**

Il s'ensuit que le polynôme interpolant obtenu par la méthode de Newton est

$$\begin{aligned}\Pi_4(x) = \mathbf{0} + \mathbf{0.9589}x - \mathbf{0.2348}x(x - 0.5) - \mathbf{0.1182}x(x - 0.5)(x - 1) + \\ + \mathbf{0.0336}x(x - 0.5)(x - 1)(x - 1.5)\end{aligned}$$

Le graphique du polynôme est tracé en Figure 5.3.

•

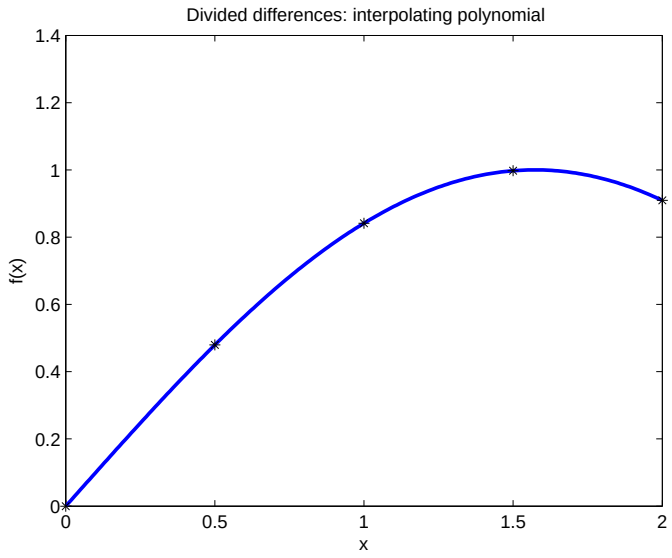


FIGURE 5.3 – Polynôme interpolant obtenu par méthode de Newton. (Script MATLAB `s_pol_new2.m`)

Les propriétés les plus importantes de la méthode des différences divisées sont les suivantes :

- $n(n+1)$ additions et $n(n+1)/2$ divisions sont nécessaires pour construire la matrice triangulaire inférieure des différences divisées.
- Si on disposait de la valeur prise par f dans un nouveau noeud x_{n+1}, on aurait simplement à calculer une ligne supplémentaire.
- Pour construire Π_{n+1} à partir de Π_n, $(n+1)$ divisions et $2(n+1)$ additions sont nécessaires. Ceci n'est pas le cas pour la méthode de Lagrange où il est nécessaire de répéter toute la procédure.

Pour conclure, nous rappelons que la méthode de Lagrange et de Newton nous amènent, pour les même données, toujours au même polynôme interpolant, en utilisant deux algorithmes numériques différents.

5.3.3 L'erreur d'interpolation

Un polynôme interpolant passe exactement à travers les $n+1$ points (x_i, y_i) qui définissent le problème d'interpolation. Toutefois, si les ordonnées y_i représentent les valeurs d'une fonction $f(\cdot)$ (Équation (5.1)) il est intéressant d'étudier le comportement du polynôme interpolant $\Pi_n(\cdot)$ en dehors des points d'interpolation.

A cette fin, nous définissons la quantité

$$E_n(x) = f(x) - \Pi_n(x) \tag{5.20}$$

l'*erreur d'interpolation* du polynôme $\Pi_n(\cdot)$. Pour que l'interpolation soit efficace, il est important que l'erreur d'interpolation soit petite partout dans le domaine de la fonction $f(\cdot)$.

5.3.3.1 L'analyse de l'erreur d'interpolation

Le théorème suivant a été démontré, à propos de la quantité $E_n(x)$.

Théorème 5.2. *Soient $x_0, x_1, \ldots, x_n$, $n+1$ noeuds distincts et soit x un point appartenant au domaine de définition de f. On suppose que $f \in C^{n+1}(I_x)$ où C^{n+1} est l'ensemble des fonctions continues d'ordre $n+1$ I_x est le plus petit intervalle contenant les noeuds $x_0, x_1, \ldots, x_n$ et x. L'erreur d'interpolation au point x est donnée par*

$$E_n(x) = f(x) - \Pi_n(x) = \frac{f^{(n+1)}(\psi)}{(n+1)!}\omega_{n+1}(x) = \\ = \frac{f^{(n+1)}(\psi)}{(n+1)!}(x-x_0)(x-x_1)\ldots(x-x_n) \quad (5.21)$$

où $\psi \in I_x$ et ω_{n+1} est le polynôme nodal de degré $n+1$.

Le théorème peut être utilisé afin de définir une borne supérieure de l'erreur d'approximation.

Soit $\|f^{(n+1)}\|_\infty = \max_{x\in[a,b]} |f^{(n+1)}(x)|$, alors on peut montrer que

$$|f(x) - \Pi_n(x)| \leq \frac{\|f^{(n+1)}\|_\infty}{(n+1)!}|\omega_{n+1}(x)| \leq \frac{\|f^{(n+1)}\|_\infty (b-a)^{n+1}}{(n+1)!} \quad (5.22)$$

ou

$$\|f - \Pi_n\|_\infty \leq \frac{\|f^{(n+1)}\|_\infty}{(n+1)!}\|\omega_{n+1}\|_\infty \quad (5.23)$$

En vertu des formules ci-dessus, nous pouvons dériver les conclusions suivantes :

- le calcul de l'erreur d'interpolation demande la connaissance de la fonction $f(\cdot)$ (parfois inconnue).
- l'erreur d'interpolation est petite pour un x près d'un noeud, puisque au moins un des termes $(x - x_i)$ sera petit.
- l'erreur d'interpolation est autant plus petite que la fonction est lisse (*smooth*), puisque dans ces cas le maximum de la dérivée est réduit.
- on peut montrer que le maximum de $|\omega_{n+1}(x)|$ est atteint toujours dans un des deux intervalles extrêmes $[x_0, x_1]$ ou $[x_{n-1}, x_n]$.
- l'erreur d'extrapolation (c-à-d. en un x en dehors de l'intervalle I_x et donc loin des valeurs x_i) est typiquement supérieure à celle d'interpolation, à cause à la taille des quantités $(x - x_i)$.
- l'erreur d'extrapolation est autant plus importante que le degré n du polynôme est grand.
- si f est un polynôme de degré n, sa dérivée d'ordre $n+1$ est nulle et l'erreur d'interpolation est nulle pour tout x. Ce résultat confirme le théorème d'unicité.

Exemple La figure 5.4 montre l'allure du polynôme nodal $\omega_7(x)$ pour un problème d'interpolation où $n = 6$ et les abscisses x_i prennent leurs valeurs dans l'ensemble $\{-3, -2, -1, 0, 1, 2, 3\}$. Notons que la valeur absolue du polynôme en dehors de l'intervalle d'interpolation est beaucoup plus grande que la valeur à l'intérieur de l'intervalle. Il s'ensuit que l'erreur d'extrapolation sera beaucoup plus importante que l'erreur d'interpolation.

•

5.3.3.2 Analyse d'erreur et méthode de Newton

En combinant la formule (5.21) et la formule des differences divisées (5.16) on obtient une relation entre la formule de Newton et l'erreur d'interpolation.

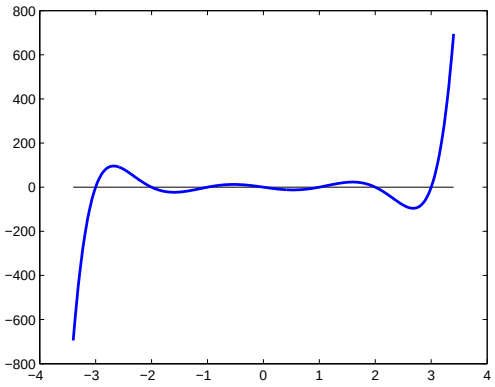


FIGURE 5.4 – Polynôme nodal d'ordre 7 pour $x_i \in \{-3, -2, -1, 0, 1, 2, 3\}$. (Script MATLAB `s_nodal`).

Considérons $\Pi_n(\cdot)$ et un noeud $x = x_{n+1}$ distinct des précédents. Soit $\Pi_{n+1}(\cdot)$ le polynôme interpolant $f(\cdot)$ aux noeuds x_k, $k = 0, \ldots, n+1$.

Puisque $\Pi_{n+1}(x) = f(x)$, en utilisant la formule des différences divisées (5.16) on obtient

$$E_n(x) = f(x) - \Pi_n(x) = \Pi_{n+1}(x) - \Pi_n(x) = \\ = (x - x_0) \ldots (x - x_n) f[x_0, \ldots, x_n, x]$$

En utilisant la formule

$$E_n(x) = f(x) - \Pi_n(x) = \frac{f^{(n+1)}(\psi)}{(n+1)!}(x - x_0)(x - x_1) \ldots (x - x_n) \tag{5.24}$$

on a

$$f[x_0, \ldots, x_n, x] = \frac{f^{(n+1)}(\psi)}{(n+1)!} \tag{5.25}$$

pour un certain $\psi \in I_x$ où I_x est le plus petit intervalle contenant les noeuds $x_0, x_1, \ldots, x_n$ et x.

5.3.3.3 Convergence uniforme

Nous analysons ici le comportement de l'erreur d'interpolation (5.20) quand le nombre n tend vers l'infini. Malgré l'intuition que l'erreur d'interpolation devrait diminuer pour un nombre croissant de couples (x_i, y_i) la propriété suivante peut être montrée :

Propriété 5.1. *Soient les noeuds d'interpolation équirépartis. Il est toujours possible de trouver une fonction f pour laquelle*

$$\lim_{n \to \infty} \|f - \Pi_n\|_\infty \neq 0 \tag{5.26}$$

où $\|f\|_\infty = \max_{x \in [a,b]} |f(x)|$.

Donc, la convergence uniforme n'est pas garantie quand les noeuds d'interpolation sont équirépartis. Ceci peut être visualisé par le fameux *contre-exemple de Runge*. Supposons qu'on approche la fonction

$$f(x) = \frac{1}{1 + x^2} \tag{5.27}$$

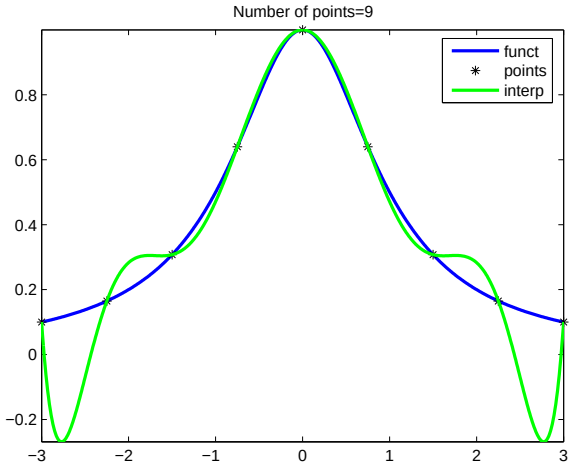


FIGURE 5.5 – Interpolation de la fonction de Runge par un polynôme d'ordre 12.

en utilisant l'interpolation de Lagrange avec noeuds équirépartis. Puisque les dérivées de la fonction ne sont pas bornées, on peut vérifier qu'il existe des points x à l'intérieur de l'intervalle d'interpolation tels que

$$\lim_{n\to\infty} |f(x) - \Pi_n(x)| \neq 0 \tag{5.28}$$

c.-à-d. l'approximation se dégrade en ajoutant des données.

Le script MATLAB `s_runge.m` (Figure 5.5) montre les différents polynômes interpolants pour un n croissant. On peut remarquer que les polynômes d'interpolation présentent des oscillations qui augmentent avec le degré du polynôme.

5.3.3.4 Instabilité de l'interpolation

Un problème connu des méthodes d'interpolation est leur instabilité dans le cas de noeuds x_i équirépartis. Ceci implique qu'une petite variation des données (x_i, y_i) peut entraîner une grande variation de la solution $\Pi_n(\cdot)$.

Le script MATLAB `s_unstable.m` représente graphiquement le problème (Figure 5.6). Plusieurs méthodes ont été proposées pour résoudre le problème de l'instabilité. Parmi eux nous mentionnons :

Méthode de Chebyshev : dans cette méthode les noeuds sont non équirépartis mais localisés en fonction de n et condensés aux extrêmes de l'intervalle. La convergence uniforme pour $n \to \infty$ est garantie.

Méthode d'interpolation de Lagrange par morceaux I_j : dans cette méthode l'interpolation est effectuée sur chaque morceau I_j de longueur h en utilisant un nombre fixe et petit de noeuds équirépartis. Ceci permet de réduire la contribution de ω_{n+1} à l'erreur sans augmenter n. De plus la convergence est uniforme pour $h \to 0$.

Méthode d'interpolation de Hermite : celle-ci généralise la méthode de Lagrange en prenant en compte aussi la valeur de la dérivée dans les noeuds.

Méthode de splines : elle sera le sujet de la prochaine section.

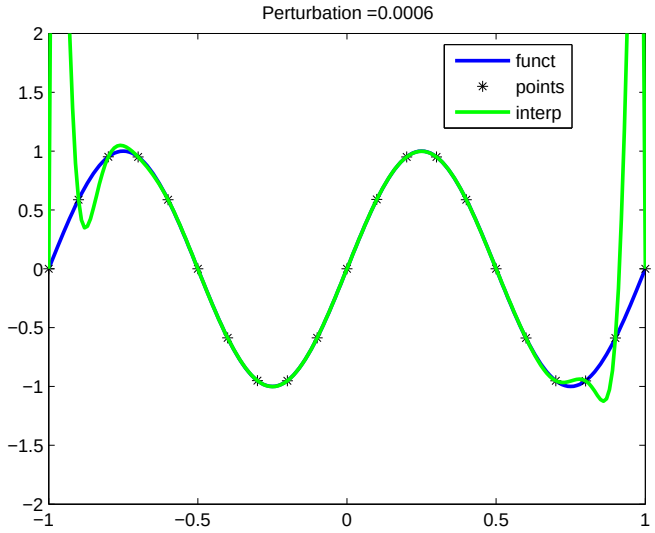


FIGURE 5.6 – Instabilité du polynôme interpolant dans le cas de noeuds équirépartis. Une petite variation des valeurs (astérisques) entraîne une large variation de la solution (en pointillé).

5.4 La méthode de splines

Dans cette section nous allons présenter la méthode d'interpolation par morceaux, dite de *splines*. Cette méthode a été proposée afin d'apporter une solution aux problèmes de convergence et d'instabilité typiques des méthodes polynomiales.

Nous commençons notre présentation par la définition de spline.

Définition 33. *Soient $x_0, \ldots, x_n$, $n+1$ noeuds distincts de $[a,b]$ avec*

$$a = x_0 < x_1 < \cdots < x_n = b.$$

La fonction $s_k(x)$ sur l'intervalle $[a,b]$ est une spline *de degré k relative aux noeuds x_i si*

$$s_{k|[x_i,x_{i+1}]} \in P^k, \ \textit{pour } i = 0, 1, \ldots, n-1 \tag{5.29}$$

$$s_k \in C^{k-1}[a,b] \tag{5.30}$$

où P_k est l'ensemble des polynômes de degré k et $C^{k-1}[a,b]$ est l'ensemble des fonctions continues d'ordre $k-1$ définies sur l'intervalle $[a,b]$.

Suite à cette définition tout polynôme de degré k sur $[a,b]$ peut être considéré comme une spline, mais en pratique, nous considérerons seulement des splines

1. constituées de n polynômes différents $s_{k,i}$ sur chaque sous-intervalle $[x_i, x_{i+1}]$,
2. présentant des discontinuités de la dérivée k-ème aux noeuds internes (dits *actifs*).

Le i-ème polynôme $s_{k,i} = s_{k|[x_i,x_{i+1}]}$, $i = 0, \ldots, n-1$, composant la spline peut être écrit sous la forme

$$s_{k,i} = \sum_{h=0}^{k} s_{hi}(x - x_i)^h \tag{5.31}$$

Il s'ensuit que calculer une spline équivaut à déterminer les $(k+1)n$ coefficients s_{hi}. Pour ce faire, nous pouvons imposer les contraintes suivantes de regularité et d'interpolation :

- $k(n-1)$ **contraintes sur les dérivées** : $s^{(m)}_{k,i-1}(x_i) = s^{(m)}_{k,i}(x_i)$, pour $i = 1,\ldots,n-1$ et $m = 0,\ldots,k-1$
- $n+1$ **contraintes d'interpolations** : $s_k(x_i) = f(x_i)$ pour $i = 0,\ldots,n$.

Il s'ensuit qu'il reste encore

$$(k+1)n - k(n-1) - (n+1) = kn + n - kn + k - n - 1 = k - 1$$

degrés de liberté à fixer. Deux options sont envisageables afin de fixer les $k-1$ contraintes additionnelles :

1. imposer une contrainte de périodicité

$$s^{(m)}_k(a) = s^{(m)}_k(b) \qquad m = 1,\ldots,k-1. \tag{5.32}$$

   Dans ce cas les splines sont dites *périodiques*

2. imposer pour $k = 2l-1$, avec $l \geq 2$

$$s^{(l+j)}_k(a) = s^{(l+j)}_k(b) = 0 \qquad j = 0,1,\ldots,l-2. \tag{5.33}$$

   Dans ce cas les splines sont dites *naturelles*

Notons que si $k = 3$, alors $l = 2$ et la contrainte naturelle équivaut à imposer

$$s^{(2)}_k(a) = s^{(2)}_k(b) = 0. \tag{5.34}$$

5.4.1 Les splines cubiques

Pour l'interpolation, les splines cubiques (degré $k = 3$) sont les plus utilisées. Elles sont importantes car

1. ce sont les splines de plus petit degré qui permettent une approximation C^2,
2. elles ont de bonnes propriétés de régularité.

Dans ce cas il faut calculer $(k+1)n = 4n$ coefficients pour déterminer de manière unique la spline $s_3(\cdot)$.

La définition des splines nous donne les contraintes suivantes :

- $s_3(x_i^-) = f(x_i) = s_3(x_i^+)$ pour $i = 1,\ldots,n-1$ [$2(n-1)$ conditions],
- $s_3(x_0) = f(a)$ [1 condition],
- $s_3(x_n) = f(b)$ [1 condition],
- $s'_3(x_i^-) = s'_3(x_i^+)$ pour $i = 1,\ldots,n-1$ [$(n-1)$ conditions],
- $s''_3(x_i^-) = s''_3(x_i^+)$ pour $i = 1,\ldots,n-1$ [$(n-1)$ conditions].

Au total, ceci correspond à $4n-2$ conditions. Les $k-1 = 2$ restantes sont imposées via les additionnelles.

La méthode la plus intuitive, mais aussi la lus coûteuse, pour calculer une spline demande donc la résolution d'un système linéaire de taille $4n$. Il est possible toutefois adopter une technique de calcul qui réduit la taille du système linéaire à résoudre. Cette technique, présentée dans la section qui suit, demande la résolution d'un système de taille $n+1$.

5.4.1.1 Le système de M-continuité

Introduisons les notations suivantes :

$$f_i = s_3(x_i),\ m_i = s'_3(x_i),\ M_i = s''_3(x_i) \tag{5.35}$$

Puisque $s_{3,i-1}(\cdot) \in P^3$, $s''_{3,i-1}(\cdot)$ est une fonction linéaire en x qui peut être écrite sous la forme

$$s''_{3,i-1}(x) = M_{i-1}\frac{x_i - x}{h_i} + M_i\frac{x - x_{i-1}}{h_i} \tag{5.36}$$

où $x \in [x_{i-1}, x_i]$, $i = 1, \ldots, n$, et $h_i = x_i - x_{i-1}$. Notons que ceci est l'équation d'une droite passant par les deux points (x_i, M_i) et (x_{i-1}, M_{i-1}).

En intégrant la relation (5.36), on obtient

$$s'_{3,i-1}(x) = -M_{i-1}\frac{(x_i - x)^2}{2h_i} + M_i\frac{(x - x_{i-1})^2}{2h_i} + C_{i-1} \tag{5.37}$$

$$s_{3,i-1}(x) = M_{i-1}\frac{(x_i - x)^3}{6h_i} + M_i\frac{(x - x_{i-1})^3}{6h_i} + C_{i-1}(x - x_{i-1}) + \tilde{C}_{i-1} \tag{5.38}$$

En imposant les valeurs aux extrémités $s_3(x_{i-1}) = f_{i-1}$ et $s_3(x_i) = f_i$, on a

$$\tilde{C}_{i-1} = f_{i-1} - M_{i-1}\frac{h_i^2}{6}$$
$$C_{i-1} = \frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(M_i - M_{i-1})$$

Puisque

$$s'_{3,i-1}(x) = -M_{i-1}\frac{(x_i - x)^2}{2h_i} + M_i\frac{(x - x_{i-1})^2}{2h_i} + \frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(M_i - M_{i-1})$$
$$s'_{3,i}(x) = -M_i\frac{(x_{i+1} - x)^2}{2h_{i+1}} + M_{i+1}\frac{(x - x_i)^2}{2h_{i+1}} + \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{h_{i+1}}{6}(M_{i+1} - M_i)$$

en imposant la continuité de la dérivée première en x_i, nous obtenons

$$s'_{3,i-1}(x_i) = s'_3(x_i^-) = \frac{h_i}{6}M_{i-1} + \frac{h_i}{3}M_i + \frac{f_i - f_{i-1}}{h_i} =$$
$$= -\frac{h_{i+1}}{3}M_i - \frac{h_{i+1}}{6}M_{i+1} + \frac{f_{i+1} - f_i}{h_{i+1}} = s'_3(x_i^+) = s'_{3,i}(x_i)$$

Ceci conduit au système linéaire (dit de *M-continuité*) avec $n+1$ inconnues et $n-1$ équations

$$\frac{h_i}{6}M_{i-1} + \frac{h_i + h_{i+1}}{3}M_i + \frac{h_{i+1}}{6}M_{i+1} = \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i} \tag{5.39}$$

pour $i = 1, \ldots, n-1$. Pour obtenir une spline naturelle on rajoute les 2 conditions supplémentaires $M_0 = 0$ et $M_n = 0$.

Exemple Calculons une spline cubique ($k = 3$) naturelle qui interpole les 5 couples ($n = 4$)

0	0.5	1	1.5	2
0	0.4794	0.8415	0.9975	0.9093

obtenus par la fonction $f(x) = \sin(x)$. En vertu de l'équation (5.39) et des conditions additionnelles (5.34) le système de M-continuité est

$$\begin{bmatrix} 1.0000 & 0 & 0 & 0 & 0 \\ 0.0833 & 0.3333 & 0.0833 & 0 & 0 \\ 0 & 0.0833 & 0.3333 & 0.0833 & 0 \\ 0 & 0 & 0.0833 & 0.3333 & 0.0833 \\ 0 & 0 & 0 & 0 & 1.0000 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} = \begin{bmatrix} 0 \\ -0.2348 \\ -0.4120 \\ -0.4884 \\ 0 \end{bmatrix} \tag{5.40}$$

La courbe correspondante est tracée en Figure 5.7

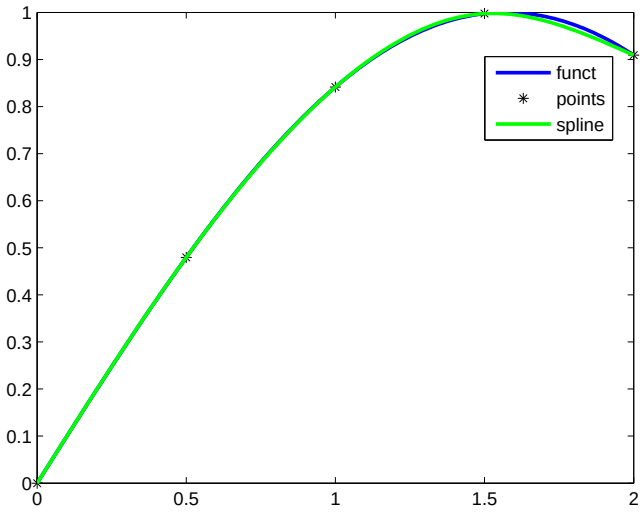


FIGURE 5.7 – Spline cubique interpolant. (Script MATLAB `s_spline2.m`)

•

Exemple Le script MATLAB `s_ru_spl.m` montre le comportament de l'interpolation de la fonction (5.27) de Runge par spline cubique. Nous pouvons observer la regularité de l'approximation pour un nombre de points n croissant (Figure 5.8).

•

5.5 Exercices

5.5.1 Interpolation de Lagrange

1. Déterminez le polynôme de Lagrange passant par les points $(0, -2)$, $(-1, -2)$ et $(1/2, -5/4)$.
2. Donnez une approximation de $\sqrt[3]{20}$ en utilisant l'interpolation de Lagrange.
3. Considérons la fonction $f(x) = \sin(x)$ et supposons que la valeur de la fonction est connue pour les 5 noeuds [0.0,0.2,0.4,0.6,0.8].
   (a) Calculez la borne supérieure de la valeur absolue de l'erreur d'approximation du polynôme interpolant Π_4 en $x = 0.28$.
   (b) Comparez la borne avec l'erreur réelle en sachant que $\sin(0.28) = 0.2763556$.

5.5.2 Interpolation de Newton

1. Déterminez le polynôme de Newton passant par les points $(6, 6)$, $(8, 24)$, $(10, 60)$ et $(12, 120)$.
2. Calculez $\varphi(n) = \sum_{k=0}^{n} k^2$ en utilisant l'interpolation de Newton avec 5 points.
3. Considérons les données fournies par le tableau suivant :

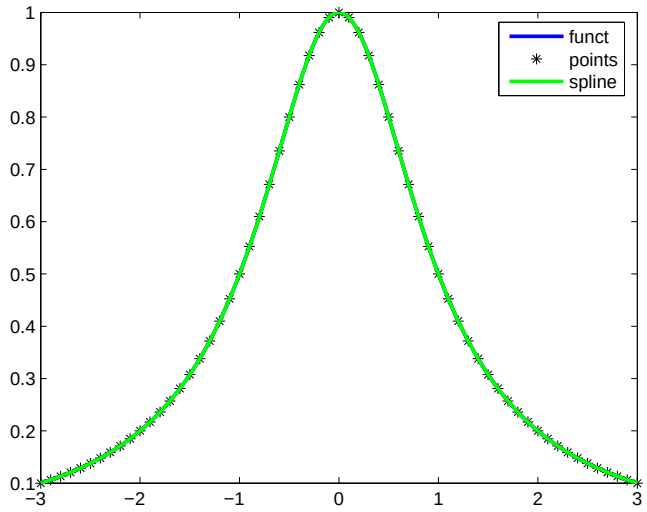


FIGURE 5.8 – Interpolation de la fonction de Runge par spline cubique. (Script MATLAB `s_ru_spl.m`)

x	$f(x)$
-0.2	1.3940
0.0	1.1884
0.1	1.1221
0.5	1.0025
0.7	1.0084

(a) Construisez le tableau des différences divisées.

(b) Estimez $f(0.15)$ en utilisant

i. un polynôme de degré 2 passant par les 3 premiers points ;

ii. un polynôme de degré 2 passant par les 3 derniers points ;

iii. un polynôme de degré 3 passant par les 4 premiers points ;

iv. un polynôme de degré 3 passant par les 4 derniers points ;

(c) Comparez les résultats.

5.5.3 Splines

1. Déterminez les splines naturelle et périodique cubiques passant par les points $(-1, 0)$, $(0, 1)$ et $(1, -1)$.
2. Déterminez la spline naturelle cubique passant par les points donnés dans le tableau suivant :

x	$f(x)$
0.0	2.0000
1.0	4.4366
1.5	6.7134
2.25	13.9130

Evaluez la spline en 0.66 et 1.75. Tirez des conclusions. (La vraie fonction est $f(x) = 2e^x - x^2$).

Chapitre 6

Curve fitting

6.1 Exemples et motivations

De nombreuses applications nécessitent de représenter de manière synthétique un grand ensemble de données pouvant résulter de mesures expérimentales. Dans ce cas, l'approche par interpolation peut être inefficace pour deux raisons :

1. si le nombre de données est grand, le polynôme interpolant peut présenter des oscillations importantes.
2. les données sont entachées de bruit.

Au lieu de l'interpolation, une approximation des données, appelée *lissage* ou *fitting* des données, peut être effectuée en utilisant la méthode discrète des moindres carrés.

Voici deux exemples qui montrent l'utilité des méthodes de curve-fitting dans les sciences expérimentales.

- En 1601, l'astronome allemand Johannes Kepler formula la troisième loi du mouvement planétaire
$$T = Cx^{3/2} \tag{6.1}$$
qui lie la distance x de la planète au soleil (en million de kilomètres) et la période orbitale T (en jours), sur base de données expérimentales. Les couples de valeurs (x_i, T_i) observés pour les planètes Mercure, Vénus, Terre et Mars sont $(58, 88)$, $(108, 225)$, $(150, 365)$ et $(228, 687)$. La valeur du coefficient $C = 0.199769$ fut trouvée grâce à la méthode de moindres carrés proposée par K. F. Gauss (*Theoria Motus Corporum Caelestium, 1809*).
- Supposons qu'on mesure 2 variables corrélées Q et I, où Q est la chaleur dissipée par une résistance $R = 2\Omega$ et I est le courant passant à travers R. L'approximation par polynôme interpolant (Figure 6.1) ne révèle pas la relation quadratique existante entre I et Q.

6.2 Précision d'une approximation

Considérons

1. $n + 1$ couples de valeurs (x_i, y_i), $i = 0, \ldots, n$ où y_i représente, par exemple, une quantité physique mesurée à la position x_i.
2. une fonction d'approximation $h(\cdot)$.

Nous définissons *l'erreur d'approximation* en x_i, $i = 0, \ldots, n$ par
$$e_i = h(x_i) - y_i \tag{6.2}$$
Plusieurs normes peuvent être considérées afin de mesurer l'éloignement de la fonction $h(\cdot)$ des données.

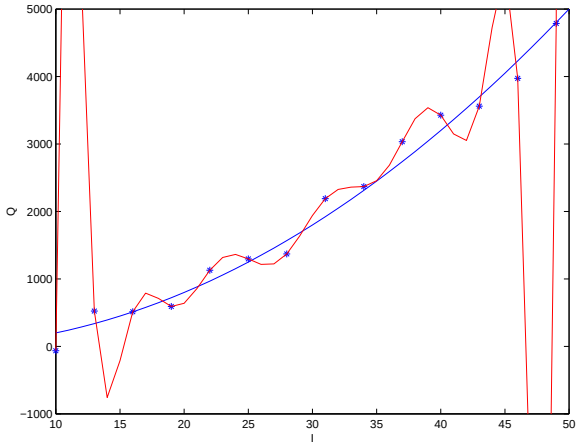


FIGURE 6.1 – Interpolation des mesures de deux variable physiques corrélées mais bruitées.

Erreur absolue maximale :

$$E_\infty(h) = \max_{0 \le i \le n} \{|h(x_i) - y_i|\} \tag{6.3}$$

Erreur absolue moyenne :

$$E_1(h) = \frac{1}{N+1} \sum_{i=0}^{n} |h(x_i) - y_i| \tag{6.4}$$

***Root-mean-square error (RMSE)* :**

$$E_2(h) = \left(\frac{1}{N+1} \sum_{i=0}^{n} |h(x_i) - y_i|^2 \right)^{1/2} \tag{6.5}$$

Dans ce qui suit, nous considérerons uniquement la norme $E_2(\cdot)$ pour les raisons suivantes :

- comme pour E_∞ et E_1, les écarts négatifs $e_i < 0$ n'effacent pas les écarts positifs $e_j > 0$,
- l'optimisation de $E_2(\cdot)$ est plus facile, grâce à sa forme quadratique
- les petits écarts sont réduits et les grands écarts sont amplifiés.

6.3 Systèmes sur-déterminés et pseudo-inverse

Afin d'introduire la méthode des moindre carrés, nous présentons la notion de système linéaire sur-déterminé. Soit donné le système linéaire

$$Ax = b \tag{6.6}$$

où A est une matrice $(n \times m)$ et b et un vecteur $(n \times 1)$. Si $n = m$ et si la matrice A est inversible alors la solution du système linéaire existe et est unique (Théorème 3.1). Si $n > m$ le système est dit *sur-déterminé*. Un système sur-déterminé n'admet pas une solution au sens classique mais il admet une solution au sens des moindres carrés.

Définition 34 (Solution au sens des moindres carrés)**.** *Étant donné une matrice $A \in \mathbb{R}^{n\times m}$ telle que $n > m$ et $b \in \mathbb{R}^n$ on dit que $x^* \in \mathbb{R}^m$ est une solution du système linéaire $Ax = b$* au sens des moindres carrés *si*

$$\Phi(x^*) = \min_{x\in\mathbb{R}^m} \Phi(x), \text{ où } \Phi(x) = \|Ax - b\|_2^2 \tag{6.7}$$

Le problème aux moindres carrés peut être formulé comme un problème d'optimisation qui consiste à minimiser la norme euclidienne du résidu. La solution peut être déterminée en imposant au gradient de la fonction $\Phi(\cdot)$ de s'annuler en x^*. Puisque

$$\Phi(x) = (Ax - b)^T(Ax - b) = x^T A^T Ax - 2x^T A^T b + b^T b \tag{6.8}$$

on a

$$\nabla\Phi(x^*) = 2A^T Ax^* - 2A^T b = 0 \tag{6.9}$$

Il en découle que x^* doit être solution du système carré $(m \times m)$

$$A^T Ax^* = A^T b \tag{6.10}$$

appelé *système des équations normales.*

Si A est de rang maximum, le système des équations normales est non singulier et la solution x^* existe et est unique. Dans ce cas, étant donné que la matrice des coefficients du système (6.10) est symétrique et définie positive, on pourrait envisager la factorisation de Cholesky (Section 3.7) comme méthode de résolution.

Malheureusement cette méthode a deux inconvénients majeurs

1. le système (6.10) est mal conditionné
2. les erreurs d'arrondi dans le calcul $A^T A$ peuvent entraîner une perte du nombre de chiffres significatifs

Il est donc en général recommandé d'utiliser une méthode alternative : la factorisation QR pour matrices rectangulaires [6].

Par contre, si A n'est pas de rang maximal, le système (6.10) a un nombre infini de solutions. Par conséquent, on doit imposer une contrainte supplémentaire pour forcer l'unicité de la solution. Un exemple de contrainte consiste à chercher à minimiser la norme euclidienne de x^*.

Le problème de moindres carrés peut alors être formulé de la manière suivante : trouver $x^* \in \mathbb{R}^m$ de norme euclidienne minimale tel que

$$\|Ax^* - b\|_2^2 \leq \min_{x\in\mathbb{R}^m} \|Ax - b\|_2^2 \tag{6.11}$$

On peut montrer que l'unique solution de ce problème est

$$x^* = A^\dagger b$$

où $A^\dagger$ est la *pseudo-inverse* de A.

La définition de matrice pseudo-inverse dérive du fait que toute matrice peut être réduite sous forme diagonale en la multipliant à droite et à gauche par des matrices orthogonales. Soit $A_{n\times m}$ une matrice réelle de rang $r \leq m$. Il existe une décomposition en valeurs singulières (Section A.9)

$$U^T AV = \Sigma = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \dots & \sigma_m \\ 0 & 0 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

où $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_m = 0$ sont appelés valeurs singulières de A, et $U_{n\times n}$ et $V_{m\times m}$ sont matrices orthogonales. Notons aussi que

$$\sigma_i = \sqrt{\lambda_i(A^T A)}, i = 1, \ldots, m$$

où λ_i sont les valeurs propres de la matrice carrée et symétrique $A^T A$.

Définition 35 (Pseudo-inverse)**.** *La matrice*

$$A^\dagger = V\Sigma^\dagger U^T \tag{6.12}$$

est appelée matrice pseudo-inverse de Moore-Penrose *ou* inverse généralisée*, où*

$$\Sigma^\dagger_{m\times n} = diag\left(\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_r}, 0, \ldots, 0\right) = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ 0 & \ddots & \ldots & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & \frac{1}{\sigma_r} & 0 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 \end{bmatrix}$$

et $\sigma_1, \ldots, \sigma_r$ sont les valeurs singulières non nulles de la matrice A.

Notons donc que
- si $r = m < n$ (rang maximal) alors $A^\dagger = (A^T A)^{-1} A^T$,
- si $r = m = n$ alors la matrice pseudo-inverse coïncide avec la matrice inverse, c.-à-d. $A^\dagger = A^{-1}$.

6.4 Approximation aux moindres carrés

Soient $n+1$ couples de valeurs (x_i, y_i), $i = 0, \ldots, n$ où y_i représente, par exemple, une quantité physique (sortie) mesurée à la position x_i (entrée).

Définition 36 (Polynôme aux moindres carrés)**.** *On appelle polynôme aux moindres carrés $\Pi_m(x)$ le polynôme de degré $m \leq n$ tel que*

$$\sum_{i=0}^{n} |y_i - \Pi_m(x_i)|^2 \leq \sum_{i=0}^{n} |y_i - q_m(x_i)|^2 \qquad \forall q_m(x) \in P^m \tag{6.13}$$

où P^m est l'ensemble des polynômes de degré m.
En notant

$$\Pi_m(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m = \sum_{j=0}^{m} a_j \varphi_j(x) \tag{6.14}$$

où $\varphi_j(x) = x^j$, $j = 0, \ldots, m$, le problème de la recherche du polynôme aux moindre carrés peut être formulé comme la résolution du problème suivant :

Trouver les coefficients $\{a_j \in \mathbb{R}, j = 0, \ldots, m\}$ tels que

$$\sum_{j=0}^{m} a_j \varphi_j(x_i) = y_i \qquad i = 0, \ldots, n \tag{6.15}$$

Si $m < n$ ce problème équivaut à résoudre le système sur-déterminé

$$Xa = y \tag{6.16}$$

où

– $X_{(n+1)\times(m+1)}$ est une matrice rectangulaire telle que $X(i+1, j+1) = \varphi_j(x_i)$
– a est le vecteur des inconnues
– $y_{(n+1)\times 1}$ est le vecteur (connu) du second membre.

La solution $a^*_{(n+1)\times 1}$ est la solution du système aux équations normales

$$X^T X a = X^T y \tag{6.17}$$

et le polynôme

$$\Pi^*_m(x) = \sum_{j=0}^{m} a^*_j \varphi_j(x) \tag{6.18}$$

est l'approximation au sens des moindres carrés des données (x_i, y_i), $i = 0, \ldots, n$.

Exemple
Considérons les $n+1$ données ($n = 3$) et le polynôme d'ordre $m = 2 < n$

x	x_0	x_1	x_2	x_3
y	y_0	y_1	y_2	y_3

$$\Pi_m(x) = a_0\varphi_0(x) + a_1\varphi_1(x) + a_2\varphi_2(x) = a_0 + a_1 x + a_2 x^2 \tag{6.19}$$

Le système correspondante $Xa = y$ est

$$\begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{6.20}$$

•

Exemple Le script MATLAB `s_least.m` calcule les coefficients du polynôme aux moindre carrés d'ordre $m = 2$ pour les $n+1$ données suivantes

x	0	0.5	1	1.5
y	0	0.4794	0.8415	0.9975

Notons que le script utilise la fonction MATLAB `pinv` pour le calcul de la matrice pseudo-inverse.

Le polynôme résultant et les données sont visualisés en Figure 6.2.

•

6.5 Régression linéaire

Si le polynôme aux moindres carrés est d'ordre $m = 1$, la solution du problème (6.14) est une fonction linéaire

$$\Pi^*_1(x) = a^*_0\varphi_0(x) + a^*_1\varphi_1(x) \tag{6.21}$$

appelée *régression linéaire* associée aux données.

Dans ce cas, le système d'équations normales correspondantes à $X^T X a = X^T y$ est

$$\sum_{j=0}^{1}\sum_{i=0}^{n} \varphi_k(x_i)\varphi_j(x_i)a_j = \sum_{i=0}^{n} \varphi_k(x_i)y_i \tag{6.22}$$

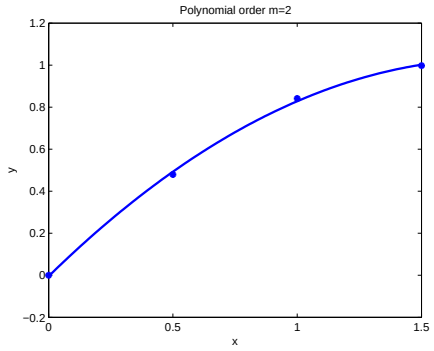


FIGURE 6.2 – 4 couples entrée-sortie et le polynôme aux moindres carrés d'ordre $m = 2$ (Script MATLAB `s_least.m`)

pour $k = 0, 1$.

En posant $(f, g)_n = \sum_{i=0}^{n} f(x_i)g(x_i)$ on peut montrer que le système (6.22) peut être mis sous la forme

$$\begin{cases} (\varphi_0, \varphi_0)_n a_0 + (\varphi_1, \varphi_0)_n a_1 &= (y, \varphi_0)_n \\ (\varphi_0, \varphi_1)_n a_0 + (\varphi_1, \varphi_1)_n a_1 &= (y, \varphi_1)_n \end{cases} \tag{6.23}$$

Si $\varphi_0(x) = 1$ et $\varphi_1(x) = x$, la solution est une droite de coefficients a_0 et a_1 qui satisfont le système à 2 équations et 2 inconnues

$$\begin{cases} (n+1)a_0 + a_1 \sum_{i=0}^{n} x_i &= \sum_{i=0}^{n} y_i \\ a_0 \sum_{i=0}^{n} x_i + a_1 \sum_{i=0}^{n} x_i^2 &= \sum_{i=0}^{n} x_i y_i \end{cases} \tag{6.24}$$

Exemple de régression linéaire Soit $n = 4$, $m = 1$, $\varphi_0(x) = 1$ et $\varphi_1(x) = x$. Étant donné

x_i	1	3	4	6	7
y_i	-2.1	-0.9	-0.6	0.6	0.9

le système aux équations normales est

$$\begin{cases} 5a_0 + 21a_1 &= -2.1 \\ 21a_0 + 111a_1 &= 2.7 \end{cases} \tag{6.25}$$

et la solution est

$$\begin{cases} a_0^* = -2.542 \\ a_1^* = 0.505 \end{cases} \tag{6.26}$$

Les données et la droite de régression sont illustrés en Figure 6.3.

•

Trade-off overfitting/underfitting Jusqu'ici, nous avons considéré le degré m du polynôme aux moindre carrés comme étant connu à l'avance. En réalité, dans les problèmes pratiques d'analyse des données, le choix de l'ordre m est considéré comme un problème serieux.

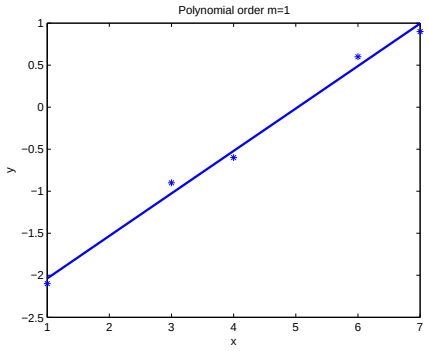


FIGURE 6.3 – Droite de régression linéaire (Script MATLAB `s_least2.m`)

Ici, nous nous limiterons à montrer l'effet de l'ordre du polynôme sur la qualité de l'approximation du polynôme. Le script `s_unstable2.m` montre, pour un ensemble fixé de 16 couples (x_i, y_i) échantillonnés à partir d'une relation non-linéaire, l'allure du polynôme aux moindres carrés pour un ordre m croissant.

Notons, que pour des petits m, le polynôme n'est pas capable de représenter la non-linéarité de la fonction sous-jacent les données (ce phénomène est connu sous le nom anglais d'*underfitting*) tandis que pour des ordres m trop grands, le polynôme affiche une complexité trop importante pour la fonction en question (en anglais *overfitting*.

L'idée qu'en augmentant l'ordre du polynôme, la précision ne peut qu'augmenter se révèle donc fausse. Au contraire, une bonne approximation dépend du choix optimal du degré du polynôme aux moindres carrés.

•

6.6 Radial Basis Functions (RBF)

Les réseaux de neurones ont été proposés à partir des années soixante en alternative aux modèles classiques d'analyse de données [1]. Les *Radial Basis Functions (RBF)* sont un exemple parmi les plus connus de réseau de neurones. Ici, elles nous servent comme un exemple de polynôme aux moindre carrés

$$h(x) = \sum_{j=0}^{m} a_j \varphi_j(x) \tag{6.27}$$

qui affiche une forme alternative de la fonction $\varphi_j(\cdot)$, $j = 0, \ldots, m$.

L'idée de RBF est de poser $\varphi_0(x) = 1$ et

$$\varphi_j(x) = \exp\left\{-\frac{(x - c_j)^2}{b_j^2}\right\} \qquad j > 1 \tag{6.28}$$

La fonction $\varphi_j(\cdot)$, appelée *fonction noyau*, a la forme d'une fonction gaussienne qui est centrée sur la valeur c_j et dont la largeur dépend du paramètre b_j. La fonction $\varphi_j(x)$ renvoie donc une valeur qui décroît pour un x qui s'eloigne du centre.

Si les termes $c_j \in \mathbb{R}$ et $b_j \in \mathbb{R}$, $j = 1, \ldots, m$, sont fixes, alors le *fitting* de la fonction $h(\cdot)$ aux données est fait par la méthode des moindres carrés.

Autrement, des techniques non linéaires sont nécessaires pour estimer les termes c_j et b_j, $j = 1, \ldots, m$.

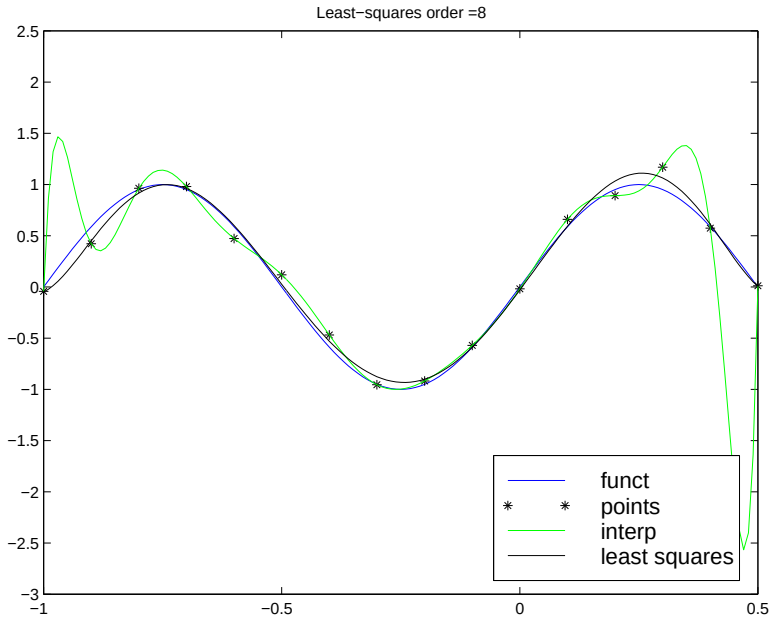


FIGURE 6.4 – Trade-of underfitting/overfitting : la fonction utilisée pour générer les données (astérisques) est très mal représentée par un polynôme avec un ordre trop important (en pointillé) (Script MATLAB `s_unstable2.m`).

Exemple Cet exemple montre le fitting d'une RBF à un jeu de données formé par 11 couples entrée/sortie. Les données et le graphique du polynôme résultant sont illustrés en Figure 6.6

•

6.6.1 Problèmes à plusieurs dimensions

L'utilisation de polynômes pour les problèmes de fitting avec $d > 1$ dimensions $\{x^{(1)}, \ldots, x^{(d)}\}$ est problématique à cause du grand nombre de paramètres.

Par exemple, l'expression d'un polynôme de degré $m = 3$ pour d dimensions est

$$\Pi_m(x^{(1)}, \ldots, x^{(d)}) = a_0 + \sum_{h=1}^{d} a_{1h} x^{(h)} + \sum_{h_1=1}^{d} \sum_{h_2=1}^{d} a_{2h_1h_2} x^{(h_1)} x^{(h_2)} + \\ + \sum_{h_1=1}^{d} \sum_{h_2=1}^{d} \sum_{h_3=1}^{d} a_{3h_1h_2h_3} x^{(h_1)} x^{(h_2)} x^{(h_3)}$$

Pour $d > 1$ le nombre de paramètres est de l'ordre $O(d^m)$.

Par contre, l'extension des RBF au cas multidimensionnel est plus facile. Il s'agit, en effet, de considérer des modèles d'approximation de la forme

$$h(x) = a_0 + \sum_{j=1}^{m} a_j \varphi_j(x^{(1)}, \ldots, x^{(d)}) \tag{6.29}$$

où

$$\varphi_j(x) = \exp\left\{ -\frac{\|x - c_j\|^2}{b_j^2} \right\} \tag{6.30}$$

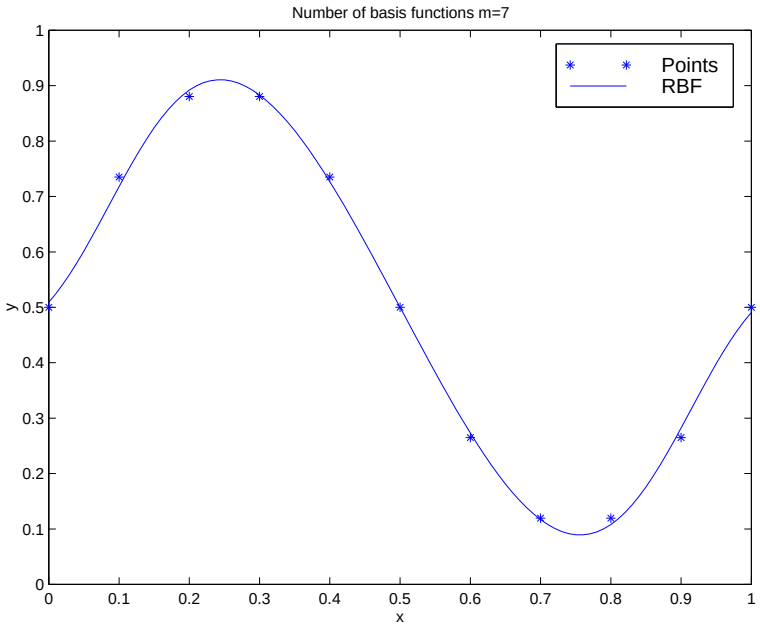


FIGURE 6.5 – Fitting d'une Radial Basis Function (Script MATLAB `s_rbf.m`)

et $x \in \mathbb{R}^d$, $c_j \in \mathbb{R}^d$, $b_j \in \mathbb{R}$. On a montré de manière théorique et expérimentale que dans ces modèles le nombre m de termes ne doit pas augmenter de façon exponentielle par rapport à la dimension d afin de garantir un bon pouvoir d'approximation.

Comme dans le cas unidimensionnel, si les paramètres c_j et b_j sont connus, les paramètres a_j peuvent être calculés par la méthode des moindres carrés.

Exemple Considérons un jeu de 400 données obtenu en échantillonnant la fonction bidimensionelle

$$z = 0.1 + \frac{(1 + \sin(2x + 3y))}{(3.5 + \sin(x - y))}$$

La Figure 6.6 illustre le graphique de l'approximation obtenu en utilisant un modèle RBF.

•

6.7 Exercices

6.7.1 Approximation au sens des moindres carrés

1. Soit $n = 5$, $m = 2$, $\varphi_0(x) = 1$, $\varphi_1(x) = x$ et $\varphi_2(x) = x^2$. Ecrivez le système aux équations normales correspondant aux données suivantes :

x_i	1	3	4	5	6	8
y_i	-2	-0.8	-0.5	0.1	0.7	1.0

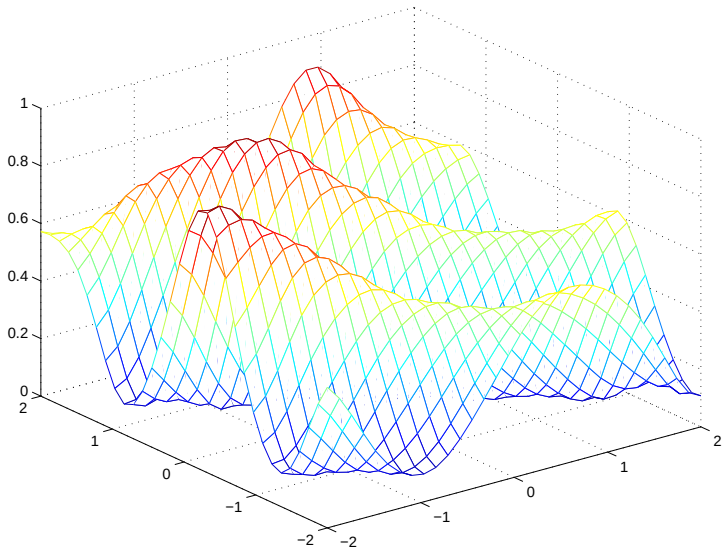


FIGURE 6.6 – RBF fitting sur deux dimensions (Script MATLAB `s_rbf2.m`)

Chapitre 7

Résolution d'équations non linéaires

Ce chapitre traite le problème du calcul des racines d'une fonction réelle d'une variable réelle

$$f : \mathcal{I} =]a, b[\subseteq \mathcal{R} \rightarrow \mathcal{R} \tag{7.1}$$

Le problème consiste à trouver les zéros de la fonction f, c.à.d. les valeurs $\alpha \in \mathbb{R}$ telles que

$$f(\alpha) = 0 \tag{7.2}$$

Notons que le nombre de racines dépend du type et de l'ordre de la fonction (Figure 7.1)

Voyons un exemple où la résolution numérique de ce problème est requise.

7.1 Exemple et motivations pratiques : balle flottante

Considérons une balle de rayon r immergée dans un liquide (Figure 7.1). Si la densité du matériel composant la balle est ρ, un problème intéressant est trouver la profondeur de l'immersion de la balle.

Selon le théorème d'Archimede, *tout corps plongé dans un fluide reçoit de la part de celui-ci une poussée verticale dirigée de bas en haut égale au poids du volume de fluide deplacé par le corps.*

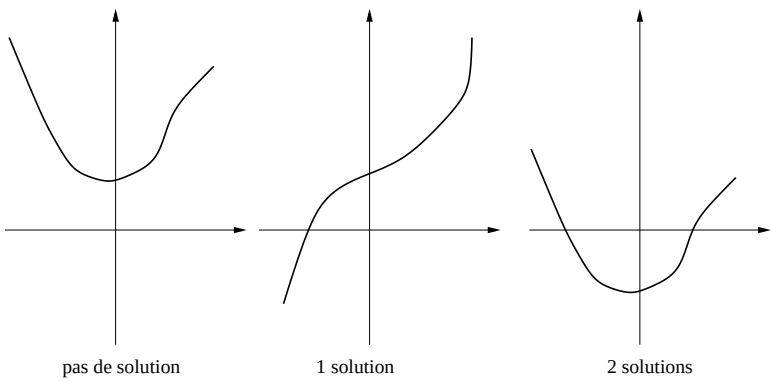
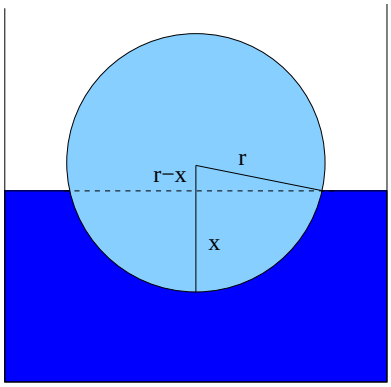


FIGURE 7.1 – Fonctions avec différents nombres de racines.



Le volume du fluide deplacé est le volume de la calotte spherique de hauter x

$$V = \int_0^x (r^2 - (r - x + z)^2)dz = \frac{1}{3}\pi x^2(3r - x).$$

Le poids du corps sphérique est

$$P = \rho(4\pi r^3)/3.$$

Afin de connaître la profondeur de l'immersion d'une balle ayant densité ρ et rayon r , il faut résoudre l'équation non linéaire

$$\pi x^2(3r - x)/3 = \rho(4\pi r^3)/3$$

Supposons que le liquide ait densité unitaire $\rho = 1$ et que $r = 1$. L'equation peut donc être simplifiée sous la forme

$$x^3 - 3x^2 + 4\rho = 0$$

Puisque l'équation est non linéaire et une solution exacte ne peut être trouvée d'une manière analytique, nous devons faire appel à une résolution approchée.

7.2 Les méthode numériques

La solution analytique du problème (7.2) est rarement disponible. Donc, on fait typiquement appel à des méthodes numériques. Les méthodes numériques pour approcher une racine α sont en général itératives. Dans ce cas la méthode vise à construire une suite $\{x^{(k)}\}$ telle que

$$\lim_{k\to\infty} x^{(k)} = \alpha \tag{7.3}$$

Notons que, en posant $g' = f$, puisque les zéros de f sont aussi les extrêmes de g, le problème de la recherche des racines est strictement lié au problème de la recherche des extrêmes d'une fonction g (problème d'optimisation).

Définition 37 (Convergence de la méthode)**.** *On dit qu'une suite* $\{x^{(k)}\}$ *converge vers* α *avec un ordre* $p \geq 1$ *si*

$$\exists C > 0 : \frac{|x^{(k+1)} - \alpha|}{|x^{(k)} - \alpha|^p} \leq C, \forall k \geq k_0 \tag{7.4}$$

où la constante C *est dite le* facteur de convergence *de la méthode.*

Si l'ordre $p = 1$, la convergence est dite *linéaire* et $C < 1$.

La convergence est autant plus rapide que la valeur de p est grande.

Contrairement à ce qui arrive dans les systèmes linéaires, la convergence des méthodes itératives pour la détermination des racines d'une équation non linéaire dépend en général du choix de la donnée initiale $x^{(0)}$. Les méthodes qui convergent vers α *pour tout choix* de $x^{(0)}$ sont dites *globalement convergentes* vers α.

Toutefois, la plupart des fois on ne peut établir que des résultats de *convergence locale*, c.à.d. valables seulement pour un $x^{(0)}$ qui appartient à un voisinage de la racine α. On verra dans la suite que typiquement les méthodes à convergence locale ont un ordre de convergence plus grand.

7.3 Conditionnement du problème

Considérons l'équation non linéaire

$$f(\alpha) = \varphi(\alpha) - d = 0 \tag{7.5}$$

Considérons la sensibilité de la solution α par rapport à des perturbations de la donnée d. Le problème n'est bien posé que si la fonction φ est localement inversible, c.à.d. $\alpha = F(d) = \varphi^{-1}(d)$. Si $f'(\alpha) \neq 0$, puisque $F'(d) = \frac{1}{f'(\alpha)}$ les définitions de conditionnement relatif et absolu (Équations 1.7 et 1.8) permettent d'obtenir

$$K(d) \approx \|F'(d)\| \frac{\|d\|}{\|F(d)\|} = \frac{1}{|f'(\alpha)|} \frac{|d|}{|\alpha|} \tag{7.6}$$

et

$$K_{abs}(d) \approx \frac{1}{|f'(\alpha)|} \tag{7.7}$$

Il s'ensuit que le problème est bien conditionné si $|f'(\alpha)|$ est grande et mal conditionné si $|f'(\alpha)|$ est petite.

Considérons maintenant le cas d'une racine α de multiplicité m. Ceci signifie que $f^{(k)}(\alpha) = 0$, pour $k = 1, \ldots, m-1$. Si α est une racine multiple de multiplicité $m > 1$, sur la base du développement de Taylor de φ au point α jusqu'à l'ordre m, on obtient

$$d + \delta d = \varphi(\alpha + \delta\alpha) \approx \varphi(\alpha) + \sum_{k=1}^{m} \frac{\varphi^{(k)}(\alpha)}{k!} (\delta\alpha)^k \tag{7.8}$$

Puisque $\varphi^{(k)}(\alpha) = 0$ pour $k = 1, \ldots, m-1$, il s'ensuit que

$$\delta d = \frac{f^{(m)}(\alpha)(\delta\alpha)^m}{m!} \tag{7.9}$$

et

$$K_{abs}(d) \approx \frac{|\delta\alpha|}{|\delta d|} \approx \left| \frac{m! \delta d}{f^{(m)}(\alpha)} \right|^{1/m} \frac{1}{|\delta d|} \tag{7.10}$$

Comme montré en Figure 7.2, la quantité $K_{abs}(d)$ peut être grande même quand δd est petit. Il s'ensuit que le problème est toujours mal conditionné dans le cas de racines multiples.

Toutefois, le problème de la détermination d'une racine d'une équation non linéaire reste bien conditionné quand

1. la racine α est une racine simple
2. la quantité $|f'(\alpha)|$ est très différent de zéro.

Dans ce qui suit nous allons introduire les méthodes le plus connues pour la solution approchée du problème (7.2).

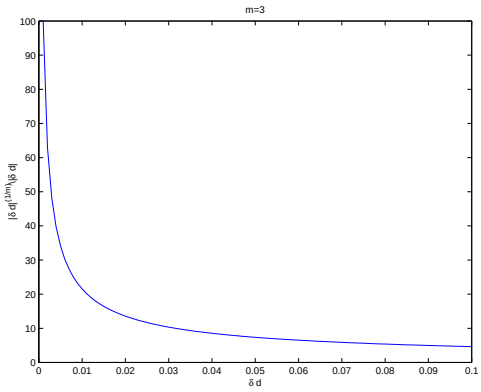


FIGURE 7.2 – Allure du conditionnement absolu pour une racine de multiplicité $m = 3$

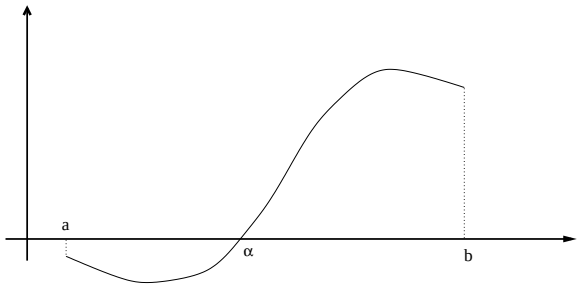


FIGURE 7.3 – Fonction continue passant par l'origine

7.4 Méthode de la bissection (ou de dichotomie)

La méthode est fondée sur le théorème suivant :

Théorème 7.1 (Zéros d'une fonction continue)**.** *Étant donnée une fonction continue $f : [a, b] \to \mathcal{R}$, si $f(a)f(b) < 0$ (Figure 7.3), alors $\exists \alpha \in]a, b[$ tel que $f(\alpha) = 0$.*

L'algorithme de bissection pose l'initialisation suivante :

$$a^{(0)} = a, \qquad b^{(0)} = b, \qquad x^{(0)} = \frac{a^{(0)} + b^{(0)}}{2} \tag{7.11}$$

Ensuite, il construit une série des solutions approchées $x^{(k)}$ de la solution exacte x de la manière suivante. Pour $k > 0$, soit $|I_k| = |b^{(k)} - a^{(k)}|$ la longueur de l'intervalle $[a^{(k)}, b^{(k)}]$, alors

– Si $f(x^{(k)})f(a^{(k)}) < 0$

$$a^{(k+1)} = a^{(k)}, \qquad b^{(k+1)} = x^{(k)} \tag{7.12}$$

– Si $f(x^{(k)})f(b^{(k)}) < 0$

$$a^{(k+1)} = x^{(k)}, \qquad b^{(k+1)} = b^{(k)} \tag{7.13}$$

et

$$x^{(k+1)} = \frac{a^{(k+1)} + b^{(k+1)}}{2} \tag{7.14}$$

Les itérations s'achèvent à la Kème étape quand

$$|x^{(K)} - \alpha| \leq |I_K| \leq \epsilon \tag{7.15}$$

Considérons à présent la convergence de l'algorithme.
Soit $|I_0| = |b - a|$ et $|I_k| = |I_0|/2^k = |b - a|/2^k$ pour $k \geq 0$.
En notant $e^{(k)} = x^{(k)} - \alpha$ l' *erreur absolue* à l'étape k, on déduit que

$$|e^{(k)}| = |x^{(k)} - \alpha| \leq \frac{|b - a|}{2^{k+1}} \text{ pour } k \geq 0 \tag{7.16}$$

ce qui entraîne

$$\lim_{k \to \infty} |e^{(k)}| = 0 \tag{7.17}$$

Donc la méthode de la bissection est *globalement convergente*.

De plus, pour avoir à la $(K-1)$-ème étape

$$|e^{(K-1)}| = |x^{(K-1)} - \alpha| \leq \epsilon \tag{7.18}$$

puisque $\log_a(x) = \frac{\ln(x)}{\ln(a)}$ où ln est le logarithme népérien, on doit prendre

$$K \geq \log_2(b - a) - \log_2 \epsilon = \frac{\ln((b - a)/\epsilon)}{\ln(2)} \approx \frac{\ln((b - a)/\epsilon)}{0.6931} \tag{7.19}$$

Etudions maintenant la vitesse de convergence de la méthode. Soit $x^{(k)}$ la solution approchée à l'instant k. Nous voulons déterminer à quelle étape $k > j$ l'erreur sera un dixième de l'erreur à la $j$ème étape, c.à.d.

$$|x^{(k)} - \alpha| = |x^{(j)} - \alpha|/10 \tag{7.20}$$

Puisque

$$|x^{(k)} - \alpha| \leq \frac{|I_0|}{2^{k+1}} \quad \text{et} \quad |x^{(k)} - \alpha| = \frac{|x^{(j)} - \alpha|}{10} \leq \frac{|I_0|}{2^{j+1} \cdot 10}$$

une estimation rapide peut être obtenue en imposant l'égalité des deux bornes supérieures. Nous obtenons

$$2^{k+1} = 2^{j+1} \cdot 10 \Rightarrow k - j = \log_2(10) \approx 3.32$$

Ceci montre que l'algorithme de bissection converge lentement. La lente convergence de la méthode suggère une utilisation de la méthode pour avoir une bonne initialisation d'une méthode d'ordre supérieur.

Voici quelques autres considérations sur la méthode :

- la méthode ne prend pas en compte les informations données par les valeurs de f et, éventuellement, par sa dérivée f'.
- la méthode ne garantit pas la réduction monotone de l'erreur.
- la méthode n'est pas une méthode d'ordre 1 au sens de la définition d'ordre de convergence.

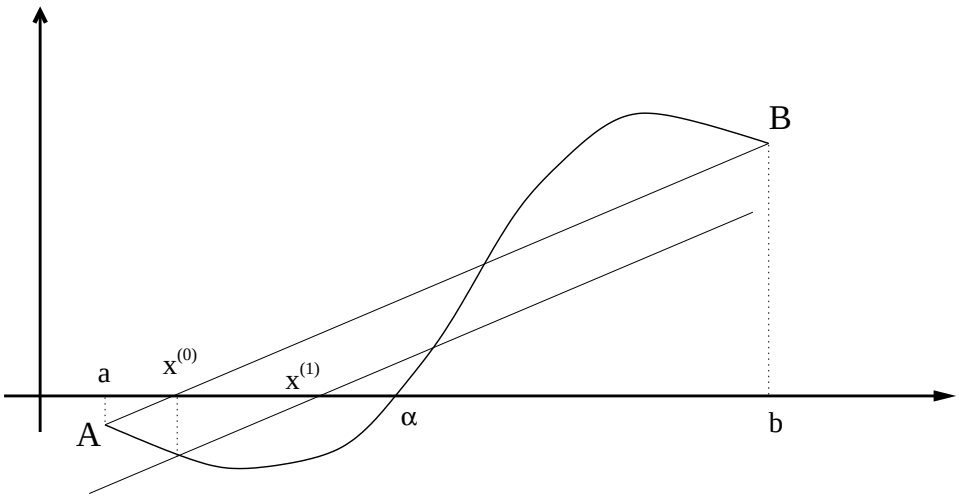


FIGURE 7.4 – Les premières étapes de la méthode itérative de la corde.

7.5 Méthodes linéarisées

Afin de garantir une convergence plus rapide que dans la méthode de bissection, il est nécessaire de prendre en compte l'information associée à la valeur de f dans la solution approchée et, éventuellement, à sa dérivée f'.

Pour cela, développons f en série de Taylor autour de α

$$f(\alpha) = 0 = f(x) + (\alpha - x)f'(\xi) \text{ où } \xi \in [\alpha, x] \tag{7.21}$$

Cette équation conduit à

$$f(x^{(k)}) + (x^{(k+1)} - x^{(k)})q_k = 0 \tag{7.22}$$

où q_k est une approximation de $f'(\xi)$, et donc à la méthode itérative suivante :

$$x^{(k+1)} = x^{(k)} - q_k^{-1} f(x^{(k)}). \tag{7.23}$$

Cela revient à

1. approcher une fonction non-linéaire par une droite et
2. déterminer $x^{(k+1)}$ comme l'intersection entre l'axe des x et la droite de pente q_k passant par $(x^{(k)}, f(x^{(k)}))$.

Nous montrerons 4 méthodes qui se différencient par le choix de q_k.

7.5.1 Méthode de la corde

La méthode de la corde pose

$$q_k = \frac{f(b) - f(a)}{b - a} \tag{7.24}$$

dans (7.23). La quantité q_k représente la pente de la droite passant par $A = (a, f(a))$ et $B = (b, f(b))$ (Figure 7.4).

Dans la Section7.6.2.1 nous allons montrer sous quelles conditions la méthode converge localement. On peut aussi montrer que si la méthode de la corde converge localement, alors elle converge avec un ordre $p = 1$.

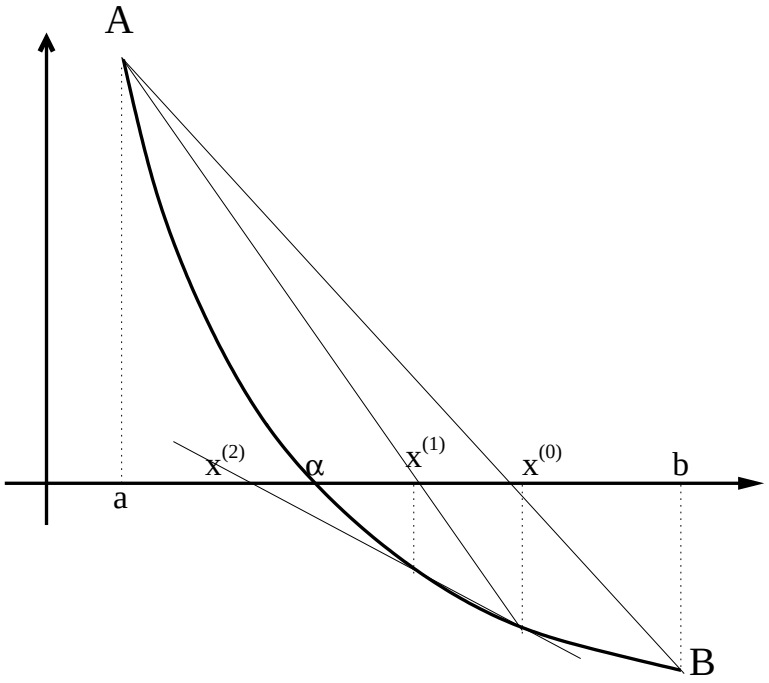


FIGURE 7.5 – Les premières étapes de la méthode itérative de la sécante.

7.5.2 Méthode de la sécante

La méthode de la sécante pose

$$q_k = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \tag{7.25}$$

dans (7.23). La quantité q_k représente la pente de la droite passant par $(x^{(k-1)}, f(x^{(k-1)}))$ et $(x^{(k)}, f(x^{(k)}))$ (Figure 7.5).

On peut montrer que la méthode de la sécante jouit de la propriété de *convergence locale*. Ceci signifie que si les données initiales sont assez proches de α et $f'(\alpha) \neq 0$, la méthode converge (avec un ordre $p = 1.63$).

7.5.3 Méthode de la fausse position

La méthode de la fausse position pose

$$q_k = \frac{f(x^{(k)}) - f(x^{(k')})}{x^{(k)} - x^{(k')}} \tag{7.26}$$

où k' est le plus grand indice inférieur à k tel que $f(x^{(k)})f(x^{(k')}) < 0$ (Figure 7.6).

On peut montrer que la méthode est *globalement convergente* et converge avec un ordre $p = 1$.

7.5.4 Méthode de Newton

Soit la fonction f continue sur $\mathcal{I}$ et $f'(\alpha) \neq 0$. La méthode de Newton pose

$$q_k = f'(x^{(k)}) \tag{7.27}$$

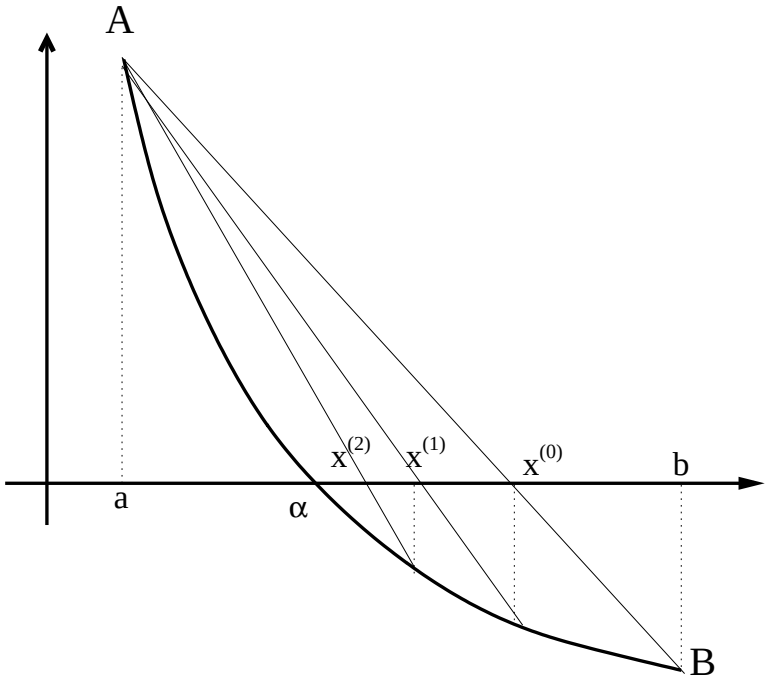


FIGURE 7.6 – Les premières étapes de la méthode itérative de la fausse position.

dans (7.23). La quantité q_k représente la pente de la tangente à f en $x^{(k)}$ (Figure 7.7).

Cette méthode nécessite à chaque étape l'évaluation des deux fonctions, c.a.d. $f(\cdot)$ et $f'(\cdot)$.

On peut montrer que la méthode jouit de la *convergence locale* : si $x^{(0)}$ est assez proche de α et $f'(\alpha) \neq 0$, la méthode converge avec un ordre $p = 2$.

Exemple Considérons le problème de la recherche de la racine de la fonction non-linéaire $f(x) = x^3 - 8$.

Le script MATLAB `s_iter.m` montre la suite des solutions approchées générées par les 4 méthodes linéarisées. Figure 7.8 montre l'affichage graphique du script pour la méthode de la corde.

•

7.6 Itérations de point fixe

Un procédé général pour trouver les racines d'une équation non linéaire $f(x) = 0$ où $f : [a, b] \to \mathbb{R}$ consiste à transformer le problème de la recherche de la racine α dans le problème équivalent

$$x - \phi(x) = 0 \text{ c.à.d. } x = \phi(x) \tag{7.28}$$

où $\phi(\cdot)$ est telle que $\phi(\alpha) = \alpha$.

Le point α est dit *point fixe* de la fonction $\phi(\cdot)$ selon la définition suivante.

Définition 38 (Point fixe)**.** *Le point fixe d'une fonction $\phi(\cdot)$ est un nombre réel p tel que $\phi(p) = p$.*

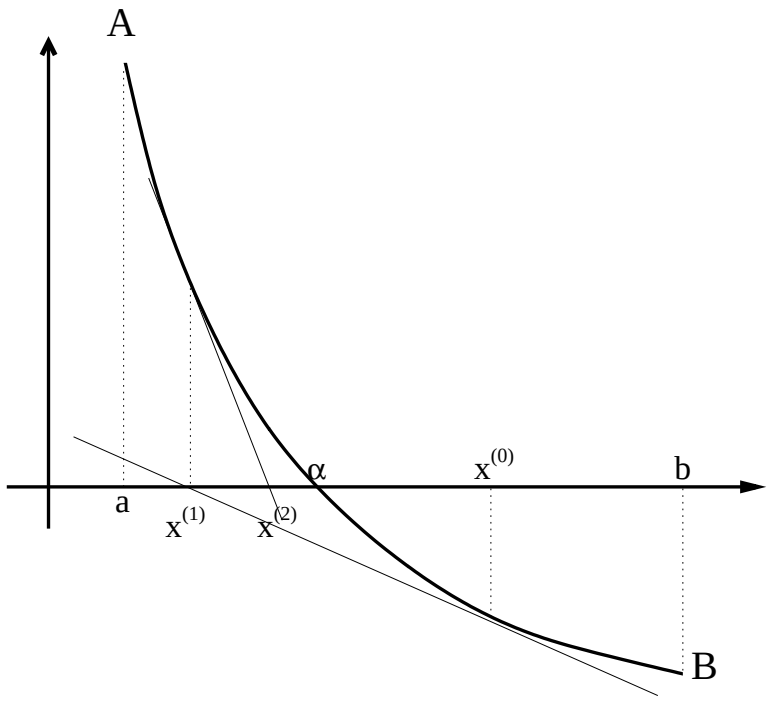


FIGURE 7.7 – Les premières étapes de la méthode itérative de Newton.

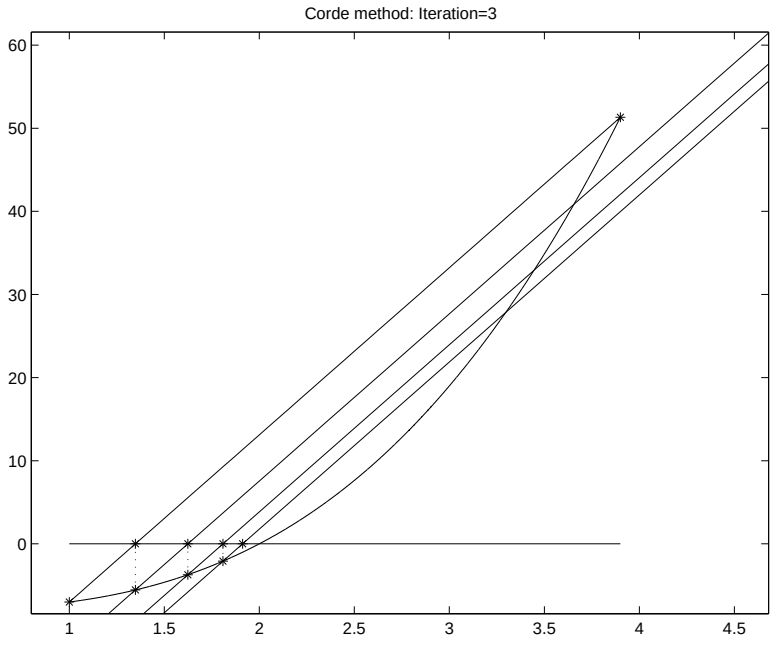


FIGURE 7.8 – Les premières étapes de la méthode itérative de la corde. (Script MATLAB `s_iter.m`)

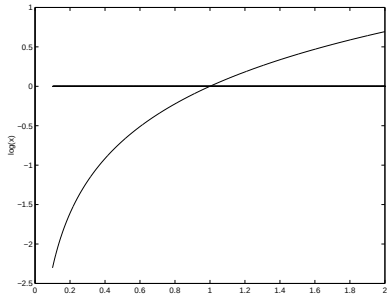


FIGURE 7.9 – Problème original : $f(x) = \log(x) = 0$.

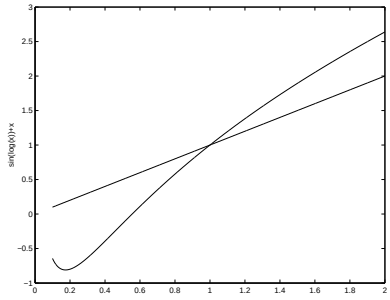


FIGURE 7.10 – Problème de point fixe : $\phi(x) = \sin(\log(x)) + x = x$.

Approcher les zéros de f se ramène donc au problème de la détermination des points fixes de ϕ.

La transformation du problème de la recherche d'une racine d'une fonction f dans le problème de la recherche du point fixe d'une fonction ϕ est toujours possible. Toutefois cette transformation n'est pas unique. Par exemple, un ensemble de transformations plausibles sont les transformations

$$\phi(x) = F(f(x)) + x \tag{7.29}$$

où $F(\cdot)$ est une fonction continue telle que $F(0) = 0$.

Un exemple de telle fonction est la fonction $F(\cdot) = \sin(\cdot)$. Ceci signifie que chercher la racine de $f(x) = 0$ peut être ramené à la recherche du point fixe de $\sin(f(x)) + x$.

Considérons la fonction $f(x) = \log(x)$ définie dans l'intervalle $I =]0.1, 2[$. Nous définissons maintenant la fonction $\phi(x) = sin(log(x)) + x$. Figure 7.9 montre l'intersection de la fonction f avec l'axe x alors que la Figure 7.10 montre l'intersection de la fonction ϕ avec la droite bissectrice. Notons que les deux intersections ont lieu pour la même valeur $\alpha = 1$.

7.6.1 Algorithme d'itération de point fixe

Comme discuté dans la section précédente, une fois effectuée la transformation du problème (7.2) dans le problème (7.28), approcher les zéros de $f(\cdot)$ revient à déterminer les points fixes de $\phi(\cdot)$.

Étant donné $x^{(0)}$, l'algorithme itératif *de point fixe* est basé sur la transformation

$$x^{(k+1)} = \phi(x^{(k)}) \text{ pour } k \geq 0 \tag{7.30}$$

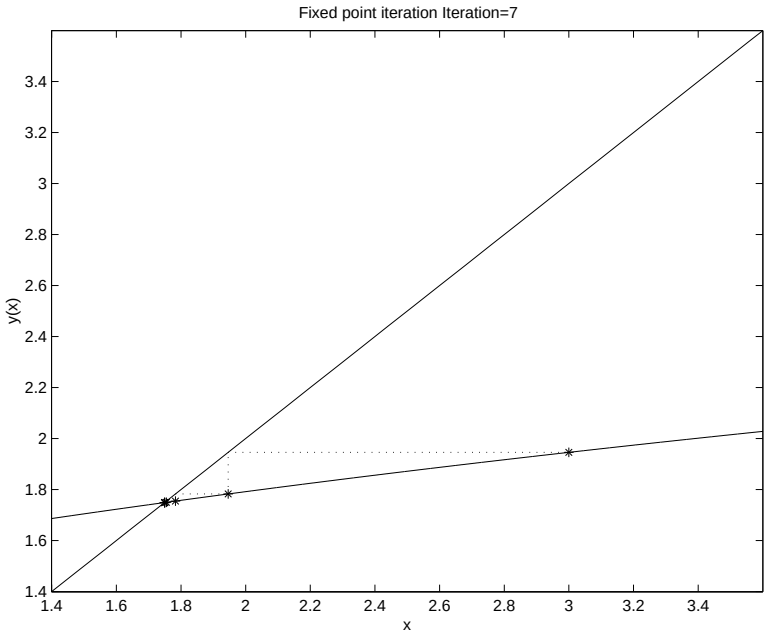


FIGURE 7.11 – Les premières étapes de la méthode itérative du point fixe. (Script MATLAB `s_fixpt.m`)

où $\phi(\cdot)$ est dite la *fonction d'itération* associée.

Exemple Considérons le problème de la recherche du point fixe de la fonction non-linéaire $\phi(x) = \log(x + 4)$.

Le script MATLAB `s_fixpt.m` montre la suite des solutions approchées générées par l'algorithme d'itération de point fixe (Figure 7.11).

•

Si α est un point fixe de $\phi(x)$, alors la méthode est *fortement consistante* selon la définition en Section 1.3.4.

Pour ce qui concerne la convergence, les théorèmes suivants ont été démontrés.

Théorème 7.2. *Soit $\phi(\cdot)$ une fonction continue et $\{x^{(k)}\}_{k=0}^{\infty}$ une suite générée par l'itération de point fixe. Si*

$$\lim_{k\to\infty} x^{(k)} = \alpha \tag{7.31}$$

alors α est un point fixe de $\phi(\cdot)$.

Théorème 7.3. *Soit $\phi(\cdot) \in C^1[a, b]$.*

- *Si $\forall x \in [a, b]$, $\phi(x) \in [a, b]$ alors $\phi(\cdot)$ a un point fixe dans $[a, b]$*
- *Si, de plus, $\exists K < 1 : |\phi'(x)| \leq K \ \forall x \in [a, b]$ alors*

  1. *$\phi(\cdot)$ a un unique point fixe α dans $[a, b]$ et la suite $\{x^{(k)}\}$ converge vers α pour tout choix de $x^{(0)}$. Dans ce cas α est appelé* point attracteur *(Figure 7.12).*
  2.
$$\lim_{k\to\infty} \frac{x^{(k+1)} - \alpha}{x^{(k)} - \alpha} = \lim_{k\to\infty} \phi'(\eta^{(k)}) = \phi'(\alpha) \tag{7.32}$$

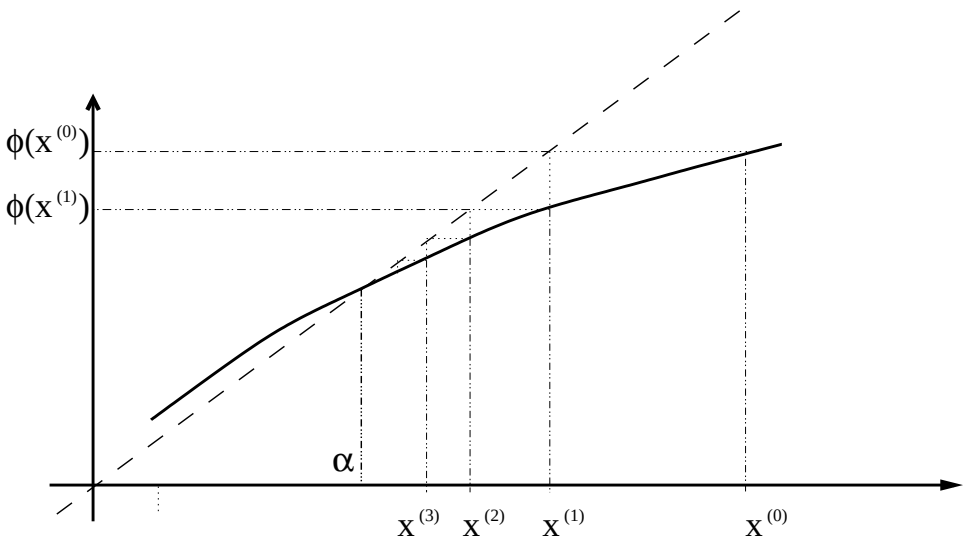


FIGURE 7.12 – Méthode de point fixe : point attracteur.

où $\eta^{(k)}$ est une valeur entre α et $x^{(k)}$ telle que

$$x^{(k+1)} - \alpha = \phi(x^{(k)}) - \phi(\alpha) = \phi'(\eta^{(k)})(x^{(k)} - \alpha)$$

– *Si $|\phi'(x)| > 1 \ \forall x \in [a, b]$ alors la suite $\{x^{(k)}\}$ ne converge pas si $x^{(0)} \neq \alpha$. Dans ce cas α est appelé* point répulsif *(Figure 7.13).*

Le théorème nous apprend que si $K < 1$ et $\phi'(\alpha) \neq 0$ la méthode converge *globalement* avec un ordre $p = 1$.

Le résultat général sur le taux de convergence des méthodes de point fixe est énoncé comme suit

Théorème 7.4. *Si*
– *la fonction $\phi \in C^{p+1}(\mathcal{I})$ pour un certain voisinage $\mathcal{I}$ de α*
– *$p \geq 1$*
– *$\phi^{(i)}(\alpha) = 0$ pour $1 \leq i \leq p$*
– *$\phi^{(p+1)}(\alpha) \neq 0$*

alors la méthode de point fixe associée à la fonction d'itération ϕ est d'ordre $p + 1$ et

$$\lim_{k \to \infty} \frac{x^{(k+1)} - \alpha}{(x^{(k)} - \alpha)^{p+1}} = \frac{\phi^{p+1}(\alpha)}{(p+1)!} \tag{7.33}$$

En pratique il est souvent difficile de déterminer l'intervalle $[a, b]$ qui satisfait les conditions de convergence globale. Dans ce cas , le résultat de convergence local suivant peut être utile

Théorème 7.5 (Ostrowski)**.** *. Soit α un point fixe d'une fonction ϕ continue et différentiable dans un voisinage $\mathcal{I}$ de α. Si $|\phi'(\alpha)| < 1$ alors il existe $\delta > 0$ tel que que la suite $\{x^{(k)}\}$ converge vers α pour tout $x^{(0)}$ tel que $|x^{(0)} - \alpha| < \delta$.*

Remarque On peut montrer que si $|\phi'(\alpha)| > 1$ la convergence est impossible. alors que si $|\phi'(\alpha)| = 1$ on ne peut en général tirer aucune conclusion.

•

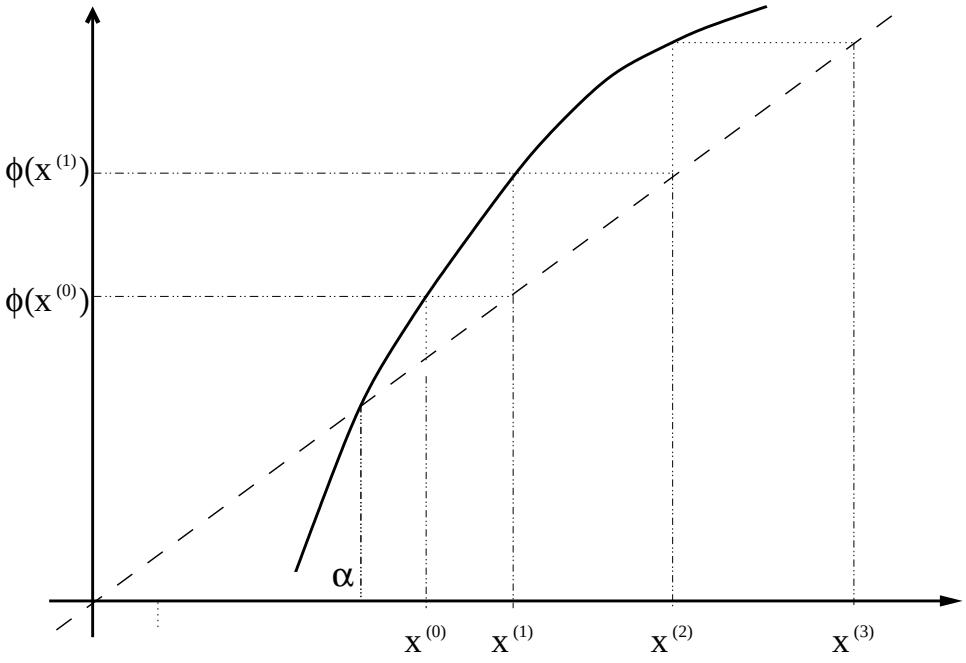


FIGURE 7.13 – Méthode de point fixe : point répulsif.

7.6.2 Analyse des méthodes linéarisées

Les résultats théoriques sur la méthode de point fixe rendent possible une analyse de convergence des méthodes linéarisées (Section 7.5). Voyons quelques exemples.

7.6.2.1 Analyse de la méthode de la corde

La méthode de la corde peut être écrite sous la forme $x^{(k+1)} = \phi(x^{(k)})$ où

$$\phi(x) = x - \frac{b-a}{f(b)-f(a)} f(x) \tag{7.34}$$

La condition de convergence locale $|\phi'(\alpha)| < 1$ (théorème 7.5) est équivalente à

$$0 < \frac{b-a}{f(b)-f(a)} f'(\alpha) < 2 \tag{7.35}$$

Sauf dans le cas exceptionnel où $\phi'(\alpha) = 0$, la convergence est linéaire.

7.6.2.2 Analyse de la méthode de Newton

La méthode de Newton peut être écrite sous la forme $x^{(k+1)} = \phi(x^{(k)})$ où

$$\phi(x) = x - \frac{f(x)}{f'(x)} \tag{7.36}$$

En supposant $f'(\alpha) \neq 0$ on obtient

$$\phi'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} \Rightarrow \phi'(\alpha) = 0 \tag{7.37}$$

La méthode est convergente localement. On peut montrer qu'elle est convergente d'ordre $p = 2$.

7.7 Les tests d'arrêt

Comme pour les cas linéaire (Section 3.12.8), il faut déterminer des critères d'arrêt des méthodes itératives qui soient basés sur des quantités observables. Il est donc essentiel d'établir la relation analytique qui existe entre les quantités observables et l'erreur $x^{(k)} - \alpha$ qu'on veut minimiser.

Dans ce qui suit nous analyserons en détail la relation entre erreur et résidu et la relation entre erreur et incrément.

7.7.1 Erreur et résidu

Soit $x^{(k)}$ une solution approchée de la racine α à la k-ème itération. Nous définissons l'erreur $e^{(k)} = x^{(k)} - \alpha$ et le résidu $r^{(k)}$ comme

$$r^{(k)} = f(x^{(k)}) - f(\alpha) = f(x^{(k)}) \tag{7.38}$$

Si α est une racine de multiplicité m

$$f(x^{(k)}) \approx f(\alpha) + \frac{f^{(m)}(\alpha)}{m!}(x^{(k)} - \alpha)^m \tag{7.39}$$

puisque $f(\alpha) = 0$, on obtient la relation suivante entre erreur et résidu

$$|e^{(k)}| \approx \left(\frac{m!}{|f^{(m)}(\alpha)|}\right)^{1/m} |r^{(k)}|^{1/m} \tag{7.40}$$

Considérons à présent un critère d'arrêt basé sur le résidu, c.-à-d. les itérations s'achèvent dès que $|r^{(k)}| = |f(x^{(k)})| < \varepsilon$.

Soit $e^{(k)} = \alpha - x^{(k)}$ et ε une tolérance fixée pour le calcul approché de α. Nous supposerons de plus f continûment différentiable dans un voisinage de α

Puisque

$$|e^{(k)}| \approx \left(\frac{m!}{|f^{(m)}(\alpha)|}\right)^{1/m} |f(x^{(k)})|^{1/m} \tag{7.41}$$

on obtient que pour $m = 1$

1. si $|f'(\alpha)| \approx 1$, alors $e^{(k)} \approx \varepsilon$.
2. si $|f'(\alpha)| << 1$, $|e^{(k)}|$ peut être assez grand par rapport à ε.
3. si $|f'(\alpha)| >> 1$, on a $|e^{(k)}| << \varepsilon$, c.à.d. le test est trop restrictif.

7.7.2 Erreur et incrément

Soit $x^{(k)}$ la suite produite par l'algorithme de point fixe. Puisque

$$e^{(k+1)} = \alpha - x^{(k+1)} = \phi(\alpha) - \phi(x^{(k)}) = \phi'(\psi^{(k)})(\alpha - x^{(k)}) = \phi'(\psi^{(k)})e^{(k)} \tag{7.42}$$

on obtient

$$x^{(k+1)} - x^{(k)} = e^{(k)} - e^{(k+1)} = \left(1 - \phi'(\psi^{(k)})\right) e^{(k)} \tag{7.43}$$

et en supposant que on puisse remplacer $\phi'(\psi^{(k)})$ par $\phi'(\alpha)$ on obtient l'approximation

$$e^{(k)} \approx \frac{1}{1 - \phi'(\alpha)}(x^{(k+1)} - x^{(k)}) \tag{7.44}$$

Soit $e^{(k)} = \alpha - x^{(k)}$ et ε une tolérance fixée pour le calcul approché de α. Nous supposerons de plus f continûment différentiable dans un voisinage de α

Si on effectue un test d'arrêt basé sur l'incrément, les itérations s'achèvent dès que $|x^{(k+1)} - x^{(k)}| < \varepsilon$.

Puisque

$$|e^{(k)}| \approx \frac{1}{|1 - \phi'(\alpha)|}|x^{(k+1)} - x^{(k)}| \tag{7.45}$$

le test
- n'est pas satisfaisant si $\phi'(\alpha)$ est proche de 1.
- est optimal si $\phi'(\alpha) = 0$ (comme pour la méthode de Newton)
- est encore satisfaisant si $-1 < \phi'(\alpha) < 0$.

7.8 Exercices

7.8.1 Méthode de la bissection

7.8.1.1

Utiliser la méthode de la bissection pour trouver une racine de la fonction $x^3 + 3x^2 - 1$ dans l'intervalle $[0, -2]$. Faites 5 itérations.

7.8.1.2

Que va-t-il se passer quand on utilisera la méthode de la bissection pour la fonction$f(x) = \tan(x)$ (x est exprimé en radians)

1. Sur l'intervalle $[3, 4]$?
2. Sur l'intervalle $[1, 3]$?

7.8.1.3

Supposons que l'on utilise la méthode de la bissection pour trouver un zero de $f(x)$ sur l'intervalle $[2, 7]$. Combien de fois devra-t-on itérer la méthode pour garantir que l'approximation C_n a une précision de 5×10^{-8} ?

7.8.2 Méthode de la corde

Utiliser la méthode de la corde pour trouver une racine de la fonction $x^3 - x^2 - 4x - 3$ dans l'intervalle $[2, 3]$. Faire 5 itérations et prendre $x^{(0)} = 2.5$.

7.8.3 Méthode de la sécante

Utiliser la méthode de la sécante pour trouver une racine de la fonction $6x^3 - 23x^2 + 20x$ dans l'intervalle $[1, 2]$. Faire 4 itérations.

7.8.4 Méthode de la fausse position ou regula falsi

Dans les exercices suivants, commencer avec l'intervalle $[a_0, b_0]$ et utiliser la méthode de la fausse position pour calculer 4 itérations.

1. $e^x - 2 - x = 0, [a_0, b_0] = [-2.4, -1.6]$
2. $cos(x) + 1 - x = 0, [a_0, b_0] = [0.8, 1.6]$
3. $ln(x) - 5 + x = 0, [a_0, b_0] = [3.2, 4.0]$
4. $x^2 - 10x + 23 = 0, [a_0, b_0] = [6.0, 6.8]$

7.8.5 Méthode de Newton-Raphson

Utiliser la méthode de Newton-Raphson pour trouver une racine de la fonction $x^3 - 9x + 2$ dans l'intervalle $[-4, -3]$. Faire 5 itérations et prendre $x^{(0)} = -3.5$.

7.8.6 Itérations de point fixe

7.8.6.1

Trouver un zéro de $f(x) = 2x^2 - 5x + 1$ en trouvant un point fixe. Utiliser d'abord 0, puis 1 comme valeur initiale. Faire 5 itérations.

7.8.6.2

Déterminer si chaque fonction a un point fixe unique sur l'intervalle donné.

1. $\phi(x) = 1 - \frac{x^2}{4}$ sur $[0, 1]$
2. $\phi(x) = 2^{-x}$ sur $[0, 1]$
3. $\phi(x) = \frac{1}{x}$ sur $[0.5, 5.2]$

Rappel : $(a^{f(x)})' = f'(x)a^{f(x)} \ln(a)$

7.8.6.3

Investiguer la nature de l'itération de point fixe quand

$$\phi(x) = -4 + 4x - \frac{1}{2}x^2$$

1. Résoudre $\phi(x) = x$ et montrer que 2 et 4 sont les deux seuls points fixes de la fonction
2. Utiliser la valeur de départ $x_0 = 1.9$ pour l'itération de point fixe et calculer 3 itérations de la méthode de point fixe.
3. Idem pour la valeur de départ $x_0 = 3.8$
4. Quelle conclusion peut-on envisager ? Comment vérifier cette hypothèse ?

7.8.6.4

Soit $f(x) = x^2 + x - 4$. Peut-on utiliser une itération de point fixe pour trouver les solutions de l'équation $f(x) = 0$? Pourquoi ?

7.8.6.5

Trouver une racine de $f(x) = x - x^3$ en utilisant une itération de point fixe.

7.8.7 Test d'arrêt

Soit la fonction $f(x) = e^{-x} - \eta$ avec $\eta > 0$. Posons $\eta = 0.05$.

1. Utiliser la méthode de Newton pour approximer la solution de l'équation $f(x) = 0$ ($x^{(0)} = 0$). Arrêter les itérations quand le résidu $r^{(k)} = |f(x^{(k)}) - f(\alpha)| = |f(x^{(k)})| < 0.1$. Quelle est la valeur de l'erreur pour la solution trouvée ?
2. Utiliser la méthode de Newton pour approximer la solution de l'équation $f(x) = 0$ ($x^{(0)} = 0$). Arrêter les itérations quand l'incrément $|x^{(k+1)} - x^{(k)}| < 0.1$. Quelle est la valeur de l'erreur pour la solution trouvée ?

Annexe A

Rappels d'algèbre linéaire

A.1 Espaces vectoriels

Définition A.1. *Un* espace vectoriel *sur un corps K ($K = \mathbb{R}$ ou $K = \mathbb{C}$) est un ensemble non vide V sur lequel on définit une loi interne notée $+$, appelée* addition*, et une loi externe, notée $\cdot$, appelée* multiplication par un scalaire*, qui possèdent les propriétés suivantes :*

1. *$(V,+)$ est un groupe commutatif,*
2. *la loi externe satisfait*
   - $\forall\ \alpha \in K,\ \forall\ \boldsymbol{v}, \boldsymbol{w} \in V, \alpha(\boldsymbol{v} + \boldsymbol{w}) = \alpha\boldsymbol{v} + \alpha\boldsymbol{w} \in V$;
   - $\forall\ \alpha, \beta \in K,\ \forall\ \boldsymbol{v} \in V, (\alpha + \beta)\boldsymbol{v} = \alpha\boldsymbol{v} + \beta\boldsymbol{v} \in V$ *et* $(\alpha\beta)\boldsymbol{v} = \alpha(\beta\boldsymbol{v})$;
   - $1 \cdot \boldsymbol{v} = \boldsymbol{v}$,

   où 1 est l'élément unité de K.

Les éléments de V sont appelés vecteurs*, ceux de K sont les* scalaires*.*

Définition A.2. *Une partie non vide W de V est un* sous-espace vectoriel *de V ssi*

$$\forall\ (\boldsymbol{v}, \boldsymbol{w})\ \in\ W^2,\ \forall\ (\alpha, \beta)\ \in\ K^2, \alpha\boldsymbol{v} + \beta\boldsymbol{w}\ \in\ W.$$

Combinaison linéaire des vecteurs $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n \in V$: vecteur de V de la forme

$$\alpha_1\boldsymbol{v}_1 + \alpha_2\boldsymbol{v}_2 + \ldots + \alpha_n\boldsymbol{v}_n$$

où les $\alpha_i\ \in\ K$.

L'ensemble W des combinaisons linéaires d'une famille de vecteurs $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n \in V$ est un sous-espace de V, appelé *sous-espace engendré* par la famille de vecteurs. On le note

$$W = \text{vect}(\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n)$$

Définition A.3. *La famille $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ est appelée* partie génératrice *de W.*

Définition A.4. *La famille $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_p$ de vecteurs de V est appelée* partie libre *si les vecteurs $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_p$ sont* linéairement indépendants *càd si*

$$\alpha_1\boldsymbol{v}_1 + \alpha_2\boldsymbol{v}_2 + \ldots + \alpha_p\boldsymbol{v}_p = \boldsymbol{0}$$

avec les $\alpha_i\ \in\ K$ implique

$$\alpha_1 = \alpha_2 = \ldots = \alpha_p = 0.$$

Définition A.5. *Base de V = partie libre et génératrice de V.*

Théorème A.1. *Si V est un espace vectoriel muni d'une base de n vecteurs, alors toute partie libre de V a au plus n éléments et toute autre base de V a exactement n éléments. Le nombre n est appelé* dimension *de V et on note dim(V)=n.*

Si pour tout n il existe n vecteurs de V linéairement indépendants, l'espace vectoriel est dit de dimension infinie.

A.2 Matrices

Soit $A = (a_{ij})$ avec $a_{ij} \in \mathbb{R}$ une matrice $m \times n$, c'est-à-dire une matrice avec m lignes et n colonnes :

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

- $A(i_1 : i_2, j_1 : j_2) = (a_{ij}) \quad i_1 \leq i \leq i_2,\ j_1 \leq j \leq j_2$ est la sous-matrice de A de taille $(i_2 - i_1 + 1) \times (j_2 - j_1 + 1)$ comprise entre les lignes i_1 et i_2 et les colonnes j_1 et j_2.
- $\boldsymbol{v}(i_1 : i_2)$ est le vecteur de taille $i_2 - i_1 + 1$ compris entre la i_1-ème et la i_2-ème composante de $\boldsymbol{v}$.

Ces notations sont utilisées, par exemple, dans Matlab.

A.3 Opérations sur les matrices

Soient $A = (a_{ij})$ et $B = (b_{ij})$ deux matrices dans $\mathbb{R}^{m \times n}$.
- $A = B$ si $a_{ij} = b_{ij}$ pour $i = 1, \dots, m$, $j = 1, \dots, n$;
- $A + B = C$ avec $c_{ij} = a_{ij} + b_{ij}$ pour $i = 1, \dots, m$, $j = 1, \dots, n$;
- élément neutre pour la somme matricielle : matrice nulle, notée 0, constituée de coefficients tous nuls ;
- le produit d'une matrice A de taille $m \times n$ par un scalaire $\lambda \in \mathbb{R}$: $\lambda A = C$ avec $c_{ij} = \lambda a_{ij}$ pour $i = 1, \dots, m$, $j = 1, \dots, n$;
- le produit d'une matrice A de taille $m \times p$ par une matrice B de taille $p \times n$ est la matrice C de taille $m \times n$ telle que

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

  pour $i = 1, \dots, m$, $j = 1, \dots, n$;
- le produit matriciel est non commutatif en général ; on dira que deux matrices commutent si $AB = BA$;
- élément neutre pour le produit matriciel : matrice *identité*, notée I, définie par $I = (\delta_{ij})$ où

$$\delta_{ij} = \begin{cases} 1, & \text{si } i = j, \\ 0, & \text{si } i \neq j. \end{cases}$$

Matrice diagonale : $D = (d_{ii}\delta_{ij}) = \mathrm{diag}(d_{11}, d_{22}, \dots, d_{nn})$.

Opérations élémentaires sur les lignes d'une matrice :
- multiplication de la $i$ème ligne d'une matrice par un scalaire α ; place ;
- échange de la $i$ème et de la $j$ème ligne d'une matrice ;
- somme de la $i$ème ligne d'une matrice avec α fois sa $j$ème ligne.

Inverse d'une matrice :

Une matrice $A \in \mathbb{R}^{n\times n}$ est appelée *matrice carrée d'ordre n.*

Définition A.6. *Une matrice carrée A d'ordre n est dite* inversible *ou* non singulière *s'il existe une matrice carrée B d'ordre n telle que* $AB = BA = I$. *On dit que B est la matrice inverse de A et on la note* A^{-1}. *Une matrice qui n'est pas inversible est dite* singulière.

Propriété A.1. *Une matrice carrée est inversible ssi ses vecteurs colonnes sont linéairement indépendants.*

Définition A.7. *On appelle transposée d'une matrice* $A \in \mathbb{R}^{m\times n}$ *la matrice* $n \times m$, *notée* A^T, *obtenue en permutant les lignes et les colonnes de A.*

Propriétés :
- $(A^T)^T = A$;
- $(A + B)^T = A^T + B^T$;
- $(AB)^T = B^T A^T$;
- $(\alpha A)^T = \alpha A^T$;
- si A est inversible : $(A^T)^{-1} = (A^{-1})^T \equiv A^{-t}$.

Définition A.8. *Une matrice* $A \in \mathbb{R}^{m\times n}$ *est dite*
- symétrique *si* $A = A^T$;
- antisymétrique *si* $A = -A^T$;

Définition A.9 (Matrice orthogonale)**.** *Une matrice* $A \in \mathbb{R}^{m\times n}$ *est dite* orthogonale *si*

$$A^T A = AA^T = I$$

c.-à-d. si $A^{-1} = A^T$.

A.4 Trace et déterminant d'une matrice

Soit A une matrice $n \times n$.

Définition A.10. *La trace de A est la somme de ces éléments diagonaux :* $tr(A) = \sum_{i=1}^{n} a_{ii}$.

Définition A.11. *Le déterminant de A est le scalaire défini par :*

$$\det(A) = \sum_{\boldsymbol{\sigma}\in P} signe(\boldsymbol{\sigma}) a_{1\sigma_1} a_{2\sigma_2} \dots a_{n\sigma_n},$$

où $P = \{\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_n)\}$ *est l'ensemble des n! multi-indices obtenus par permutation du multi-indice* $\boldsymbol{i} = (1, \dots, n)$ *et* $signe(\boldsymbol{\sigma})$ *vaut 1 (respectivement -1) si on effectue un nombre pair (respectivement impair) de transpositions pour obtenir* $\boldsymbol{\sigma}$ *à partir de* $\boldsymbol{i}$.

Appelons M_{ij} la sous-matrice carrée d'ordre $n-1$ de A, obtenue en supprimant la $i$ème ligne et la $j$ème colonne de A. Le déterminant $\det(M_{ij})$ est appelé le *mineur* de l'élément a_{ij} de A.

Le *$k$ème mineur principal* de A est le déterminant de la sous-matrice principale d'ordre k, $A_k = A(1:k, 1:k)$.

Le *cofacteur* de a_{ij}, noté A_{ij}, est défini par :

$$A_{ij} = (-1)^{i+j} \det(M_{ij}).$$

Théorème A.2. *Le déterminant de A est égal à la somme des produits obtenus en multipliant les éléments d'une ligne (resp. d'une colonne) quelconque par leurs cofacteurs respectifs.*

Propriétés : Pour des matrices carrées A et B d'ordre n :
- $\det(A) = \det(A^T)$;
- $\det(AB) = \det(A)\det(B)$;
- $\det(A^{-1}) = \frac{1}{\det(A)}$;
- $\det(\alpha A) = \alpha^n \det(A),\ \forall\ \alpha\ \in\ K$;
- si 2 lignes ou 2 colonnes de A coincident, alors $\det(A) = 0$;
- si on échange 2 lignes (ou 2 colonnes) de A, alors on change le signe du déterminant ;
- si A est diagonale, alors $\det(A) = \prod_{i=1}^{n} a_{ii}$;
- si A est orthogonale, alors A^{-1} existe, $A^{-1} = A^T$ et $\det(A) = \pm 1$.

Exercice A.1. *Montrer que le déterminant d'une matrice orthogonale est égal à* $\pm$ *1.*

A.5 Matrices triangulaires

Une matrice *triangulaire* d'ordre n est une matrice carrée de la forme

$$L = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \quad \text{ou} \quad U = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix}.$$

La matrice L est dite *triangulaire inférieure* tandis que U est dite *triangulaire supérieure.*

Propriétés :
- le déterminant d'une matrice triangulaire inférieure est le produit de ses éléments diagonaux ;
- l'inverse d'une matrice triangulaire inférieure est encore une matrice triangulaire inférieure ;
- le produit de deux matrices triangulaires inférieures est encore une matrice triangulaire inférieure ;
- le produit de deux matrices triangulaires inférieures dont les éléments diagonaux sont égaux à 1 est encore une matrice triangulaire inférieure dont les éléments diagonaux sont égaux à 1 ;

Ces propriétés restent vraies si on remplace *inférieure* par *supérieure.*

A.6 Valeurs propres et vecteurs propres

Définition A.12. *Soit A une matrice carrée d'ordre n. On dit que $\lambda \in \mathbb{C}$ est une* valeur propre *de A s'il existe une vecteur non nul $\boldsymbol{x} \in \mathbb{C}^n$ tel que $A\boldsymbol{x} = \lambda\boldsymbol{x}$. Le vecteur $\boldsymbol{x}$ est le* vecteur propre *associé à la valeur propre λ. On dit que $\boldsymbol{x}$ et $\boldsymbol{y}$ sont respectivement* vecteur propre à droite *et* vecteur propre à gauche *de A associés à la valeur propre λ, si $A\boldsymbol{x} = \lambda\boldsymbol{x}$ et $\bar{\boldsymbol{y}}^T A = \lambda\bar{\boldsymbol{y}}^T$.*

Définition A.13. *Le spectre $\sigma(A)$ de A est l'ensemble des valeurs propres de A.*

La valeur propre λ correspondant au vecteur propre $\boldsymbol{x}$ peut être déterminée en calculant le *quotient de Rayleigh*

$$\lambda = \frac{\bar{\boldsymbol{x}}^T A \boldsymbol{x}}{(\bar{\boldsymbol{x}}^T \boldsymbol{x})}.$$

Le nombre λ est solution de l'*équation caractéristique*

$$p_A(\lambda) \equiv \det(A - \lambda I) = 0,$$

où $p_A(\lambda)$ est le *polynôme caractéristique* de A. Ce polynôme étant de degré n par rapport à λ, on sait qu'il existe n valeurs propres (non nécessairement distinctes) ; notons ces valeurs propres $\lambda_1, \ldots, \lambda_n$.

Exercice A.2. *Calculer les valeurs propres et les vecteurs propres des matrices suivantes :*

$$A = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 2 & -1 \\ -1 & 1 & 4 \end{bmatrix} \quad et \quad B = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Propriétés :
- $\mathrm{tr}(A) = \sum_{i=1}^n \lambda_i$;
- $\det(A) = \prod_{i=1}^n \lambda_i$;
- $\sigma(A) = \sigma(A^T)$;
- si A est diagonale ou triangulaire, alors $\sigma(A) = \{a_{11}, a_{22}, \ldots, a_{nn}\}$.
- si A est est réelle, les valeurs propres de A sont soit des nombres réels, soit des nombres complexes deux à deux conjugués ;
- si les valeurs propres d'une matrice carrée sont distinctes, les vecteurs propres correspondants sont linéairement indépendants.

Exercice A.3. *Montrer qu'une matrice est singulière ssi elle possède une valeur propre nulle.*

Définition A.14 (Rayon spectral)**.** *Le plus grand des module des valeurs propres de A est appelé* rayon spectral *de A et est noté $\rho(A)$:*

$$\rho(A) = \max_{\lambda \in \sigma(A)} |\lambda|.$$

Exercice A.4. *Montrer que si λ est valeur propre de A, λ^m et $\frac{1}{\lambda}$ sont valeurs propres de A^m et A^{-1}, respectivement. En déduire que $\lambda_{max}(A^{-1}) = \frac{1}{\lambda_{min}(A)}$.*

Exercice A.5. *Montrer que les valeurs propres d'une matrice orthogonale sont de module 1.*

Théorème A.3. *Soit A une matrice carrée, alors*

$$\lim_{k \to \infty} A^k = 0 \Leftrightarrow \rho(A) < 1. \tag{A.1}$$

A.7 Matrices semblables

Définition A.15. *Deux matrices carrées A et B d'ordre n sont dites* semblables *s'il existe une matrice non singulière P d'ordre n telle que $B = P^{-1}AP$.*

Exercice A.6. *Montrer que deux matrices semblables possèdent le même spectre. Quel est la relation entre les vecteurs propres des deux matrices ?*

On a alors :
- $\mathrm{tr}(P^{-1}AP) = \mathrm{tr}(A)$;
- $\det(P^{-1}AP) = \det(A)$.

Propriété A.2. ***(décomposition de Schur)*** *Soit $A \in \mathbb{R}^{n\times n}$, il exite U orthogonale telle que*

$$U^{-1}AU = U^TAU = \begin{bmatrix} \lambda_1 & b_{12} & \dots & b_{1n} \\ 0 & \lambda_2 & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} = T,$$

où les λ_i sont les valeurs propres de A.

Conséquence : Toute matrice symétrique est *orthogonalement semblable* à une matrice diagonale réelle. Ainsi, quand A est symétrique, toute décomposition de Schur de A est diagonale. Dans ce cas, puisque $U^{-1}AU = \Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$ on a $AU = U\Lambda$, de sorte que les vecteurs colonnes de U sont les vecteurs propres de A.

A.8 Matrices diagonalisables

Définition A.16. *Une matrice* diagonalisable *est une matrice carrée semblable à une matrice diagonale.*

Propriété A.3. *Les valeurs propres d'une matrice diagonalisable sont les termes diagonaux de la matrice diagonale semblable.*

Propriété A.4. *Une matrice est diagonalisable ssi ses vecteurs propres sont linéairement indépendants.*

A.9 Décomposition en valeurs singulières

Propriété A.5. *Soit $A \in \mathbb{R}^{m\times n}$. Il existe deux matrices orthogonales réelles $U \in \mathbb{R}^{m\times m}$ et $V \in \mathbb{R}^{n\times n}$ telles que*

$$U^TAV \equiv \Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_p) \tag{A.2}$$

avec $p = min(m, n)$ et $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$. Cette relation est appelée décomposition en valeurs singulières *de A et les scalaires σ_i (ou $\sigma_i(A)$) sont appelés* valeurs singulières *de A.*

Exercice A.7. *Montrer que la $i$ème valeur singulière de A est donnée par*

$$\sigma_i(A) = \sqrt{\lambda_i(A^TA)}$$

où $\lambda_i(A^TA)$ est la $i$ème valeur propre de la matrice A^TA.

Exercice A.8. *Montrer que si A est une matrice carrée symétrique d'ordre n, alors ses valeurs singulières coincident avec les modules de ses valeurs propres :*

$$\sigma_i(A) = \mid \lambda_i(A) \mid .$$

A.10 Normes vectorielles

Définition A.17. *Soit V un espace vectoriel sur K ($K = \mathbb{R}$ ou $K = \mathbb{C}$). On dit qu'une application $||\cdot||$ de V dans $\mathbb{R}$ est une* norme *sur V si :*

1. $||\boldsymbol{v}|| \geq 0 \ \forall \ \boldsymbol{v} \in V$ *et* $||\boldsymbol{v}|| = 0$ *ssi* $\boldsymbol{v} = 0$;
2. $||\alpha \boldsymbol{v}|| = \mid \alpha \mid ||\boldsymbol{v}|| \ \forall \ \boldsymbol{v} \in V, \ \forall \ \alpha \in K$;
3. $||\boldsymbol{v} + \boldsymbol{w}|| \leq ||\boldsymbol{v}|| + ||\boldsymbol{w}||, \ \forall \ \boldsymbol{v}, \boldsymbol{w} \in V$ *(inégalité triangulaire).*

Définition A.18. *La* p-norme *est définie par*

$$||\boldsymbol{v}||_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{1/p}, \quad \textit{pour } 1 \leq p < \infty,$$

où les v_i sont les composantes de $\boldsymbol{v}$.

Définition A.19. *La* norme infinie *est définie par*

$$||\boldsymbol{v}||_\infty = \max_{1 \leq i \leq n} |v_i|.$$

Produit scalaire euclidien : $(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} x_i y_i$.

Pour toute matrice carrée A d'ordre n et pour tout $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, on a $(A\boldsymbol{x}, y) = (\boldsymbol{x}, A\boldsymbol{y})$.

A.11 Normes matricielles

Définition A.20. *Une* norme matricielle *est une application* $||\cdot|| \ : \ \mathbb{R}^{m \times n} \to \mathbb{R}$ *telle que :*

1. $||A|| \geq 0 \ \forall \ A \in \mathbb{R}^{m \times n}$ *et* $||A|| = 0$ *ssi* $A = 0$;
2. $||\alpha A|| = \mid \alpha \mid ||A|| \ \forall \ A \in \mathbb{R}^{m \times n}, \ \forall \ \alpha \in \mathbb{R}$;
3. $||A + B|| \leq ||A|| + ||B||, \ \forall \ A, B \in \mathbb{R}^{m \times n}$ *(inégalité triangulaire).*

Définition A.21. *La* p-norme matricielle *est définie par*

$$||\boldsymbol{A}||_p = \sup_{\boldsymbol{x} \neq 0} \frac{||A\boldsymbol{x}||_p}{||\boldsymbol{x}||_p}.$$

Définition A.22. *La* 1-norme matricielle *et la* norme infinie matricielle *sont définies par*

$$||\boldsymbol{A}||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|,$$

$$||A||_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|,$$

et sont appelées norme somme des colonnes *et* norme somme des lignes *respectivement.*

Théorème A.4. *Soit $\sigma_1(A)$ la plus grande valeur singulière d'une matrice A à composantes réelles. Alors,*

$$||A||_2 = \sqrt{\rho(A^T A)} = \sqrt{\rho(A A^T)} = \sigma_1(A).$$

En particulier, si A est symétrique, alors

$$||A||_2 = \rho(A),$$

tandis que si A est orthogonale, alors

$$||A||_2 = 1.$$

Définition A.23 (Norme matricielle consistante)**.** *On dit qu'une norme matricielle est* compatible *ou* consistante *avec une norme vectorielle si*

$$||A\boldsymbol{x}|| \leq ||A||\ ||\boldsymbol{x}||, \quad \forall\ \boldsymbol{x} \in \mathbb{R}^n.$$

Théorème A.5. *Si* $||\cdot||$ *est une norme matricielle consistante, alors*

$$\rho(A) \leq ||A||, \quad \forall\ A \in \mathbb{R}^{n\times n}. \tag{A.3}$$

Exercice A.9. *Démontrer ce théorème.*

Propriété : toutes les *p-norme* matricielles sont consistantes.

A.12 Matrices définies positives, matrices à diagonale dominante

Définition A.24. *Une matrice* $A \in \mathbb{R}^{n\times n}$ *est*
- définie positive *sur* $\mathbb{R}^n$ *si* $(A\boldsymbol{x}, \boldsymbol{x}) > 0,\ \forall\ \boldsymbol{x} \in \mathbb{R}^n,\ \boldsymbol{x} \neq \mathbf{0}$,
- semi-définie positive *sur* $\mathbb{R}^n$ *si* $(A\boldsymbol{x}, \boldsymbol{x}) \geq 0,\ \forall\ \boldsymbol{x} \in \mathbb{R}^n,\ \boldsymbol{x} \neq \mathbf{0}$.

Propriété : Tous les termes diagonaux d'une matrice définie positive sont strictement positifs.

Propriété A.6. *Soit* $A \in \mathbb{R}^{n\times n}$ *une matrice symétrique. Alors, A est définie positive ssi une des propriétés suivantes est satisfaite :*

1. $(A\boldsymbol{x}, \boldsymbol{x}) > 0\ \forall \boldsymbol{x} \neq \mathbf{0}$ *avec* $\boldsymbol{x} \in \mathbb{R}^n$ *;*
2. *les valeurs propres des sous-matrices principales de A sont toutes positives ;*
3. *les mineurs principaux de A sont tous positifs, c-à-d.* $\det(A(1:k, 1:k) > 0$, $k = 1, \ldots, n$ *;*
4. *il existe une matrice inversible H telle que* $A = HH^T$.

Définition A.25. *Si A est symétrique définie positive et si* $A^{1/2}$ *est l'unique matrice définie positive solution de l'équation* $X^2 = A$*, l'application* $||\cdot||_A$ *donnée par*

$$||\boldsymbol{x}||_A = ||A^{1/2}\boldsymbol{x}||_2 = (A\boldsymbol{x}, \boldsymbol{x})^{1/2}$$

définit une norme vectorielle, appelée norme de l'énergie *du vecteur* $\boldsymbol{x}$*. On associe à la norme de l'énergie le* produit scalaire de l'énergie *donné par* $(\boldsymbol{x}, \boldsymbol{y})_A = (A\boldsymbol{x}, \boldsymbol{y})$.

Définition A.26. *Une matrice* $A \in \mathbb{R}^{n\times n}$ *est dite* à diagonale dominante par lignes *ou simplement* à diagonale dominante *si*

$$|\ a_{ii}\ | \geq \sum_{j=1, j\neq i}^{n} |\ a_{ij}\ |,\ \textit{avec } i = 1, \ldots, n,$$

tandis que'elle est dite à diagonale dominante par colonnes *si*

$$|\ a_{ii}\ | \geq \sum_{j=1, j\neq i}^{n} |\ a_{ji}\ |,\ \textit{avec } i = 1, \ldots, n.$$

Si les inégalités ci-dessus sont strictes, A est dite à diagonale dominante stricte *(par lignes et par colonnes respectivement).*

A.13 Formes quadratiques

Une *forme quadratique* des variables $x_1, x_2, \ldots, x_n$ est un polynôme

$$q(x_1, x_2, \ldots, x_n) = \sum_{i<j} c_{ij} x_i x_j. \tag{A.4}$$

La forme quadratique est *diagonalisée* si

$$q(x_1, x_2, \ldots, x_n) = c_{11}x_1^2 + c_{22}x_2^2 + \ldots + c_{nn}x_n^2.$$

La forme quadratique (A.4) s'écrit de manière unique sous la forme matricielle

$$q(\boldsymbol{x}) = \boldsymbol{x}^T A \boldsymbol{x}$$

où $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ et $A = (a_{ij})$ est une matrice symétrique. Les éléments de A peuvent être obtenus à partir de la formule (A.4) en posant

$$a_{ii} = c_{ii} \text{ et } a_{ij} = a_{ji} = c_{ij}/2 \text{ pour } i \neq j.$$

La matrice symétrique A s'appelle la *représentation matricielle* de la forme quadratique q.

Toute matrice symétrique A définit une forme quadratique q.

De plus, une forme quadratique q est diagonalisée ssi la matrice symétrique correspondante A est diagonale.

A.14 Règle de Cramer pour la résolution de systèmes d'équations linéaires

Soit un système de n équations linéaires à n inconnues :

$$\left\{\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &=& b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &=& b_2 \\ \vdots & & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &=& b_n. \end{array}\right.$$

Ce système peut s'écrire sous la forme $A\boldsymbol{x} = \boldsymbol{b}$ où A est la matrice carrée des coefficients et $\boldsymbol{b}$ est le vecteur colonne des termes constants. Soit A_i la matrice obtenue à partir de A en remplaçant le $i$ème colonne de A par le vecteur colonne $\boldsymbol{b}$.

Théorème A.6. *Le système précedent admet une solution unique ssi* $\det(A) \neq 0$. *Dans ce cas, la solution unique est donnée par*

$$\begin{array}{rcl} x_1 &=& \dfrac{\det(A_1)}{\det(A)}, \\ x_2 &=& \dfrac{\det(A_2)}{\det(A)}, \\ \vdots & & \\ x_n &=& \dfrac{\det(A_n)}{\det(A)}. \end{array}$$

A.15 Relations matricielles utiles

Si x et a sont des vecteurs $(n \times 1)$ et C une matrice symétrique $(n \times n)$

$$\frac{d}{dx}(x^T C x) = 2Cx \tag{A.5}$$

$$\frac{d}{dx}(x^T a) = a \tag{A.6}$$

Annexe B

Exercices MATLAB

B.1 Introduction

1. – Démarrer MATLAB . Dans la fenêtre de commande taper `x=-1:0.1:1` Ensuite exécuter chacune des commandes suivantes :
```
sqrt(x)
sin(x)
x.^3
plot(x,cos(x.^2))
cos(x)
x.^2
plot(x,sin(x.^3))
```
   – Exécuter les commandes suivantes et expliquer les résultats :
```
x=[2 3 4 5]
y=-1:1:2
x.^y
x.*y
x./y
```

2. – Initialiser la matrice `A=[1 5 8;84 81 7;12 34 71]` et examiner le contenu de `A(1,1), A(2,1), A(1,2), A(3,3), A(1:2,:), A(:,1), A(3,:), A(:,2:3)`.
   – Que produisent les commandes MATLAB suivantes ?
```
x=1:1:10
z=rand(10)
y=[z;x]
c=rand(4)
e=[c eye(size(c)); eye(size(c)) ones(size(c))]
d=sqrt(c)
t1=d*d
t2=d.*d
```
3. Initialiser une matrice $A_{4\times4}$. Étant donné que la fonction `sum(x)` donnera la somme des éléments du vecteur `x`, utiliser la fonction `sum` pour trouver les sommes de la première ligne et de la deuxième colonne de la matrice.
4. On a trois matrices A, B et C :

$$\mathbf{A} = \begin{bmatrix} 2 & -3 \\ 4 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 44 & 101 \\ -18 & -41 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 2 \\ -2 & -5 \end{bmatrix}$$

   Au moyen de MATLAB , calculer $C^{-1} \times A \times C$. Que peut on en conclure ?

5. Écrire un script simple qui prend en entrée deux matrices carrées A et B : ensuite, les additionner, les soustraire et les multiplier. Commenter le script et utiliser `disp` pour afficher des titres correspondants.

6. Écrire un script simple pour dessiner les deux fonctions $y_1 = x^2 cosx$ et $y_2 = x^2 sinx$ sur le même graphe. Insérer des commentaires dans les scripts et prendre `x=-2:0.1:2`.

7. Une équation itérative pour résoudre l'équation $x^2 - x - 1 = 0$ est donnée par

   $x_{r+1} = 1 + (1/x_r)$ pour r = 0,1,2,...

   Étant donné que x_0 vaut 2, écrire un script MATLAB pour résoudre l'équation. Une précision suffisante est obtenue quand

   $$abs(x_{r+1} - x_r) < 0.0005.$$

   Inclure une vérification dans la réponse.

8. Étant donné une matrice $A_{4\times5}$, écrire un script pour trouver les sommes de chacune des colonnes en utilisant

   (a) La commande *for*

   (b) La fonction *sum*

9. Étant donné un vecteur X de n éléments, écrire un script MATLAB pour former le produit

   $$p_k = x_1 x_2 \dots x_{k-1} x_{k+1} \dots x_n$$

   pour $k = 1, 2, \dots, n$.. Autrement dit, p_k contiendra les produits de tous les éléments du vecteur excepté le $k$ème. Tester le script avec des valeurs extrêmes de x et n.

10. Écrire un script MATLAB pour générer une matrice dont les coefficients le long de la diagonale principale valent d et dont les coefficients au-dessus et au-dessous de la diagonale principale valent c, le reste de la matrice étant constitué de zéros. Le script devrait donner des indications claires sur le format des données (c et d) et afficher le résultat avec une en-tête convenable.

11. Écrire une fonction MATLAB pour résoudre l'équation suivante :

    $$ax^2 + bx + c = 0$$

    La fonction utilisera trois paramètres en entrée a, b et c et renverra comme résultat les valeurs des deux racines.

12. Il peut être prouvé que la série $(I - A)^{-1} = I + A + A^2 + A^3 + \dots$, où A est une matrice $n \times n$, converge si les valeurs propres de A sont plus petites que l'unité. La matrice $n \times n$ suivante satisfait cette condition si $a + 2b < 1$:

    $$\begin{bmatrix} a & b & 0 & \dots & 0 & 0 & 0 \\ b & a & b & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & b & a & b \\ 0 & 0 & 0 & \dots & 0 & b & a \end{bmatrix}$$

    Expérimenter avec cette matrice pour différentes valeurs de n, a et b pour illustrer que les séries convergent sous les conditions annoncées.

B.2 Résolution de systèmes linéaires

1. Résoudre le système $A\boldsymbol{x} = \boldsymbol{b}$, avec

$$A = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix} \quad \text{et} \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}.$$

   en utilisant d'abord l'opérateur \ de MATLAB.

2. Écrire une fonction backsub(A,B) en MATLAB qui résout un système d'équations triangulaire supérieur $Ax = b$ par la méthode de substitution directe. Tout d'abord, calculer $x_N = b_N/A_{NN}$ et ensuite, utiliser la règle

$$x_k = \frac{b_k - \sum_{j=k+1}^{N} a_{kj}x_j}{a_{kk}} \quad \text{pour} \quad k = N-1, N-2, ..., 1.$$

3. Trouver des matrices P, L et U telles que L est triangulaire inférieure, U triangulaire supérieure et $PA = LU$ en utilisant MATLAB. A est la matrice des coefficients du système suivant :

$$\begin{cases} 2x_1 - 3x_2 + 2x_3 + 5x_4 = 3 \\ x_1 - x_2 + x_3 + 2x_4 = 1 \\ 3x_1 + 2x_2 + 2x_3 + x_4 = 0 \\ x_1 + x_2 - 3x_3 - x_4 = 0 \end{cases}$$

4. Ecrire une fonction Doolittle(A) en MATLAB qui renvoie les matrices L et U obtenues à partir de la matrice A par la méthode de Doolittle. Vérifier le bon fonctionnement de votre fonction avec la matrice de l'exercice précédent.

   Pour rappel, la méthode de Doolittle : on pose $l_{kk} = 1$. Ensuite, pour pour $k = 1, \ldots, n$ on calcule d'abord la k-ième ligne de U, puis la k-ième colonne de L, selon les formules :

$$u_{kj} = a_{kj} - \sum_{r=1}^{k-1} l_{kr}u_{rj}, j = k, \ldots, n,$$

$$l_{ik} = \frac{1}{u_{kk}}\Big(a_{ik} - \sum_{r=1}^{k-1} l_{ir}u_{rk}\Big), i = k+1, \ldots, n.$$

5. Pour une solution approchée x^*

$$\begin{aligned} e &= x - x^* \\ r &= b - Ax^* = b - A(x - e) = Ae \end{aligned}$$

   Posant $x^* = x^{(0)}$, on peut effectuer un raffinement itératif

$$\begin{cases} r^{(i)} = b - Ax^{(i)} \\ Ae = r^{(i)} \\ x^{(i+1)} = x^{(i)} + e \end{cases} \quad \text{jusqu'à} \quad \frac{\| e \|}{\| x^{(i)} \|} < tol$$

   Si le système n'est pas trop mal conditionné, le raffinement itératif converge très rapidement.

   Pour le système suivant :

$$\begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 23 \\ 32 \\ 33 \\ 31 \end{bmatrix} \quad \text{la solution est} \quad x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

```
function X=jacobi(A,B,P,delta,max1)
% Input -A is an N x N nonsingular matrix
%       -B is an N x 1 matrix
%       -P is an N x 1 matrix; the initial guess
%       -delta is the tolerance for P
%       -max1 is the maximum number of iterations
% Output-X is an N x 1 matrix : the jacobi approximation to
%        the solutions of AX= B

N= length(B);

for k=1:max1
    for j=1:N
        X(j)=(B(j)-A(j,[1:j-1,j+1:N])*P([1:j-1,j+1:N]))/A(j,j);
    end
    err=abs(norm(X'-P));
    relerr=err/(norm(X)+eps);
    P=X';
 if (err<delta)|(relerr<delta)
         break
    end
end
X=X';
```

FIGURE B.1 – Code MATLAB implémetant la méthode de Jacobi

Calculer à partir de

$$x^* = x^{(0)} = \begin{bmatrix} 0.9881 \\ 1.0073 \\ 1.0028 \\ 0.9983 \end{bmatrix}$$

la valeur de $x^{(1)}$ en utilisant MATLAB.

6. Pour le système

$$A = \begin{bmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{bmatrix} \quad \text{et} \quad \boldsymbol{b} = \begin{bmatrix} 0.8642 \\ 0.1440 \end{bmatrix} \quad \text{et la solution approchée} \quad \boldsymbol{x^*} = \begin{bmatrix} 0.9911 \\ 0.4870 \end{bmatrix}$$

   – Comparer x^* au résultat exact du système
   – Calculer le résidu $r = b - Ax^*$. Que peut-on en conclure à propos du résidu ?

7. Considérons le système $A\boldsymbol{x} = \boldsymbol{b}$ où

$$A = \begin{bmatrix} 10 & -2 & 1 \\ -2 & 10 & -2 \\ -2 & -5 & 10 \end{bmatrix} \quad \text{et} \quad \boldsymbol{b} = \begin{bmatrix} 9 \\ 12 \\ 18 \end{bmatrix}$$

   dont la solution exacte est $\boldsymbol{x} = (1, 2, 3)^t$.
   En utilisant les scripts de la figure B.1 et B.2 :

   (a) Déterminer une solution approchée du système par la méthode de Jacobi ; calculer trois itérations en partant du vecteur initial $\boldsymbol{x}^{(0)} = (0, 0, 0)^t$.

   (b) Idem mais avec la méthode de Gauss-Seidel.

   Que peut-on conclure ?

```
function X=gseid(A,B,P,delta,max1)
% Input -A is an N x N nonsingular matrix
%       -B is an N x 1 matrix
%       -P is an N x 1 matrix; the initial guess
%       -delta is the tolerance for P
%       -max1 is the maximum number of iterations
% Output-X is an N x 1 matrix : the jacobi approximation to
%        the solutions of AX= B

N= length(B);

for k=1:max1
    for j=1:N
        if j==1
           X(1)=(B(1)-A(1,2:N)*P(2:N))/A(1,1);
        elseif j==N
               X(N)=(B(N)-A(N,1:N-1)*(X(1:N-1))')/A(N,N);
        else
             % X contains th kth approximations and P th (k-1)st
             X(j)=(B(j)-A(j,1:j-1)*X(1:j-1)-A(j,j+1:N)*P(j+1:N))/A(j,j);
     end
end
    err=abs(norm(X'-P));
    relerr=err/(norm(X)+eps);
    P=X';
 if (err<delta)|(relerr<delta)
         break
    end
end
X=X';
```

FIGURE B.2 – Code MATLAB implémetant la méthode de Gauss-Seidel

8. En utilisant le script MATLAB implantant la méthode de Jacobi qui vous a été fourni, résoudre, avec une précision de 0.000005, le système d'équations $Ax = b$ où les éléments de A sont

$$\begin{array}{rcl} a_{ii} & = & -4 \quad \text{et} \\ a_{ij} & = & 2 \quad \text{si} \quad |i-j| = 1 \\ & = & 0 \quad \text{si} \quad [i-j| \geq 2 \quad \text{où} \quad i,j = 1,2,\ldots,10 \end{array}$$

et

$$b^T = [2\ 3\ 4\ \ldots\ 11]$$

Vérifier votre résultat en utilisant l'opérateur \ de MATLAB.

9. Le fait qu'une matrice A soit à diagonale dominante stricte, c.-à-d. que

$$|a_{kk}| > \sum_{\substack{j=1 \\ j \neq k}}^{N} |a_{kj}|$$

pour $k = 1, 2, \ldots, N$. est une condition suffisante mais pas nécessaire pour déterminer si la méthode de Jacobi va converger. A partir de différents vecteurs initiaux P_0, calculer des solutions approchées pour le système suivant :

$$\left\{ \begin{array}{rcrcrcl} x & & & + & z & = & 2 \\ -x & + & y & & & = & 0 \\ x & + & 2y & - & 3z & = & 0 \end{array} \right.$$

en utilisant d'abord la méthode de Jacobi, ensuite celle de Gauss-Seidel. Conseil : augmenter progressivement le nombre d'itérations. Que constate-t-on ?

B.3 Résolution d'équations non linéaires

B.3.1 Méthode de la sécante

En utilisant la fonction fournie en annexe(secant.m), qui approxime la solution d'une équation non linéaire par la méthode de la sécante, écrire une fonction qui dessine les différentes étapes de la méthode de la sécante. Le but est que le script dessine la fonction dont on cherche une racine sur l'intervalle et les différentes droites de pente

$$q_k = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}$$

passant par les points

$$(x^{(k)}, f(x^{(k)})$$

Essayer ce script sur la fonction $x^2 - 1$ sur l'intervalle [0, 2] avec 3 itérations de la méthode.

B.4 Manipulation de fonctions en MATLAB

Ecrire une fonction `sinxdiva(n,a,b,pas)` affichant sur une même figure les n fonctions $sin(x/i), i = 1, \ldots, n$ sur l'intervalle $[a, b]$. Utiliser pour l'affichage des points espacés de `pas`.

B.5 Interpolation

Ecrire une fonction qui calcule la matrice des différences divisées étant donnés deux vecteurs x et y représentant des couples d'abscisse et d'ordonnée. Le tester sur les séries de points suivantes :

1.

x	y
0	1
0.2	1.5646
0.4	1.9320
0.8	1.6755
1.2	0.5575
1.6	0.0038
2.0	0.7206

2.

x	y
0	1
0.00004	1.0001
0.00008	1.0002
0.00012	1.0004
0.00016	1.0005
0.0002	1.0006

Sachant que la fonction à interpoler est $f(x) = 1 + sin(3x)$, que peut-on penser des résultats obtenus ?

B.5.1 Interpolation de Lagrange en 3 dimensions

Soit Ω le produit tensoriel de 2 intervalles *i.e.*, $\Omega = [a, b] \times [c, d]$. Dans ce cas, en introduisant les noeuds $a = x_1 < x_2 < \cdots < x_n = b$ et $c = y_1 < y_2 < \cdots < y_m = d$, le polynôme d'interpolation $\Pi_{n,m} f$ s'écrit

$$\Pi_{n,m} f(u, v) = \sum_{i=1}^{n} \sum_{j=1}^{m} f(x_i, y_j) l_i(u) l_j(v)$$

où les polynômes $l_i(u)$ et $l_j(v)$ sont les polynômes caractéristiques de Lagrange unidimensionnels en u et v.

$$l_i \in \mathbb{P}_n \ : l_i(u) = \prod_{j=1, j\neq i}^{n} \frac{u - x_j}{x_i - x_j}, i = 1, \cdots, n$$

Ecrire une fonction, qui étant donnés

- une fonction f,
- deux vecteurs colonnes X et Y respectivement de taille n et m, les points utilisés pour calculer le polynôme d'interpolation,
- deux vecteurs U et V respectivement de taille o et p,

calcule les valeurs de $\Pi_{n,m} f(u_i, v_j)$ pour $i = 1, \cdots, o$ $j = 1, \cdots, p$ et les place dans une matrice Z. Afficher le résultat au moyen de la commande `mesh(U,V,Z)`. Pour tester ce script, utiliser la fonction $z = x^2 + y^2$ sur l'intervalle $[-5, 5] \times [-5, 5]$, les vecteurs

$$X = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, Y = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

et $U = V = -5; 0.5; 5$.(voir figure B.3)

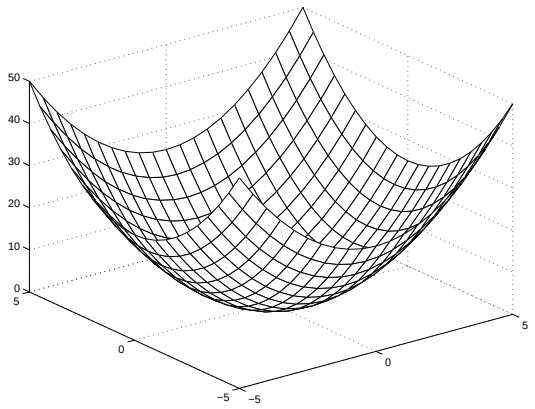


FIGURE B.3 – Le résultat attendu à l'exercice B.5.1

Bibliographie

[1] C. M. Bishop. *Neural Networks for Statistical Pattern Recognition*. Oxford University Press, Oxford, UK, 1994.

[2] Jean Paul Doignon. *Cours de Mathématiques (syllabus)*. Université Libre de Bruxelles.

[3] J. Hadamard. *Lectures on the Cauchy Problem in Linear Partial Differential Equations*. Yale University Press, 1923.

[4] D. Harrison and D.L. Rubinfeld. Hedonic prices and the demand for clean air. *J. Environ. Economics and Management*, 5 :81–102, 1978.

[5] A.J. Lotka. *Elements of physical biology*. Baltimore : Williams & Wilkins Co., 1925.

[6] A. Quarteroni, R. Sacco, and F. Saleri. *Méthodes numériques pour le calcul scientifique*. Springer, 2000.

[7] L. N. Trefethen. The definition of numerical analysis. *SIAM News*, 1992. disponible sur http ://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/.

[8] V. Volterra. *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi.*, volume 2 of *VI*. Mem. R. Accad. Naz. dei Lincei., 1926.