# Méthode d'analyse numérique.

Pascal Viot

# Méthodes d'analyse numérique
# Cours du DEA "Modélisation Dynamique et Statistique des Systèmes Complexes"

Pascal Viot
Laboratoire de Physique Théorique des Liquides, Boîte 121,
4, Place Jussieu, 75252 Paris Cedex 05
Email : viot@lptl.jussieu.fr

17 janvier 2003

Ce cours est une introduction aux méthodes d'analyse numérique très largement utilisées en physique afin de résoudre les équations algébriques ou différentielles que l'on rencontre dans la modélisation de phénomènes physiques, chimiques ou biologiques.

Ce domaine particulièrement vaste nécessite simultanément des connaissances mathématiques, informatiques et physiques. De larges classes de problèmes numériques sont abordées dans ces notes et montrent la nécessité de bien caractériser les propriétés mathématiques du problème considéré afin de choisir la méthode numérique la mieux adaptée pour le traitement numérique.

# Chapitre 1

# Intégration et sommes discrètes

**Contenu**



## 1.1 Introduction

Les sciences physiques se fixent l'objectif de prédire les phénomènes à partir de la connaissance d'un nombre fini de grandeurs microscopiques intervenant dans la modélisation. Celle-ci doit, dans un premier temps, décrire les faits expérimentaux observés et, dans un second temps, permettre d'en prédire de nouveaux. La description des phénomènes peut être soit microscopique, soit phénoménologique.

La modélisation se divise naturellement en trois étapes : une première étape consiste à déterminer les paramètres microscopiques essentiels qui interviennent pour décrire le phénomène puis à choisir le modèle adapté pour décrire le phénomène. La second étape consiste à établir les équations (très souvent différentielles) qui décrivent le phénomène. La troisième étape consiste à résoudre les équations précédemment établies afin de donner des réponses quantitatives. Ces réponses permettent de valider ou d'invalider le modèle, soit en comparant les prédictions avec l'expérience, soit en analysant la cohérence interne du modèle à travers ses prédictions.

Dans la situation la plus optimiste où il a été possible de réaliser les trois étapes précédentes, l'obtention d'un résultat numérique nécessite au moins le calcul d'une intégrale simple. Pour réaliser ce 'simple' travail, il existe bien souvent plusieurs méthodes et la tâche principale consiste à sélectionner celle

qui est la mieux adaptée pour satisfaire l'objectif. Ce principe est très général et s'applique à l'ensemble des problèmes numériques que nous allons aborder tout au long de ce cours. Ce cours a donc pour objectifs de présenter quelques algorithmes associés à chacun des problèmes rencontrés. Il existe des logiciels gratuits ou payants qui utilisent les méthodes numériques que nous allons rencontrer dans ce cours ; la connaissance de ces méthodes est donc un préalable pour choisir la manière de résoudre un problème numérique donné le plus efficacement possible.

Le cas typique d'une intégrale définie à évaluer est

$$I = \int_a^b f(x)dx. \tag{1.1}$$

Comme l'évaluation de la fonction $f$ pour une infinité de points est impossible, l'intégration numérique consiste à remplacer l'intégrale Eq. (1.1) par une somme discrète sur un nombre fini de points.

$$I_N = \sum_{i=1}^{N} a_i f(x_i) \tag{1.2}$$

où $a_i$ et $x_i$ sont des variables que nous allons préciser dans la suite. Pour que l'évaluation numérique soit correcte, il est nécessaire d'imposer que toute méthode vérifie que

$$\lim_{N\to\infty} I_N = I \tag{1.3}$$

Au delà de la vérification de ce critère Eq. (1.3), la qualité d'une méthode sera évaluée par la manière dont la convergence vers le résultat exact s'effectue. Dans la suite nous allons considérer des fonctions de classe $C_1$[1] (continûment dérivable) sur le support $[a, b]$[2]. La dernière restriction imposée à la fonction est que

$$|f'(x)| < K \quad \forall x \in [a, b] \tag{1.4}$$

où $K$ est une constante finie. Cela revient à supposer que la dérivée de la fonction $f$ n'est jamais singulière sur le support $[a, b]$.

## 1.2 Les méthodes de Côtes

Mathématicien anglais, contemporain et collaborateur de Newton, Roger Côtes s'est intéressé à des méthodes de calculs numériques et exacts pour l'intégration et explique que les méthodes suivantes portent son nom.

[1] On peut calculer l'intégrale d'une fonction plus générale à condition que cette fonction ait la même mesure ( mesure définie au sens de Lebesgue) qu'une fonction de classe $C_1$, c'est-à-dire que les fonctions ne diffèrent que sur un ensemble de mesure nulle. Comme exemple simple d'ensemble de mesure nulle, citons les ensembles dénombrables.

[2] Si la fonction est continûment dérivable par morceaux sur un nombre fini d'intervalles sur l'intervalle $[a, b]$, on peut se ramener au cas précédent sur chaque intervalle, et donc évaluer l'intégrale sur l'ensemble de l'intervalle $[a, b]$.

Les méthodes les plus simples que l'on peut utiliser pour calculer une intégrale simple sont celles où les abscisses sont choisies de manière régulièrement espacées. Si on a $N+1$ abscisses, on repère celles-ci simplement par la relation

$$x_i = x_0 + ih \tag{1.5}$$

avec $x_0 = a$, $x_N = b$ et $h$ est appelé le pas de l'intégration. Pour simplifier les notations, on pose

$$f_i = f(x_i) \tag{1.6}$$

### 1.2.1 Trapèze

La méthode des trapèzes consiste à approximer la fonction entre deux abscisses successives par une droite, ce qui donne.

$$\int_{x_i}^{x_{i+1}} f(x)dx = \frac{h}{2}(f_i + f_{i+1}) + \mathcal{O}(h^3 f''). \tag{1.7}$$

Le terme d'erreur indique la qualité de l'évaluation de l'intégration et dépend de manière cubique du pas d'intégration ; $f''$ se réfère à un point situé à l'intérieur de l'intervalle. Pour que cette méthode converge rapidement il est nécessaire de choisir un pas $h$ inférieur à $f''$. A noter que cette formule devient exacte quand la fonction est un polynôme de degré 1 sur l'intervalle $[x_1, x_2]$.

Sur l'intervalle $[a, b]$, on a

$$\int_a^b f(x) = h\sum_{i=1}^{N-1} f_i + \frac{h}{2}(f_0 + f_N) + \mathcal{O}\left(\frac{(b-a)^3 f''}{N^2}\right) \tag{1.8}$$

où on a utilisé que $h = (b-a)/N$

### 1.2.2 Simpson

La méthode de Simpson consiste à remplacer la fonction par un polynôme de degré 2 sur un intervalle constitué de trois abscisses consécutives

$$\int_{x_i}^{x_{i+2}} f(x)dx = h\left(\frac{1}{3}f_i + \frac{4}{3}f_{i+1} + \frac{1}{3}f_{i+2}\right) + \mathcal{O}(h^5 f^{(4)}). \tag{1.9}$$

Il se trouve que cette formule est exacte jusqu'à des polynômes de degré 3 ce qui implique que l'erreur dépende de $h$ à la puissance 5.

On peut aussi déterminer la formule à 4 points qui est aussi exacte pour les polynômes de degré 3. Cette formule s'appelle aussi Simpson 3/8 à cause des coefficients du développement

$$\int_{x_i}^{x_{i+3}} f(x)dx = h\left(\frac{3}{8}f_i + \frac{9}{8}f_{i+1} + \frac{9}{8}f_{i+2} + \frac{3}{8}f_{i+3}\right) + \mathcal{O}(h^5 f^{(4)}). \tag{1.10}$$

Sur un intervalle complet, en choisissant $N$ pair, la méthode de Simpson donne une estimation de l´intégrale sur l´intervalle $[a, b]$.

$$\int_a^b f(x) = \frac{h}{3}\left[f_0 + f_N + 2\sum_{i=1}^{N/2-1}(2f_{2i-1} + f_{2i})\right] + \mathcal{O}\left(\frac{1}{N^4}\right) \tag{1.11}$$

La méthode de Simpson est donc de deux ordres de grandeur plus efficace que la méthode des trapèzes. Mais il est possible de mieux utiliser la méthode des trapèzes. Sachant que cette dernière méthode converge en $1/N^2$, on peut évaluer l'intégrale deux fois sur le même intervalle par la méthode des trapèzes ; la première fois avec $N$ points et la seconde avec $2N$ points, pui en combinant les deux résultats de la manière suivante

$$S = \frac{4}{3}S_{2N} - \frac{1}{3}S_N \tag{1.12}$$

Sachant que le développement asymptotique de la méthode des trapèzes est une fonction paire de $1/N^2$, on en déduit que la formule (1.12) donne une estimation de l'intégrale en $1/N^4$, et ce résultat redonne une évaluation analogue à la méthode de Simpson.

## 1.3 Méthode de Romberg

L'idée de la méthode de Romberg s'inspire directement de la remarque faite au paragraphe précédent. Si on calcule successivement par la méthode des trapèzes les intégrales avec un nombres de points $N/2^k, N/2^{k-1}, \ldots, N$, on peut rapidement avoir une estimation de l'intégrale avec une erreur en $\mathcal{O}(1/N^{2k})$ où $k$ est le nombre de fois que l'on a calculé l'intégrale. La formule itérative utilisée est la suivante

$$S_{k+1}(h) = S_k(h) + \frac{(S_k(h) - S_k(2h))}{(4^k - 1)} \tag{1.13}$$

Le tableau 1.1 résume la procédure récursive employée dans la méthode de Romberg.

| Pas | Trapèzes | Simpson | Boole | troisième amélioration |
|---|---|---|---|---|
| $h$ | $S_1(h)$ | $S_2(h)$ | $S_3(h)$ | $S_4(h)$ |
| $2h$ | $S_1(2h)$ | $S_2(2h)$ | $S_3(2h)$ | |
| $4h$ | $S_1(4h)$ | $S_2(4h)$ | | |
| $8h$ | $S_1(8h)$ | | | |

TAB. 1.1 – Table de Romberg

## 1.4 Méthodes de Gauss

Dans les méthodes précédentes, nous avons vu qu'en changeant les coefficients de pondération des valeurs de la fonction à intégrer aux abscisses régulièrement espacées, on pouvait grandement améliorer la convergence de la méthode. Les méthodes de Gauss ajoutent aux méthodes précédentes de pouvoir utiliser des abscisses non régulièrement espacées.

Soit $W(x)$ une fonction strictement positive sur l´intervalle $[a, b]$, appelée fonction de poids, on choisit une suite de points $x_i$ telle que l'intégrale soit

approchée par une somme discrète de la forme

$$\int_a^b W(x)f(x)dx = \sum_{i=1}^{N} w_i f_i \tag{1.14}$$

Quand les abscisses sont régulièrement espacées, les coefficients inconnus sont les poids $w_i$ ; cela implique que pour $N$ points d´intégration, on peut obtenir une évaluation exacte de l´intégrale jusqu'à un polynôme de degré $N-1$ si $N$ est pair (trapèzes) et $N$ si $N$ est impair (Simpson). Les méthodes de Gauss utilise le fait si les abscisses et les poids sont des inconnues à déterminer ; la formule d´intégration de Gauss à $N$ points devient exacte jusqu'à des polynômes de degré $2N-1$, ce qui augmente la précision de l´évaluation sans qu'il soit nécessaire d´augmenter le nombre de points à calculer.

Pour déterminer ces $2N$ paramètres, on s´appuie sur la construction de polynômes orthogonaux. Le principe de cette construction remonte à Gauss et Jacobi et a été largement développé par Christoffel. Soit un intervalle $(a, b)$, on introduit le produit scalaire de deux fonctions $f$ et $g$ avec la fonction de poids $W$ par :

$$< f|g > \equiv \int_a^b W(x)f(x)g(x)dx \tag{1.15}$$

Les fonctions sont dites orthogonales si leur produit scalaire est nul. La fonction est normalisée quand $< f|f >= 1$. Un ensemble orthonormé de fonctions est un ensemble de fonctions toutes normalisées et orthogonales deux à deux.

On peut construire de manière systématique une suite de polynômes telle que le coefficient du monôme de degré le plus élevé soit égal à un et telle qu'ils soient tous orthogonaux.

La relation de récurrence est la suivante.

$$p_{-1}(x) = 0 \tag{1.16}$$

$$p_0(x) = 1 \tag{1.17}$$

$$p_{i+1}(x) = (x - a_i)p_i(x) - b_i p_{i-1}(x) \tag{1.18}$$

avec les coefficients $a_i$ et $b_i$ déterminés de la manière suivante

$$a_i = \frac{< xp_i|p_i >}{< p_i|p_i >} \tag{1.19}$$

$$b_i = \frac{< xp_i|p_{i-1} >}{< p_{i-1}|p_{i-1} >} \tag{1.20}$$

$$\tag{1.21}$$

Si l'on divise chaque polynôme par $< p_i|p_i >^{1/2}$, on obtient alors des polynômes normalisés.

Une propriété fondamentale de ces polynômes ainsi construits est la suivante : Le polynôme $p_i$ a exactement $j$ racines distinctes placées sur l'intervalle $[a, b]$. Chaque racine du polynôme $p_i$ se trouve entre deux racines consécutives du polynôme $p_{i+1}$.

Les $N$ racines du polynôme $p_N$ sont choisies comme abscisses dans l'évaluation de l'intégrale de Gauss et les poids $w_i$ sont calculés à partir de la formule

$$w_i = \frac{< p_{N-1}|\frac{1}{x-x_i}|p_{N-1} >}{p_{N-1}(x_i)p'_N(x_i)} \tag{1.22}$$

$$= \frac{d_N}{d_{N-1}} \frac{< p_{N-1}|p_{N-1} >}{p_{N-1}(x_i)p'_N(x_i)} \tag{1.23}$$

où le symbole prime désigne la dérivée du polynôme et $d_n$ le coefficient du monôme le plus élevé du polynôme $p_N$[3].

Avec ce choix, on peut montrer en utilisant la relation de récurrence, Eq. (1.18), que $< p_i|p_1 >= 0$.

A titre d'exemples voici les fonctions de poids classiques que l'on utilise pour calculer une intégrale par la méthode de Gauss

- *Gauss-Legendre*
  La fonction de poids est $W = 1$ sur l'intervalle $[-1, 1]$. La relation de récurrence pour les polynômes de Legendre est

$$(i+1)P_{i+1} = (2i+1)xP_i - iP_{i-1} \tag{1.24}$$

- *Gauss-Hermite*
  La fonction de poids est $W = \exp(-x^2)$ sur la droite réelle. La relation de récurrence pour les polynômes d'Hermite est

$$H_{i+1} = 2xT_i - 2iT_{i-1} \tag{1.25}$$

- *Gauss-Laguerre*
  La fonction de poids est $W = x^\alpha e^{-x}$ sur l'intervalle $[0, +\infty[$. La relation de récurrence pour les polynômes de Laguerre est

$$(i+1)L^\alpha_{i+1} = (-x + 2i + \alpha + 1)L^\alpha_i - (i+\alpha)L^\alpha_{i-1} \tag{1.26}$$

- *Gauss-Jacobi*
  La fonction de poids est $W = (1-x)^\alpha(1+x)^\beta$ sur l'intervalle $]-1, 1[$. La relation de récurrence pour les polynômes de Jacobi est

$$c_i P^{(\alpha,\beta)}_{i+1} = (d_i + e_i x)P^{(\alpha,\beta)}_i - f_i P^{(\alpha,\beta)}_{i-1} \tag{1.27}$$

  où les coefficients $c_i$, $d_i$, $e_i$ et $f_i$ sont donnés par les relations

$$c_i = 2(i+1)(i+\alpha+\beta+1)(2i+\alpha+\beta) \tag{1.28}$$

$$d_i = (2i+\alpha+\beta+1)(\alpha^2-\beta^2) \tag{1.29}$$

$$e_i = (2i+\alpha+\beta)(2i+\alpha+\beta+1)(2i+\alpha+\beta+2) \tag{1.30}$$

$$f_i = 2(i+\alpha)(i+\beta)(2i+\alpha+\beta+2) \tag{1.31}$$

On peut utiliser des fonctions de poids qui sont intégrables sur l´intervalle, sans être nécessairement bornées.

[3] Attention, la formule 4.5.9 de la référence[1] est incorrecte et doit être remlacée par l'équation (1.22)

– *Gauss-Chebyshev*[4] La fonction de poids est $W = (1 - x^2)^{-1/2}$ sur l'intervalle $[-1, 1]$. La relation de récurrence pour les polynômes de Chebyshev.

$$T_{i+1} = 2xT_i - T_{i-1} \tag{1.32}$$

## 1.5 Intégrales multiples

Le problème de l'évaluation des intégrales multiples est étroitement lié à la difficulté de l'évaluation numérique d'un très grand nombre de points pour la fonction considérée. Par exemple dans un espace à trois dimensions, si on utilise 30 points dans chacune des directions, il est nécessaire de calculer la fonction pour $30^3$ points. La situation s'aggrave très rapidement quand la dimension de l'espace augmente ! Le calcul des intégrales multiples est néanmoins possible dans un certain nombre de cas. Si la fonction possède une symétrie importante, par exemple la symétrie sphérique dans un espace de dimension $d$, on peut se ramener de manière analytique à une intégrale à une dimension (Voir appendice A). De manière générale, en utilisant une symétrie de la fonction, on peut ramener le calcul de l'intégrale de dimension $n$ à celui d´une intégrale de dimension $n' << n$ où les méthodes précédentes peuvent encore s'appliquer.

Dans le cas où il n'existe pas de symétrie, la méthode la plus efficace est la méthode dite de Monte Carlo dont le principe est décrit dans le cours *Simulation numérique en Physique Statistique.*(http ://pascal.viot.com)

[4]Pafnuty Lvovich Chebyshev (1821-1894) a un nom dont l´orthographe varie un peu selon les langues, puisqu´il s'agit d´une traduction phonétique. En Français, son nom est généralement orthographié Tchebytchev.

# Chapitre 2

# Fonctions spéciales et évaluation de fonctions

**Contenu**



## 2.1 Introduction

Les fonctions spéciales sont définies de manière assez imprécise, puisqu'elles regroupent les fonctions que l'usage (ou la fréquence d'utilisation) a fini par associer à un nom. Parmi ces fonctions, on trouve un grand nombre de fonctions qui sont des solutions d'équations différentielles du second ordre, sans que cette propriété soit exclusive. Ces fonctions sont toutefois très utiles, car elles apparaissent très souvent, dès que l'on cherche à résoudre des équations différentielles du second ordre dont les coefficients ne sont pas constants. Les fonctions spéciales sont disponibles en programmation sous la forme de bibliothèques. Elles sont aussi définies pour un grand nombre d'entre elles dans les logiciels de calcul symbolique (Maple, Mathematica,...). Dans la suite de ce cours, nous allons définir une partie d'entre elles et décrire les méthodes numériques utilisées dans les bibliothèques de programmes pour le calcul de ces fonctions.

## 2.2 Fonctions transcendantes simples

Les fonctions les plus simples que l'on rencontre lors de l'apprentissage des mathématiques sont tout d'abord les monômes, puis les polynômes, et enfin les fractions rationnelles. Le calcul de la valeur de la fonction pour un argument réel ou complexe nécessite un nombre fini des quatre opérations élémentaires que sont l'addition, la soustraction, la multiplication et la division.

Les premières fonctions transcendantes que l'on rencontre sont alors les fonctions trigonométriques (sin, cos,tan, arccos, arcsin) ainsi que leurs fonctions inverses. Les fonctions exp et log représentent généralement le reste de l'arsenal des fonctions transcendantes définies dans un cursus de premier, voire de second cycle universitaire.

## 2.3 Fonction Gamma

### 2.3.1 Définition et propriétés

La fonction Gamma est généralement définie par l'intégrale suivante

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt \tag{2.1}$$

quand la partie réelle de $z$ est strictement positive, $Re(z) > 0$).

La formule d'Euler donne une expression de la fonction $\Gamma$ pour toute valeur de $z$ complexe hormis les valeurs de $z$ entières négatives où la fonction possède des pôles :

$$\Gamma(z) = \lim_{n\to\infty} \frac{n! n^z}{z(z+1)\dots(z+n)} \tag{2.2}$$

En intégrant par parties l'équation (2.1), on peut facilement montrer que

$$\Gamma(z+1) = z\Gamma(z) \tag{2.3}$$

En vérifiant que $\Gamma(1) = 1$, on obtient par récurrence que

$$\Gamma(n+1) = n! \tag{2.4}$$

Avec cette définition, la fonction $\Gamma$ apparaît comme un prolongement analytique de la fonction factorielle définie sur $\mathbb{N}$. D'après l'équation (2.2), la fonction $\Gamma$ a un pôle en 0 et pour toutes les valeurs entières négatives.

La formule suivante permet de relier la fonction entre les valeurs situées dans le demi-plan complexe où $Re(z) > 1$ et celui où $Re(z) < 1$ :

$$\Gamma(1-z) = \frac{\pi}{\Gamma(z)\sin(\pi z)} \tag{2.5}$$

Pour calculer numériquement la fonction $\Gamma$ pour une valeur de $z$ en dehors des pôles, il est nécessaire de développer cette fonction sur la base des polynômes et des exponentielles. La formule la plus précise est celle de Lanczòs. Ce développement est spécifique à la fonction $\Gamma$. La formule qui s'inspire de la formule de

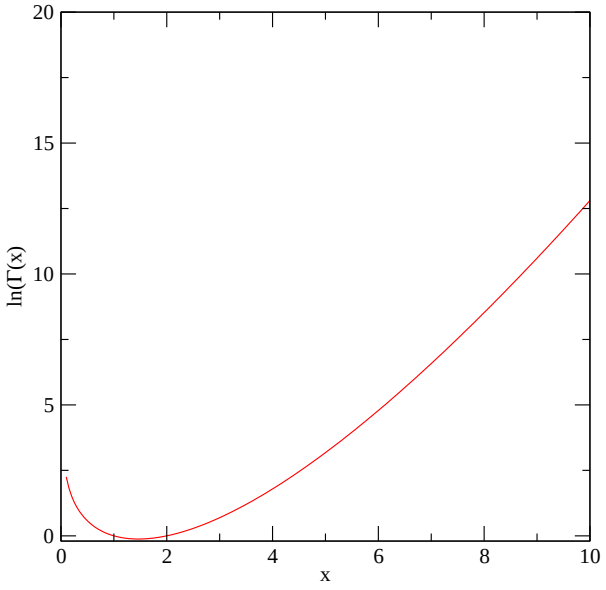


FIG. 2.1 – La fonction $\ln(\Gamma(x))$ en fonction de $x$.

Stirling bien connue pour la fonction factorielle n'est valable que pour $Re(z) > 0$ et est donnée par

$$\Gamma(z+1) = \left(z+\gamma+\frac{1}{2}\right)^{z+\frac{1}{2}} e^{-(z+\gamma+\frac{1}{2})} \times \sqrt{2\pi}\left[c_0 + \frac{c_1}{z+1} + \frac{c_2}{z+2} + \ldots + \frac{c_N}{z+N} + \epsilon\right] \tag{2.6}$$

où $\epsilon$ est le paramètre estimant l'erreur. Pour le choix particulier $\gamma = 5$, $N = 6$ et $c_0$ très voisin de 1, on a $|\epsilon| < 2.10^{-10}$.

Il est difficile de calculer la fonction $\Gamma$ pour des valeurs de $z$ un peu importantes. Cela résulte de la croissance très rapide de la fonction $\Gamma$. On peut montrer que la fonction $\Gamma$ croît plus vite que toute exponentielle, comme de manière analogue on montre que l'exponentielle croît plus vite que tout polynôme. On dit parfois que la fonction $\Gamma$ a une croissance super-exponentielle.

Dans de nombreuses formules, la fonction $\Gamma$ apparaît à la fois au numérateur et au dénominateur d'une expression. Chacun des termes peut être très important, mais le rapport est souvent un nombre relativement modeste. Pour calculer numériquement ce type d'expression, il est préférable de calculer $\ln(\Gamma(z))$ (voir Fig. 2.1). Ainsi la fraction est alors l'exponentielle de la différence de deux logarithmes. Tous les nombres qui interviennent dans ce calcul sont exponentiellement plus petits que ceux qui apparaissent dans un calcul direct, on évite ainsi le dépassement de capacité de l'ordinateur.

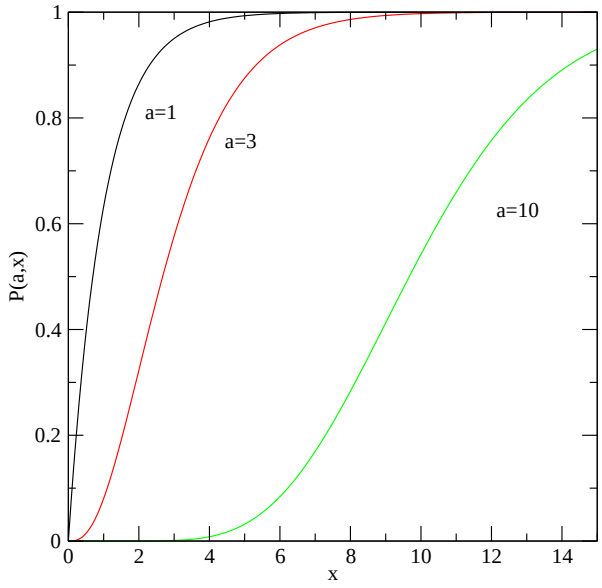


FIG. 2.2 – La fonction $P(a,x)$ en fonction de $x$ pour trois valeurs de $a$ ($a = 1, 3, 10$).

On définit la fonction $\gamma$ incomplète[1] comme

$$\gamma(a,x) = \int_0^x t^{a-1} e^{-t} dt \tag{2.7}$$

La fonction normalisée suivante $P(a,x)$

$$P(a,x) = \frac{\gamma(a,x)}{\Gamma(a)} \tag{2.8}$$

est parfois appelée aussi fonction Gamma incomplète. On peut montrer que $P(a,x)$ est monotone croissante avec $x$. La fonction est très proche de 0 quand $x$ est inférieur à $a-1$ et proche de 1 quand $x$ est très supérieur. La variation entre ces deux valeurs apparaît autour de l'abscisse $a-1$ et sur une largeur de l'ordre de $\sqrt{a}$ (voir figure 2.2).

### 2.3.2 Fonctions reliées : $\Psi$, $B$

A partir de la fonction $\Gamma$, on définit des fonctions dérivées. En raison de leur grande fréquence d'utilisation, elles ont "reçu" un nom. Ainsi, la fonction

[1] Noter que dans le logiciel Maple, la fonction Gamma et les fonctions Gamma incomplètes sont les seules fonctions définies avec des lettres capitales.

$\Psi$, appelée aussi fonction Digamma est définie comme la dérivée logarithmique de la fonction Gamma[2] :

$$\Psi(x) = \frac{d\ln(\Gamma(x))}{dx} \tag{2.10}$$

Parmi les propriétés remarquables de la fonction $\Psi$, notons que, pour des valeurs entières, on a

$$\Psi(n) = -\gamma + \sum_{i=1}^{n-1} \frac{1}{i} \tag{2.11}$$

où $\gamma = 0.577\ldots$ est la constante d´Euler.

Les fonctions Beta qui sont notées paradoxalement avec un $B$ sont définies par la relation :

$$B(z,w) = \frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)} \tag{2.12}$$

## 2.4 Fonctions de Bessel

Les fonctions de Bessel sont définies de la manière suivante : considérons l'équation différentielle du second ordre

$$x^2y'' + xy' + (x^2 - \nu^2)y = 0 \tag{2.13}$$

Les solutions de cette équation sont appelées fonctions de Bessel de première et de deuxième espèce : La solution finie à l'origine et notée $J_\nu(x)$ est appelée fonction de Bessel de première espèce et la seconde solution notée $Y_\nu(x)$ est appelée fonction de Bessel de deuxième espèce. Si $\nu$ n'est pas un entier, ces fonctions sont reliées par la relation suivante :

$$Y_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)} \tag{2.14}$$

La figure 2.3 représente graphiquement les fonctions de Bessel de première et de seconde espèces pour les quatre premières valeurs entières de $\nu$

Le comportement asymptotique des fonctions de Bessel de première et de seconde espèce est le suivant

$$J_\nu(x) \simeq \sqrt{\frac{2}{\pi x}}(\cos(x - \nu\pi/2 - \pi/4)) \tag{2.15}$$

$$Y_\nu(x) \simeq \sqrt{\frac{2}{\pi x}}(\sin(x - \nu\pi/2 - \pi/4)) \tag{2.16}$$

Soit l'équation différentielle du second ordre

$$x^2y'' + xy' - (x^2 - \nu^2)y = 0 \tag{2.17}$$

[2]On définit aussi les fonctions polygamma comme une généralisation de la fonction Digamma

$$\Psi(n,x) = \frac{d^n\Psi(x)}{dx^n} \tag{2.9}$$

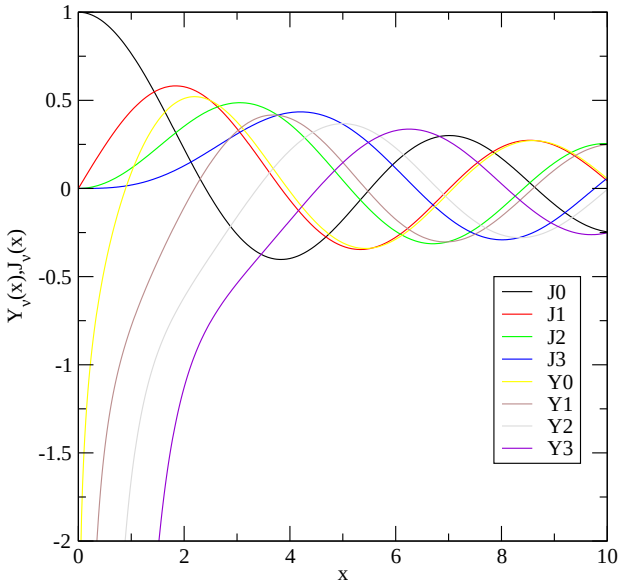


FIG. 2.3 – Les quatre premières fonctions de Bessel entières de première espèce et de deuxième espèce. Ces fonctions sont présentes dans toutes les bibliothèques mathématiques de programmation, dans les logiciels de calcul symbolique comme Maple et dans un logiciel graphique comme xmgrace.

Les solutions de cette équation sont appelées fonctions de Bessel modifiées . La solution finie à l'origine et notée $I_\nu(x)$ est appelée fonction de Bessel modifiée de première espèce et la seconde solution notée $K_\nu(x)$ est appelée fonction de Bessel modifiée de seconde espèce. Ces fonctions sont reliées par la relation suivante :

$$K_\nu(x) = \frac{\pi(I_\nu(-x) - I_\nu(x))}{2\sin(\nu\pi)} \tag{2.18}$$

La figure 2.4 représente graphiquement les fonctions de Bessel modifiées de première et de deuxième espèce pour les quatre premières valeurs entières de $\nu$.

Le comportement asymptotique des fonctions de Bessel modifiées est le suivant :

$$I_\nu(x) \simeq \frac{e^z}{\sqrt{2\pi x}} \tag{2.19}$$

$$K_\nu(x) \simeq \sqrt{\frac{\pi}{2x}} e^{-z} \tag{2.20}$$

Les fonctions de Hankel $H_{1,\nu}$ and $H_{2,\nu}$ sont appelées fonctions de Bessel de

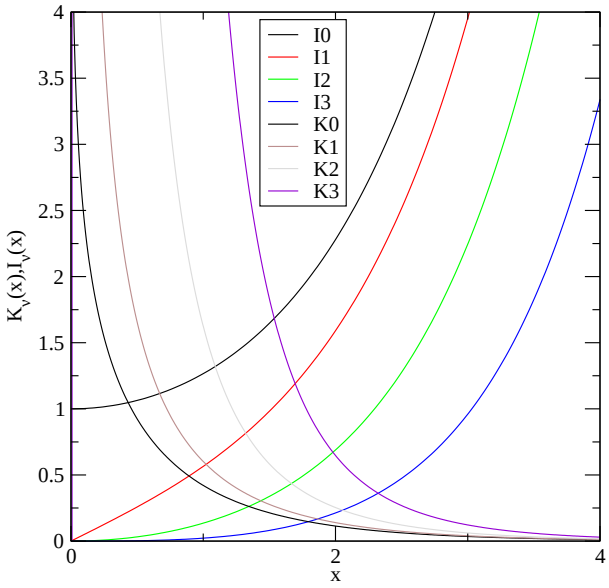


FIG. 2.4 – Les quatre premières fonctions de Bessel entières modifiées de première et de deuxième espèce. Ces fonctions sont présentes dans toutes les bibliothèques mathématiques de programmation, dans les logiciels de calcul symbolique comme Maple et dans un logiciel graphique comme xmgrace.

troisième espèce et sont définies par la relation

$$H_{1,\nu}(x) = J_\nu(x) + iY_\nu(x) \tag{2.21}$$
$$H_{2,\nu}(x) = J_\nu(x) - iY_\nu(x) \tag{2.22}$$

## 2.5 Fonctions Hypergéométriques

### 2.5.1 Fonction Hypergéométrique Gaussienne

Les fonctions hypergéométriques gaussiennes sont définies comme étant les solutions de l'équation différentielle suivante.

$$x(1-x)y'' + [c - (a+b+1)x]y' - aby = 0 \tag{2.23}$$

où $a$, $b$ et $c$ sont des constantes.

Si $c$, $a-b$ et $c-a-b$ sont non entiers, la solution générale de cette équation est

$$y = F(a,b;c;x) + Bx^{1-c}F(a-c+1, b-c+1; 2-c; x) \tag{2.24}$$

La fonction $F$ peut être exprimée sous la forme d'une série

$$\begin{aligned} F(a,b;c;z) &\equiv {}_2F_1(a,b,c;z) \\ &= \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{z^n}{n!} \end{aligned} \tag{2.25}$$

Cette série converge uniformément à l'intérieur du disque unité. Dès que $a$, $b$ ou $c$ sont entiers, la fonction hypergéométrique peut se réduire à une fonction transcendante plus simple. Par exemple, ${}_2F_1(1,1,2;z) = -z^{-1}\ln(1-z)$

### 2.5.2 Fonctions Hypergéométriques généralisées

On définit des fonctions hypergéométriques généralisées de la manière suivante : soit le rapport

$${}_pF_q\left[\begin{array}{c} a_1, a_2, \ldots, a_p \\ b1, b_2, \ldots, b_q \end{array} ; z\right] = \sum_{k=0}^{\infty} \frac{(a_1)_k (a_2)_k \ldots (a_p)_k}{(b_1)_k (b_2)_k \ldots (b_q)_k} \frac{x^k}{k!} \tag{2.26}$$

où on a utilisé la notation de Pochhammer

$$(a)_k = \frac{\Gamma(a+k)}{\Gamma(a)} \tag{2.27}$$

## 2.6 Fonction erreur, exponentielle intégrale

La fonction erreur et la fonction erreur complémentaire sont définies comme

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{2.28}$$

$$\begin{aligned} erfc(x) &= 1 - erf(x) \\ &= \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt \end{aligned} \tag{2.29}$$

La fonction $erf$ est aussi présente dans toutes les bibliothèques standard de programmation.

La fonction exponentielle intégrale $Ei$ est définie comme la valeur principale de l´intégrale suivante pour $x > 0$.

$$Ei(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t} dt \tag{2.30}$$

$$= \int_{-\infty}^{x} \frac{e^t}{t} \tag{2.31}$$

Le développement en série de cette fonction donne

$$Ei(x) = \gamma + \ln(x) + \sum_{n=1}^{\infty} \frac{x^n}{n\, n!} \tag{2.32}$$

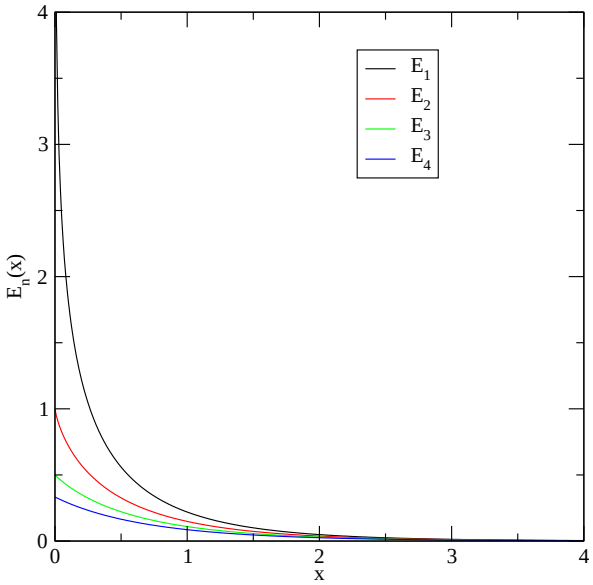


FIG. 2.5 – Les quatre premières fonctions exponentielles intégrales (fonctions $E_n$. Ces fonctions sont présentes dans toutes les bibliothèques mathématiques de programmation, dans les logiciels de calcul symbolique comme Maple et dans un logiciel graphique comme xmgrace.

Pour des grandes valeurs de $x$, on a le développement asymptotique suivant

$$Ei(x) \simeq \frac{e^x}{x}\left(1\frac{1}{x} + \ldots\right) \tag{2.33}$$

De manière générale, on définit les exponentielles intégrales $E_n(x)$ comme

$$E_n(z) = \int_1^\infty \frac{e^{-zt}}{t^n} dt \tag{2.34}$$

La Figure 2.5 représente les quatre premières exponentielles intégrales. Le développement en série de cette fonction donne

$$E_1(x) = -(\gamma + \ln(x)) + \sum_{n=1}^{\infty}(-1)^n \frac{x^n}{n\, n!} \tag{2.35}$$

La fonction $E_i(1, x)$ n´est définie que pour des arguments réels : Pour $x < 0$, on a

$$E_i(x) = -E_i(1, -x) \tag{2.36}$$

On peut noter que les exponentielles intégrales $E_n(x)$ sont reliées à la fonction $\gamma$ par la relation

$$E_n(x) = x^{n-1}\gamma(1 - n, x) \tag{2.37}$$

## 2.7 Conclusion

Cette introduction aux fonctions spéciales est très loin d'être exhaustive ; il existe de nombreuses autres fonctions dites spéciales : les fonctions elliptiques, les fonctions de Fresnel, les fonctions de Meier,.... Le développement de bibliothèques qui permettent de calculer les valeurs de ces fonctions est un secteur très actif et nous disposerons dans les années futures de bibliothèques encore plus performantes.

# Chapitre 3

# Racines d'équations

**Contenu**



## 3.1 Introduction

L'une des tâches rencontrées fréquemment lors d'un calcul est la recherche de la racine d'une équation. Sans perte de généralité, on peut toujours écrire une équation où le membre de droite est égal à zéro,

$$f(x) = 0 \tag{3.1}$$

Si $x$ est une variable scalaire, le problème est unidimensionnel. Si $x$ est une variable vectorielle (à $N$ dimensions) et que l'on a $N$ équations à satisfaire, on peut formellement écrire sous une notation vectorielle

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{3.2}$$

Malgré la similarité des équations (3.1) et (3.2), un système d'équations à $N$ variables est considérablement plus compliqué à résoudre qu'un système unidimensionnel. La raison vient du fait que la méthode générale pour la recherche de racines est liée à la capacité d'encadrer numériquement la région où le système d'équations possède une racine particulière.

On exclut de ce chapitre le cas des systèmes linéaires qui sera traité dans le chapitre de l'algèbre linéaire. Le principe dominant la recherche de racines

d'équations est celui de méthodes itératives, où en partant d'une valeur d'essai (ou un couple de valeurs d'essai), on s'approche de plus en plus près de la solution exacte. Il est évident qu'une estimation de départ raisonnable associée à une fonction $f$ qui varie suffisamment lentement est nécessaire pour obtenir une convergence vers la solution recherchée.

Nous allons considérer le problème unidimensionnel pour lequel plusieurs méthodes sont disponibles afin de choisir la méthode la mieux adaptée compte tenu des informations que l'on dispose sur la fonction $f$. Une dernière partie de ce chapitre sera consacrée aux méthodes plus spécifiques pour la recherche de racines de polynômes

## 3.2 Dichotomie

Comme nous l'avons mentionné ci-dessus, la clé de la recherche de racines d'équations repose sur l'existence d'un encadrement préalable de cette racine. S'il existe un couple $(a, b)$ tel que le produit $f(a)f(b) < 0$ et si la fonction est continue, le théorème de la valeur intermédiaire nous dit que fonction s'annule au moins une fois à l'intérieur de cet intervalle.

La méthode de dichotomie est une méthode qui ne peut pas échouer, mais sa rapidité de convergence n'est pas la meilleure en comparaison avec les autres méthodes. L'idée de cette méthode est la suivante : soit une fonction $f$ monotone sur un intervalle $[a_0, b_0]$ telle que $f(a_0)f(b_0) < 0$ , on sait alors qu'il existe une et une seule racine comprise dans cet intervalle.

L´algorithme de la méthode de dichotomie est le suivante : tout d'abord, on calcule $f(\frac{a_0+b_0}{2})$.

- Si $f(\frac{a_0+b_0}{2})f(a_0) < 0$, on définit un nouvel encadrement de la racine par le couple $(a_1, b_1)$ tel que

$$a_1 = a_0 \tag{3.3}$$

$$b_1 = \frac{a_0 + b_0}{2}. \tag{3.4}$$

- Si $f(\frac{a_0+b_0}{2})f(a_0) > 0$, alors on définit un nouvel encadrement de la racine par le couple $(a_1, b_1)$ tel que

$$a_1 = \frac{a_0 + b_0}{2} \tag{3.5}$$

$$b_1 = b_0. \tag{3.6}$$

En itérant cette méthode, on obtient une suite de couple $(a_n, b_n)$ telle que $\epsilon_n = b_n - a_n$ vérifie la relation

$$\epsilon_{n+1} = \frac{\epsilon_n}{2} \tag{3.7}$$

où $\epsilon_0 = (b_0 - a_0)/2$ Cela signifie que si l'on se fixe la tolérance $\epsilon$ qui représente la précision à laquelle on souhaite obtenir la racine, on a un nombre d'itérations à effectuer égal à

$$n = \ln_2 \left( \frac{|b_0 - a_0|}{\epsilon} \right) \tag{3.8}$$

où la notation $\ln_2$ signifie le logarithme en base 2.

Le choix de la valeur de la tolérance nécessite quelques précautions. Si la racine recherchée est de l'ordre de l'unité, on peut très raisonnablement choisir $\epsilon_0$ de l'ordre de $10^{-6}$ à $10^{-13}$ selon que l'on travaille en simple ou double précision. Par contre pour une racine dont la valeur est de l'ordre de $10^{10}$, une précision de $10^{-4}$ sera la valeur maximale que l'on peut atteindre en double précision. Inversement, pour une racine proche de zéro, la précision peut être meilleure que $10^{-14}$.

## 3.3 Méthode de Ridder

### 3.3.1 Méthode de la position fausse

La méthode de dichotomie sous-exploite le fait que l'on peut mieux utiliser la fonction à l'intérieur de l'encadrement effectué à chaque itération. La méthode de la position fausse approche la fonction de manière linéaire dans l'intervalle considéré.

Soit la droite $y = cx+d$ passant par $f(a_0)$ et $f(b_0)$ en $a_0$ et $b_0$ respectivement, on obtient facilement que

$$c = \frac{f(b_0) - f(a_0)}{b_0 - a_0} \tag{3.9}$$

$$d = \frac{b_0 f(a_0) - a_0 f(b_0)}{b_0 - a_0} \tag{3.10}$$

La nouvelle abscisse estimée pour la racine de l'équation est donnée par $y = cx + d = 0$, ce qui donne

$$x = -\frac{d}{c} \tag{3.11}$$

$$= \frac{b_0 f(a_0) - a_0 f(b_0)}{f(b_0) - f(a_0)} \tag{3.12}$$

soit encore

$$x = a_0 - (b_0 - a_0)\frac{f(a_0)}{f(b_0) - f(a_0)} \tag{3.13}$$

$$= b_0 + (b_0 - a_0)\frac{f(b_0)}{f(b_0) - f(a_0)} \tag{3.14}$$

On reprend à ce stade le principe de l'algorithme précédent si $f(x)f(a_0) > 0$ alors

$$a_1 = x \tag{3.15}$$

$$b_1 = b_0 \tag{3.16}$$

sinon

$$a_1 = a_0 \tag{3.17}$$

$$b_1 = x \tag{3.18}$$

La figure

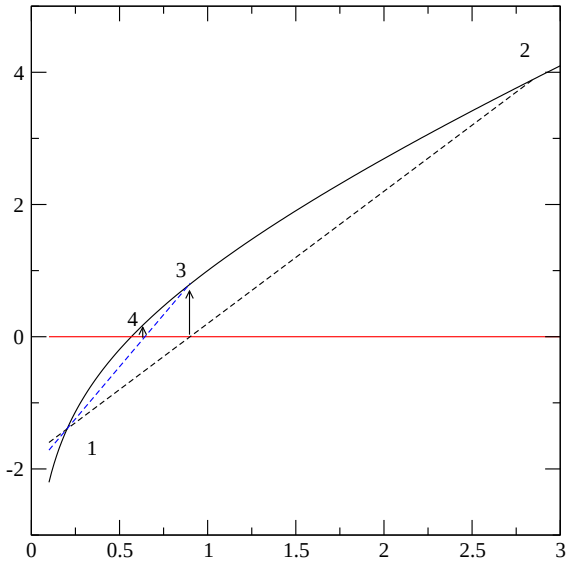


FIG. 3.1 – Schéma illustrant le principe de la position fausse. Les lignes en pointillé correspondent aux interpolations linéaires. A partir de l'encadrement repéré par les points 1 et 2, le processus itératif conduit aux points 3, puis 4...

### 3.3.2 Méthode de Ridder

Une variante de la méthode précédente qui est très efficace est basée sur l'algorithme suivant. On évalue la fonction au point $x = \frac{a_0+b_0}{2}$ et on résout l'équation en $z$

$$f(a_0) - 2f(x)z + f(b_0)z^2 = 0 \tag{3.19}$$

La solution positive est donnée par

$$z = \frac{f(x) + sgn(f(x))\sqrt{f(x)^2 - f(a_0)f(b_0)}}{f(b_0)} \tag{3.20}$$

En appliquant la méthode de la position fausse non pas à $f(a_0)$, $f(x)$ et $f(b_0)$, mais à $f(a_0)$, $f(x)z$ et $f(b_0)z^2$, on obtient une approximation de la racine, notée $x_4$ et donnée par

$$x_4 = x + (x - a_0)\frac{sgn(f(a_0) - f(b_0))f(x)}{\sqrt{f(x)^2 - f(a_0)f(b_0)}} \tag{3.21}$$

Parmi les propriétés remarquables, notons que $x_4$ est toujours située à l'intérieur de l'intervalle $[a_0, b_0]$. Si le produit $f(x_3)f(x_4)$ est négatif, on prend l´intervalle $[x_3, x_4]$ comme nouvelle encadrement, sinon si on considère le produit $f(x_1)f(x_4)$ ; si celui est négatif, le nouvel encadrement est $[x_1, x_4]$, sinon on prend $[x_4, x_2]$. On itère ensuite le procédé.

## 3.4 Méthode de Brent

Le principe de cette méthode est de combiner les avantages des méthodes précédemment exposées en utilisant, le principe de l'encadrement de la racine, la dichotomie, et l'interpolation quadratique inverse. Cela nécessite de connaître trois valeurs de la fonction $f$ dont la racine est à déterminer. Soit $(a, f(a))$, $(b, f(b))$, et $(c, f(c))$ la formule d'interpolation est donnée par

$$\begin{aligned} x = & \frac{(y-f(a))(y-f(b))c}{(f(c)-f(a))(f(c)-f(b))} \\ & + \frac{(y-f(b))(y-f(c))a}{(f(a)-f(b))(f(a)-f(c))} + \frac{(y-f(c))(y-f(a))b}{(f(b)-f(c))(f(b)-f(a))} \end{aligned} \tag{3.22}$$

En choisissant $y = 0$, on peut écrire l'équation (3.22) comme

$$x = b + \frac{P}{Q} \tag{3.23}$$

où $P$ et $Q$ sont donnés par

$$P = S[T(R-T)(c-b) - (1-R)(b-a)] \tag{3.24}$$

$$Q = (T-1)(R-1)(S-1) \tag{3.25}$$

où $R$, $S$ et $T$ s'expriment comme

$$R = \frac{f(b)}{f(c)} \tag{3.26}$$

$$S = \frac{f(b)}{f(a)} \tag{3.27}$$

$$T = \frac{f(a)}{f(c)} \tag{3.28}$$

En pratique, $b$ est une première estimation de la racine et $\frac{P}{Q}$ une petite correction. Quand $Q \to 0$ la valeur de $\frac{P}{Q}$ peut devenir très grande et l'itération par la méthode de Brent est remplacée par une itération de dichotomie.

## 3.5 Newton-Raphson

Toutes les méthodes précédentes ne nécessitaient que la connaissance de la fonction en différents points de l'intervalle encadrant la racine. Sous réserve que la variation de la fonction ne soit pas trop rapide, seule une hypothèse de continuité est nécessaire.

La méthode de Newton-Raphson nécessite de plus que la fonction $f$ dont on cherche à déterminer une racine, soit dérivable au voisinage de celle-ci.

Les itérations successives de la méthode de Newton-Raphson sont basées sur le développement limité de la fonction autour d'un point

$$f(x+\delta) = f(x) + f'(x)\delta + \frac{f''(x)}{2}\delta^2 + \dots \tag{3.29}$$

Si $\delta$ est suffisamment petit, on peut négliger les termes non linéaires et une estimation de la racine est donnée par $f(x+\delta)=0$.

$$\delta = -\frac{f(x)}{f'(x)} \tag{3.30}$$

On voit immédiatement qu'il est nécessaire que la dérivée de la fonction ne s'annule pas dans le voisinage de $x$, sous peine que l'estimation de $\delta$ devienne très grande et ne permette pas à la méthode de converger.

Si les conditions précédemment énoncées sont vérifiées, on a une méthode qui converge de manière quadratique.

En effet, la relation de récurrence entre estimations successives est donnée par

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{3.31}$$

En posant $\epsilon_{i+1} = x_{i+1} - x$, où $x$ est la racine exacte, on a

$$\epsilon_{i+1} = \epsilon_i - \frac{f(x_i)}{f'(x_i)} \tag{3.32}$$

Si on utilise un développement limité de $f$ au deuxième ordre au point $x_i$ (ce qui suppose que la fonction est deux fois dérivable au voisinage de la racine), on obtient

$$\epsilon_{i+1} = -\epsilon_i^2 \frac{f''(x_i)}{2f'(x_i)} \tag{3.33}$$

La méthode converge donc très rapidement par comparaison avec les méthodes précédentes.

A noter que si la dérivée de la fonction n'est pas connue analytiquement, l'évaluation numérique de sa dérivée est possible par une formule d'accroissement

$$f'(x) \simeq \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{3.34}$$

Dans ce cas, la méthode de Newton-Raphson se réduit à une méthode d'intersection et la convergence de celle-ci est moins rapide que la convergence quadratique.

## 3.6 Racines de Polynômes

### 3.6.1 Réduction polynomiale

La recherche de racines d'un polynôme se construit de la manière suivante : soit $P_n(x)$ un polynôme de degré $n$. Si on obtient une première racine, on peut écrire

$$P_n(x) = (x - x_1)P_{n-1}(x) \tag{3.35}$$

où $P_{n-1}(x)$ est un polynôme de degré $n-1$. Ainsi, théoriquement, une fois obtenue une première racine, on peut recommencer la recherche d'une autre racine pour une polynôme de degré strictement inférieur. Successivement, on poursuit cette procédure jusqu'à l'obtention de l'ensemble des $n$ racines du polynôme

$P_n(x)$. Rappelons que les polynômes à coefficients complexes se factorisent en un produit de monômes de degré 1. Cette propriété exprime le fait que les polynômes à coefficients complexes ont l'ensemble de leurs racines dans le plan complexe,

$$P_n(x) = \prod_{i=1}^{n}(x - x_i) \tag{3.36}$$

($\mathbb{C}$ est un corps algébriquement clos).

### 3.6.2 Méthode de Laguerre

Les méthodes de recherche de zéros de polynômes sont nombreuses et une présentation détaillée dépasse largement le cadre de ce cours. Nous avons choisi de présenter une méthode dont le principe est assez simple. La méthode de Laguerre utilise le fait que les dérivées logarithmiques successives d'un polynôme divergent au voisinage d'une racine. En prenant le logarithme de l'équation (3.36), on obtient

$$\ln(|P_n(x)|) = \sum_{i=1}^{n} \ln(|x - x_i|) \tag{3.37}$$

En dérivant l'équation (3.37), on obtient

$$\begin{aligned} \frac{d\ln(|P_n(x)|)}{dx} &= \sum_{i=1}^{n} \frac{1}{x - x_i} \\ &= \frac{P_n'(x)}{P_n(x)} \qquad (3.38) \\ &= G \qquad (3.39) \end{aligned}$$

En dérivant l'équation (3.38), il vient

$$\begin{aligned} -\frac{d^2\ln(|P_n(x)|)}{dx^2} &= \sum_{i=1}^{n} \frac{1}{(x - x_i)^2} \\ &= \left[\frac{P_n'(x)}{P_n(x)}\right]^2 - \frac{P_n''(x)}{P_n(x)} \\ &= H \qquad (3.40) \end{aligned}$$

Soit la racine $x_1$ à déterminer, on suppose que la valeur de départ $x$ est située à une distance $a$ de $x_1$ et que l'ensemble des autres racines sont situées à une distance supposée identique et qui vaut $b$

$$x - x_1 = a \tag{3.41}$$

$$x - x_i = b \qquad i \in [2, n] \tag{3.42}$$

En insérant les équations (3.41) et (3.42) dans les équations (3.38), (3.40), on en déduit respectivement les relations suivantes

$$\frac{1}{a} + \frac{n-1}{b} = G \tag{3.43}$$

$$\frac{1}{a^2} + \frac{n-1}{b^2} = H \tag{3.44}$$

Après élimination de $b$, la valeur de $a$ est

$$a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}} \tag{3.45}$$

Le signe placé devant la racine du dénominateur est choisi tel que le dénominateur soit le plus grand possible. $x - a$ devient alors la nouvelle valeur de départ et on itère le processus. En combinant cette méthode avec celle de la réduction polynomiale, on peut calculer l'ensemble des racines. En pratique, comme chaque racine n'est déterminée qu'avec une précision finie, il est nécessaire d'ajouter une procédure dite de lissage pour éviter les problèmes d'instabilité numérique.

# Chapitre 4

# Equations différentielles

**Contenu**



## 4.1 Introduction

La résolution numérique d'équations différentielles est très souvent nécessaire, faute de l'existence de solutions analytiques. Le but de ce chapitre est de montrer que la meilleure méthode, ou la plus efficace à utiliser pour obtenir une solution, nécessite de connaître la nature de l'équation différentielle à résoudre. Les méthodes les plus standards que nous allons présenter sont largement présentes dans les logiciels de calcul comme Maple, Matlab, Scilab, Octave, ou Mathematica, et surtout dans les bibliothèques pour la programmation (IMSL, NAG, GSL). Il est donc préférable d'utiliser ces bibliothèques plutôt que de réécrire un code peu performant et probablement faux dans un premier temps.

## 4.2 Définitions

Soit une fonction numérique notée $y(x)$ définie sur un intervalle fermé $[a, b]$ de $\mathbb{R}$ et de classe $C_p$ (continûment dérivable d'ordre $p$). On appelle équation différentielle d'ordre $p$ une équation de la forme

$$F(x, y, y', y'', \dots y^{(p)}) = 0 \tag{4.1}$$

où $y'$ représente la dérivée première par rapport à $x$, $y''$ la dérivée seconde, etc... Plus généralement, on appelle système différentiel un ensemble d'équations différentielles reliant une variable $x$ et un certain nombre de fonction $y_i(x)$ ainsi que leurs dérivées. L'ordre du système différentiel correspond à l'ordre de dérivation le plus élevé parmi l'ensemble des fonsctions.

On appelle solution de l'équation différentielle (4.1) toute fonction $y(x)$ définie sur l'intervalle $[a, b]$ de classe $C_p$ qui vérifie l'équation (4.1).

Une classe restreinte d'équations différentielles ont des solutions mathématiques "simples". Les équations différentielles à coefficients constants, mais aussi (voir chapitre 2) l'équation hypergéométrique, les équations de Bessel,...

On appelle forme canonique d'une équation différentielle une expression du type

$$y^{(p)} = f(x, y, y', y'', \dots y^{(p-1)}) \tag{4.2}$$

Seul ce type d'équations sera considéré dans ce chapitre.

Il est facile de vérifier que toute équation différentielle canonique peut être écrite comme un système d'équations différentielles du premier ordre.

Si on introduit $p - 1$ fonctions définies comme

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ &\dots \\ y_p &= y^{(p-1)} \end{aligned} \tag{4.3}$$

on peut exprimer l'équation (4.2) sous la forme

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ &\dots \\ y_p' &= f(x, y, y_1, y_2, \dots y_p) \end{aligned} \tag{4.4}$$

Par souci de commodité, on note dans la suite de ce chapitre $y$ une fonction dont telle que $y(x) \in \mathbb{R}^p$ où $p$ est entier positif.

Soit une équation ou système différentiel (canonique) défini sur un intervalle $[a, b]$ de $\mathbb{R}$

$$y' = f(x, y) \quad x \in [a, b] \quad y(x) \text{ et } y'(x) \in \mathbb{R}^p \tag{4.5}$$

On appelle condition de Cauchy la condition initiale

$$y(a) = y_0 \tag{4.6}$$

où $y_0$ est un vecteur de $\mathbb{R}^p$.

Le problème de Cauchy est la recherche d'une solution vérifiant la condition initiale (4.6).

Pour tout couple de vecteurs $y$ et $y^*$, et $\forall x$, la fonction $f(x, y)$ satisfait la condition de Lipschitz si

$$\|f(x, y) - f(x, y^*)\| \leq \|y - y^*\| L \tag{4.7}$$

où $L$ est appelée constante de Lipschitz et où la notation $\|\|$ désigne la norme.

Si dans le domaine $[a, b] \times \mathbb{R}^p$, la fonction $f(x, y)$ est continue par rapport au couple $(x, y)$ et si pour tout $x$ appartenant à l'intervalle $[a, b]$ elle satisfait une condition de Lipschitz, alors $\forall (x_0, y_0)$, il existe un voisinage de ce point tel que l'équation (4.5) satisfaisant la condition initiale (4.6) a une solution unique dans ce voisinage.

La condition de Lipschitz est une condition suffisante de l'existence et de l'unicité de la solution. Dans le cas où la fonction $f(x, y)$ est différentiable bornée ($||\partial f(x, y)/\partial y|| < L$), le critère de Cauchy est vérifié.

Soit l'équation différentielle suivante

$$1 + y' = \frac{y}{x} \tag{4.8}$$

$$y(0) = 0 \tag{4.9}$$

On vérifie aisément que la condition de Cauchy n'est pas vérifiée et dans ce cas, il existe une infinité de solutions $y = (C - \ln(x))x$ où $C$ est une constante quelconque qui satisfait la condition initiale.

## 4.3 Méthodes d'intégration à pas séparé

### 4.3.1 Introduction

Soit l'équation différentielle définie par les équations (4.5). On suppose que la fonction $f$ satisfait une condition de Lipschitz afin d'être certain que la solution existe, est unique et que le problème est bien posé.

On cherche à calculer une approximation de la solution $y(x)$ en un certain nombre de points $x_1, x_2, \ldots x_N$ de l'intervalle $[a, b]$, appelé maillage de l'intervalle, avec $x_0 = a$ et $x_N = b$

Nous supposons que la suite des points est choisie de manière à ce que la distance entre deux points consécutifs soit constante et on pose

$$h = \frac{(b - a)}{N} \tag{4.10}$$

ce qui donne

$$x_k = a + kh \tag{4.11}$$

avec $k = 0, 1, \ldots, N$

On appelle méthode d'intégration à pas séparé toute formule de récurrence de la forme

$$\begin{aligned} y_{k+1} &= y_k + h\phi(x_k, y_k, h) \\ k &= 0, 1, \ldots N \qquad \text{avec } y_0 \text{ donné} \end{aligned} \tag{4.12}$$

la fonction $\phi$ est supposée continue par rapport aux trois variables $x, y, h$.

On appelle méthode à pas multiples les méthodes telles que $y_{k+1}$ dépend de plusieurs valeurs précédentes $y_k, y_{k-1}, \dots y_{k-r}$.

### 4.3.2 Méthode d'Euler

Cette méthode est définie par

$$y_{k+1} = y_k + hf(y_k, x_k) \tag{4.13}$$

avec $y_0$ donné.

Cette méthode revient à approximer la solution au voisinage de $x_k$ par sa tangente et nous allons voir qu'elle est d'ordre 1. En effet, si la solution est suffisamment dérivable, on peut écrire

$$y(x_k + h) = y(x_k) + hy'(x_k) + \frac{h^2}{2}y''(x_k + \theta h) \tag{4.14}$$

avec $0 \leq \theta \leq 1$. ce qui donne

$$y(x_k + h) = y(x_k) + hf(y_k, x_k) + \frac{h^2}{2}y''(x_k + \theta h) \tag{4.15}$$

D'après la définition de la méthode

$$\frac{1}{h}\left(y(x_k + h) - y(x_k) - \phi(y_k(x_k), x_k, h)\right) = \frac{y(x_k + h) - y(x_k)}{h} - f(y(x_k), x_k) \tag{4.16}$$

ou

$$\frac{1}{h}\left(y(x_k + h) - y(x_k) - \phi(y_k(x_k), x_k, h)\right) = h\frac{y''(x_k + \theta h)}{2} \tag{4.17}$$

Si la dérivée seconde de $y$ est bornée par une constante $K$ dans l'intervalle d'intégration $[a, b]$, on aura

$$\max \|\frac{1}{h}(y(x_k + h) - y(x_k) - \phi(y_k(x_k), x_k, h))\| \leq Kh \tag{4.18}$$

ce qui montre que la méthode est d'ordre un.

La méthode d'Euler est une méthode numérique peu coûteuse numériquement, mais peu précise quand on intègre sur plusieurs pas de temps. Des améliorations sont possibles dès que l'on considère des points intermédiaires, ce que nous allons voir ci-dessous en considérant des méthodes dites de Runge-Kutta

### 4.3.3 Méthode RK explicites à un point

$$\begin{aligned} y_{k,1} &= y_k + \frac{h}{2\alpha}f(x_k, y_k) \\ y_{k+1} &= y_k + h(1-\alpha)f(x_k, y_k) + \alpha f(x_k + \frac{h}{2\alpha}, y_{k,1}) \\ & y_0 \text{ donné} \end{aligned}$$

avec $\alpha$ un nombre réel compris entre 0 et 1. Les valeurs de $\alpha$ couramment utilisées sont $\alpha = 1$, $\alpha = 1/2$ et $\alpha = 3/4$. Ces méthodes sont d'ordre 2.

### 4.3.4 Méthodes RK implicites à un point

La formule de récurrence est définie par la relation

$$y_{k+1} = y_k + h[(1-\theta)f(x_k, y_k) + \theta f(x_{k+1}, y_{k+1})] \tag{4.19}$$

où $\theta$ est un nombre réel appartenant à l'intervalle $]0, 1]$ (si $\theta = 0$, on retrouve la méthode d'Euler). Si $\theta = 1/2$, la méthode est d'ordre 2 et s'appelle la méthode des trapèzes. Si $\theta \neq 1/2$, la méthode est d'ordre 1.

### 4.3.5 Méthodes RK explicites à 2 points intermédiaires

Ces méthodes sont définies par les relations

$$\begin{aligned} y_{k,1} &= y_k + \frac{h}{3} f(x_k, y_k) \\ y_{k,2} &= y_k + \frac{2h}{3} f\left(x_k + \frac{h}{3}, y_{k,1}\right) \\ y_{k+1} &= y_k + \frac{h}{4}\left(f(x_k, y_k) + 3f\left(x_k + \frac{2h}{3}, y_{k,2}\right)\right) \end{aligned} \tag{4.20}$$

ou par

$$\begin{aligned} y_{k,1} &= y_k + \frac{h}{2} f(x_k, y_k) \\ y_{k,2} &= y_k + h\left(f(x_k, y_k) + 2f\left(x_k + \frac{h}{2}, y_{k,1}\right)\right) \\ y_{k+1} &= y_k + \frac{h}{6}\left(f(x_k, y_k) + 4f\left(x_k + \frac{h}{2}, y_{k,1}\right) + f\left(x_{k+1}, y_{k,2}\right)\right) \end{aligned} \tag{4.21}$$

Ces deux méthodes sont d'ordre 3. La première est parfois appelée méthode de Heun.

### 4.3.6 Méthodes RK explicites à 3 points intermédiaires

La méthode suivante est de loin la plus connue et utilisée. Les relations de récurrence sont les suivantes.

$$\begin{aligned} y_{k,1} &= y_k + \frac{h}{2} f(x_k, y_k) \\ y_{k,2} &= y_k + \frac{h}{2} f\left(x_k + \frac{h}{2}, y_{k,1}\right) \\ y_{k,3} &= y_k + \frac{h}{2} f\left(x_k + \frac{h}{2}, y_{k,2}\right) \\ y_{k+1} &= y_k + \frac{h}{6}\left(f(x_k, y_k) + 2f\left(x_k + \frac{h}{2}, y_{k,1}\right) + 2f\left(x_k + \frac{h}{2}, y_{k,2}\right) + f\left(x_{k+1}, y_{k,3}\right)\right) \end{aligned} \tag{4.22}$$

Cette méthode est d'ordre 4.

### 4.3.7 Formule générale des méthodes RK explicites

Les méthodes de Runge Kutta s'écrivent de manière générale

$$\begin{aligned}
K_1 &= h[f(x_k + \theta_1 h, y_k + \alpha_{1,1}K_1 + \alpha_{1,2}K_2 + \ldots + \alpha_{1,n}K_n)] \\
K_2 &= h[f(x_k + \theta_2 h, y_k + \alpha_{2,1}K_1 + \alpha_{2,2}K_2 + \ldots + \alpha_{2,n}K_n)] \\
&\ldots\ldots \\
K_n &= h[f(x_k + \theta_n h, y_k + \alpha_{n,1}K_1 + \alpha_{n,2}K_2 + \ldots + \alpha_{n,n}K_n)] \\
y_{k+1} &= y_k + h[\gamma_1 K_1 + \gamma_2 K_2 + \ldots \gamma_n K_n]
\end{aligned} \tag{4.23}$$

Les coefficients sont déterminés afin que l'ordre soit le plus élevé possible. On note $A$ la matrice de coefficients $(\alpha_{i,j})$, $\Gamma$ le vecteur des coefficients $\gamma_i$ et $\Theta$ le vecteur des coefficients $\theta_i$.

Quand la matrice $A$ est triangulaire inférieure stricte, $\alpha_{ij} = 0$ pour $j \geq i$, on dit que la méthode est explicite. Si seule la partie triangulaire supérieure est nulle, $\alpha_{ij} = 0$ pour $j > i$, la méthode est dite implicite ; sinon elle est totalement implicite.

Une représentation en forme de tableau des équations (4.23) donne

| | $\gamma_1$ | $\gamma_2$ | $\ldots$ | $\gamma_n$ |
|---|---|---|---|---|
| $\theta_1$ | $\alpha_{1,1}$ | $\alpha_{1,2}$ | $\ldots$ | $\alpha_{1,n}$ |
| $\theta_2$ | $\alpha_{2,1}$ | $\alpha_{2,2}$ | $\ldots$ | $\alpha_{2,n}$ |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | |
| $\theta_n$ | $\alpha_{n,1}$ | $\alpha_{n,2}$ | $\ldots$ | $\alpha_{n,n}$ |

Avec cette représentation, la méthode Runge-Kutta explicite à deux points qui est d'ordre 4 est représentée par le tableau suivant

| | 1/6 | 1/3 | 1/3 | 1/6 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1/2 | 1/2 | 0 | 0 | 0 |
| 1/2 | 0 | 1/2 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |

## 4.4 Méthode d'intégration à pas variables

### 4.4.1 Introduction

Un intégrateur "intelligent" possède une procédure de contrôle de la méthode de convergence, c'est à dire un moyen d'estimer l'erreur commise par le calcul sur un pas d´intégration et la possibilité de choisir en conséquence un nouveau pas si le système différentiel aborde une région où la fonction prend des valeurs plus importantes. Le calcul de cette estimation entraîne un surcoût de calcul, qu'il convient de bien gérer afin de minimiser cet effort supplémentaire.

L'idée la plus simple pour estimer cette erreur consiste à calculer la solution donnée par un algorithme (de Runge-Kutta d'ordre 4 par exemple) pour deux pas d´intégration différents, $h$ et $2h$. Soit $y(x + 2h)$ la solution exacte à $x + 2h$

et $y(x+h)$ la solution exacte à $x+h$, on a

$$y(x+2h) = y_1 + (2h)^5\phi + O(h^6) \tag{4.24}$$
$$y(x+2h) = y_2 + 2(h^5)\phi + O(h^6) \tag{4.25}$$

où $\phi$ est une fonction qui reste constante sur l'intervalle $x, x+2h$ à l'ordre $h^5$. La première équation correspond à une intégration avec un pas égal à $2h$ tandis que la seconde correspond à deux intégrations successives avec un pas de $h$. La différence

$$\Delta = y_2 - y_1 \tag{4.26}$$

fournit une estimation de l'erreur commise avec un pas d'intégration $h$.

## 4.5 Méthodes de Runge-Kutta "embarquées"

Une autre méthode pour estimer l'erreur commise par l'utilisation d'un pas $h$ est due à Fehlberg. Il utilise le fait qu'en choisissant des valeurs particulières de $\gamma_i$ (voir section 4.3.7), on peut changer l'ordre de l'évaluation de la solution pour un pas de temps $h$ donné.

$$y_1 = y(x) + \sum_{i=1}^{6} \gamma_i K_i + 0(h^6) \tag{4.27}$$

$$y_2 = y(x) + \sum_{i=1}^{6} \gamma_i^* K_i + 0(h^5) \tag{4.28}$$

ce qui conduit à une estimation de l'erreur

$$\Delta = \sum_{i=1}^{6} (\gamma_i - \gamma_i^*) K_i \tag{4.29}$$

Pour déterminer la valeur du pas la plus adaptée, on note tout d'abord que $\Delta$ est calculé à l'ordre $h^5$. Si on a un pas $h_1$ qui donne une erreur $\Delta_1$, le pas $h_0$ donné pour une erreur $\Delta_0$ fixée à l'avance, est donné par la relation

$$h_0 = h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{1/5} \tag{4.30}$$

Il est donc possible, pour une valeur de $|\Delta_0|$ donnée à l'avance de diminuer $h_0$ pour obtenir une erreur plus faible ou d'augmenter $h_0$ de manière raisonnable si $\Delta_1$ est inférieur en valeur absolue à $|\Delta_0|$.

Une difficulté apparaît pour ce type de méthode quand on considère un système différentiel à plusieurs variables. L'estimation de l'erreur est alors donnée à priori par un vecteur. La généralisation de la procédure ébauchée reste possible, mais nous ne détaillerons pas ce type de subtilité dans ce chapitre.

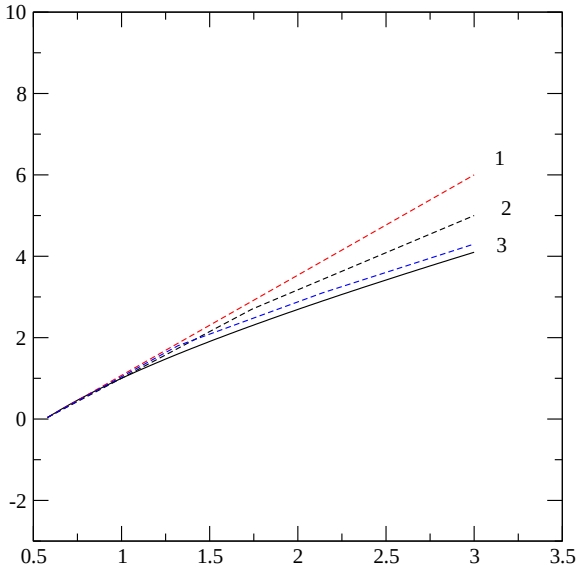


FIG. 4.1 – Schéma illustrant le principe de la méthode de Burlish-Stoer. Les lignes en pointillé correspondent aux intégrations réalisées avec trois pas d'intégration différents. La courbe en trait plein correspond à la solution exacte.

## 4.6 Méthode de Bulirsh-Stoer

L'idée de cette méthode repose sur les trois principes suivants : la résolution de l'équation différentielle pour un accroissement de $\Delta x$ est donnée par une fonction qui dépend de $h$, mais qui tend vers une limite finie (indépendante de $h$) ; on cherche à estimer la valeur exacte du système différentiel à intégrer en calculant pour différents pas et en prenant la limite d'un pas tendant vers zéro.

La figure 4.1 illustre graphiquement les différentes estimations de la valeur de la fonction solution en utilisant trois pas d'intégration différents. La limite d'un pas de temps nul correspond à la solution exacte donnée par la courbe en trait plein. Ce point est très comparable à celui de la méthode de Romberg décrite pour l'intégration.

Le second principe de la méthode consiste à extrapoler cette limite non pas sur la base de développements polynômiaux mais de développements en fractions rationnelles. Le troisième principe consiste à utiliser des fonctions d'erreur qui sont paires en pas d´intégration.

## 4.7 Conclusion

Les méthodes exposées ci-dessus permettent d'obtenir une solution de plus en plus précise pour des systèmes différentiels où le nombre de fonctions n'est pas très important. En simulation de Dynamique Moléculaire, un critère de qualité pour le choix d'une méthode est le respect de la propriété d'invariance par renversement du temps, qui est une propriété satisfaite pour les systèmes hamiltoniens. Les algorithmes satisfaisant cette propriété sont appelés symplectiques ; un exemple de ce type d'algorithme est l´algorithme dit de Verlet dont le principe est décrit dans le cours *Simulation numérique en Physique Statistique.*(http ://pascal.viot.com)

# Chapitre 5

# Transformée de Fourier rapide

**Contenu**



## 5.1 Introduction

Largement utilisée en Physique, la transformée de Fourier d'une fonction dépendant d'une variable (par exemple du temps) est devenue si naturelle que sa représentation graphique est généralement aussi utile, voire plus, que celle de la fonction elle-même. Après un rappel des propriétés élémentaires et fondamentales des transformées de Fourier pour la résolution de problèmes mathématiques, nous allons présenter le principe de la méthode numérique de l'évaluation de cette transformée. Plus spécifiquement, depuis près de 40 ans est apparu un algorithme performant pour le calcul de la transformée de Fourier dont le temps de calcul varie essentiellement comme $N \ln_2(N)$ où $N$ est le nombre de points où la fonction $f$ a été évaluée. Par opposition à une approche trop naïve, où le nombre d'opérations croît comme $N^2$ cette méthode a reçu le nom de transformée de Fourier rapide (FFT, Fast Fourier Transform, en anglais), que toute bibliothèque mathématique propose généralement à son catalogue.

## 5.2 Propriétés

Soit une fonction $f$ définie sur $\mathbb{R}$, on appelle transformée de Fourier de $f$, la fonction $\hat{f}$

$$\hat{f}(\nu) = \int_{-\infty}^{+\infty} f(t) e^{2\pi i \nu t} dt \tag{5.1}$$

La transformée de Fourier inverse est définie comme

$$f_1(t) = \int_{-\infty}^{+\infty} \hat{f}(\nu) e^{-2\pi i \nu t} d\nu \tag{5.2}$$

Pour une fonction $f$ intégrable, il y a identité entre la fonction $f_1$ et la fonction $f$, hormis éventuellement sur un support de $\mathbb{R}$ de mesure nulle.

Une autre définition de la transformée de Fourier rencontrée dans la littérature est celle qui correspond à une représentation en pulsation au lieu de celle en fréquence donnée ci-dessus.

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(t) e^{i\omega t} dt \tag{5.3}$$

La transformée de Fourier inverse correspondante est définie comme[1]

$$f_1(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(\omega) e^{-i\omega t} d\omega \tag{5.6}$$

Avec le logiciel Maple, la transformée de Fourier est définie d'une manière encore différente !

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(t) e^{-i\omega t} dt \tag{5.7}$$

avec bien entendu la transformée de Fourier inverse correspondante

$$f_1(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(\omega) e^{i\omega t} d\omega \tag{5.8}$$

Compte tenu des différentes définitions des transformées de Fourier et en l'absence de documentation précise dans certaines bibliothèques mathématiques, il est fortement recommandé de tester la transformée de Fourier numérique que l'on va utiliser, en testant une fonction dont la transformée de Fourier est connue analytiquement pour déterminer de manière simple la convention utilisée dans la bibliothèque disponible. Au delà du principe général de la transformée de Fourier rapide, il existe des optimisations possibles compte tenu de l'architecture de l'ordinateur sur lequel le programme sera exécuté. Cela renforce l'aspect totalement inutile de la réécriture d'un code réalisant la dite transformée, ce code serait en général très peu performant comparé aux nombreux programmes disponibles.

Dans le cas où la fonction $f$ possède des symétries, sa transformée de Fourier présente des symétries en quelque sorte duales de la fonction initiale

[1] Pour symétriser les expressions, la transformée de Fourier est parfois définie comme

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(t) e^{i\omega t} dt \tag{5.4}$$

et la transformée de Fourier inverse comme

$$f_1(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} \hat{f}(\omega) e^{-i\omega t} d\omega \tag{5.5}$$

- Si $f(t)$ est réelle, la transformée de Fourier des fréquences négatives est complexe conjuguée de celle des arguments positifs $\hat{f}(-\nu) = (\hat{f}(\nu))^*$.
- Si $f(t)$ est imaginaire, la transformée de Fourier des fréquences négatives est égale à l'opposée du complexe conjugué de celle des arguments positifs $\hat{f}(-\nu) = -(\hat{f}(\nu))^*$.
- Si $f(t)$ est paire, la transformée de Fourier est aussi paire, $\hat{f}(-\nu) = \hat{f}(\nu)$.
- Si $f(t)$ est impaire, la transformée de Fourier est aussi impaire, $\hat{f}(-\nu) = -\hat{f}(\nu)$.
- Si $f(t)$ est paire et réelle, la transformée de Fourier l'est aussi.
- Si $f(t)$ est impaire et réelle, la transformée de Fourier est imaginaire et impaire.
- Si $f(t)$ est paire et imaginaire, la transformée de Fourier l'est aussi.
- Si $f(t)$ est impaire et imaginaire, la transformée de Fourier est réelle et impaire.

Des propriétés complémentaires sont associées aux opérations de translation et dilatation

- Soit $f(at)$, sa transformée de Fourier est donnée par $\frac{1}{|a|}\hat{f}(\frac{\nu}{a})$.
- Soit $\frac{1}{|b|}f(\frac{t}{|b|})$, la transformée de Fourier est donnée par $\hat{f}(b\nu)$.
- Soit $f(t - t_0)$, la transformée de Fourier est donnée par $\hat{f}(\nu)e^{2\pi i\nu t_0}$.
- Soit $f(t)e^{-2\pi i\nu t_0}$, la transformée de Fourier est donnée par $\hat{f}(\nu - \nu_0)$.

Une identité particulièrement utile concernant les transformées de Fourier concerne la distribution de Dirac $\delta$ : soit $f$ une fonction continue définie sur $\mathbb{R}$, La distribution $\delta$ est définie à partir de la relation suivante

$$\int_{-\infty}^{+\infty} dx f(x)\delta(x - x_0) = f(x_0) \tag{5.9}$$

Ainsi, on en déduit facilement que

$$\int_{-\infty}^{+\infty} dx \delta(x - x_0)e^{2\pi kix} = e^{2\pi kix_0} \tag{5.10}$$

et donc que

$$\int_{-\infty}^{+\infty} dx \delta(x)e^{2\pi ikx} = 1 \tag{5.11}$$

Par inversion de cette relation, on a la représentation intégrale de la distribution $\delta$

$$\delta(x) = \int_{-\infty}^{+\infty} dk e^{-2\pi ikx} \tag{5.12}$$

De manière équivalente, on montre que

$$\delta(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} dk e^{-ikx} \tag{5.13}$$

La propriété sans doute la plus importante concernant les transformées de Fourier concerne la convolution. Soit deux fonctions $f$ et $g$ définie sur $\mathbb{C}$, on définit la convolution de $f$ avec $g$ notée $f * g$ comme

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)d\tau \tag{5.14}$$

On vérifie immédiatement que $f * g = g * f$. L'opérateur de convolution est commutatif, car la multiplication de deux nombres complexes l'est aussi.

La transformée de Fourier de la convolution de deux fonctions $f$ et $g$ est égale au produit de leurs transformées de Fourier.

$$\widehat{(f * g)}(\nu) = \hat{f}(\nu)\hat{g}(\nu) \tag{5.15}$$

Soit une fonction réelle $f$, on définit la fonction d'autocorrélation comme

$$C_f(t) = \int_{-\infty}^{+\infty} du f(u) f(t+u) \tag{5.16}$$

La transformée de Fourier de la fonction $C_f$ est alors donnée par

$$\hat{C}_f(\nu) = \hat{f}(\nu)\hat{f}(-\nu) \tag{5.17}$$

Compte tenu du fait que $f$ est réelle, on a $\hat{f}(-\nu) = (\hat{f}(\nu))^*$, ce qui donne

$$\widehat{C_f}(\nu) = |\hat{f}(\nu)|^2 \tag{5.18}$$

Cette relation est appelée théorème de Wiener-Khinchin.

Le théorème de Parseval donne que

$$\int_{-\infty}^{+\infty} |f(t)|^2 dt = \int_{-\infty}^{+\infty} |\hat{f}(\nu)|^2 d\nu \tag{5.19}$$

## 5.3 Discrétisation de la transformée de Fourier

### 5.3.1 Échantillonage

Dans la situation la plus fréquemment rencontrée, la fonction $f$ est échantillonnée à intervalle régulier. Soit $\Delta$ le pas séparant deux "mesures" consécutives, on a

$$f_n = f(n\Delta) \qquad \text{avec } n = \ldots, -2, -1, 0, 1, 2, 3, \ldots \tag{5.20}$$

Pour le pas de l'échantillonnage $\Delta$, il existe une fréquence critique, appelée fréquence de Nyquist,

$$\nu_c = \frac{1}{2\Delta} \tag{5.21}$$

au delà de laquelle il n'est pas possible d'avoir une information sur le spectre de la fonction échantillonnée. Supposons que la fonction soit une sinusoïde de fréquence égale à celle de Nyquist. Si la fonction est maximale pour un point, elle est minimale au point suivant et ainsi de suite.

La conséquence de ce résultat est que si on sait que le support de la transformée de Fourier est strictement limité à l'intervalle $[-\nu_c, \nu_c]$, la fonction $f(t)$ est alors donnée par la formule

$$f(t) = \Delta \sum_{n=-\infty}^{+\infty} f_n \frac{\sin(2\pi\nu_c(t - n\Delta))}{\pi(t - n\Delta)} \tag{5.22}$$

Si la fonction $f(t)$ est multipliée par la fonction $e^{2\pi\nu_1 t}$ avec $\nu_1$ qui est un multiple de $1/\Delta$, les points échantillonnés sont exactement les mêmes. Une conséquence de cette propriété est l'apparition du phénomène de duplication ("aliasing" en anglais). Cela se manifeste de la manière suivante : supposons que la transformée de Fourier exacte d'une fonction ait un spectre plus large que celui donné par l'intervalle de Nyquist, le spectre situé à droite se replie à gauche et celui à droite se replie à gauche.

### 5.3.2 Transformée de Fourier discrète

Soit un nombre fini de points où la fonction $f$ a été échantillonnée

$$f_k = f(t_k) \qquad \text{avec } k = 0, 1, 2, \ldots, N-1 \tag{5.23}$$

Avec $N$ nombres fournis, il semble évident que l'on peut obtenir $N$ fréquences pour la transformée de Fourier. Soit la suite

$$\nu_n = \frac{n}{N\Delta} \qquad \text{avec } n = -N/2, \ldots, 0, \ldots, N/2 \tag{5.24}$$

Le nombre de fréquences est a priori égale à $N+1$, mais comme les bornes inférieure et supérieure de l'intervalle en fréquence correspondent aux bornes définies par la fréquence critique de Nyquist, les valeurs obtenues pour $n = -N/2$ et $n = N/2$ sont identiques et nous avons donc bien $N$ points indépendants.

Soit un nombre fini de points noté $h_k$, on définit la transformée de Fourier discrète comme

$$\hat{h}_n = \frac{1}{N} \sum_{k=0}^{N-1} h_k e^{2\pi i k n/N} \tag{5.25}$$

La transformée de Fourier discrète inverse est donnée par

$$h_k = \sum_{n=0}^{N-1} \hat{h}_n e^{-2\pi i k n/N} \tag{5.26}$$

Ces transformées de Fourier sont définies indépendamment des abscisses correspondant aux valeurs originales où la fonction $h$ a été évaluée. Il est toutefois possible de relier la transformée de Fourier discrète à la transformée de Fourier de départ par la relation

$$\hat{f}(\nu_n) \simeq \Delta \hat{f}_n \tag{5.27}$$

[2] Notons que les propriétés établies pour les transformées de Fourier au début de ce chapitre, selon le cas où la fonction est paire ou impaire, réelle ou imaginaire se transposent complètement au cas de la transformée de Fourier discrète. Les

[2] La transformée de Fourier continue est approchée de la manière suivante

$$\hat{f}(\nu_n) = \int_{-\infty}^{+\infty} f(t) e^{2\pi i \nu_n t} dt \tag{5.28}$$

$$\approx \Delta \sum_{k=0}^{N-1} f_k e^{2\pi i \nu_n t_k} \tag{5.29}$$

règles concernant la convolution se retrouvent bien évidemment et par exemple l'expression du théorème de Parseval est alors

$$\sum_{k=0}^{N-1} |\hat{h}_k|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |h_k|^2 \tag{5.30}$$

Le passage de la transformée de Fourier continue à la transformée de Fourier discrète est en fait un changement de mesure, ce qui explique que les propriétés établies pour l'une restent vraies pour la seconde.

On peut noter que les différences entre la transformée de Fourier discrète et son inverse sont au nombre de deux : un changement de signe dans l'exponentielle complexe et un facteur de normalisation en $1/N$. Ces faibles différences expliquent pourquoi les procédures sont souvent identiques pour le calcul de la transformée de Fourier et de son inverse dans les bibliothèques mathématiques.

## 5.4 Transformée de Fourier rapide

L'idée de base de la transformée de Fourier rapide repose sur la remarque suivante : posons

$$W = e^{2\pi i/N} \tag{5.31}$$

La composante de Fourier de h s'exprime alors comme

$$\hat{h}_n = \sum_{k=0}^{N-1} W^{nk} h_k \tag{5.32}$$

Ainsi le vecteur $h$ de composante $h_k$ est multiplié par une matrice de coefficients $a_{nk} = W^{nk}$. Sans utiliser d'astuces particulières, le temps de calcul pour une telle opération est alors en $N^2$. Pour améliorer de manière spectaculaire la rapidité de ce traitement, on note que la transformée de Fourier discrète de longueur $N$ peut être écrite comme la somme de deux transformées de Fourier discrète chacune de longueur $N/2$. En effet, en supposant que le nombre $N$ est pair, on peut séparer la contribution des termes pairs et celle des termes impairs dans l'équation

$$F_k = \sum_{j=0}^{N-1} e^{2\pi ijk/N} f_j \tag{5.33}$$

$$= \sum_{j=0}^{N/2-1} e^{2\pi i(2j)k/N} f_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi i(2j+1)k/N} f_{2j+1} \tag{5.34}$$

$$= \sum_{j=0}^{N/2-1} e^{2\pi ijk/(N/2)} f_{2j} + W^k \sum_{j=0}^{N/2-1} e^{2\pi ijk/(N/2)} f_{2j+1} \tag{5.35}$$

$$= F_k^p + W^k F_k^i \tag{5.36}$$

où $F_k^p$ représente la kième composante de Fourier pour les composantes paires de la fonction de départ et $F_k^i$ représente la kième composante de Fourier pour

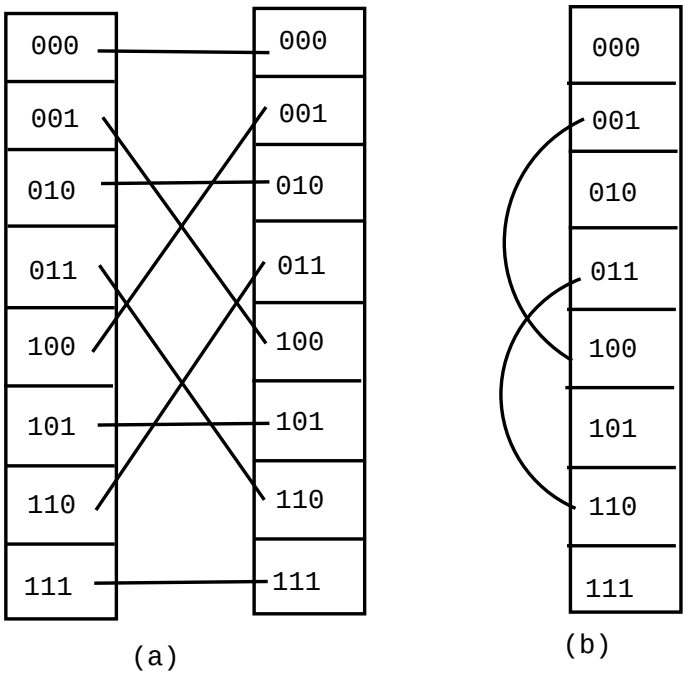


FIG. 5.1 – Schéma illustrant le principe de la mise en ordre du tableau par renversement de bits. (a) Sur un tableau de 8 éléments étiquettés en binaire, on note les opérations de déplacements. (b) Bilan des déplacements à effectuer : deux opérations d'échange d'éléments.

les composantes impaires de la fonction de départ. A noter que ces deux composantes sont périodiques en $k$ avec une période $N/2$.

Pour itérer ce type de procédure, nous allons supposer que le nombre de points de la fonction de départ est dorénavant une puissance de deux. On peut alors exprimer chaque composante de Fourier en fonction de deux nouvelles composantes de Fourier sur un intervalle de longueur $N/4$ et ainsi de suite. Une fois obtenue un intervalle de longueur 1, on a la transformée de Fourier pour cet intervalle qui est égale au nombre $f_n$ correspondant. Le nombre d'indices pour caractériser ce nombre est égal à $\ln_2(N)$

$$F_k^{iipip\ldots ppi} = f_n \tag{5.37}$$

On remarque donc que pour chaque valeur de $n$ on a un alphabet fini d'indice $iipip\ldots ppi$ constitué de $i$ et de $p$. Cette correspondance est biunivoque. Si on pose que $i = 1$ et $p = 0$, on a l'expression inversée en binaire de chaque valeur de $n$.

La procédure de transformée de Fourier rapide est alors constituée d'une première opération de tri pour réordonner la matrice selon la lecture de l'alphabet généré, puis on calcule successivement les composantes de Fourier de longueur 2, puis 4, jusqu'à $N$. La figure 5.1 illustre le nombre d'opérations à effectuer pour inverser les éléments du tableau selon le mécanisme du renversement de bits.

De nombreuses variantes de cet algorithme original sont disponibles, qui permettent d'obtenir la transformée de Fourier pour un nombre de points différent d'une puissance de deux.

# Chapitre 6

# Algèbre linéaire

**Contenu**



## 6.1 Introduction

Les deux chapitres qui suivent concernent le traitement numérique des matrices Ce premier chapitre est consacré aux méthodes employées pour résoudre les quatre tâches les plus fréquemment rencontrées pour les systèmes linéaires. Par définition, on peut écrire celles-ci comme

$$A.x = b \tag{6.1}$$

où $A$ est une matrice $M \times N$

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \ddots & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{pmatrix} \tag{6.2}$$

$x$ est un vecteur colonne de $M$ éléments et $b$ un vecteur colonne de $N$ éléments.

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_N \end{pmatrix} \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_M \end{pmatrix} \tag{6.3}$$

Si $N = M$, il y autant d'équations que d'inconnues et si aucune des équations n'est une combinaison linéaire des $N-1$ autres, la solution existe et est unique.[1]

Hormis pour des valeurs de $N$ très petites (typiquement $N \leq 4$), où les formules sont simples à exprimer, il est généralement nécessaire de procéder à un calcul numérique pour obtenir la solution d'un système d'équations linéaires. Comme nous allons le voir ci-dessous, si la matrice $A$ n'est pas une matrice creuse (matrice dont la majorité des éléments est nulle), il est nécessaire d'appliquer une méthode générale qui revient en quelque sorte à inverser la matrice $A$ (ou à la factoriser), ce qui nécessite un grand nombre d'opérations qui augmente comme le cube de la dimension linéaire de la matrice, $N^3$.

Même si la procédure numérique est censée conduire à une solution dans le cas d'une matrice non singulière ($det(A) \neq 0$), l'accumulation d'erreurs d'arrondi, souvent liée à des soustractions de nombres voisins fournit un vecteur $x$ erroné. Dans le cas d'une matrice dont le déterminant est très voisin de zéro (matrice presque singulière), les solutions obtenues peuvent être aussi fausses. L'énorme bibliothèque de sous-programmes pour les problèmes d'algèbre linéaire montre l'importance de ces problèmes dans le calcul numérique et la nécessité de choisir une méthode spécifique dès que la matrice a des propriétés particulières.

Parmi les tâches typiques d'algèbre linéaire hormis la résolution d'un système linéaire, on peut citer

- la détermination des solutions d'un ensemble de systèmes, par exemple $A.x_j = b_j$ où $x_j$ et $b_j$ sont des vecteurs à $N$ composantes et $j$ un indice parcourant un ensemble fini d'entiers,
- le calcul de l'inverse de $A$ : $A^{-1}$,
- le calcul du déterminant de $A$.

De manière encore plus aiguë que dans les chapitres précédents, une méthode de force brute est bien moins efficace pour résoudre un problème d'algèbre linéaire qu'une méthode spécifique. Nous allons donc voir tout d'abord le principe d'une méthode générale pour une matrice quelconque, puis considérer quelques unes des techniques spécifiques qui dépendent de la structure des matrices.

[1] Si $N > M$ , les équations sont indépendantes entre elles et il n'y a pas de solution. Si $N < M$, il y a une indétermination et il existe une infinité de solutions.

## 6.2 Élimination de Gauss-Jordan

### 6.2.1 Rappels sur les matrices

Il est utile de remarquer que le calcul des quatre tâches précédentes peut être a priori exécuté de manière similaire : en effet, si on considère par exemple un système $4 \times 4$, on a

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{132} & a_{133} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \cdot \left( \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \sqcup \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{pmatrix} \sqcup \begin{pmatrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \\ y_{31} & y_{132} & y_{133} & y_{34} \\ y_{41} & y_{42} & y_{43} & y_{44} \end{pmatrix} \right) =$$
$$= \left( \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} \sqcup \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} \sqcup \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \right) \tag{6.4}$$

où l'opérateur $\sqcup$ désigne l'opérateur réunion de colonnes. Ainsi on voit que l'inversion d'une matrice est identique à la résolution d'un ensemble de $N$ systèmes linéaires pour lequel le vecteur situé dans le membre de droite a tous les éléments nuls sauf un seul qui est différent pour chaque colonne.

Les trois règles élémentaires que l'on peut réaliser sur les matrices sont les suivantes.

- Échanger les lignes de $A$ et de $b$ (ou $c$ ou de la matrice identité) et en gardant $x$ (ou $t$ ou $Y$) ne change pas la solution du système linéaire. Il s'agit d'une réécriture des équations dans un ordre différent.
- De manière similaire, la solution du système linéaire est inchangée si toute ligne de $A$ est remplacée par une combinaison linéaire d'elle-même et des autres lignes, en effectuant une opération analogue pour $b$ (ou $c$ ou la matrice identité).
- L'échange de deux colonnes de $A$ avec échange simultané des lignes correspondantes de $x$ (ou $t$ ou $Y$) conduit à la même solution. Toutefois, si on souhaite obtenir la matrice $Y$ dans l'ordre initialement choisi, il est nécessaire de procéder à l'opération inverse une fois la solution obtenue.

### 6.2.2 Méthode sans pivot

Pour la simplicité de l'exposé, on considère tout d'abord une matrice dont les éléments diagonaux sont strictement différents de zéro.

On multiplie la première ligne de $A$ par $1/a_{11}$ (et la ligne correspondante de $b$ ou de la matrice identité). On soustrait $a_{21}$ fois la 1ère ligne à la deuxième ligne, $a_{31}$ fois la 1ère ligne à la troisième ligne et ainsi de suite jusqu'à la dernière ligne. La matrice $A$ a maintenant la structure suivante

$$\begin{pmatrix} 1 & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{pmatrix} \tag{6.5}$$

où les coefficients $a'_{ij}$ s'expriment en fonction des coefficients où les coefficients $a_{ij}$.

On multiplie alors la deuxième ligne par $1/a'_{22}$, puis on soustrait $a'_{12}$ fois la deuxième ligne à la première ligne, $a'_{32}$ fois la deuxième ligne à la troisième ligne et ainsi de suite jusqu'à la dernière ligne. La matrice $A$ a maintenant la structure suivante

$$\begin{pmatrix} 1 & 0 & a''_{13} & a''_{14} \\ 0 & 1 & a''_{23} & a''_{24} \\ 0 & 0 & a'_{33} & a''_{34} \\ 0 & 0 & a''_{43} & a''_{44} \end{pmatrix} \tag{6.6}$$

On itère le procédé jusqu'à la dernière ligne et on obtient alors la matrice identité. On peut obtenir alors facilement la solution du système d'équations.

### 6.2.3 Méthode avec pivot

La méthode précédente souffre du défaut de ne s'appliquer qu'aux matrices dont tous les éléments de la diagonale sont non nuls. Or, une matrice n'est pas nécessairement singulière si un seul élément de la diagonale est nul. Pour permettre une plus grande généralité de la méthode, on ajoute aux opérations précédentes la troisième règle énoncée ci-dessus. On peut en effet, en utilisant l'inversion des colonnes, placer sur la diagonale un élément non nul de la ligne (si tous les éléments d'une ligne sont nuls, cela signifie que le déterminant de la matrice est nul et que la matrice est singulière, ce qui a été exclu par hypothèse). De plus, on peut montrer que la recherche du plus grand élément de la ligne pour faire la permutation des colonnes rend la procédure bien plus stable, en particulier en augmentant de manière importante la précision numérique des solutions.

Ce type d'algorithme est disponible dans de nombreuses bibliothèques et il est parfaitement inutile de chercher à réécrire un code qui nécessite un temps important à la fois pour sa réalisation et pour sa validation.

## 6.3 Élimination gaussienne avec substitution

Pour la résolution stricte d'un système linéaire, il n'est pas nécessaire d'obtenir une matrice diagonale comme celle obtenue par la méthode précédente. Ainsi en effectuant à chaque ligne la soustraction des lignes situées au dessous de la ligne dont le coefficient de la diagonale vient d'être réduit à l'unité, on obtient une matrice triangulaire supérieure dont la structure est la suivante

$$\begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{133} & a'_{34} \\ 0 & 0 & 0 & a'_{44} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{pmatrix} \tag{6.7}$$

Il est possible de faire la résolution de ce système linéaire par une procédure de substitution successive

$$x_4 = \frac{b'_4}{a'_{44}} \tag{6.8}$$

$$x_3 = \frac{1}{a'_{33}}(b'_3 - x_4 a'_{34}) \tag{6.9}$$

et de manière générale

$$x_i = \frac{1}{a'_{ii}}(b_i - \sum_{j=i+1}^{N} a'_{ij} x_j) \tag{6.10}$$

Cette méthode est avantageuse dans le cas des systèmes linéaires car le nombre d'opérations à effectuer est d'ordre $N^2$, contrairement à la méthode précédente qui nécessite de calculer complètement l'inverse de la matrice et nécessite un temps de calcul en $N^3$.

S'il faut résoudre un ensemble de systèmes linéaires comprenant $N$ termes, cela est alors équivalent à résoudre l'inversion d'une matrice et l'on retrouve un temps de calcul de l'ordre de $N^3$.

## 6.4 Décomposition LU

Les méthodes précédentes nécessitent de connaître à l'avance le membre de gauche de l'équation (6.1). Les méthodes qui suivent consiste à réécrire la matrice $A$ afin que la résolution du système d'équations soit exécutée plus facilement.

### 6.4.1 Principe

Nous allons montrer que toute matrice $N \times N$ peut se décomposer de la manière suivante.

$$A = L.U \tag{6.11}$$

où $L$ est une matrice triangulaire inférieure et $U$ une matrice triangulaire supérieure, soit

$$L = \begin{pmatrix} \alpha_{11} & 0 & 0 & \dots & 0 \\ \alpha_{21} & \alpha_{22} & 0 & \dots\dots\dots & 0 \\ \vdots & \vdots & \ddots & \dots & \vdots \\ \vdots & \vdots & \vdots & \alpha_{(N-1)(N-1)} & 0 \\ \alpha_{N1} & \alpha_{N2} & \dots & \dots & \alpha_{NN} \end{pmatrix} \tag{6.12}$$

et

$$U = \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \dots & \beta_{1N} \\ 0 & \beta_{22} & \beta_{23} & \dots\dots\dots & \beta_{2N} \\ \vdots & \vdots & \ddots & \dots & \vdots \\ \vdots & \vdots & \vdots & \beta_{(N-1)(N-1)} & \beta_{(N-1)N} \\ 0 & 0 & \dots & 0 & \beta_{NN} \end{pmatrix} \tag{6.13}$$

En effectuant la multiplication matricielle de $L$ par $U$, on obtient les relations suivantes

$$a_{ij} = \sum_{l=1}^{i} \alpha_{il}\beta_{lj} \qquad i \leq j \tag{6.14}$$

$$a_{ij} = \sum_{l=1}^{j} \alpha_{il}\beta_{lj} \qquad i > j \tag{6.15}$$

Ce système d'équations linéaires donne $N^2$ équations et le nombre d'inconnues est $2\left(\frac{N(N+1)}{2}\right)$. Ce système est donc surdéterminé. On peut donc choisir $N$ équations supplémentaires afin d'obtenir une et une seule solution. On fixe donc la valeur de la diagonale de la matrice $L$

$$\alpha_{ii} = 1 \tag{6.16}$$

pour $i \in [1, N]$.

L'algorithme de Crout permet de calculer simplement les $N^2$ coefficients restants, en utilisant les relations suivantes

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj} \qquad i \leq j \tag{6.17}$$

$$\alpha_{ij} = \frac{1}{\beta_{jj}}(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj}) \qquad i \geq j+1 \tag{6.18}$$

Il faut noter que les sous-programmes créant cette décomposition s'appuie sur la recherche du meilleur pivot afin que la méthode soit stable.[2]

### 6.4.2 Résolution d'un système linéaire

La résolution d'un système linéaire devient très simple en introduisant le vecteur $y$.

$$A.x = L.U.x \tag{6.20}$$

$$= L.(Ux) = b \tag{6.21}$$

Soit

$$L.y = b \tag{6.22}$$

[2]Sachant que le nombre d'inconnues est devenu égale à $N^2$, on peut écrire les deux matrices sous la forme d'une seule matrice de la forme

$$\begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \dots & \beta_{1N} \\ \alpha_{21} & \beta_{22} & \beta_{23} & \dots\dots\dots & \beta_{2N} \\ \vdots & \vdots & \ddots & \dots & \vdots \\ \vdots & \vdots & \vdots & \beta_{(N-1)(N-1)} & 0 \\ \alpha_{N1} & \alpha_{N2} & \dots & \dots & \beta_{NN} \end{pmatrix} \tag{6.19}$$

en se rappelant que les éléments diagonaux de la matrice $\alpha$ ont été choisis égaux à un.

$$U.x = y \tag{6.23}$$

Chaque système peut être résolu par une procédure de substitution.

$$y_1 = \frac{b_1}{\alpha_{11}} \tag{6.24}$$

$$y_i = \frac{1}{\alpha_{ii}}(b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j) \tag{6.25}$$

(substitution à partir du premier élément car la matrice est triangulaire inférieure). On obtient la solution pour le vecteur $x$ en utilisant

$$x_N = \frac{y_N}{\beta_{NN}} \tag{6.26}$$

$$x_i = \frac{1}{\beta_{ii}}(y_i - \sum_{j=i+1}^{N} \beta_{ij} x_j) \tag{6.27}$$

car la matrice est triangulaire supérieure.

Une fois effectuée la décomposition $LU$, le calcul du déterminant d'une matrice devient très simple. On sait que le déterminant d'un produit de matrices est égale au produit des déterminants. De plus pour une matrice triangulaire, le déterminant de la matrice est égal au produit des éléments de sa diagonale. Comme les éléments de la diagonale de $L$ ont été choisis égaux à 1, le déterminant de cette matrice est donc égal à 1, et le déterminant de $A$ est donc égal à

$$det(A) = \prod_{j=1}^{N} \beta_{jj} \tag{6.28}$$

## 6.5 Matrices creuses

### 6.5.1 Introduction

On appelle matrice creuse une matrice dont la plupart des éléments sont égaux à zéro. Si de plus, la structure des éléments non nuls est simple, il n'est pas nécessaire de réserver une quantité de mémoire égale à celle de la matrice complète. Des algorithmes spécifiques permettent de réduire le temps de calcul de manière considérable. Parmi les cas simples de matrices creuses, citons les matrices

- tridiagonales : les éléments de matrice non nuls sont sur la diagonale et de part et d'autre de celle-ci sur les deux lignes adjacentes. On a $a_{ij} = 0$ pour $|i - j| > 1$,
- diagonales par bande de largeur $M$ : les éléments de matrice tels que $|i - j| > M$ sont nuls $a_{ij} = 0$,
- simplement ou doublement bordées : par rapport à la définition précédente, des éléments non nuls supplémentaires existent le long des lignes ou colonnes du bord de la matrice.

### 6.5.2 Matrices tridiagonales

Soit le système suivant

$$\begin{pmatrix} b_1 & c_1 & 0 & \dots & 0 \\ a_2 & b_2 & c_2 & \dots & 0 \\ 0 & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \vdots & b_{N-1} & c_{N-1} \\ 0 & 0 & \dots & \dots & b_{NN} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_N \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_N \end{pmatrix} \tag{6.29}$$

Noter qu'avec cette notation $a_1$ et $c_N$ ne sont pas définis. Il y a un vaste de choix de bibliothèques disponibles pour calculer les solutions qui prennent un temps de calcul proportionnel à $N$. De manière générale, on peut obtenir aussi avoir un algorithme proportionnel à $N$ pour une matrice à bandes. Le préfacteur de l'algorithme est proportionnel à $M$.

### 6.5.3 Formule de Sherman-Morison

Supposons que l'on ait une matrice $A$ dont on a facilement calculé l'inverse (cas d'une matrice tridiagonal). Si on fait un petit changement dans $A$ en modifiant par exemple un ou quelques éléments de l'ensemble de la matrice, peut-on calculer encore facilement l'inverse de cette nouvelle matrice ?

$$B = A + u \otimes v \tag{6.30}$$

ou le symbole $\otimes$ désigne le produit extérieur. $u \otimes v$ représente une matrice dont l'élément $ij$ est le produit de la $i$ème composante de $u$ par la $j$ème composante de $v$.

La formule de Sherman-Morison donne l'inverse de $B$

$$\begin{aligned} B^{-1} &= (1 + A^{-1}.u \otimes v)^{-1}.A^{-1} && (6.31)\\ &= (1 - A^{-1}.u \otimes v + A^{-1}.u \otimes v.A^{-1}.u \otimes v - \dots).A^{-1} && (6.32)\\ &= A^{-1} - A^{-1}.u \otimes vA^{-1}(1 - \lambda + \lambda^2 + \dots) && (6.33)\\ &= A^{-1} - \frac{(A^{-1}.u) \otimes (v.A^{-1})}{1 + \lambda} && (6.34) \end{aligned}$$

où $\lambda = v.A^{-1}u$. On a utilisé l'associativité du produit extérieur et produit interne.

Posons $z = A^{-1}u$ et $w = (A^{-1})^T v$, on a $\lambda = v.z$ et

$$B^{-1} = A^{-1} - \frac{z \otimes w}{1 + \lambda} \tag{6.35}$$

## 6.6 Décomposition de Choleski

Une matrice est définie positive symétrique ($a_{ij} = a_{ji}$) quand $\forall x$, on a

$$x.A.x > 0 \tag{6.36}$$

quel que soit $x$. Il existe alors une matrice $L$ triangulaire inférieure telle que

$$A = L.L^T \tag{6.37}$$

Les éléments de matrice sont déterminés par les relations

$$L_{ii} = (a_{ii} - \sum_{k=1}^{i-1} L_{ik}^2)^{1/2} \tag{6.38}$$

$$L_{ji} = \frac{1}{L_{ii}}(a_{ij} - \sum_{k=1}^{i-1} L_{ik}L_{jk}) \tag{6.39}$$

avec $j \in [i+1, N]$. Une fois cette construction réalisée, on peut résoudre un système linéaire simplement en utilisant la procédure de substitution précédemment détaillée dans la section 6.4.2.

## 6.7 Conclusion

Avant de résoudre des systèmes linéaires de grande dimension, il est impératif de commencer par une analyse des propriétés de la matrice afin de déterminer la méthode la plus adaptée afin d'obtenir une solution avec une précision correcte et pour un temps de calcul qui sera minimal. Les différentes méthodes présentées dans ce chapitre ne sont qu'une introduction à ce très vaste sujet.

# Chapitre 7

# Analyse spectrale

**Contenu**



## 7.1 Introduction

Ce chapitre est consacré aux opérations que l'on peut réaliser sur des matrices. Plus spécifiquement, nous allons nous intéresser à la détermination des valeurs propres et/ou vecteurs propres correspondants. Tout d'abord, quelques rappels utiles pour la suite de ce chapitre : soit une matrice carrée $A$ de dimension $N$, on appelle un vecteur propre $x$ associé à la valeur propre $\lambda$, un vecteur qui satisfait la relation

$$A.x = \lambda x \tag{7.1}$$

Si $x$ est un vecteur propre, pour tout réel $\alpha \neq 0$, $\alpha x$ est aussi un vecteur propre avec la même valeur propre $\lambda$.

Les valeurs propres d'une matrice peuvent être déterminées comme les racines du polynôme caractéristique de degré $N$

$$det(A - \lambda.1) = 0 \tag{7.2}$$

où 1 désigne la matrice identité.

Compte tenu du fait que dans $\mathbb{C}$, tout polynôme de degré $N$ a $N$ racines, la matrice $A$ possède $N$ valeurs propres complexes.

Quand une racine du polynôme caractéristique est multiple, on dit que la valeur propre est dégénérée, et la dimension de l'espace associé à cette valeur propre est supérieure ou égale à deux.

La détermination des valeurs propres d'une matrice à partir de l'équation caractéristique n'est pas efficace sur le plan numérique. Des méthodes plus adaptées sont exposées dans la suite de ce chapitre.

Si une matrice possède une valeur propre nulle, la matrice est dite singulière.

– Une matrice est symétrique si elle est égale à sa transposée

$$A = A^T \tag{7.3}$$

$$a_{ij} = a_{ji} \quad \forall i, j \tag{7.4}$$

– Une matrice est hermitienne ou auto-adjointe si elle est égale au complexe conjugué de sa transposée.

$$A = A^\dagger \tag{7.5}$$

– Une matrice est orthogonale si son inverse est égale à sa transposée

$$A.A^T = A^T.A = 1 \tag{7.6}$$

– Une matrice est unitaire si sa matrice adjointe est égale à son inverse

$$A.A^\dagger = A^\dagger.A = 1 \tag{7.7}$$

Pour les matrices à coefficients réels, il y a identité de définition entre matrice symétrique et Hermitienne, entre matrice orthogonale et unitaire.

Une matrice est dite normale si elle commute avec son adjointe.

## 7.2 Propriétés des matrices

Les valeurs propres d'une matrice Hermitienne sont toutes réelles. Parmi les matrices Hermitiennes très utilisées en Physique, il vient à l'esprit la représentation matricielle de l'opérateur de Schrödinger en mécanique quantique[1]

Un corollaire de la propriété précédente est que les valeurs propres d'une matrice réelle symétrique sont elles aussi toutes réelles.

Les vecteurs propres d'une matrice normale ne possédant que des valeurs propres non dégénérées forment une base d'un espace vectoriel de dimension $N$. Pour une matrice normale avec des valeurs propres dégénérées, les vecteurs propres correspondant à une valeur propre dégénérée peuvent être remplacés par une combinaison linéaire de ceux-ci.

Pour une matrice quelconque, l'ensemble des vecteurs propres ne constituent pas nécessairement une base d'un espace de dimension $N$.

Pour une matrice non normale, les vecteurs propres ne sont pas orthogonaux. On appelle vecteur à droite les vecteurs tels que

$$A.x_i^R = \lambda_i x_i^R \tag{7.8}$$

[1] La représentation de cet opérateur correspond en général à une matrice de dimension infinie, mais nous ne considérons ici que les systèmes ou la représentation matricielle est possible dans un espace de dimension finie.

où $\lambda_i$ est la $i$ème valeur propre. De manière similaire, on appelle vecteurs propres à gauche, les vecteurs tels que

$$x_i^L.A = \lambda_i x_i^L \tag{7.9}$$

Le transposé du vecteur à gauche de $A$ est le vecteur propre à droite de la transposée de la même matrice. Si la matrice est symétrique, les vecteurs propres à gauche sont les transposés des vecteurs propres à droite[2].

Si la matrice est Hermitienne, les vecteurs propres à gauche sont les transposés des vecteurs propres conjugués à droite.

Dans le cas d'une matrice non normale, on définit la matrice $X_R$ comme la matrice constituée de colonnes formées par les vecteurs à droite. On introduit la matrice $X_L$ formée par les lignes des vecteurs à gauche. On obtient par définition que

$$A.X_R = X_R.diag(\lambda_1, \ldots, \lambda_n) \tag{7.10}$$

de même on a

$$X_L.A = diag(\lambda_1, \ldots, \lambda_n).X_L \tag{7.11}$$

En multipliant l'équation (7.10) par $X_L$ et l'équation (7.11) par $X_R$, on obtient

$$X_L.X_R.diag(\lambda_1, \ldots, \lambda_n) = diag(\lambda_1, \ldots, \lambda_n)X_L.X_R \tag{7.12}$$

ce qui montre que la matrice diagonale formée par les valeurs propres de $A$ commute avec le produit $X_L.X_R$. Sachant que les seules matrices qui commutent avec une matrice diagonale constituée d'éléments différents sont elles-mêmes diagonales, on en déduit que chaque vecteur à gauche est orthogonal à chaque vecteur à droite et réciproquement. En normalisant les vecteurs à droite et à gauche, on peut obtenir que la matrice $X_L.X_R$ soit égale à l'identité.

Dans le cas où l'ensemble des vecteurs propres ne constitue une base complète, il est toujours possible de compléter cet ensemble afin d'avoir une matrice telle que $X_L.X_R = 1$.

Si la matrice $A$ est inversible, on obtient en multipliant l'équation (7.10) par $X_R^{-1}$ que

$$X_R^{-1}.A.X_R = diag(\lambda_1, \ldots, \lambda_n) \tag{7.13}$$

Nous avons alors construit une matrice de transformation similaire de $A$

Rappelons la propriété suivante : soit $B$ une matrice telle que

$$B = P^{-1}.A.P \tag{7.14}$$

où $P$ est une matrice inversible. On a

$$\begin{aligned} det(B - \lambda 1) &= det(P^{-1}.A.P - \lambda.1) && (7.15)\\ &= det(P^{-1}.(A - \lambda.1)P) && (7.16)\\ &= det(A - \lambda.1) && (7.17) \end{aligned}$$

[2]Puisque le déterminant d'une matrice et de sa transposée sont les mêmes, les valeurs propres de ces deux matrices sont identiques.

Ainsi, on a montré que l'on peut construire une matrice de transformation similaire où la matrice $A$ devient diagonale dans cette nouvelle base.

Pour une matrice réelle symétrique, la matrice de passage est une matrice orthogonale.

La stratégie générale pour déterminer les valeurs propres d'une matrice consiste à construire une suite de transformations de similarité jusqu'à l'obtention d'une matrice diagonale, ou plus simplement jusqu'à l'obtention d'une matrice tridiagonale à partir de laquelle il est possible de déterminer assez facilement les valeurs propres.

Pour réaliser ces transformations, deux grandes classes de méthodes sont disponibles : les méthodes directes et les méthodes itératives.

Les premières consistent à effectuer une suite de transformations similaires et ne s'appliquent qu'aux matrices de taille relativement modeste, car le temps de calcul croît comme le cube de la dimension linéaire de la matrice.

Pour les matrices de grande taille et généralement creuses, les méthodes itératives sont plus adaptées. Dans la mesure où la plupart du temps, le spectre complet d'une très grande matrice n'est pas recherchée, mais seulement une partie, les méthodes itératives peuvent réaliser plus efficacement cette tâche. Nous allons voir dans la suite de ce chapitre quelques algorithmes de base, tout en ayant à l'esprit qu'il existe une très vaste littérature sur ce sujet, et pour un problème particulier, il est nécessaire de commencer par analyser précisément le type de matrice dont on souhaite obtenir le spectre, afin de choisir la méthode la plus adaptée pour résoudre ce problème.

## 7.3 Méthodes directes

### 7.3.1 Méthode de Jacobi

La méthode de Jacobi revient à effectuer une suite de transformations similaires orthogonales. Chaque transformation est une simple rotation planaire qui permet d'annuler un élément de la matrice $A$ initial.

La rotation élémentaire $P_{pq}$ est donné par la matrice

$$P_{pq} = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & 1 & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & c & \dots & \dots & s & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & \dots & \dots & 1 & \dots & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & -s & \dots & \dots & c & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 1 & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & \dots & 1 \end{pmatrix} \tag{7.18}$$

avec la condition que

$$c^2 + s^2 = 1. \tag{7.19}$$

Soit la matrice $A'$ telle que

$$A' = P_{pq}^T . A . P_{pq} \tag{7.20}$$

En notant les coefficients de la matrice $a_{ij}$, on obtient après calculs que

$$a'_{rp} = ca_{rp} - sa_{rq} \tag{7.21}$$
$$a'_{rq} = ca_{rq} + sa_{rp} \tag{7.22}$$
$$a'_{pp} = c^2 a_{pp} + s^2 a_{qq} - 2csa_{pq} \tag{7.23}$$
$$a'_{qq} = s^2 a_{pp} + c^2 a_{qq} + 2csa_{pq} \tag{7.24}$$
$$a'_{pq} = (c^2 - s^2)a_{pq} + cs(a_{pp} - a_{qq}) \tag{7.25}$$

avec $r \neq p$ et $r \neq q$.

Si on annule le terme $a'_{pq}$, en introduisant l'angle de rotation $\phi$, ($c = \cos(\phi)$, $s = \sin(\phi)$) on a le rapport $\theta$

$$\theta = \frac{c^2 - s^2}{2sc} \tag{7.26}$$
$$= \cot(2\phi) \tag{7.27}$$
$$= \frac{a_{qq} - a_{pp}}{a_{pq}} \tag{7.28}$$

Si on appelle $t = s/c$, on obtient en utilisant l'équation (7.26)

$$t^2 + 2t\theta - 1 = 0 \tag{7.29}$$

La plus petite des racines correspond à un angle de rotation inférieur à $\pi/4$ et donne la méthode la plus stable numériquement. Cette racine[3] peut s'exprimer sous la forme

$$t = \frac{sgn(\theta)}{|\theta| + \sqrt{\theta^2 + 1}} \tag{7.30}$$

En utilisant que $c^2 + s^2 = 1$„ on obtient pour $c$ que

$$c = \frac{1}{\sqrt{1 + t^2}} \tag{7.31}$$

et on a immédiatement $s = tc$. En imposant que le terme $a'_{pq}$ s'annule, on a finalement les relations suivantes

$$a'_{pp} = a_{pp} - ta_{pq} \tag{7.32}$$
$$a'_{qq} = a_{qq} + ta_{pq} \tag{7.33}$$
$$a'_{rp} = a_{rp} - s(a_{rq} + \tau a_{rq}) \tag{7.34}$$
$$a'_{rq} = a_{rq} + s(a_{rp} + \tau a_{rq}) \tag{7.35}$$

avec $\tau$ défini par

$$\tau = \frac{s}{1 + c} \tag{7.36}$$

En calculant la somme $S$

$$S = \sum_{r \neq s} |a_{rs}|^2 \tag{7.37}$$

[3]Pour éviter les dépassements de capacité de l'ordinateur, on choisit $t = 1/2\theta$

on peut obtenir une estimation de la convergence de la méthode. Pour une transformation similaire élémentaire, on a

$$S' = S - 2|a_{pq}|^2 \tag{7.38}$$

Ainsi la suite des transformations conduit à faire décroître la contribution des éléments non diagonaux. Comme la transformation est orthogonale, la somme des carrés des éléments de la matrice est conservée, ce qui revient à ce que la somme des carrés de la diagonale augmente de ce qui a été perdu par les éléments non diagonaux. Ainsi formellement, on peut choisir les éléments de la matrice $A$ dans n'importe quel ordre et on obtient une méthode qui converge vers une matrice diagonale. Au terme de cette procédure, on a

$$D = V^T.A.V \tag{7.39}$$

où $D$ est une matrice diagonale contenant les différentes valeurs propres et $V$ est une matrice contenant les vecteurs propres correspondants.

### 7.3.2 Réduction de Householder

La méthode précédente est très coûteuse en temps de calcul. Pour réduire celui-ci, la procédure de Householder se propose de transformer une matrice symétrique en une matrice tridiagonale par une série de transformations orthogonales suivantes.

Une matrice de Householder est définie par la relation suivante

$$P = 1 - 2w.w^T \tag{7.40}$$

où $w$ est un vecteur réel normalisé, $w^T.w = |w|^2 = 1$.

Vérifions que la matrice $P$ est une matrice orthogonale

$$P^2 = (1 - 2w.w^T).(1 - 2w.w^T) \tag{7.41}$$

$$= 1 - 4w.w^T + 4w.(w^T.w).w^T \tag{7.42}$$

$$= 1 \tag{7.43}$$

Cela implique que $P = P^{-1}$. En utilisant la définition de $P$, on vérifie facilement que $P^T = P$, et on a donc bien construit une transformation orthogonale.

Nous allons maintenant appliquer à $P$ le vecteur $x$ constitué de la première colonne de $A$. Pour cela, on exprime la matrice $P$ sous la forme suivante

$$P = 1 - \frac{u.u^T}{H} \tag{7.44}$$

avec $H = |u|^2/2$. Si on choisit le vecteur $u$ tel que

$$u = x \mp |x|e_1 \tag{7.45}$$

où $e_1$ est le vecteur colonne unitaire tel que seule la première composante est non nulle et égale à 1. On obtient facilement la valeur de $H$

$$H = 2(|x^2| \pm |x|x_1) \tag{7.46}$$

En appliquant $P$ à $x$

$$P.x = x - \frac{u}{H}.(x \mp |x|e_1)^T.x \tag{7.47}$$

$$= x - \frac{2u.(|x|^2 \mp |x|x_1)}{2|x|^2 \mp 2|x|x_1} \tag{7.48}$$

$$= x - u \tag{7.49}$$

$$= \pm\, |x|e_1 \tag{7.50}$$

Cela montre que la matrice $P$ annule tous les éléments du vecteur $x$ hormis le premier.

La stratégie pour construire les matrices de Householder est la suivante : on choisit un vecteur $x$ constitué des $n-1$ derniers éléments de la première colonne pour construire la matrice $P_1$. En conséquence, on obtient la structure suivante pour $P_1$

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & & & & & \\ 0 & & & & & \\ 0 & & & & & \\ \vdots & & & P_1^{(n-1)} & & \\ 0 & & & & & \\ 0 & & & & & \end{pmatrix} \tag{7.51}$$

En appliquant la transformation orthogonale à la matrice $A$, on obtient

$$A' = P.A.P \tag{7.52}$$

$$= \begin{pmatrix} a_{11} & k & 0 & \dots & \dots & 0 \\ k & & & & & \\ 0 & & & & & \\ 0 & & & & & \\ \vdots & & & & & \\ 0 & & & & & \\ 0 & & & & & \end{pmatrix} \tag{7.53}$$

où le nombre $k$ est au signe près la norme du vecteur $(a_{21}, \dots, a_{n1})^T$.

On choisit la seconde matrice de Householder avec un vecteur $x$ qui constitué avec les $(n-2)$ derniers éléments de la seconde colonne

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & 0 \\ 0 & 0 & & & & \\ 0 & 0 & & & & \\ \vdots & \vdots & & P_2^{(n-2)} & & \\ 0 & 0 & & & & \\ 0 & 0 & & & & \end{pmatrix} \tag{7.54}$$

La tridiagonalisation de la partie supérieure de la matrice précédente est préservée, on construit ainsi colonne par colonne une tridiagonalisation de la matrice

$A$. La procédure est complète après $(n-2)$ transformations similaires de Householder.

Pratiquement, pour éviter la multiplication de matrices, très coûteuse numériquement, on peut exprimer le produit $P.A.P$ en introduisant la notation suivante

$$p = \frac{A.u}{H} \tag{7.55}$$

En conséquence, la première multiplication matricielle peut être écrite comme

$$A.P = A.(1 - \frac{u.u^T}{H}) \tag{7.56}$$

$$= A - p.u^T \tag{7.57}$$

de même pour la seconde

$$A' = P.A.P = A - p.u^T - u.p^T + 2Ku.u^T \tag{7.58}$$

avec

$$K = \frac{u^T.p}{2H} \tag{7.59}$$

En posant

$$q = p - Ku \tag{7.60}$$

on a la matrice $A'$ qui s'exprime alors simplement

$$A' = A - q.u^T - u.q^T \tag{7.61}$$

### 7.3.3 Algorithme QL

En utilisant une suite de transformations de Householder, on peut écrire tout matrice réelle sous la forme

$$A = Q.R \tag{7.62}$$

où $Q$ est une matrice orthogonale et $R$ une matrice triangulaire supérieure. Pour obtenir une telle décomposition, les matrices de Householder sont construites dans ce cas de manière à ce que la première colonne ne possède que le premier élément non nul une fois la transformation effectuée, pour la deuxième colonne, on choisit le vecteur de la matrice de Householder pour que tous les éléments de la matrice transformée soient nuls sous la diagonale et ainsi de suite jusqu'à former une matrice triangulaire supérieure. Le nombre de matrices de Householder pour obtenir ce résultat est égal à $(n-1)$.

De manière analogue, on peut montrer qu'il existe une décomposition de la forme

$$A = Q.L \tag{7.63}$$

. où $Q$ est une matrice orthogonale et $L$ une matrice triangulaire inférieure. Les transformations de Householder correspondantes consistent à annuler, colonne par colonne, les éléments de la matrice transformée qui sont situés au dessus de la diagonale.

Si on définit la matrice $A'$ comme

$$A' = L.Q \tag{7.64}$$

Puisque $Q$ est orthogonal, on en déduit de l'équation (7.63) que

$$L = Q^{-1}.A \tag{7.65}$$

ce qui donne

$$A' = Q^T.A.Q \tag{7.66}$$

ce qui montre que $A'$ est une transformation orthogonale de $A$.

Pour des raisons de minimisation d'erreurs d'arrondi, il est préférable d'utiliser la décomposition $QL$ au lieu de la décomposition $QR$.

L'algorithme $QL$ est défini par la suite suivante

$$A_s = Q_s.L_s \tag{7.67}$$
$$A_{s+1} = L_s.Q_s \tag{7.68}$$

La méthode repose sur les bases suivantes : (i) Si $A$ a des valeurs propres toutes distinctes de valeur absolue $|\lambda_i|$, alors $A_s$ tend vers une matrice triangulaire inférieure quand $s \to \infty$. Les valeurs propres apparaissent sur la diagonale par valeur absolue croissante (ii) Si $A$ a une valeur propre dégénérée de multiplicité $p$,quand $s \to \infty$, $A_s$ tend vers une matrice triangulaire inférieure, exceptée pour un bloc d'ordre $p$ correspondant à la valeur propre dégénérée. Pour une matrice quelconque, une itération a un coût de calcul proportionnel à $n^3$, mais pour une matrice tridiagonale, ce coût est linéaire avec $n$. On peut montrer que la convergence dépend de la différence entre deux valeurs propres successives. Quand deux valeurs propres sont trop proches, il est quand même possible d'améliorer la convergence de l'algorithme en déplaçant ces valeurs propres successives.

### 7.3.4 Factorisation de Schur

La factorization de Schur consiste à réécrire une matrice carrée $A$ sous la forme suivante

– Si la matrice $A$ est complexe

$$A = ZTZ^\dagger \tag{7.69}$$

où $Z$ est unitaire et $T$ est une matrice triangulaire supérieure.

– Si la matrice $A$ est réelle

$$A = ZTZ^T \tag{7.70}$$

où $Z$ est orthogonale et $T$ est une matrice quasi-triangulaire supérieure, ce qui signifie que la diagonale est constituée soit de blocs $1 \times 1$ soit de blocs $2 \times 2$.

Les colonnes de $Z$ sont appelées les vecteurs de Schur. Les valeurs propres de $A$ apparaissent sur la diagonale de $T$ ; les valeurs propres complexes conjuguées d'une matrice $A$ réelle correspondent aux blocs $2x2$ de la diagonale.

L'algorithme utilisée dans la bibliothèque LAPACK, on commence par transformer la matrice $A$ en la transformant en une matrice de Hessenberg, qui est une matrice triangulaire supérieure bordée une ligne d'éléménts nuls sous la diagonale.

– Si la matrice $A$ est complexe

$$A = QHQ^{\dagger} \tag{7.71}$$

où $Q$ est unitaire et $H$ est une matrice de Hessenberg.

– Si la matrice $A$ est réelle

$$A = QTQ^{T} \tag{7.72}$$

où $Q$ est orthogonale et $H$ est une matrice de Hessenberg.

Dans une deuxième étape, on transforme la matrice de Hessenberg en une matrice de Schur.

## 7.4 Méthode itératives

Ces méthodes sont principalement appliquées à des matrices de grande taille et largement creuses. Les calculs croissent dans ce cas linéairement avec $n$. Pour toute méthode itérative, il convient de s'assurer que la convergence est suffisamment rapide pour que le temps de calcul ne soit pas consommé sans que la recherche d'une solution ne soit réellement effectuée.

### 7.4.1 Méthodes des puissances

Le principe de cette méthode est très simple, car il repose sur le fait qu'en appliquant un grand nombre de fois la matrice sur un vecteur initial quelconque, les vecteurs successifs vont prendre une direction qui se rapproche du vecteur propre de la plus grande valeur propre (en valeur absolue). Le principe itératif de cette méthode est la suivante : soit $x_0$ un vecteur initial quelconque, et $A$ la matrice dont on cherche à déterminer la plus grande valeur propre. On effectue l'opération suivante

$$x_1 = A.x_0 \tag{7.73}$$

Si on désigne $\alpha$ comme l'angle entre ces deux vecteurs, on a

$$\cos(\alpha) = \frac{x_1.x_0}{|x_1|.|x_0|} \tag{7.74}$$

Si $x_0$ n'est pas perpendiculaire au vecteur recherché (ce qui est rarement le cas pour un vecteur choisi au départ aléatoirement), le cosinus de l'angle entre $x_0$ et $x_1$ est différent de zéro. En appliquant à nouveau la matrice $A$ sur le vecteur $x_1$ on crée un vecteur $x_2$ et on calcule l'angle entre $x_1$ et $x_2$ qui est inférieur au précédent en valeur absolue. On continue jusqu'à ce que l'angle entre deux vecteurs successifs devienne plus petit qu'une nombre $\epsilon$ choisi initialement.

On en déduit alors le vecteur propre recherché et donné par $x_n$, ainsi que la valeur propre correspondante. La convergence de cette méthode varie comme $(\lambda_1/\lambda_2)$ ce qui peut devenir assez lent quand les valeurs propres deviennent quasi-dégénérées. Cette méthode n'est pas très efficace, mais possède le mérite de s'écrire rapidement.

### 7.4.2 Méthode de Lanczòs

La méthode de Lanczòs consiste à la fois à calculer les puissances successives de $A$, en s'inspirant de la méthode précédente, mais de manière bien plus efficace en construisant un ensemble de vecteurs orthogonaux. Par simplicité, nous allons voir la méthode pour des matrices hermitiennes, mais la méthode peut être étendue pour des matrices plus générales, en particulier quand les vecteurs à gauche diffèrent des vecteurs à droite.

Soit un vecteur de départ normalisé $u_0$ : ce vecteur est convenablement choisi c´est à dire que sa projection sur la base de la valeur propre à déterminer est non nulle. On construit les vecteurs successifs de la base dite de Krylov à partir de la relation

$$\beta_2 u_1 = A.u_0 - \alpha_1 u_0 \tag{7.75}$$

où $\beta_2$ et $\alpha_1$ sont des constantes déterminées de la manière suivante : $\alpha_1$ est choisi de manière à ce que le vecteur $u_1$ soit orthogonal au vecteur $u_0$

$$\alpha_1 = u_0^T.A.u_0 \tag{7.76}$$

et $\beta_2$ est déterminé de manière à ce que le vecteur $u_1$ soit normalisé.

$$\beta_2 = \sqrt{u_0^T.A^2.u_0 - \alpha_1^2} \tag{7.77}$$

Pour le vecteur suivant $u_2$, on utilise la relation :

$$\beta_3 u_2 = A.u_1 - \alpha_2 u_1 - \beta_2 u_0 \tag{7.78}$$

On note tout d'abord qu'avec cette construction, le vecteur $u_2$ est orthogonal au vecteur $u_0$. De manière similaire, on impose que $u_2$ soit un vecteur orthogonal à $u_1$ ce qui conduit à la relation

$$\alpha_2 = u_1^T.A.u_1 \tag{7.79}$$

et la normalisation de $u_2$ est imposée en choisissant $\beta_3$ comme

$$\beta_3 = \sqrt{u_1^T.A^2.u_1 - \alpha_2^2 - \beta_2} \tag{7.80}$$

Par itération, on obtient pour le $i$ème vecteur

$$\beta_{i+1} u_i = A.u_{i-1} - \alpha_i u_{i-1} - \beta_i u_{i-2} \tag{7.81}$$

On vérifie facilement que tous les vecteurs $u_j$ avec $j < i-2$ sont orthogonaux avec le vecteur $u_{i+1}$ en raison de la construction dans un sous espace orthonormé sur ses vecteurs. Les valeurs $\alpha_i$ et $\beta_{i+1}$ sont déterminées par les relations suivantes

$$\alpha_i = u_{i-1}^T.A.u_{i+1} \tag{7.82}$$

$$\beta_{i+1} = \sqrt{u_{i-1}^T.A^2.u_{i-1} - \alpha_i^2 - \beta_i} \tag{7.83}$$

Dans cette base, la matrice $A'$ est tridiagonale et est donnée

$$A' = \begin{pmatrix} \alpha_1 & \beta_2 & 0 & \dots & \dots & \dots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & 0 & \dots & \dots & 0 \\ 0 & \beta_3 & \alpha_3 & \beta_4 & 0 & \dots & 0 \\ 0 & \dots & & \dots & & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & \dots & 0 & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ 0 & \dots & \dots & \dots & 0 & \beta_n & \alpha_n \end{pmatrix} \tag{7.84}$$

Il est possible d´utiliser alors une méthode de type QL pour obtenir rapidement les valeurs propres de la matrice $A'$, c´est à dire aussi la matrice $A$.

Quelques remarques : en construisant progressivement une base de vecteurs de plus en plus grande, on voit que la valeur propre recherchée peut être estimée en utilisant les sous espaces successifs. On peut donc arrêter l´itération quand la différence entre deux estimations successives de la valeur propre est devenue suffisamment petite.

Dans le cas où la matrice $A$ est une représentation tronquée d´un opérateur dont la base propre est infinie (par exemple, un opérateur de Schröringer), le procédé itératif conduit progressivement à des valeurs de $\beta_i$ de plus en en plus petites. On peut considérer que la base de Krylov est complète quand $\beta_n$ est devenu inférieure en valeur absolue à une valeur $\epsilon$ choisie à l´avance (généralement $10^{12}$).

# Chapitre 8

# Equations intégrales

**Contenu**



## 8.1 Introduction

Les équations intégrales sont a priori moins simples à résoudre que les équations algébriques ou les équations différentielles. Nous allons voir dans ce chapitre que pour des équations intégrales linéaires, une fois réalisée la discrétisation de ces équations, on se ramène au problème de la recherche de solutions d'un système linéaire que nous avons vu chapitre 6.

## 8.2 Equation de Fredholm

### 8.2.1 Equation de première espèce

L'équation de Fredholm inhomogène de première espèce est définie par la relation suivante

$$\int_a^b K(t,s)f(s)ds = g(t) \tag{8.1}$$

où $f(t)$ est la fonction inconnue que l'on souhaite déterminer. $g(t)$ est le terme de source et $K(t,s)$ est appelé le noyau.

En notant $g_i = g(t_i)$, $K_{ij} = K(s_i, t_j)$ et $f_j = f(t_j)$ où $i$ est un indice variant de 1 à $N$ et $j$ un indice variant de 1 à $M$ ($M$ peut être différent de $N$).

L'équation (8.1) se réécrit alors comme

$$\sum_{j=1}^{M} K_{ij} f_j = g_i \tag{8.2}$$

Soit encore sous une forme matricielle

$$K.f = g \tag{8.3}$$

Formellement, si le noyau $K$ n'est pas singulier, la solution existe, est unique et est donnée par la relation

$$f = K^{-1}.g \tag{8.4}$$

### 8.2.2 Equation de seconde espèce

L'équation de Fredholm inhomogène de deuxième espèce est définie par la relation suivante

$$\lambda f(t) = \int_a^b K(t,s) f(s) ds + g(t) \tag{8.5}$$

où $f(t)$ est la fonction inconnue que l'on souhaite déterminer. $g(t)$ est le terme de source, $K(t,s)$ est appelé le noyau et $\lambda$ est un scalaire introduit par commodité pour la suite de cette équation.

Suivant le même principe que celui défini ci-dessus, une fois discrétisée, l'équation (8.5) se réexprime comme

$$\lambda f_i = \sum_{j=1}^{M} K_{ij} f_j + g_i \tag{8.6}$$

Soit encore sous une forme matricielle

$$(K - \lambda 1).f = -g \tag{8.7}$$

Si la fonction $g$ est nulle, le problème se ramène à la détermination des valeurs propres de la matrice $K$, et on parle d'équation de Fredholm homogène. Dans le cas où $g$ est différent de zéro, la solution existe et est unique si $\lambda$ n'est pas l'une des valeurs propres du noyau, sinon la matrice à inverser $K - \lambda.1$ devient singulière. Si cette dernière est inversible, on a formellement la solution comme

$$f = (\lambda.1 - K)^{-1}.g \tag{8.8}$$

La résolution numérique des équations de Fredholm homogène de première espèce est généralement délicate car le noyau correspond souvent à une matrice presque non inversible (mal conditionnée), c'est-à-dire avec un déterminant voisin de zéro. Corrélativement, si $\lambda$ est suffisament différent de zéro, la solution des équations de Fredholm de seconde espèce est relativement simple à obtenir.

## 8.3 Equation de Volterra

Les équations de Volterra sont des cas particuliers de ceux de Fredholm dans lesquelles le noyau $K$ est tel que

$$K(t,s) = 0 \qquad \text{pour } s > t \tag{8.9}$$

### 8.3.1 Equation de première espèce

L'équation de Volterra homogène de première espèce est définie par la relation suivante

$$g(t) = \int_a^t K(t,s) f(s) ds \tag{8.10}$$

où $f(t)$ est la fonction inconnue que l'on souhaite déterminer. $g(t)$ est le terme de source et $K(t,s)$ est appelé le noyau.

La réécriture matricielle de l'équation de Volterra est identique formellement à celle de Fredholm, mais avec la particularité que la matrice associée au noyau $K$ est une matrice triangulaire inférieure. Ce type d'équations linéaires, comme nous l'avons vu au chapitre 6, peut être résolu par une méthode de substitution. Alors que les équations de Fredholm de première espèce sont généralement mal conditionnées, les équations de Volterra ne le sont pas.

### 8.3.2 Equation de seconde espèce

De manière similaire, l'équation de Volterra de première espèce inhomogène s'écrit comme

$$\lambda f(t) = \int_a^t K(t,s) f(s) ds + g(t) \tag{8.11}$$

De manière identique, la représentation matricielle de l'équation de Volterra est identique à celle correspondante de Fredholm, seule la structure du noyau $K$ est différente, puisque comme pour toutes les équations de Volterra linéaires, la matrice $K$ triangulaire inférieure.

## 8.4 Conclusion

L'existence de structure de noyaux presque singuliers nécessite d'utiliser des méthodes plus complexes qui dépassent largement le cadre de ce cours introductif aux méthodes numériques.

# Chapitre 9

# Equations aux dérivées partielles

**Contenu**



## 9.1 Introduction

Les équations aux dérivées partielles interviennent dans de nombreux domaines de physique, qui comprennent les problèmes de diffusion, les phénomènes de propagation, ainsi que le domaine de la mécanique des fluides décrite par les équations hydrodynamiques comme celles de Navier-Stokes et l'équation de Schrödinger dépendante du temps pour la mécanique quantique. Ces équations différentielles n'ont généralement pas de solutions analytiques et une résolution numérique de ces équations est alors nécessaire.

Une équation aux dérivées partielles est une relation liant une fonction de $n$ variables à ses dérivées partielles. L'ordre de l'équation est donné par l'ordre le plus élévé des dérivées partielles apparaissant dans l'équation. La forme générale d'une équation aux dérivées partielles linéaires est

$$L[f(\mathbf{x})] = g(\mathbf{x}) \tag{9.1}$$

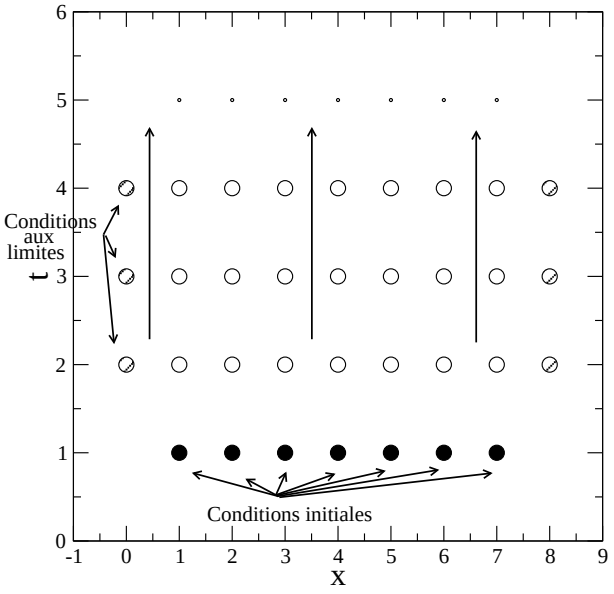


FIG. 9.1 – Schéma illustrant le principe de la résolution d'une équation aux dérivées partielles avec des conditions initiales et des conditions aux limites : La propagation de la solution le long de l'axe temporel (axe vertical) se fait par "tranche horizontale".

où un $\mathbf{x}$ est un vecteur de composante $(x_1, x_2, \ldots, x_n)$ et où $L$ est un opérateur défini par la relation

$$L = p_0(\mathbf{x}) + \sum_{i=1}^{n} p_i(\mathbf{x})\partial_i + \sum_{i,j=1}^{n} p_{ij}(\mathbf{x})\partial_i\partial_j + \ldots \tag{9.2}$$

Si $g(\mathbf{x}) = 0$, on dit que l'équation est homogène.

Dans ce cours d'introduction, nous allons nous limiter aux équations aux dérivées partielles linéaires. Une première classification non exhaustive fait apparaître trois types d'équations : équations hyperboliques, paraboliques et elliptiques. Cette classification provient de l'analyse des dérivées partielles d'ordre le plus élévé. Pour les équations d'ordre deux, une solution unique existe si la relation appeléé équation aux caractéristiques

$$\frac{p_{12} \pm \sqrt{p_{12}^2 - p_{11}p_{22}}}{p_{11}} \neq 0 \tag{9.3}$$

est vraie. Si la valeur du membre de gauche de l'équation (9.3) est réelle, on dit que l'on a une équation hyperbolique. Si la valeur est complexe, on dit que l'on a une équation elliptique et si elle est nulle, on a une équation parabolique.

On peut illustrer ces trois catégories par les exemples physiques suivants :

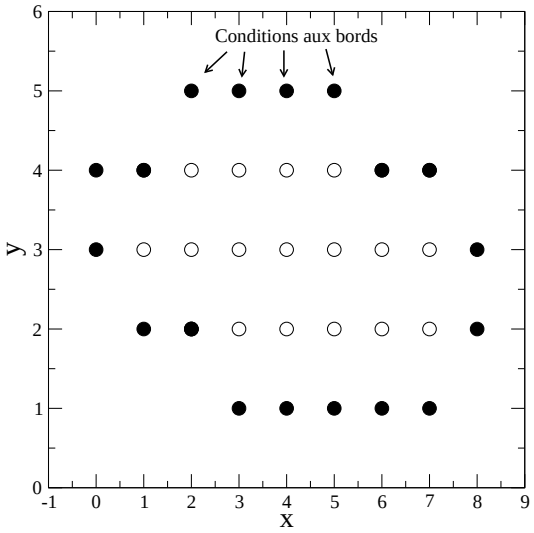


FIG. 9.2 – Schéma illustrant le principe de la résolution d'une équation aux dérivées partielles avec des conditions aux bords : l´ensemble des points de la grille doit être gardé tout au long du calcul.

**Equation hyperbolique.** La propagation des ondes dans un milieu continu (unidimensionnel) obéit à l'équation suivante

$$\frac{\partial^2 u(x,t)}{\partial t^2} = c^2 \frac{\partial^2 u(x,t)}{\partial x^2} \tag{9.4}$$

où $u(x,t)$ peut désigner une variable scalaire comme la densité locale $\rho(x,t)$ ou une variable vectorielle comme la vitesse locale $\mathbf{v}(x,t)$.

**Equation parabolique.** La diffusion de particules dans un milieu continu obéit à l'équation suivante

$$\frac{\partial \rho(x,t)}{\partial t} = D \frac{\partial^2 \rho(x,t)}{\partial x^2} \tag{9.5}$$

où $D$ est la constante de diffusion et $\rho(x,t)$ la densité locale instantanée des particules diffusantes.

**Equation elliptique.** En dimensions deux, le potentiel coulombien induit par une densité de charges électriques satisfait l'équation suivante (avec des unités choisies de manière appropriée)

$$-\rho(x,y) = \frac{\partial^2 V(x,y,t)}{\partial x^2} + \frac{\partial^2 V(x,y,t)}{\partial y^2} \tag{9.6}$$

où $\rho(x,y)$ est la densité locale de charge et $V(x,y)$ le potentiel électrostatique à déterminer.

La résolution numérique repose sur une distinction importante entre ces type d'équations : Les deux premières équations correspondent à une évolution temporelle tandis que la troisième équation correspond à un problème statique. Dans le premier cas, la solution est donnée à partir d'une fonction (ou condition initiale) que l'on doit propager le long de l'axe du temps ; en complément on doit tenir compte des conditions au limites du problème. En discrétisant la résolution, on obtient un schéma du type de celui illustré par la figure 9.1.

En effet pour une équation dépendante du temps du type équation de diffusion, la résolution de l'équation se fait pour un temps donné en fonction des valeurs de la fonction à l'instant antérieur précédent (ou de quelques instants antérieurs précédents). Il n'est donc pas nécessaire de garder en mémoire la solution à tous les temps.

Pour l'équation de Poisson, la solution est unique une fois donnée la fonction $\rho(x, y)$. La solution ne pouvant être obtenue numériquement directement à partir de l'intégration à partir d'une frontière du système, il est nécessaire de procéder à des réajustements, ce qui implique de garder en mémoire la totalité de valeurs de la fonction $V$ à déterminer tout au long de la résolution numérique (voir figure 9.2).

## 9.2 Equations avec conditions aux frontières

### 9.2.1 Introduction

Le prototype de ces équations est la résolution de l'équation de Poisson. Ces équations sont a priori plus simples à résoudre car le problème de la stabilité de l'algorithme se pose moins fréquemment que dans pour des équations avec conditions initiales, que nous allons discuter dans la section suivante.

Avant de procéder à la résolution numérique, il est important de bien distinguer le type de conditions aux frontières du problème : condition de Dirichlet, qui spécifie les valeurs des points à la frontière ; condition de Neumann qui spécifie les valeurs des gradients normaux à la frontière, voire des conditions mixtes.

### 9.2.2 Différences finies

La première étape consiste à choisir un maillage régulier de l'espace (que nous choisissons bidimensionnel dans notre exemple). Soit la fonction de deux variables $V(x, y)$

$$x_i = x_0 + j\Delta \qquad j = 0, 1, ..., J \tag{9.7}$$

$$y_k = x_0 + k\Delta \qquad k = 0, 1, ..., K \tag{9.8}$$

où $\Delta$ est le pas de la grille. On note $V_{jk} = V(x_j, y_k)$. On approxime le laplacien par la formule de différence finie suivante

$$\frac{V_{j+1,k} + V_{j-1,k} - 2V_{j,k}}{\Delta^2} + \frac{V_{j,k+1} + V_{j,k-1} - 2V_{j,k}}{\Delta^2} = -\rho_{jk} \tag{9.9}$$

que l'on peut écrire simplement

$$V_{j+1,k} + V_{j-1,k} + V_{j,k+1} + V_{j,k-1} - 4V_{j,k} = -\Delta^2 \rho_{jk} \tag{9.10}$$

Ce système d'équations linéaires peut s'exprimer sous une forme matricielle en utilisant l'indexation suivante

$$i = j(K+1) + k \qquad j = 0, 1, ..., J \text{ et } k = 0, 1, ..., K \tag{9.11}$$

L'équation (9.9) est correcte pour l'ensemble des points intérieurs au rectangle. Pour les points aux frontières, les valeurs de $V$ ou de ces dérivées sont données par le problème lui-même. On obtient finalement la structure matricielle suivante.

$$A.Y = b \tag{9.12}$$

La matrice $A$ est une matrice tridiagonale avec des franges, le vecteur $Y$ est un vecteur à $JK$ composantes ($Y_i \equiv V_{jk}$) et le vecteur $b$ à $JK$ composantes ($b_i \equiv -\rho_{jk}$) Ce type de représentation est similaire pour des équations elliptiques du second ordre

$$\begin{aligned} a(x,y)\frac{\partial^2 V}{\partial x^2} + b(x,y)\frac{\partial V}{\partial x} + c(x,y)\frac{\partial^2 V}{\partial y^2} \\ + d(x,y)\frac{\partial V}{\partial y} + e(x,y)\frac{\partial^2 V}{\partial x \partial y} + f(x,y)V = g(x,y) \end{aligned} \tag{9.13}$$

En choisissant une classification très rapide, on peut dire qu'il existe trois types de méthodes pour résoudre ce type d'équations : les méthodes de relaxation, les méthodes de Fourier et les méthodes matricielles directes.

### 9.2.3 Méthodes matricielles

Les méthodes matricielles ou méthodes directes consistent à déterminer les valeurs de la fonction $V$ en calculant la matrice inverse de $A$. La difficulté de cette méthode repose sur la taille considérable de la matrice que l'on doit inverser. En effet, si on considère un réseau bidimensionnel dont la taille est $100 \times 100$, et compte tenu de l'équation (9.11), la matrice $A$ est alors une matrice carrée de taille $10000 \times 10000$, contenant $10^8$ éléments. Il est donc nécessaire que cette matrice soit très creuse et de structure simple afin que le calcul de son inverse soit effectué avec un algorithme rapide.

Pratiquement, la matrice inverse est facile à obtenir quand les éléments de matrice ne dépendent pas des abscisses du réseau (cas d'une équation elliptique à coefficients constants).

### 9.2.4 Méthodes de relaxation

Les méthodes de relaxation reposent sur le schéma itératif suivant. On décompose la matrice $A$ comme

$$A = E - F \tag{9.14}$$

où $E$ est une matrice facilement inversible et $F$ le reste. On peut écrire alors que

$$E.u = F.u + b \tag{9.15}$$

En choisissant un vecteur $u^{(0)}$, on procède à la succession de calculs suivants

$$E.u^{(r)} = F.u^{(r-1)} + b \tag{9.16}$$

La convergence de la méthode est obtenue quand la différence entre deux valeurs de la fonction est inférieure à $\epsilon$, $\|u^{(r)} - u^{(r-1)}\| < \epsilon$. La norme $\|u\|$ désigne par exemple $\sum_{ij} |u_{ij}|^2$.

### 9.2.5 Méthodes de Fourier

**Principe**

On exprime la transformée de Fourier discrète de $V$ dans les deux directions $Ox$ et $Oy$.

$$V_{kl} = \frac{1}{KL} \sum_{m=0}^{K-1} \sum_{n=0}^{L-1} \hat{V}_{mn} e^{-2i\pi km/K} e^{-2i\pi ln/L} \tag{9.17}$$

Une expression analogue pour la fonction $\rho$ peut être obtenue

$$\rho_{kl} = \frac{1}{KL} \sum_{m=0}^{K-1} \sum_{n=0}^{L-1} \hat{\rho}_{mn} e^{-2i\pi km/K} e^{-2i\pi ln/L} \tag{9.18}$$

En prenant la transformée de Fourier discrète de l'équation (9.10), les composantes de Fourier de $V$ sont reliées à celles de $\rho$ par la relation

$$\hat{V}_{mn} \left( e^{2i\pi m/K} + e^{-2i\pi m/K} + e^{2i\pi n/L} + e^{-2i\pi n/L} - 4 \right) = -\hat{\rho}_{mn} \Delta^2 \tag{9.19}$$

ce qui peut se réécrire plus simplement pour si $n$ et $m$ différents de zéro comme

$$\hat{V}_{mn} = \frac{-\hat{\rho}_{mn} \Delta^2}{2 \left( \cos\left(\frac{2\pi m}{K}\right) + \cos\left(\frac{2\pi n}{L}\right) - 2 \right)} \tag{9.20}$$

La relation algébrique entre les composantes de Fourier de $V$ et de $\rho$ indique que le calcul dans l'espace de Fourier est élémentaire et revient à multiplier individuellement chaque composante de $\rho$ par un scalaire.

La méthode de résolution numérique est donc la suivante :

- Calculer les composantes de Fourier de $\rho$ : $\hat{\rho}_{mn}$
- Calculer les composantes de Fourier de $V$ en utilisant la relation (9.20) : $\hat{V}_{mn}$
- Calculer la transformée de Fourier inverse de $\hat{V}$ pour obtenir $V$.

Noter que la transformée de Fourier obtenue donne les relations suivantes

$$u_{jk} = u_{j+J,k} = u_{j,k+K} \tag{9.21}$$

ce qui en d'autres termes correspond à une solution satisfaisant des conditions aux limites périodiques. A partir de l'équation (9.20), on voit que l'on peut choisir $\hat{V}_{00} = 0$, mais qu'il est nécéssaire que $\hat{\rho}_{00} = 0$. Si cette dernière relation n'est pas satisfaite, il faut modifier la solution choisie pour $V$ comme nous allons le voir ci-dessous.

### Condition de Dirichlet

Si on impose des conditions de Dirichlet aux bords du rectangle, c'est-à-dire $u = 0$ pour $j = 0$ et $j = J$ et pour $k = 0$ et $k = K$, la transformée de Fourier adaptée est alors celle en sinus.

$$V_{kl} = \frac{2}{KL} \sum_{m=0}^{K-1} \sum_{n=0}^{L-1} \hat{V}_{mn} \sin\left(\frac{\pi km}{K}\right) \sin\left(\frac{\pi ln}{L}\right) \tag{9.22}$$

Une expression similaire pour la fonction $\rho$ peut être écrite. Un calcul simple montre que la relation (9.20) relie les composantes de Fourier en sinus des deux fonctions. et la méthode de calcul donnée dans le paragraphe précédent s'applique à nouveau à condition de considérer les transformées de Fourier en sinus.

### Condition de Neumann

Si on impose des conditions de Neumann aux bords du rectangle, c'est-à-dire $\nabla u = 0$ pour $j = 0$ et $j = J$ et pour $k = 0$ et $k = K$, la transformée de Fourier adaptée est alors celle en cosinus.

$$V_{kl} = \frac{2}{KL} \sum_{m=0}^{K-1} \sum_{n=0}^{L-1} \hat{V}_{mn} \cos\left(\frac{\pi km}{K}\right) \cos\left(\frac{\pi ln}{L}\right) \tag{9.23}$$

ainsi que pour la fonction $\rho$. A nouveau, les composantes de Fourier sont reliées par la relation (9.20). Le même schéma donné ci-dessus s'applique alors.

### Conditions complexes

Ceci représente le cas général où par exemple la fonction $u(x, y)$ s'annule sur l'une des frontières ($j = 0$), mais vaut $u = f(y)$ sur la frontière opposée ($j = J$). La méthode consiste alors à écrire que la solution de ce problème est la somme de deux contributions : celle de la solution de l'équation pour une condition de Dirichlet et celle de la solution qui est nulle à l'intérieur du rectangle. En écrivant la solution sous la forme

$$u = u' + u^B \tag{9.24}$$

avec $u^B = 0$ sauf $u^B = f$ à la frontière, on obtient

$$\nabla^2 u' = -\nabla^2 u^B + \rho \tag{9.25}$$

ce qui montre que la fonction satisfait alors une condition de Dirichlet. Le calcul de $u'$ est effectué en ajoutant un terme de source à $\rho$. Pratiquement, cela ajoute une contribution sur la ligne située juste avant la frontière.

De manière analogue, si la condition aux frontières est donnée par un gradient normal non nul, on décompose la solution d'une première contribution correspondant à la solution du problème de Neumann et d'une seconde contribution nulle en dehors de la frontière. En utilisant une décomposition analogue à celle introduite ci-dessus, cela revient à résoudre le problème de Neumann en modifiant le terme de source.

## 9.3 Equations avec conditions initiales

### 9.3.1 Equations à flux conservatif

Les équations à flux conservatif dans un espace unidimensionel peuvent s'écrire sous la forme

$$\frac{\partial u}{\partial t} = -\frac{\partial F(u)}{\partial x} \tag{9.26}$$

où $u$ est une fonction scalaire et $F$ un vecteur, appelé flux conservé, on peut citer l´exemple de l'équation de la conservation de la masse

$$\frac{\partial \rho}{\partial t} = -\nabla.(\rho v) \tag{9.27}$$

Si de plus on a une relation constitutive, comme la loi de Fick, $F = -D\nabla(\rho)$, on obtient l'équation parabolique de la diffusion.

Si $u$ est un vecteur, l'équation de propagation des ondes (sonores ou électromagnétiques, par exemple) se ramène à ce type d'équation. En effet soit l'équation des ondes suivante,

$$\frac{\partial^2 v}{\partial t^2} = c^2 \frac{\partial^2 v}{\partial x^2} \tag{9.28}$$

on introduit les fonctions auxiliaires $r$ et $s$

$$r = c\frac{\partial v}{\partial x} \tag{9.29}$$

$$s = \frac{\partial v}{\partial t} \tag{9.30}$$

et l'équation de propagation s'exprime alors par un ensemble de deux équations aux dérivées partielles du premier ordre

$$\frac{\partial r}{\partial t} = c\frac{\partial s}{\partial x} \tag{9.31}$$

$$\frac{\partial s}{\partial t} = c\frac{\partial r}{\partial x} \tag{9.32}$$

Si on considère que $r$ et $s$ sont les deux composantes du vecteur $u$, l'équation de propagation est alors donnée par l'équation (9.26) avec la relation matricielle suivante

$$F(u) = \begin{pmatrix} 0 & -v \\ -v & 0 \end{pmatrix} .u \tag{9.33}$$

Ainsi les équations à flux conservé comprennent à la fois les équations aux dérivées partielles paraboliques et hyperboliques.

Par souci de simplicité, nous allons considérer par la suite l'équation à flux conservatif suivante

$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x} \tag{9.34}$$

La solution de cette équation est bien évidemment connue $u = f(x - ct)$, mais nous allons chercher à construire une procédure numérique pour déterminer la solution.

### 9.3.2 Une approche naïve

Une première étape consiste à discrétiser l'espace et le temps sur un réseau régulier

$$x_j = x_0 + j\Delta x \qquad j = 0, 1, \dots J \tag{9.35}$$

$$t_n = t_0 + n\Delta t \qquad n = 0, 1, \dots N \tag{9.36}$$

où $\Delta x$ et $\Delta t$ sont les pas d'espace et de temps respectivement.

La difficulté réside maintenant dans le choix de l'algorithme d'intégration. La méthode la plus simple consiste à utiliser un algorithme de type Euler

$$\left.\frac{\partial u}{\partial t}\right|_{j,n} = \frac{u_j^{n+1} - u_j^n}{\Delta t} + \mathcal{O}(\Delta t) \tag{9.37}$$

Même si la précision de l'algorithme n'est pas très importante, cela permet d'exprimer la solution à l'instant $n+1$ uniquement en fonction de l'instant $n$.

Pour intégrer spatialement, on utilise un algorithme du second ordre

$$\left.\frac{\partial u}{\partial x}\right|_{j,n} = \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} + \mathcal{O}(\Delta x^2) \tag{9.38}$$

ce qui donne finalement

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} \tag{9.39}$$

soit encore

$$u_j^{n+1} = u_j^n - c\Delta t\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} \tag{9.40}$$

La détermination de $u_j^{n+1}$ se fait en fonction des trois points : $u_j^n$, $u_{j+1}^n$ et $u_{j-1}^n$. On parle de méthode explicite. Malheureusement, ce schéma très simple ne fonctionne pas en ce sens qu'il ne permet pas d'obtenir une solution correcte de l'équation aux dérivées partielles. En d'autres termes, on dit qu'il est instable.

### 9.3.3 Critère de Stabilité de Von Neumann

Pour faire l'analyse de la stabilité des algorithmes, on suppose que les coefficients de l'équation aux différences varient très peu dans l'espace et le temps. Les modes propres solutions de cette équation peuvent alors s'écrire sous la forme

$$u_j^n = \xi(k)^n e^{ikj\Delta x} \tag{9.41}$$

où $k$ est un vecteur d'onde réel et $\xi(k)$ une fonction complexe qui dépend de $k$. On voit facilement que l'intégration temporelle conduit à une progression géométrique pour les valeurs de $u_j^n$. Afin que la solution soit stable, il est nécessaire que les modes exponentiellement divergents n'existent pas. Cela revient à dire que l'algorithme est instable dès qu'il existe un vecteur d'onde $k_0$ tel que

$$|\xi(k_0)| > 1 \tag{9.42}$$

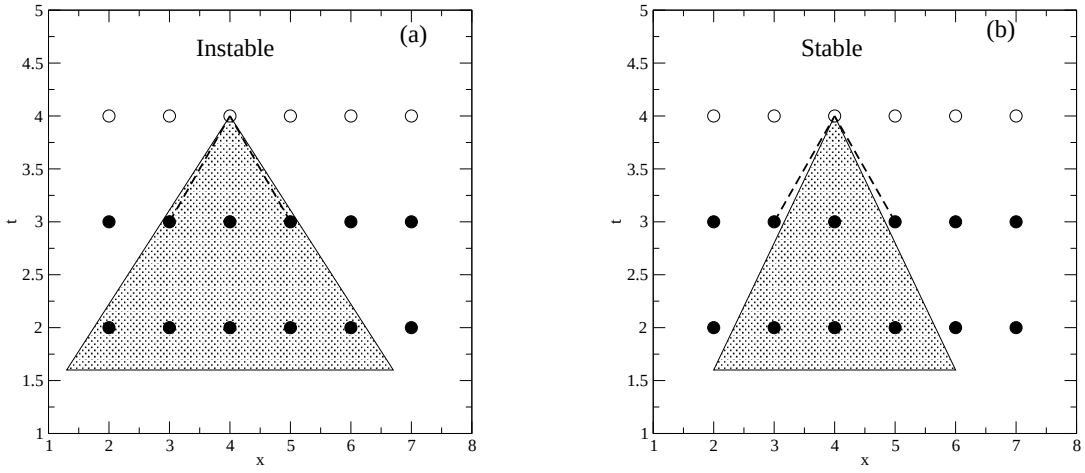


FIG. 9.3 – Schéma illustrant le principe de la résolution d'une équation aux dérivées partielles avec la méthode de Lax : La zone hachurée indique la zone de dépendance (a) Méthode instable (b) Méthode stable

En insérant l'équation (9.41) dans l'équation (9.39) et on obtient

$$\xi(k) = 1 - i\frac{c\Delta t}{\Delta x}\sin(k\Delta x) \tag{9.43}$$

Le module de $\xi(k)$ est strictement supérieur à 1 pour tous les vecteurs d'onde, hormis ceux où $k\Delta x = p\pi$ où $p$ est un entier. Ceci permet de comprendre la nature catastrophique de la méthode précédente.

### 9.3.4 Méthode de Lax

Pour corriger le grave problème d'instabilité de la méthode précédente, Lax a proposé la modification suivante : le terme $u_j^n$ provenant de la dérivée temporelle est remplacé par la moyenne sur les deux points adjacents :

$$u_j^n \to \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) \tag{9.44}$$

ce qui donne la relation itérative suivante

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) + \Delta t\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} \tag{9.45}$$

En insérant la solution (9.41) dans l'équation (9.45), on obtient une expression pour $\xi(k)$

$$\xi(k) = \cos(k\Delta x) - i\frac{c\Delta t}{\Delta x}\sin(k\Delta x) \tag{9.46}$$

La condition de stabilité $|\xi(k)| < 1$ équivaut à choisir

$$\frac{c\Delta t}{\Delta x} \leq 1 \tag{9.47}$$

Cette condition, dite de Courant est appelée aussi condition de stabilité de Courant-Friedrichs-Lewy. Il est facile de comprendre que la discrétisation spatio-temporelle doit se faire à une vitesse qui est donnée par $\frac{\Delta x}{\Delta t}$ inférieur à la vitesse de propagation du phénomène $c$. Pour comprendre graphiquement la raison pour laquelle le rapport du pas en temps sur celui de la position ne doit pas excéder la vitesse de propagation, considérons les figures suivantes : les zones hachurées correspondent à la solution donnée par la solution exacte de l'équation de propagation : si la valeur du point à l'étape $n + 1$ dépend des valeurs des points situées dans la zone hachurée, l'algorithme est instable. Inversement dans la figure de droite, le sommet de cette zone dépend de points situés à l'extérieur de la zone hachurée et l'algorithme est stable.

## 9.4 Conclusion

Les difficultés des méthodes numériques pour la résolution des équations dérivées partielles ont des origines diverses : elles proviennent soit des subtilités liées au choix de l´algorithme comme dans le cas des équations avec conditions initiales, soit des problèmes de stockage en mémoire dans le cas des équations avec conditions aux frontières. Des difficultés supplémentaires apparaissent pour des équations aux dérivées partielles non-linéaires (cas de l'hydrodynamique) où il est nécessaire d'introduire des nouveaux critères pour l'analyse de la stabilité des algorithmes.

De tels développements dépassent largement le cadre de ce cours introductif, mais cela illustre la grande complexité de ce sujet. En effet, Le domaine de la résolution des équations aux dérivées partielles reste un domaine très actif de recherche en analyse numérique, soulignant à la fois l'intérêt de nouvelles méthodes et la nécessite d'obtenir des méthodes de plus en plus performantes.

# Annexe A

# Coordonnées hypersphériques

Il convient de distinguer le cas pair du cas impair pour le système de coordonnées. Les calculs analytiques et numériques sont plus simples dans le cas impair, car en particulier le calcul de la transformée de Fourier d'une fonction à symétrie sphérique se ramène au calcul d'une transformée de Fourier à une dimension sur l'axe réel positif.

Considérons tout d'abord le cas où $n$ est impair : A $n$ dimensions, on définit les coordonnées hypersphériques de la manière suivante

$$\begin{aligned}
x_1 &= \rho \sin(\phi_1) \sin(\phi_2) \sin(\phi_3) \ldots \sin(\phi_{n-3}) \sin(\phi_{n-2}) \sin(\phi_{n-1}) \\
x_2 &= \rho \sin(\phi_1) \sin(\phi_2) \sin(\phi_3) \ldots \sin(\phi_{n-3}) \sin(\phi_{n-2}) \cos(\phi_{n-1}) \\
x_3 &= \rho \sin(\phi_1) \sin(\phi_2) \sin(\phi_3) \ldots \sin(\phi_{n-3}) \cos(\phi_{n-2}) \\
x_4 &= \rho \sin(\phi_1) \sin(\phi_2) \sin(\phi_3) \ldots \cos(\phi_{n-3}) \\
&\ldots \\
x_{n-1} &= \rho \sin(\phi_1) \cos(\phi_2) \\
x_n &= \rho \cos(\phi_1)
\end{aligned} \tag{A.1}$$

où les variables $\phi_i$ varient de 0 à $\pi$ pour $j$ compris entre 1 et $n-2$, la variable $\phi_{n-1}$ varie entre 0 et $2\pi$, et la variable $\rho$ est positive ou nulle.

Pour $n$ pair, on définit les coordonnées hypersphériques de la manière suivante

$$\begin{aligned}
x_1 &= \rho \sin(\phi_1) \sin(\phi_2) \sin(\phi_3) \ldots \sin(\phi_{n-3}) \sin(\phi_{n-2}) \sin(\phi_{n-1}) \\
x_2 &= \rho \sin(\phi_1) \sin(\phi_2) \sin(\phi_3) \ldots \sin(\phi_{n-3}) \sin(\phi_{n-2}) \cos(\phi_{n-1}) \\
x_3 &= \rho \sin(\phi_1) \sin(\phi_2) \sin(\phi_3) \ldots \sin(\phi_{n-3}) \cos(\phi_{n-2}) \\
x_4 &= \rho \sin(\phi_1) \sin(\phi_2) \sin(\phi_3) \ldots \cos(\phi_{n-3}) \cos(\phi_{n-2}) \\
&\ldots \\
x_{n-1} &= \rho \sin(\phi_2) \cos(\phi_1) \\
x_n &= \rho \cos(\phi_2) \cos(\phi_1)
\end{aligned} \tag{A.2}$$

On vérifie facilement que quel que soit $n$

$$-\rho \leq x_i \leq \rho \tag{A.3}$$

pour toute valeur de $i$. De plus, on a

$$\sum_{i=1}^{n} x_i^2 = \rho^2 \tag{A.4}$$

Pour évaluer une intégrale à $n$ dimensions, il est nécessaire de calculer le Jacobien de la transformation des coordonnées cartésiennes aux coordonnées hypersphériques pour $n$ impair

$$J = \frac{\partial(x_1 x_2 \dots x_n)}{\partial(\rho\phi_1\phi_2 \dots \phi_n)} = \rho^{n-1} \times \begin{vmatrix} \sin(\phi_1)\sin(\phi_2)\dots\sin(\phi_{n-1}) & \cos(\phi_1)\dots\sin(\phi_{n-1}) & \dots & \sin(\phi_1)\dots\cos(\phi_{n-1}) \\ \sin(\phi_1)\sin(\phi_2)\dots\cos(\phi_{n-1} & \cos(\phi_1)\dots\cos(\phi_{n-1}) & \dots & -\sin(\phi_1)\dots\sin(\phi_{n-1}) \\ \dots & \dots & \dots & \dots \\ \cos(\phi_1) & -\sin(\phi_1) & 0 & 0 \end{vmatrix} \tag{A.5}$$

Par récurrence, on montre que le jacobien $J$ se réexprime sous la forme

$$J = \rho^{n-1} \prod_{j=1}^{n-2} \sin^{n-1-j}(\phi_j) = \prod_{j=1}^{'n-2} \sin(\phi_{n-1-j}))^j \tag{A.6}$$

On peut vérifier que la formule convient aussi pour $n$ pair.

Quand l'integrant est une fonction qui ne dépend que de $\rho$, on peut intégrer sur tous les angles et le calcul se ramène à celui d'une intégrale simple sur l'axe réel positif. Par exemple on peut calculer le volume d'une hypersphère de rayon $a$ (qui correspond à un intégrant égal à 1)

$$\begin{aligned} V_n &= \int_0^a \rho^{n-1} d\rho \prod_{j=1}^{n-2} \int_0^{\pi} (\sin(\phi_j))^{n-1-j} d\phi_j \int_0^{2\pi} d\phi_{n-1} \\ &= \frac{\pi^{n/2} a^n}{\Gamma(n/2+1)} \\ &= \begin{cases} \dfrac{\pi^{n/2} a^n}{(n/2)!} & \text{pour } n \text{ pair} \\ \dfrac{\pi^{(n-1)/2}(2a)^n((n-1)/2)!}{n!} & \text{pour } n \text{ impair} \end{cases} \end{aligned} \tag{A.7}$$

L'élément de surface de l'hypersphère de rayon $a$ est donné par

$$dS_n = a^{n-1} \prod_{i=1}^{n-2} \sin^{n-1-i}(\phi_i) d\phi_i \tag{A.8}$$

On peut en déduire la surface d'une hypersphère dans un espace à $n$ dimensions.

$$\begin{aligned} S_n = \frac{dV_n}{da} &= \frac{n\pi^{n/2} a^{n-1}}{\Gamma(n/2+1)!} \\ &= \begin{cases} \dfrac{n\pi^{n/2} a^{n-1}}{(n/2)!} & \text{pour } n \text{ pair} \\ \dfrac{2n\pi^{(n-1)/2}(2a)^{n-1}((n-1)/2)!}{(n-1)!} & \text{pour } n \text{ impair} \end{cases} \end{aligned} \tag{A.9}$$

# Annexe B

# Les bibliothèques BLAS et Lapack

**Contenu**



## B.1 Introduction

La bibliothèque BLAS est un bibliothèque qui réalise les opérations d'algèbre linéaire de bas niveau. Elle est disponible en Fortran et en C par l'intermédiaire de la bibliothèque GSL, par exemple. Ces opérations sont classées par ordre de "difficulté" croissante :

1. opérations vectorielles
$$y = \alpha x + y \tag{B.1}$$
2. opérations matrices-vecteurs
$$y = \alpha A x + \beta y \tag{B.2}$$
3. opérations matrices-matrices
$$C = \alpha AB + C \tag{B.3}$$

LAPACK dont le nom vient de Linear Algebra PACKage est une bibliothèque écrite en fortran 77 et qui comprend un ensemble de sous-programmes pour résoudre les problèmes suivants : système d'équations linéaires : problèmes aux valeurs propres. Diverses factorisations matricielles sont disponibles comme la décomposition LU, QR Cholesky, SVD, Schur et Schur generalisée. Les matrices considérées sont des matrices soit pleines soit à bande. Le cas des matrices creuses n'est pas spécifiquement traité. Les sous-programmes sont fournis pour des problèmes dans lesquelles les matrices sont soit réelles, soit complexes, en simple ou en double précision.

La bibliothèque LAPACK s´appuie sur des appels à des sous-programmes de plus bas niveau qui sont ceux du BLAS. Cela permet une meilleure optimisation car des bibliothèques BLAS spécifiques existent pour chaque type de processeur et utilisent leurs caractéristiques. En l'absence de bibliothèque BLAS optimisée, il est possible de disposer de la bibliothèque BLAS de base disponible sur le site de netlib[4].

La version installée sur les stations de travail de l'école doctorale est la dernière version disponible (version 3.0) qui date de 2000. Elle est prévue pour des programmes Fortran. Une version *C* de Lapack (clapack) est disponible sur le site de netlib, mais n'est pas installée par défaut sur les stations.

## B.2 Terminologie

Une terminologie mnémotechnique a été choisie pour retrouver "facilement" les noms des sous-programmes à appeler. Le principe est le même pour Lapack et le BLAS.

Les sous-programmes ont un suffixe qui indique le type d'objets sur lesquels on effectue les opérations.

- S pour type REAL
- D pour type DOUBLE PRECISION
- C pour type COMPLEX
- Z pour type COMPLEX*16

[1]

Pour le BLAS de niveau 2, 3 et Lapack, un groupe de deux lettres désigne le type de matrices que considère le sous-programme.

- GE pour une matrice générale
- SY pour une matrice symétrique
- HE pour une matrice hermitienne
- TR pour une matrice triangulaire
- GB pour une matrice à bande générale
- SB pour une matrice symétrique à bande
- HB pour une matrice hermitienne à bande
- TB pour une matrice triangulaire à bande

Pour la recherche du nom précis d'un sous-programme, il existe un petit index référençant (disponible sur le site netlib ou sur les stations de travail) de tous les sous-programmes avec la liste des arguments à fournir. Il existe aussi un site internet très utile qui propose un moyen de rechercher le sous-programme adapté à partir de la sélection de la tâche à effectuer. L'adresse de ce site est http ://www.cs.colorado.edu/∼lapack/drivers.html

[1] Pour le Blas de Niveau 2, il y a un ensemble de sous-programmes pour des calculs en précision étendue. Les préfixes sont ES, ED, EC, EZ

# Annexe C

# La bibliothèque GSL

## Contenu

## C.1 Introduction

Parmi les nombreuses bibliothèques mathématiques que l'on peut utiliser pour résoudre des problèmes numériques, il existe une version évolutive basée sur le concept de logiciel libre qui s'appelle la GSL (acronyme de Gnu Scientific Library). La version actuelle est une version stable 1.2. Cette bibliothèque est écrite en C et est installée sur les stations de travail de l'école doctorale (avec la version 1.1.1), ce qui permet à tout utilisateur d'en disposer pleinement.

Cette bibliothèque offre des sous-programmes pour les problèmes suivants :

| | |
|---|---|
| Nombres complexes | Racines de polynômes |
| Fonctions spéciales | Vecteurs et Matrices |
| Permutations | Combinatoire |
| Tri | BLAS |
| Algèbre linéaire | FFT |
| Valeurs propres | Générateurs de nombres aléatoires |
| Histogrammes | Statistiques |
| Intégration de Monte Carlo | Equations différentielles |
| Recuit simulé | Différentiation numérique |
| Interpolation | Séries accélératrices |
| Approximations de Chebyshev | Recherche de racines |
| Transformées de Hankel discrètes | Moindre carré |
| Minimisation | Constantes physiques |

La diversité des problèmes numériques traités par cette bibliothèque rend très difficile une présentation rapide. Le manuel proprement dit comprend actuellement plus de 450 pages ! Le but de cet appendice est donc de donner quelques clés pour une utilisation aussi simple que possible de cette bibliothèque et par effet d'apprentissage d'autres bibliothèques dont les noms de sous-programmes sont généralement moins éloquents.

Les fichiers d'entêtes nécessaires à la compilation d'un programme sont placés dans le répertoire /usr/include/gsl.

Pour que les exécutables générés soient les plus petits possibles, les fichiers d'entêtes sont très nombreux (206 dans la version 1.1.1), et leur nom correspondent à une classe de sous-programes que l'on souhaite utilisés selon la règle suivante :

- Le nom commence toujours par le préfixe gsl_. (Moyen simple d'identification par rapport à d'autres bibliothèques qui seraient à utiliser dans le même programme).
- La seconde partie correspond à la classe de problème. Par exemple sf_ correspond à un fonction spéciale, matrix_ à un problème sur les matrices,...

– La troisième correspond au problème spécifique. Par exemple, gsl_sf_expint.h est le fichier pour les fonctions exponentielles intégrales, gsl_sf_bessel.h aux fonctions de bessel, ...
– Si l'on veut disposer d'une classe de fonctions plus directement au lieu de charger un à un les fichiers d'entêtes, on peut utiliser le fichier qui regroupe. Pour l'exemple précédent, on peut utiliser gsl_sf.h qui contient les définitions de toutes les fonctions spéciales.

## C.2 Sous programmes

La Liste des fonctions disponibles est la suivante

### C.2.1 BLAS

Les fonctions du BLAS écrites. Elles réalisent les trois niveaux qui sont les suivants :

1. opérations vectorielles
$$y = \alpha x + y \tag{C.1}$$
2. opérations matrices-vecteurs
$$y = \alpha A x + \beta y \tag{C.2}$$
3. opérations matrices-matrices
$$C = \alpha AB + C \tag{C.3}$$

cblas_caxpy, cblas_ccopy, cblas_cdotc_sub, cblas_cdotu_sub, cblas_cgbmv, cblas_cgemm, cblas_cgemv, cblas_cgerc, cblas_cgeru, cblas_chbmv, cblas_chemm, cblas_chemv, cblas_cher, cblas_cher2, cblas_cher2k, cblas_cherk, cblas_chpmv, cblas_chpr, cblas_chpr2, cblas_cscal, cblas_csscal, cblas_cswap, cblas_csymm, cblas_csyr2k, cblas_csyrk, cblas_ctbmv, cblas_ctbsv, cblas_ctpmv, cblas_ctpsv, cblas_ctrmm, cblas_ctrmv, cblas_ctrsm, cblas_ctrsv, cblas_dasum, cblas_daxpy, cblas_dcopy, cblas_ddot, cblas_dgbmv, cblas_dgemm, cblas_dgemv, cblas_dger, cblas_dnrm2, cblas_drot, cblas_drotg, cblas_drotm, cblas_drotmg, cblas_dsbmv, cblas_dscal, cblas_dsdot, cblas_dspmv, cblas_dspr, cblas_dspr2, cblas_dswap, cblas_dsymm, cblas_dsymv, cblas_dsyr, cblas_dsyr2, cblas_dsyr2k, cblas_dsyrk, cblas_dtbmv, cblas_dtbsv, cblas_dtpmv, cblas_dtpsv, cblas_dtrmm, cblas_dtrmv, cblas_dtrsm, cblas_dtrsv, cblas_dzasum, cblas_dznrm2, cblas_icamax, cblas_idamax, cblas_isamax, cblas_izamax, cblas_sasum, cblas_saxpy, cblas_scasum, cblas_scnrm2, cblas_scopy, cblas_sdot, cblas_sdsdot, cblas_sgbmv, cblas_sgemm, cblas_sgemv, cblas_sger, cblas_snrm2, cblas_srot, cblas_srotg, cblas_srotm, cblas_srotmg, cblas_ssbmv, cblas_sscal, cblas_sspmv, cblas_sspr, cblas_sspr2, cblas_sswap, cblas_ssymm, cblas_ssymv, cblas_ssyr, cblas_ssyr2, cblas_ssyr2k, cblas_ssyrk, cblas_stbmv, cblas_stbsv, cblas_stpmv, cblas_stpsv, cblas_strmm, cblas_strmv, cblas_strsm, cblas_strsv, cblas_xerbla, cblas_zaxpy, cblas_zcopy, cblas_zdotc_sub, cblas_zdotu_sub, cblas_zdscal, cblas_zgbmv, cblas_zgemm, cblas_zgemv, cblas_zgerc, cblas_zgeru, cblas_zhbmv, cblas_zhemm, cblas_zhemv, cblas_zher, cblas_zher2, cblas_zher2k, cblas_zherk, cblas_zhpmv, cblas_zhpr, cblas_zhpr2, cblas_zscal, cblas_zswap, cblas_zsymm, cblas_zsyr2k, cblas_zsyrk, cblas_ztbmv, cblas_ztbsv, cblas_ztpmv, cblas_ztpsv, cblas_ztrmm, cblas_ztrmv, cblas_ztrsm, cblas_ztrsv

### C.2.2 fonctions simples

Les fonctions simples avec une grande précision
gsl_acosh, gsl_asinh, gsl_atanh, gsl_expm1, gsl_log1p

### C.2.3 interfaces entre GSL et le BLAS

gsl_blas_caxpy, gsl_blas_ccopy, gsl_blas_cdotc, gsl_blas_cdotu, gsl_blas_cgemm, gsl_blas_cgemv, gsl_blas_cgerc, gsl_blas_cgeru, gsl_blas_chemm, gsl_blas_chemv, gsl_blas_cher, gsl_blas_cher2, gsl_blas_cher2k, gsl_blas_cherk, gsl_blas_cscal, gsl_blas_csscal, gsl_blas_cswap, gsl_blas_csymm, gsl_blas_csyr2k, gsl_blas_csyrk, gsl_blas_ctrmm, gsl_blas_ctrmv, gsl_blas_ctrsm, gsl_blas_ctrsv, gsl_blas_dasum, gsl_blas_daxpy, gsl_blas_dcopy, gsl_blas_ddot, gsl_blas_dgemm, gsl_blas_dgemv, gsl_blas_dger, gsl_blas_dnrm2, gsl_blas_drot, gsl_blas_drotg, gsl_blas_drotm, gsl_blas_drotmg, gsl_blas_dscal, gsl_blas_dsdot, gsl_blas_dswap, gsl_blas_dsymm, gsl_blas_dsymv, gsl_blas_dsyr, gsl_blas_dsyr2, gsl_blas_dsyr2k, gsl_blas_dsyrk, gsl_blas_dtrmm, gsl_blas_dtrmv, gsl_blas_dtrsm, gsl_blas_dtrsv, gsl_blas_dzasum, gsl_blas_dznrm2, gsl_blas_icamax, gsl_blas_idamax, gsl_blas_isamax, gsl_blas_izamax, gsl_blas_sasum, gsl_blas_saxpy, gsl_blas_scasum, gsl_blas_scnrm2, gsl_blas_scopy, gsl_blas_sdot, gsl_blas_sdsdot, gsl_blas_sgemm, gsl_blas_sgemv, gsl_blas_sger, gsl_blas_snrm2, gsl_blas_srot, gsl_blas_srotg, gsl_blas_srotm, gsl_blas_srotmg, gsl_blas_sscal, gsl_blas_sswap, gsl_blas_ssymm, gsl_blas_ssymv, gsl_blas_ssyr, gsl_blas_ssyr2, gsl_blas_ssyr2k, gsl_blas_ssyrk, gsl_blas_strmm, gsl_blas_strmv, gsl_blas_strsm, gsl_blas_strsv, gsl_blas_zaxpy, gsl_blas_zcopy, gsl_blas_zdotc, gsl_blas_zdotu, gsl_blas_zdscal, gsl_blas_zgemm, gsl_blas_zgemv, gsl_blas_zgerc, gsl_blas_zgeru, gsl_blas_zhemm, gsl_blas_zhemv, gsl_blas_zher, gsl_blas_zher2, gsl_blas_zher2k, gsl_blas_zherk, gsl_blas_zscal, gsl_blas_zswap, gsl_blas_zsymm, gsl_blas_zsyr2k, gsl_blas_zsyrk, gsl_blas_ztrmm, gsl_blas_ztrmv, gsl_blas_ztrsm, gsl_blas_ztrsv

### C.2.4 Blocs

Les fonctions pour les blocs de mémoire , gsl_block_alloc, gsl_block_calloc, gsl_block_fprintf, gsl_block_fread, gsl_block_free, gsl_block_fscanf, gsl_block_fwrite

### C.2.5 Séries de Chebyshev.

gsl_cheb_alloc, gsl_cheb_calc_deriv, gsl_cheb_calc_integ, gsl_cheb_eval, gsl_cheb_eval_err, gsl_cheb_eval_n, gsl_cheb_eval_n_err, gsl_cheb_free, gsl_cheb_init

### C.2.6 Combinatoire

gsl_combination_alloc, gsl_combination_calloc, gsl_combination_data, gsl_combination_fprintf, gsl_combination_fread, gsl_combination_free , gsl_combination_fscanf, gsl_combination_fwrite, gsl_combination_get, gsl_combination_init_first, gsl_combination_init_last, gsl_combination_k, gsl_combination_n, gsl_combination_next, gsl_combination_prev , gsl_combination_valid

### C.2.7 Complexes

Les fonctions d'opérations sur les complexes gsl_complex_abs, gsl_complex_abs2, gsl_complex_add, gsl_complex_add_imag, gsl_complex_add_real, gsl_complex_arccos, gsl_complex_arccos_real, gsl_complex_arccosh, gsl_complex_arccosh_real, gsl_complex_arccot,

gsl_complex_arccoth, gsl_complex_arccsc, gsl_complex_arccsc_real, gsl_complex_arccsch, gsl_complex_arcsec, gsl_complex_arcsec_real, gsl_complex_arcsech, gsl_complex_arcsin , gsl_complex_arcsin_real, gsl_complex_arcsinh, gsl_complex_arctan, gsl_complex_arctanh, gsl_complex_arctanh_real, gsl_complex_arg, gsl_complex_conjugate, gsl_complex_cos, gsl_complex_cosh, gsl_complex_cot, gsl_complex_coth, gsl_complex_csc, gsl_complex_csch, gsl_complex_div, gsl_complex_div_imag, gsl_complex_div_real, gsl_complex_exp, gsl_complex_inverse, gsl_complex_linalg_LU_invert, gsl_complex_log, gsl_complex_log10, gsl_complex_log_b, gsl_complex_logabs, gsl_complex_mul, gsl_complex_mul_imag, gsl_complex_mul_real, gsl_complex_negative, gsl_complex_polar, gsl_complex_pow, gsl_complex_pow_real, gsl_complex_rect, gsl_complex_sec, gsl_complex_sech, gsl_complex_sin, gsl_complex_sinh, gsl_complex_sqrt, gsl_complex_sqrt_real, gsl_complex_sub, gsl_complex_sub_imag, gsl_complex_sub_real, gsl_complex_tan, gsl_complex_tanh

### C.2.8 Hankel

Les fonctions pour les transformées de Hankel

gsl_dht_alloc, gsl_dht_apply, gsl_dht_free, gsl_dht_init, gsl_dht_k_sample, gsl_dht_new, gsl_dht_x_sample

### C.2.9 Dérivées

gsl_diff_backward, gsl_diff_central, gsl_diff_forward

### C.2.10 Valeurs et vecteurs propres

gsl_eigen_herm, gsl_eigen_herm_alloc, gsl_eigen_herm_free, gsl_eigen_hermv, gsl_eigen_hermv_alloc, gsl_eigen_hermv_free, gsl_eigen_hermv_sort, gsl_eigen_symm, gsl_eigen_symm_alloc, gsl_eigen_symm_free, gsl_eigen_symmv, gsl_eigen_symmv_alloc, gsl_eigen_symmv_free, gsl_eigen_symmv_sort

### C.2.11 Transformées de Fourier

gsl_fft_complex_backward, gsl_fft_complex_forward, gsl_fft_complex_inverse, gsl_fft_complex_radix2_backward, gsl_fft_complex_radix2_dif_backward, gsl_fft_complex_radix2_dif_forward, gsl_fft_complex_radix2_dif_inverse, gsl_fft_complex_radix2_dif_transform, gsl_fft_complex_radix2_forward, gsl_fft_complex_radix2_inverse, gsl_fft_complex_radix2_transform, gsl_fft_complex_transform, gsl_fft_complex_wavetable_alloc, gsl_fft_complex_wavetable_free, gsl_fft_complex_workspace_alloc, gsl_fft_complex_workspace_free, gsl_fft_halfcomplex_radix2_backward, gsl_fft_halfcomplex_radix2_inverse, gsl_fft_halfcomplex_transform, gsl_fft_halfcomplex_unpack, gsl_fft_halfcomplex_wavetable_alloc, gsl_fft_halfcomplex_wavetable_free, gsl_fft_real_radix2_transform, gsl_fft_real_transform, gsl_fft_real_unpack, gsl_fft_real_wavetable_alloc, gsl_fft_real_wavetable_free, gsl_fft_real_workspace_alloc, gsl_fft_real_workspace_free

gsl_finite

### C.2.12 Ajustements

gsl_fit_linear, gsl_fit_linear_est, gsl_fit_mul, gsl_fit_mul_est, gsl_fit_wlinear, gsl_fit_wmul

### C.2.13 Histogrammes

gsl_histogram2d_accumulate, gsl_histogram2d_add, gsl_histogram2d_alloc, gsl_histogram2d_clone, gsl_histogram2d_cov, gsl_histogram2d_div, gsl_histogram2d_equal_bins_p, gsl_histogram2d_find, gsl_histogram2d_fprintf, gsl_histogram2d_fread , gsl_histogram2d_free, gsl_histogram2d_fscanf, gsl_histogram2d_fwrite , gsl_histogram2d_get, gsl_histogram2d_get_xrange, gsl_histogram2d_get_yrange, gsl_histogram2d_increment, gsl_histogram2d_max_bin, gsl_histogram2d_max_val, gsl_histogram2d_memcpy, gsl_histogram2d_min_bin, gsl_histogram2d_min_val, gsl_histogram2d_mul, gsl_histogram2d_nx, gsl_histogram2d_ny, gsl_histogram2d_pdf_alloc, gsl_histogram2d_pdf_free, gsl_histogram2d_pdf_init, gsl_histogram2d_pdf_sample, gsl_histogram2d_reset, gsl_histogram2d_scale, gsl_histogram2d_set_ranges, gsl_histogram2d_set_ranges_uniform, gsl_histogram2d_shift, gsl_histogram2d_sub, gsl_histogram2d_sum, gsl_histogram2d_xmax, gsl_histogram2d_xmean, gsl_histogram2d_xmin, gsl_histogram2d_xsigma, gsl_histogram2d_ymax, gsl_histogram2d_ymean, gsl_histogram2d_ymin, gsl_histogram2d_ysigma, gsl_histogram_accumulate, gsl_histogram_add, gsl_histogram_alloc, gsl_histogram_bins, gsl_histogram_clone, gsl_histogram_div, gsl_histogram_equal_bins_p, gsl_histogram_find, gsl_histogram_fprintf, gsl_histogram_fread, gsl_histogram_free, gsl_histogram_fscanf, gsl_histogram_fwrite, gsl_histogram_get, gsl_histogram_get_range, gsl_histogram_increment, gsl_histogram_max, gsl_histogram_max_bin, gsl_histogram_max_val, gsl_histogram_mean, gsl_histogram_memcpy, gsl_histogram_min, gsl_histogram_min_bin, gsl_histogram_min_val, gsl_histogram_mul, gsl_histogram_pdf_alloc, gsl_histogram_pdf_free , gsl_histogram_pdf_init, gsl_histogram_pdf_sample, gsl_histogram_reset, gsl_histogram_scale, gsl_histogram_set_ranges, gsl_histogram_set_ranges_uniform, gsl_histogram_shift, gsl_histogram_sigma, gsl_histogram_sub, gsl_histogram_sum

gsl_hypot

### C.2.14 IEEE

gsl_ieee_env_setup, gsl_ieee_fprintf_double, gsl_ieee_fprintf_float, gsl_ieee_printf_double, gsl_ieee_printf_float

### C.2.15 Intégration

gsl_integration_qag, gsl_integration_qagi, gsl_integration_qagil, gsl_integration_qagiu, gsl_integration_qagp, gsl_integration_qags, gsl_integration_qawc, gsl_integration_qawf, gsl_integration_qawo, gsl_integration_qawo_table_alloc, gsl_integration_qawo_table_free, gsl_integration_qawo_table_set, gsl_integration_qawo_table_set_length, gsl_integration_qaws, gsl_integration_qaws_table_alloc, gsl_integration_qaws_table_free, gsl_integration_qaws_table_set, gsl_integration_qng, gsl_integration_workspace_alloc, gsl_integration_workspace_free

### C.2.16 Interpolation

gsl_interp_accel_alloc, gsl_interp_accel_find , gsl_interp_accel_free, gsl_interp_akima, gsl_interp_akima_periodic, gsl_interp_alloc, gsl_interp_bsearch, gsl_interp_cspline, gsl_interp_cspline_periodic, gsl_interp_eval, gsl_interp_eval_deriv, gsl_interp_eval_deriv2, gsl_interp_eval_deriv2_e, gsl_interp_eval_deriv_e, gsl_interp_eval_e, gsl_interp_eval_integ, gsl_interp_eval_integ_e, gsl_interp_free, gsl_interp_init, gsl_interp_linear, gsl_interp_min_size, gsl_interp_name, gsl_interp_polynomial

gsl_isinf, gsl_isnan

### C.2.17 Algèbre linéaire

gsl_linalg_bidiag_decomp, gsl_linalg_bidiag_unpack, gsl_linalg_bidiag_unpack2, gsl_linalg_bidiag_unpack_B, gsl_linalg_cholesky_decomp, gsl_linalg_cholesky_solve, gsl_linalg_cholesky_svx, gsl_linalg_complex_LU_decomp, gsl_linalg_complex_LU_det, gsl_linalg_complex_LU_lndet, gsl_linalg_complex_LU_refine, gsl_linalg_complex_LU_sgndet, gsl_linalg_complex_LU_solve, gsl_linalg_complex_LU_svx, gsl_linalg_hermtd_decomp, gsl_linalg_hermtd_unpack, gsl_linalg_hermtd_unpack_T, gsl_linalg_HH_solve, gsl_linalg_HH_svx, gsl_linalg_LU_decomp, gsl_linalg_LU_det, gsl_linalg_LU_invert, gsl_linalg_LU_lndet, gsl_linalg_LU_refine, gsl_linalg_LU_sgndet, gsl_linalg_LU_solve, gsl_linalg_LU_svx, gsl_linalg_QR_decomp, gsl_linalg_QR_lssolve, gsl_linalg_QR_QRsolve, gsl_linalg_QR_QTvec, gsl_linalg_QR_Qvec, gsl_linalg_QR_Rsolve, gsl_linalg_QR_Rsvx, gsl_linalg_QR_solve, gsl_linalg_QR_svx, gsl_linalg_QR_unpack, gsl_linalg_QR_update, gsl_linalg_QRPT_decomp , gsl_linalg_QRPT_decomp2, gsl_linalg_QRPT_QRsolve, gsl_linalg_QRPT_Rsolve, gsl_linalg_QRPT_Rsvx, gsl_linalg_QRPT_solve, gsl_linalg_QRPT_svx, gsl_linalg_QRPT_update, gsl_linalg_R_solve, gsl_linalg_R_svx, gsl_linalg_solve_symm_cyc_tridiag, gsl_linalg_solve_symm_tridiag, gsl_linalg_SV_decomp, gsl_linalg_SV_decomp_jacobi, gsl_linalg_SV_decomp_mod, gsl_linalg_SV_solve, gsl_linalg_symmtd_decomp, gsl_linalg_symmtd_unpack, gsl_linalg_symmtd_unpack_T

### C.2.18 Matrices

gsl_matrix_add, gsl_matrix_add_constant, gsl_matrix_alloc, gsl_matrix_calloc, gsl_matrix_column, gsl_matrix_const_column, gsl_matrix_const_diagonal , gsl_matrix_const_row, gsl_matrix_const_subdiagonal, gsl_matrix_const_submatrix, gsl_matrix_const_superdiagonal, gsl_matrix_const_view_array, gsl_matrix_const_view_array_with_tda, gsl_matrix_const_view_vector, gsl_matrix_const_view_vector_with_tda, gsl_matrix_diagonal, gsl_matrix_div_elements, gsl_matrix_fprintf, gsl_matrix_fread, gsl_matrix_free, gsl_matrix_fscanf, gsl_matrix_fwrite, gsl_matrix_get, gsl_matrix_get_col, gsl_matrix_get_row, gsl_matrix_isnull, gsl_matrix_max, gsl_matrix_max_index, gsl_matrix_memcpy, gsl_matrix_min, gsl_matrix_min_index, gsl_matrix_minmax, gsl_matrix_minmax_index, gsl_matrix_mul_elements, gsl_matrix_ptr, gsl_matrix_ptr, gsl_matrix_row, gsl_matrix_scale, gsl_matrix_set, gsl_matrix_set_all, gsl_matrix_set_col, gsl_matrix_set_identity, gsl_matrix_set_row, gsl_matrix_set_zero, gsl_matrix_sub, gsl_matrix_subdiagonal, gsl_matrix_submatrix, gsl_matrix_superdiagonal, gsl_matrix_swap, gsl_matrix_swap_columns, gsl_matrix_swap_rowcol, gsl_matrix_swap_rows, gsl_matrix_transpose, gsl_matrix_transpose_memcpy , gsl_matrix_view_array, gsl_matrix_view_array_with_tda, gsl_matrix_view_vector, gsl_matrix_view_vector_with_tda

### C.2.19 Minimisation

gsl_min_fminimizer_alloc, gsl_min_fminimizer_brent, gsl_min_fminimizer_f_lower, gsl_min_fminimizer_f_minimum, gsl_min_fminimizer_f_upper, gsl_min_fminimizer_free, gsl_min_fminimizer_goldensection, gsl_min_fminimizer_iterate, gsl_min_fminimizer_name, gsl_min_fminimizer_set , gsl_min_fminimizer_set_with_values, gsl_min_fminimizer_x_lower, gsl_min_fminimizer_x_minimum, gsl_min_fminimizer_x_upper, gsl_min_test_interval

### C.2.20 Monte Carlo

gsl_monte_miser_alloc, gsl_monte_miser_free, gsl_monte_miser_init , gsl_monte_miser_integrate, gsl_monte_plain_alloc, gsl_monte_plain_free, gsl_monte_plain_init, gsl_monte_plain_integrate,

gsl_monte_vegas_alloc, gsl_monte_vegas_free, gsl_monte_vegas_init , gsl_monte_vegas_integrate

### C.2.21 Ajustements non-linéaires

gsl_multifit_covar, gsl_multifit_fdfsolver_alloc, gsl_multifit_fdfsolver_free, gsl_multifit_fdfsolver_iterate, gsl_multifit_fdfsolver_lmder, gsl_multifit_fdfsolver_lmsder, gsl_multifit_fdfsolver_name, gsl_multifit_fdfsolver_position, gsl_multifit_fdfsolver_set, gsl_multifit_fsolver_alloc, gsl_multifit_fsolver_free, gsl_multifit_fsolver_iterate, gsl_multifit_fsolver_name, gsl_multifit_fsolver_position, gsl_multifit_fsolver_set, gsl_multifit_gradient, gsl_multifit_linear, gsl_multifit_linear_alloc, gsl_multifit_linear_free, gsl_multifit_test_delta, gsl_multifit_test_gradient, gsl_multifit_wlinear, gsl_multimin_fdfminimizer_alloc, gsl_multimin_fdfminimizer_conjugate_fr, gsl_multimin_fdfminimizer_conjugate_pr, gsl_multimin_fdfminimizer_free, gsl_multimin_fdfminimizer_gradient, gsl_multimin_fdfminimizer_iterate, gsl_multimin_fdfminimizer_minimum, gsl_multimin_fdfminimizer_name, gsl_multimin_fdfminimizer_restart, gsl_multimin_fdfminimizer_set, gsl_multimin_fdfminimizer_steepest_descent, gsl_multimin_fdfminimizer_vector_bfgs, gsl_multimin_fdfminimizer_x, gsl_multimin_test_gradient, gsl_multiroot_fdfsolver_alloc, gsl_multiroot_fdfsolver_dx, gsl_multiroot_fdfsolver_f, gsl_multiroot_fdfsolver_free, gsl_multiroot_fdfsolver_gnewton, gsl_multiroot_fdfsolver_hybridj, gsl_multiroot_fdfsolver_hybridsj, gsl_multiroot_fdfsolver_iterate, gsl_multiroot_fdfsolver_name, gsl_multiroot_fdfsolver_newton, gsl_multiroot_fdfsolver_root, gsl_multiroot_fdfsolver_set, gsl_multiroot_fsolver_alloc, gsl_multiroot_fsolver_broyden, gsl_multiroot_fsolver_dnewton, gsl_multiroot_fsolver_dx , gsl_multiroot_fsolver_f, gsl_multiroot_fsolver_free, gsl_multiroot_fsolver_hybrid, gsl_multiroot_fsolver_hybrids, gsl_multiroot_fsolver_iterate, gsl_multiroot_fsolver_name, gsl_multiroot_fsolver_root, gsl_multiroot_fsolver_set, gsl_multiroot_test_delta, gsl_multiroot_test_residual

gsl_ntuple_bookdata, gsl_ntuple_close, gsl_ntuple_create, gsl_ntuple_open, gsl_ntuple_project, gsl_ntuple_read, gsl_ntuple_write.

### C.2.22 Equations différentielles

gsl_odeiv_control_alloc, gsl_odeiv_control_free, gsl_odeiv_control_hadjust, gsl_odeiv_control_init, gsl_odeiv_control_name, gsl_odeiv_control_standard_new, gsl_odeiv_control_y_new, gsl_odeiv_control_yp_new, gsl_odeiv_evolve_alloc, gsl_odeiv_evolve_apply, gsl_odeiv_evolve_free, gsl_odeiv_evolve_reset, gsl_odeiv_step_alloc, gsl_odeiv_step_apply, gsl_odeiv_step_bsimp, gsl_odeiv_step_free, gsl_odeiv_step_gear1, gsl_odeiv_step_gear2, gsl_odeiv_step_name, gsl_odeiv_step_order, gsl_odeiv_step_reset, gsl_odeiv_step_rk2, gsl_odeiv_step_rk2imp, gsl_odeiv_step_rk4, gsl_odeiv_step_rk4imp, gsl_odeiv_step_rk8pd, gsl_odeiv_step_rkck, gsl_odeiv_step_rkf45

### C.2.23 Permutation

gsl_permutation_alloc, gsl_permutation_calloc, gsl_permutation_canonical_cycles, gsl_permutation_canonical_to gsl_permutation_data, gsl_permutation_fprintf, gsl_permutation_fread, gsl_permutation_free, gsl_permutation_fscanf, gsl_permutation_fwrite, gsl_permutation_get, gsl_permutation_init, gsl_permutation_inverse, gsl_permutation_inversions, gsl_permutation_linear_cycles, gsl_permutation_linear_to_canc gsl_permutation_memcpy, gsl_permutation_mul, gsl_permutation_next, gsl_permutation_prev, gsl_permutation_reverse, gsl_permutation_size, gsl_permutation_swap, gsl_permutation_valid, gsl_permute, gsl_permute_inverse, gsl_permute_vector, gsl_permute_vector_inverse

### C.2.24 Polynômes

gsl_poly_complex_solve, gsl_poly_complex_solve_cubic, gsl_poly_complex_solve_quadratic, gsl_poly_complex_workspace_alloc, gsl_poly_complex_workspace_free, gsl_poly_dd_eval, gsl_poly_dd_init, gsl_poly_dd_taylor, gsl_poly_eval, gsl_poly_solve_cubic, gsl_poly_solve_quadratic

### C.2.25 Puissances

gsl_pow_2, gsl_pow_3, gsl_pow_4, gsl_pow_5, gsl_pow_6, gsl_pow_7, gsl_pow_8, gsl_pow_9, gsl_pow_int

### C.2.26 Nombres aléatoires

gsl_qrng_alloc, gsl_qrng_clone, gsl_qrng_free, gsl_qrng_get , gsl_qrng_init, gsl_qrng_memcpy, gsl_qrng_name, gsl_qrng_niederreiter_2, gsl_qrng_size, gsl_qrng_sobol, gsl_qrng_state, gsl_ran_bernoulli, gsl_ran_bernoulli_pdf, gsl_ran_beta, gsl_ran_beta_pdf, gsl_ran_binomial, gsl_ran_binomial_pdf, gsl_ran_bivariate_gaussian, gsl_ran_bivariate_gaussian_pdf, gsl_ran_cauchy, gsl_ran_cauchy_pdf, gsl_ran_chisq, gsl_ran_chisq_pdf, gsl_ran_choose, gsl_ran_dir_2d, gsl_ran_dir_2d_trig_method, gsl_ran_dir_3d, gsl_ran_dir_nd, gsl_ran_discrete, gsl_ran_discrete_free, gsl_ran_discrete_pdf, gsl_ran_discrete_preproc, gsl_ran_exponential, gsl_ran_exponential_pdf, gsl_ran_exppow, gsl_ran_exppow_pdf, gsl_ran_fdist, gsl_ran_fdist_pdf, gsl_ran_flat, gsl_ran_flat_pdf, gsl_ran_gamma, gsl_ran_gamma_pdf, gsl_ran_gaussian, gsl_ran_gaussian_pdf, gsl_ran_gaussian_ratio_method, gsl_ran_gaussian_tail, gsl_ran_gaussian_tail_pdf, gsl_ran_geometric , gsl_ran_geometric_pdf, gsl_ran_gumbel1, gsl_ran_gumbel1_pdf, gsl_ran_gumbel2, gsl_ran_gumbel2_pdf, gsl_ran_hypergeometric, gsl_ran_hypergeometric_pdf, gsl_ran_landau, gsl_ran_landau_pdf, gsl_ran_laplace, gsl_ran_laplace_pdf, gsl_ran_levy, gsl_ran_levy_skew, gsl_ran_logarithmic, gsl_ran_logarithmic_pdf, gsl_ran_logistic, gsl_ran_logistic_pdf, gsl_ran_lognormal, gsl_ran_lognormal_pdf, gsl_ran_negative_binomial, gsl_ran_negative_binomial_pdf, gsl_ran_pareto, gsl_ran_pareto_pdf, gsl_ran_pascal, gsl_ran_pascal_pdf, gsl_ran_poisson, gsl_ran_poisson_pdf, gsl_ran_rayleigh, gsl_ran_rayleigh_pdf, gsl_ran_rayleigh_tail, gsl_ran_rayleigh_tail_pdf, gsl_ran_sample, gsl_ran_shuffle, gsl_ran_tdist, gsl_ran_tdist_pdf, gsl_ran_ugaussian, gsl_ran_ugaussian_pdf, gsl_ran_ugaussian_ratio_method, gsl_ran_ugaussian_tail, gsl_ran_ugaussian_tail_pdf, gsl_ran_weibull, gsl_ran_weibull_pdf, GSL_REAL, gsl_rng_alloc, gsl_rng_borosh13, gsl_rng_clone, gsl_rng_cmrg, gsl_rng_coveyou, gsl_rng_env_setup, gsl_rng_fishman18, gsl_rng_fishman20, gsl_rng_fishman2x, gsl_rng_free, gsl_rng_get, gsl_rng_gfsr4, gsl_rng_knuthran, gsl_rng_knuthran2, gsl_rng_lecuyer21, gsl_rng_max, gsl_rng_memcpy, gsl_rng_min, gsl_rng_minstd, gsl_rng_mrg, gsl_rng_mt19937, gsl_rng_name, gsl_rng_print_state, gsl_rng_r250, gsl_rng_ran0, gsl_rng_ran1, gsl_rng_ran2, gsl_rng_ran3, gsl_rng_rand, gsl_rng_rand48, gsl_rng_random_bsd, gsl_rng_random_glibc2, gsl_rng_random_libc5, gsl_rng_randu, gsl_rng_ranf, gsl_rng_ranlux, gsl_rng_ranlux389, gsl_rng_ranlxd1, gsl_rng_ranlxd2, gsl_rng_ranlxs0, gsl_rng_ranlxs1, gsl_rng_ranlxs2, gsl_rng_ranmar, gsl_rng_set, gsl_rng_size, gsl_rng_slatec, gsl_rng_state, gsl_rng_taus, gsl_rng_taus2, gsl_rng_transputer, gsl_rng_tt800, gsl_rng_types_setup, gsl_rng_uni, gsl_rng_uni32, gsl_rng_uniform, gsl_rng_uniform_int, gsl_rng_uniform_pos, gsl_rng_vax, gsl_rng_waterman14, gsl_rng_zuf

### C.2.27 Racines unidimensionnelles

gsl_root_fdfsolver_alloc, gsl_root_fdfsolver_free, gsl_root_fdfsolver_iterate, gsl_root_fdfsolver_name,

gsl_root_fdfsolver_newton, gsl_root_fdfsolver_root, gsl_root_fdfsolver_secant, gsl_root_fdfsolver_set, gsl_root_fdfsolver_steffenson, gsl_root_fsolver_alloc, gsl_root_fsolver_bisection, gsl_root_fsolver_brent, gsl_root_fsolver_falsepos, gsl_root_fsolver_free, gsl_root_fsolver_iterate, gsl_root_fsolver_name, gsl_root_fsolver_root, gsl_root_fsolver_set, gsl_root_fsolver_x_lower, gsl_root_fsolver_x_upper, gsl_root_test_delta, gsl_root_test_interval , gsl_root_test_residual

gsl_set_error_handler, gsl_set_error_handler_off

### C.2.28 Fonctions spéciales

#### Airy

gsl_sf_airy_Ai, gsl_sf_airy_Ai_deriv, gsl_sf_airy_Ai_deriv_e, gsl_sf_airy_Ai_deriv_scaled, gsl_sf_airy_Ai_deriv_scaled_e, gsl_sf_airy_Ai_e, gsl_sf_airy_Ai_scaled, gsl_sf_airy_Ai_scaled_e, gsl_sf_airy_Bi, gsl_sf_airy_Bi_deriv, gsl_sf_airy_Bi_deriv_e, gsl_sf_airy_Bi_deriv_scaled, gsl_sf_airy_Bi_deriv_scaled_e, gsl_sf_airy_Bi_e, gsl_sf_airy_Bi_scaled, gsl_sf_airy_Bi_scaled_e, gsl_sf_airy_zero_Ai, gsl_sf_airy_zero_Ai_deriv, gsl_sf_airy_zero_Ai_deriv_e, gsl_sf_airy_zero_Ai_e, gsl_sf_airy_zero_Bi, gsl_sf_airy_zero_Bi_deriv, gsl_sf_airy_zero_Bi_deriv_e, gsl_sf_airy_zero_Bi_e

#### Divers

gsl_sf_angle_restrict_pos, gsl_sf_angle_restrict_pos_e, gsl_sf_angle_restrict_symm, gsl_sf_angle_restrict_symm_e, gsl_sf_atanint, gsl_sf_atanint_e

#### Bessel

gsl_sf_bessel_I0, gsl_sf_bessel_I0_e, gsl_sf_bessel_I0_scaled, gsl_sf_bessel_i0_scaled, gsl_sf_bessel_I0_scaled_e, gsl_sf_bessel_i0_scaled_e, gsl_sf_bessel_I1, gsl_sf_bessel_I1_e, gsl_sf_bessel_I1_scaled, gsl_sf_bessel_i1_scaled, gsl_sf_bessel_I1_scaled_e, gsl_sf_bessel_i1_scaled_e, gsl_sf_bessel_i2_scaled, gsl_sf_bessel_i2_scaled_e, gsl_sf_bessel_il_scaled, gsl_sf_bessel_il_scaled_array, gsl_sf_bessel_il_scaled_e, gsl_sf_bessel_In, gsl_sf_bessel_In_array, gsl_sf_bessel_In_e, gsl_sf_bessel_In_scaled, gsl_sf_bessel_In_scaled_array, gsl_sf_bessel_In_scaled_e, gsl_sf_bessel_Inu, gsl_sf_bessel_Inu_e, gsl_sf_bessel_Inu_scaled, gsl_sf_bessel_Inu_scaled_e, gsl_sf_bessel_J0, gsl_sf_bessel_j0, gsl_sf_bessel_j0_e, gsl_sf_bessel_J0_e, gsl_sf_bessel_J1, gsl_sf_bessel_j1, gsl_sf_bessel_J1_e, gsl_sf_bessel_j1_e, gsl_sf_bessel_j2 , gsl_sf_bessel_j2_e, gsl_sf_bessel_jl, gsl_sf_bessel_jl_array, gsl_sf_bessel_jl_e, gsl_sf_bessel_jl_steed_array, gsl_sf_bessel_Jn, gsl_sf_bessel_Jn_array, gsl_sf_bessel_Jn_e, gsl_sf_bessel_Jnu, gsl_sf_bessel_Jnu_e, gsl_sf_bessel_K0, gsl_sf_bessel_K0_e, gsl_sf_bessel_K0_scaled, gsl_sf_bessel_k0_scaled, gsl_sf_bessel_K0_scaled_e, gsl_sf_bessel_k0_scaled_e, gsl_sf_bessel_K1, gsl_sf_bessel_K1_e, gsl_sf_bessel_K1_scaled, gsl_sf_bessel_k1_scaled, gsl_sf_bessel_K1_scaled_e, gsl_sf_bessel_k1_scaled_e, gsl_sf_bessel_k2_scaled, gsl_sf_bessel_k2_scaled_e, gsl_sf_bessel_kl_scaled, gsl_sf_bessel_kl_scaled_array, gsl_sf_bessel_kl_scaled_e, gsl_sf_bessel_Kn, gsl_sf_bessel_Kn_array, gsl_sf_bessel_Kn_e, gsl_sf_bessel_Kn_scaled, gsl_sf_bessel_Kn_scaled_array, gsl_sf_bessel_Kn_scaled_e, gsl_sf_bessel_Knu, gsl_sf_bessel_Knu_e, gsl_sf_bessel_Knu_scaled, gsl_sf_bessel_Knu_scaled_e, gsl_sf_bessel_lnKnu, gsl_sf_bessel_lnKnu_e, gsl_sf_bessel_sequence_Jnu_e, gsl_sf_bessel_y0, gsl_sf_bessel_Y0, gsl_sf_bessel_y0_e, gsl_sf_bessel_Y0_e, gsl_sf_bessel_y1, gsl_sf_bessel_Y1, gsl_sf_bessel_Y1_e, gsl_sf_bessel_y1_e, gsl_sf_bessel_y2, gsl_sf_bessel_y2_e, gsl_sf_bessel_yl, gsl_sf_bessel_yl_array, gsl_sf_bessel_yl_e, gsl_sf_bessel_Yn, gsl_sf_bessel_Yn_array, gsl_sf_bessel_Yn_e, gsl_sf_bessel_Ynu, gsl_sf_bessel_Ynu_e, gsl_sf_bessel_zero_J0, gsl_sf_bessel_zero_J0_e, gsl_sf_bessel_zero_J1, gsl_sf_bessel_zero_J1_e, gsl_sf_bessel_zero_Jnu, gsl_sf_bessel_zero_Jnu_e

### Beta, Coniques,Elliptiques, ...

gsl_sf_beta, gsl_sf_beta_e, gsl_sf_beta_inc, gsl_sf_beta_inc_e, gsl_sf_Chi, gsl_sf_Chi_e, gsl_sf_choose, gsl_sf_choose_e, gsl_sf_Ci, gsl_sf_Ci_e, gsl_sf_clausen, gsl_sf_clausen_e, gsl_sf_complex_cos_e, gsl_sf_complex_dilog_e, gsl_sf_complex_log_e, gsl_sf_complex_logsin_e, gsl_sf_complex_sin_e, gsl_sf_conicalP_0, gsl_sf_conicalP_0_e, gsl_sf_conicalP_1 , gsl_sf_conicalP_1_e, gsl_sf_conicalP_cyl_reg, gsl_sf_conicalP_cyl_reg_e, gsl_sf_conicalP_half, gsl_sf_conicalP_half_e, gsl_sf_conicalP_mhalf, gsl_sf_conicalP_mhalf_e, gsl_sf_conicalP_sph_reg, gsl_sf_conicalP_sph_reg_e, gsl_sf_cos, gsl_sf_cos_e, gsl_sf_cos_err, gsl_sf_cos_err_e, gsl_sf_coulomb_CL_array , gsl_sf_coulomb_CL_e, gsl_sf_coulomb_wave_F_array, gsl_sf_coulomb_wave_FG_array, gsl_sf_coulomb_wave_FG_e, gsl_sf_coulomb_wave_FGp_array , gsl_sf_coulomb_wave_sphF_array, gsl_sf_coupling_3j, gsl_sf_coupling_3j_e, gsl_sf_coupling_6j, gsl_sf_coupling_6j_e, gsl_sf_coupling_9j, gsl_sf_coupling_9j_e, gsl_sf_dawson, gsl_sf_dawson_e, gsl_sf_debye_1, gsl_sf_debye_1_e, gsl_sf_debye_2, gsl_sf_debye_2_e, gsl_sf_debye_3, gsl_sf_debye_3_e, gsl_sf_debye_4, gsl_sf_debye_4_e, gsl_sf_dilog, gsl_sf_dilog_e, gsl_sf_doublefact, gsl_sf_doublefact_e, gsl_sf_ellint_D, gsl_sf_ellint_D_e, gsl_sf_ellint_E, gsl_sf_ellint_E_e, gsl_sf_ellint_Ecomp, gsl_sf_ellint_Ecomp_e, gsl_sf_ellint_F, gsl_sf_ellint_F_e, gsl_sf_ellint_Kcomp, gsl_sf_ellint_Kcomp_e, gsl_sf_ellint_P, gsl_sf_ellint_P_e, gsl_sf_ellint_RC, gsl_sf_ellint_RC_e, gsl_sf_ellint_RD, gsl_sf_ellint_RD_e, gsl_sf_ellint_RF, gsl_sf_ellint_RF_e, gsl_sf_ellint_RJ, gsl_sf_ellint_RJ_e, gsl_sf_elljac_e, gsl_sf_erf, gsl_sf_erf_e, gsl_sf_erf_Q, gsl_sf_erf_Q_e, gsl_sf_erf_Z, gsl_sf_erf_Z_e, gsl_sf_erfc, gsl_sf_erfc_e, gsl_sf_eta, gsl_sf_eta_e, gsl_sf_eta_int, gsl_sf_eta_int_e, gsl_sf_exp, gsl_sf_exp_e, gsl_sf_exp_e10_e , gsl_sf_exp_err_e, gsl_sf_exp_err_e10_e, gsl_sf_exp_mult, gsl_sf_exp_mult_e, gsl_sf_exp_mult_e10_e, gsl_sf_exp_mult_err_e, gsl_sf_exp_mult_err_e10_e, gsl_sf_expint_3, gsl_sf_expint_3_e, gsl_sf_expint_E1, gsl_sf_expint_E1_e, gsl_sf_expint_E2, gsl_sf_expint_E2_e, gsl_sf_expint_Ei, gsl_sf_expint_Ei_e, gsl_sf_expm1, gsl_sf_expm1_e, gsl_sf_exprel, gsl_sf_exprel_2, gsl_sf_exprel_2_e, gsl_sf_exprel_e, gsl_sf_exprel_n, gsl_sf_exprel_n_e, gsl_sf_fact, gsl_sf_fact_e, gsl_sf_fermi_dirac_0, gsl_sf_fermi_dirac_0_e, gsl_sf_fermi_dirac_1, gsl_sf_fermi_dirac_1_e, gsl_sf_fermi_dirac_2, gsl_sf_fermi_dirac_2_e, gsl_sf_fermi_dirac_3half, gsl_sf_fermi_dirac_3half_e, gsl_sf_fermi_dirac_half, gsl_sf_fermi_dirac_half_e, gsl_sf_fermi_dirac_inc_0, gsl_sf_fermi_dirac_inc_0_e, gsl_sf_fermi_dirac_int, gsl_sf_fermi_dirac_int_e, gsl_sf_fermi_dirac_m1, gsl_sf_fermi_dirac_m1_e, gsl_sf_fermi_dirac_mhalf, gsl_sf_fermi_dirac_mhalf_e, gsl_sf_gamma, gsl_sf_gamma_e, gsl_sf_gamma_inc_P, gsl_sf_gamma_inc_P_e, gsl_sf_gamma_inc_Q, gsl_sf_gamma_inc_Q_e, gsl_sf_gammainv , gsl_sf_gammainv_e, gsl_sf_gammastar, gsl_sf_gammastar_e, gsl_sf_gegenpoly_1, gsl_sf_gegenpoly_1_e, gsl_sf_gegenpoly_2, gsl_sf_gegenpoly_2_e, gsl_sf_gegenpoly_3, gsl_sf_gegenpoly_3_e, gsl_sf_gegenpoly_array, gsl_sf_gegenpoly_n, gsl_sf_gegenpoly_n_e, gsl_sf_hydrogenicR, gsl_sf_hydrogenicR_1, gsl_sf_hydrogenicR_1_e, gsl_sf_hydrogenicR_e, gsl_sf_hyperg_0F1, gsl_sf_hyperg_0F1_e, gsl_sf_hyperg_1F1, gsl_sf_hyperg_1F1_e, gsl_sf_hyperg_1F1_int, gsl_sf_hyperg_1F1_int_e, gsl_sf_hyperg_2F0, gsl_sf_hyperg_2F0_e, gsl_sf_hyperg_2F1, gsl_sf_hyperg_2F1_conj, gsl_sf_hyperg_2F1_conj_e, gsl_sf_hyperg_2F1_conj_renorm , gsl_sf_hyperg_2F1_conj_renorm_e, gsl_sf_hyperg_2F1_e, gsl_sf_hyperg_2F1_renorm, gsl_sf_hyperg_2F1_renorm_e, gsl_sf_hyperg_U , gsl_sf_hyperg_U_e, gsl_sf_hyperg_U_e10_e, gsl_sf_hyperg_U_int, gsl_sf_hyperg_U_int_e, gsl_sf_hyperg_U_int_e10_e, gsl_sf_hypot, gsl_sf_hypot_e, gsl_sf_hzeta, gsl_sf_hzeta_e, gsl_sf_laguerre_1, gsl_sf_laguerre_1_e, gsl_sf_laguerre_2, gsl_sf_laguerre_2_e, gsl_sf_laguerre_3 , gsl_sf_laguerre_3_e, gsl_sf_laguerre_n, gsl_sf_laguerre_n_e, gsl_sf_lambert_W0, gsl_sf_lambert_W0_e, gsl_sf_lambert_Wm1, gsl_sf_lambert_Wm1_e, gsl_sf_legendre_array_size, gsl_sf_legendre_H3d, gsl_sf_legendre_H3d_0, gsl_sf_legendre_H3d_0_e, gsl_sf_legendre_H3d_1, gsl_sf_legendre_H3d_1_e, gsl_sf_legendre_H3d_array, gsl_sf_legendre_H3d_e, gsl_sf_legendre_P1, gsl_sf_legendre_P1_e, gsl_sf_legendre_P2, gsl_sf_legendre_P2_e, gsl_sf_legendre_P3, gsl_sf_legendre_P3_e, gsl_sf_legendre_Pl, gsl_sf_legendre_Pl_array, gsl_sf_legendre_Pl_e,

gsl_sf_legendre_Plm, gsl_sf_legendre_Plm_array, gsl_sf_legendre_Plm_e, gsl_sf_legendre_Q0, gsl_sf_legendre_Q0_e, gsl_sf_legendre_Q1, gsl_sf_legendre_Q1_e, gsl_sf_legendre_Ql, gsl_sf_legendre_Ql_e, gsl_sf_legendre_sphPlm, gsl_sf_legendre_sphPlm_array, gsl_sf_legendre_sphPlm_e, gsl_sf_lnbeta, gsl_sf_lnbeta_e, gsl_sf_lnchoose, gsl_sf_lnchoose_e, gsl_sf_lncosh, gsl_sf_lncosh_e, gsl_sf_lndoublefact, gsl_sf_lndoublefact_e, gsl_sf_lnfact, gsl_sf_lnfact_e, gsl_sf_lngamma, gsl_sf_lngamma_complex_e, gsl_sf_lngamma_e, gsl_sf_lngamma_sgn_e, gsl_sf_lnpoch, gsl_sf_lnpoch_e, gsl_sf_lnpoch_sgn_e, gsl_sf_lnsinh, gsl_sf_lnsinh_e, gsl_sf_log, gsl_sf_log_1plusx, gsl_sf_log_1plusx_e, gsl_sf_log_1plusx_mx, gsl_sf_log_1plusx_mx_e, gsl_sf_log_abs, gsl_sf_log_abs_e , gsl_sf_log_e, gsl_sf_log_erfc, gsl_sf_log_erfc_e, gsl_sf_multiply_e, gsl_sf_multiply_err_e, gsl_sf_poch, gsl_sf_poch_e, gsl_sf_pochrel, gsl_sf_pochrel_e, gsl_sf_polar_to_rect, gsl_sf_pow_int, gsl_sf_pow_int_e, gsl_sf_psi, gsl_sf_psi_1_int, gsl_sf_psi_1_int_e, gsl_sf_psi_1piy, gsl_sf_psi_1piy_e, gsl_sf_psi_e, gsl_sf_psi_int, gsl_sf_psi_int_e, gsl_sf_psi_n, gsl_sf_psi_n_e, gsl_sf_rect_to_polar, gsl_sf_Shi, gsl_sf_Shi_e, gsl_sf_Si, gsl_sf_Si_e, gsl_sf_sin, gsl_sf_sin_e, gsl_sf_sin_err, gsl_sf_sin_err_e, gsl_sf_sinc, gsl_sf_sinc_e, gsl_sf_synchrotron_1, gsl_sf_synchrotron_1_e, gsl_sf_synchrotron_2, gsl_sf_synchrotron_2_e, gsl_sf_taylorcoeff, gsl_sf_taylorcoeff_e, gsl_sf_transport_2, gsl_sf_transport_2_e, gsl_sf_transport_3, gsl_sf_transport_3_e, gsl_sf_transport_4, gsl_sf_transport_4_e, gsl_sf_transport_5, gsl_sf_transport_5_e, gsl_sf_zeta, gsl_sf_zeta_e, gsl_sf_zeta_int, gsl_sf_zeta_int_e,

gsl_siman_solve

### C.2.29 Tris

gsl_sort, gsl_sort_index, gsl_sort_largest, gsl_sort_largest_index, gsl_sort_smallest, gsl_sort_smallest_index, gsl_sort_vector, gsl_sort_vector_index, gsl_sort_vector_largest, gsl_sort_vector_largest_index, gsl_sort_vector_smallest, gsl_sort_vector_smallest_index, gsl_heapsort, gsl_heapsort_index

### C.2.30 Lissage

gsl_spline_alloc, gsl_spline_eval, gsl_spline_eval_deriv, gsl_spline_eval_deriv2, gsl_spline_eval_deriv2_e, gsl_spline_eval_deriv_e, gsl_spline_eval_e, gsl_spline_eval_integ, gsl_spline_eval_integ_e, gsl_spline_free, gsl_spline_init

### C.2.31 Statistique

gsl_stats_absdev, gsl_stats_absdev_m, gsl_stats_covariance, gsl_stats_covariance_m, gsl_stats_kurtosis, gsl_stats_kurtosis_m_sd, gsl_stats_lag1_autocorrelation , gsl_stats_lag1_autocorrelation_m, gsl_stats_max, gsl_stats_max_index, gsl_stats_mean, gsl_stats_median_from_sorted_data, gsl_stats_min, gsl_stats_min_index, gsl_stats_minmax, gsl_stats_minmax_index, gsl_stats_quantile_from_sorted_data, gsl_stats_sd, gsl_stats_sd_m, gsl_stats_sd_with_fixed_mean, gsl_stats_skew, gsl_stats_skew_m_sd, gsl_stats_variance, gsl_stats_variance_m , gsl_stats_variance_with_fixed_mean, gsl_stats_wabsdev, gsl_stats_wabsdev_m, gsl_stats_wkurtosis, gsl_stats_wkurtosis_m_sd, gsl_stats_wmean, gsl_stats_wsd, gsl_stats_wsd_m, gsl_stats_wsd_with_fixed_mean, gsl_stats_wskew, gsl_stats_wskew_m_sd, gsl_stats_wvariance, gsl_stats_wvariance_m, gsl_stats_wvariance_with_fixed_mean

gsl_strerror

### C.2.32 Transformées de Levin

gsl_sum_levin_u_accel, gsl_sum_levin_u_alloc, gsl_sum_levin_u_free, gsl_sum_levin_utrunc_accel, gsl_sum_levin_utrunc_alloc, gsl_sum_levin_utrunc_free

### C.2.33 Vecteurs

gsl_vector_add, gsl_vector_add_constant, gsl_vector_alloc, gsl_vector_calloc, gsl_vector_complex_const_imag, gsl_vector_complex_const_real , gsl_vector_complex_imag, gsl_vector_complex_real, gsl_vector_const_subvector , gsl_vector_const_subvector_with_stride, gsl_vector_const_view_array, gsl_vector_const_view_array_with_stride, gsl_vector_div, gsl_vector_fprintf, gsl_vector_fread, gsl_vector_free, gsl_vector_fscanf, gsl_vector_fwrite, gsl_vector_get, gsl_vector_isnull, gsl_vector_max, gsl_vector_max_index, gsl_vector_memcpy, gsl_vector_min, gsl_vector_min_index, gsl_vector_minmax, gsl_vector_minmax_index, gsl_vector_mul, gsl_vector_ptr, gsl_vector_ptr, gsl_vector_reverse, gsl_vector_scale, gsl_vector_set, gsl_vector_set_all, gsl_vector_set_basis, gsl_vector_set_zero, gsl_vector_sub, gsl_vector_subvector, gsl_vector_subvector_with_stride, gsl_vector_swap, gsl_vector_swap_elements, gsl_vector_view_array gsl_vector_view_array_with_stride

Nous renvoyons au manuel pour une présentation détaillée de chacune de ces fonctions.

# Bibliographie

[1] W. H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, Cambridge University Press (1992).

[2] http ://sources.redhat.com/gsl/

[3] http ://www.netlib.org/lapack/

[4] http ://www.netlib.org/blas/

[5] http ://beige.ucs.indiana.edu/B673/

[6] http ://dmawww.epfl.ch/rappaz.mosaic/index.html

# Table des matières