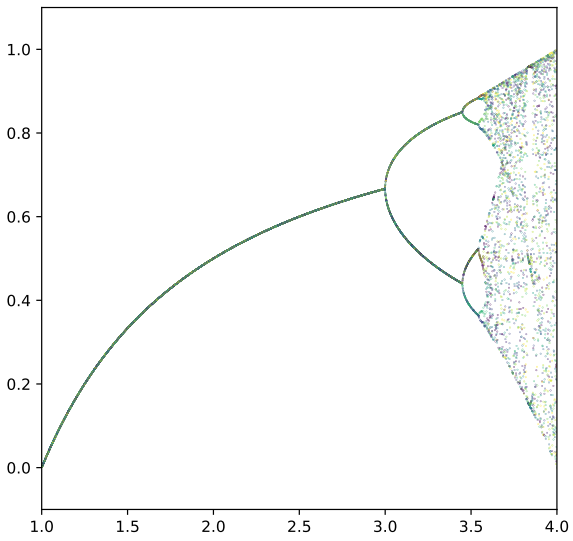
Sorbonne Université

MU4PY109
Mention Physique et Applications

Notes de cours
Année universitaire 2021-2022

Méthodes numériques

Auteur :
Pascal Viot



2 juillet 2021

Introduction

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. Chaque figure illustrant les concepts développés est réalisé en Python et le code associé à cette figure sera donné. La connaissance des méthodes numériques est donc un préalable pour choisir la méthode informatique pour résoudre un problème numérique le plus efficacement possible.

1 Intégration et sommes discrètes

Le cas typique d'une intégrale définie à évaluer est

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

Comme l'évaluation exacte de l'intégrale nécessite l'évaluation exacte de la fonction f pour une infinité de points, il convient de choisir une méthode qui remplace le calcul de l'intégrale Eq. (1.1) par une somme discrète avec 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_n, où n est un entier qui dépend de la méthode. [1] (continûment dérivable) sur le support $[a,b]$ [2]. La dernière restriction imposée à la fonction est que l'ensemble des dérivées en tout point de l'intervalle reste finie.

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

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

1.1 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.

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

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

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 $[a,b]$.

avec $x_0 = a$, $x_N = b$, i est un entier allant de 0 à N et h est appelé le pas de l'intégration. Pour simplifier les notations, on pose

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

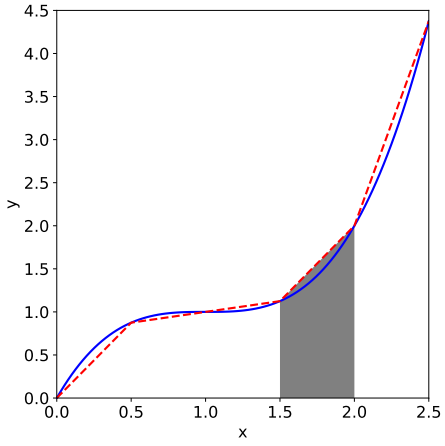


FIGURE 1.1 – Illustration de la méthode des trapèzes : la partie grisée correspond à l'aire calculée par l'équation (1.7)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: viot
"""
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
import numpy as np
def f(x):
    return(x**3-3*x**2+3*x)
fig, ax = plt.subplots(1,1,figsize=(6, 6))
x=np.linspace(0,2.5,100)
x2=np.linspace(0,2.5,6)
plt.plot(x,f(x),'-b',lw='2')
plt.plot(x2,f(x2),'--r',lw='2')
plt.xlabel('x',fontsize=15)
plt.ylabel('y',fontsize=15)
trap=[x2[3],0],[x2[3],f(x2[3])],[x2[4],f(x2[4])
    ↪ ],[x2[4],0]
poly=Polygon(trap,facecolor='grey')
ax.add_patch(poly)
plt.xlim(0,2.5)
plt.ylim(0,4.5)
plt.tick_params(labelsize=15)
plt.tight_layout()
plt.savefig("trap.pdf")
```

1.1.1 Trapèze

La méthode des trapèzes consiste à approximer la fonction entre deux abscisses successives par une droite (voir Fig. (1.1)), 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 $h = (b-a)/N$

1.1.2 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 un nombre de points pair, $N+1$, c'est-à-dire N impair, la méthode de Simpson donne une estimation de l'intégrale sur l'intervalle $[a,b]$.

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

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 $N/2$ points et la seconde avec N points, puis en combinant les deux résultats de la manière suivante

$$S_N = \frac{4}{3}T_N - \frac{1}{3}T_{N/2} \tag{1.12}$$

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.

1.2 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	$S_1(h)$	$S_2(h)$	$S_3(h)$	$S_4(h)$
$2h$	$S_1(2h)$	$S_2(2h)$	$S_3(2h)$	
$4h$	$S_1(4h)$	$S_2(4h)$		
$8h$	$S_1(8h)$			

TABLE 1.1 – Table de Romberg

1.3 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

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

Comme l'illustre la figure 1.2, la parité des polynômes correspond à la parité de l'ordre.

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

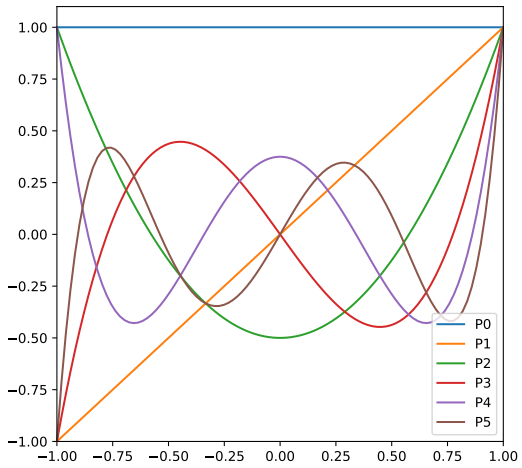


FIGURE 1.2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 1 19:36:43 2020

@author: viot
"""

from scipy.special import legendre
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-1,1,100)
plt.figure(figsize=(6,6))
for n in np.arange(0,6):
    Pn = legendre(n)
    y = Pn(x)
    plt.plot(x, y,label='P'+str(n))
plt.xlim(-1.0,1.0)
plt.ylim(-1.0,1.1)
plt.legend()
plt.savefig('legendre.pdf')
```

- *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
  $$H_{i+1} = 2xH_i - 2iH_{i-1} \tag{1.25}$$
- *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
  $$(i+1)L^\alpha_{i+1} = (-x+2i+\alpha+1)L^\alpha_i - (i+\alpha)L^\alpha_{i-1} \tag{1.26}$$
  La figure 1.3 illustre les premiers polynômes de Laguerre $L_n^{(1)}$.

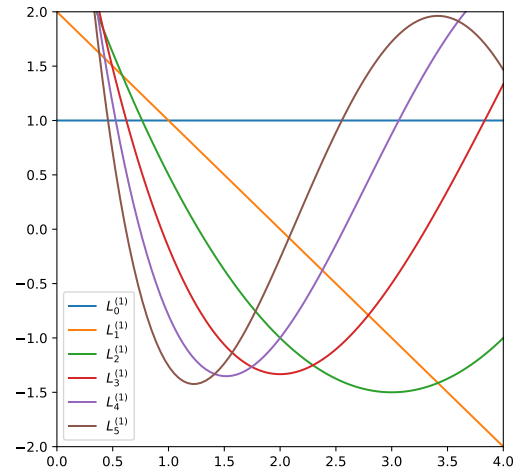


FIGURE 1.3 – Polynômes de Laguerre

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

@author: viot
"""

from scipy.special import genlaguerre
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,4,100)
plt.figure(figsize=(6,6))
for n in np.arange(0,6):
    Ln = genlaguerre(n,1)
    y = Ln(x)
    plt.plot(x, y,label=r'$L^{(1)}_'+str(n)+'$'
        ↪ )
plt.xlim(0,4.0)
plt.ylim(-2.0,2)
plt.legend()
plt.savefig('laguerre.pdf')
```

- *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
  $$c_i P_{i+1}^{(\alpha,\beta)} = (d_i + e_i x) P_i^{(\alpha,\beta)} - f_i P_{i-1}^{(\alpha,\beta)} \tag{1.27}$$
  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.

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

1.4 Méthode de Gauss-Kronrod et méthodes adaptatives

Pour déterminer la précision numérique d'une intégrale, il est nécessaire de faire deux évaluations différentes et d'utiliser la formule de Romberg pour obtenir une estimation de la précision. Avec une méthode de Côtes, dans lesquelles les abscisses sont régulièrement espacés, on peut sans effort de calcul supplémentaire (le coût provient essentiellement de l'évaluation de la fonction aux différents abscisses) obtenir une évaluation de l'intégrale en utilisant un point sur deux, ce qui donne une estimation pour un pas double. Dans une méthode de Gauss, les zéros d'un polynôme ne coïncident jamais avec ceux d'un polynôme de degré plus élévé. Kronrod, mathématicien russe a montré que l'on peut choisir un polynôme de degré $n+p$ dont n racines sont les racines du polynôme de Gauss. Notons $G(p+n,x)$ le polynôme de degré $p+n$ dont les racines correspondent aux $n+p$ abscisses de la formule d'intégration. Un polynôme de degré $n+2p-1$ peut s'exprimer sous la forme

$$f(x) = G(n+p)h(x) + g(x) \tag{1.33}$$

où

$$g(x) = \sum_{k=0}^{n+p-1} a_k x^k \tag{1.34}$$

et

$$h(x) = \sum_{k=0}^{p-1} b_k x^k \tag{1.35}$$

On impose les poids pour la nouvelle formule telle que $g(x)$ soit intégrée exactement. Cette condition s'exprime de la manière suivante

$$\int_{-1}^{1} G(n+p,x)h(x)dx = 0 \tag{1.36}$$

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.

Comme tout polynôme de degré $p-1$ peut s'exprimer comme une combinaison linéaire de Polynôme de Legendre de degré $p-1$, on a le système d'équation

$$\int_{-1}^{1} G(n+p,x)P(k,x)dx = 0 \tag{1.37}$$

avec $k = 0,1,....p-1$

Si $p = n+1$, on note $G(2n+1,x) = K(n+1,x)P(n,x)$. La condition Eq (7.1) est alors donnée par

$$\int_{-1}^{1} K(n+1,x)P(n,x)P(k,x)dx = 0 \tag{1.38}$$

avec $k = 0,1,....p-1$. Considérant le cas où n est impair, on a $K(n+1,x)$ qui est une fonction paire et se décompose sur les polynômes de Legendre comme

$$K(n+1,x) = \sum_{i=1}^{\frac{n+3}{2}} a_i P(2i-2,x) \tag{1.39}$$

Les coefficients a_i sont calculés par les relations

$$\sum_{i=1}^{\frac{n+3}{2}} a_i \int_{-1}^{1} P(2i-2,x)P(n,x)P(k,x)dx \tag{1.40}$$

pour $k = 1,\dots n$ (on choisit $a_0 = 1$). Pour k pair, ces équations sont automatiquement satisfaites, les équations restantes permettent d'évaluer les coefficients a_i (cette méthode s'appelle méthode de Patterson).

Les méthodes adaptatives partent du principe que l'erreur commise dans le calcul de l'intégrale dépend souvent de la portion du segment où les évaluations de fonctions sont faites. Si la fonction varie rapidement dans un intervalle restreint de l'ensemble du domaine d'intégration, il est préférable d'augmenter la précision du calcul dans cette région plutôt que sur la totalité de l'intervalle. Les méthodes adaptatives consistent donc à couper le segment en deux et évaluer chaque partie, on compare l'estimation de l'erreur par rappport à la tolérance imposée et l'on procède à une nouvelle division de l'intervalle dans les régions les plus difficiles.

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.

2 Interpolation de fonctions

2.1 Introduction

Pour évaluer une fonction, il est fréquent de ne disposer que de ses valeurs sur un ensemble fini de points. Dans le cas le plus simple d'une fonction à une variable, ces points sont souvent disposés sur un réseau régulier unidimensionnel. Il est nécessaire de pouvoir évaluer cette fonction en dehors de cet ensemble de points. Si le point à évaluer se situe à l'intérieur d'un intervalle élémentaire constitué de deux points consécutifs du réseau, la procédure d'approximation de la fonction par une fonction plus simple (très souvent un polynôme) s'appelle une interpolation. Quand le point à évaluer se situe en dehors des bornes du réseau, la procédure d'approximation s'appelle une extrapolation.

La mise en place d'une interpolation pour une fonction évaluée initialement sur un réseau consiste à optimiser la fonction approchante en utilisant l'ensemble des données. Le problème est de savoir si l'ensemble des données doit être utilisée en une seule fois pour déterminer une unique fonction sur l'ensemble de l'intervalle ou si on doit construire une succession de fonctions approchantes en utilisant des données locales. Nous allons voir que les réponses sont adaptées en fonction des propriétés de régularité de la fonction que l'on veut interpoler ainsi que le type de fonctions approchantes utilisées.

Sur la base de ce qui nous avons vu dans le chapitre sur l'intégration, les polynômes sont a priori de bons candidats pour approximer des fonctions ; nous allons voir ci-dessous que cela n'est pas aussi simple et qu'il existe des situations où le fait de prendre un polynôme de degré de plus en plus élevé détériore la qualité de l'approximation. En général, une fonction dont les dérivées restent bornées peut être approchée avec précision avec une interpolation polynomiale de degré élevé qui nécessite de faire intervenir les valeurs de la fonction sur une ensemble de points assez grand. Dans le cas où la fonction possède des discontinuités dans sa dérivée, une interpolation locale basée sur un petit nombre de points est alors plus précise.

2.2 Fonctions à une variable

2.2.1 Algorithme de Neville

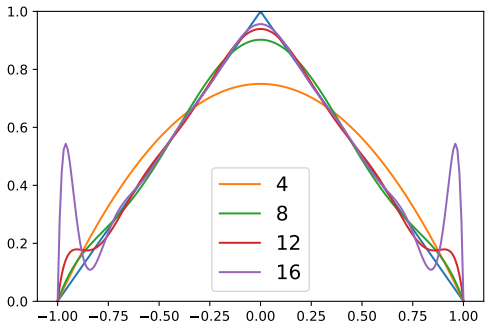


FIGURE 2.1 – Tracé de la fonction "triangle" et des interpolations polynomiales de degrés 4, 8, 12, 16.

```
from scipy.interpolate import lagrange
import numpy as np
import matplotlib.pyplot as plt
def triangle(x):
    return(np.heaviside(x,1)*(1-x)+np.heaviside
        ↪ (-x,1)*(1+x))

x2=np.linspace(-1,1,200)
y2=triangle(x2)
plt.plot(x2,y2)
plt.ylim(0,1)

for j in np.arange(4,20,4):
    x=np.linspace(-1,1,j)
    y=triangle(x)
    poly=lagrange(x,y)
    plt.plot(x2,poly(x2),label=str(j))
plt.legend(fontsize=15)
plt.savefig("interpolation.pdf")
```

Une approche simple consiste à utiliser les N points où la fonction a été évaluée $y_i = f(x_i)$ pour $i = 1, n$ et à interpoler par un polynôme d'ordre N avec la formule de Lagrange.

$$\begin{aligned} P(x) = & \frac{(x-x_2)(x-x_3)...(x_1-x_N)}{(x_1-x_2)(x_1-x_3)...(x_1-x_N)} y_1 + \\ & + \frac{(x-x_1)(x-x_3)...(x_1-x_N)}{(x_2-x_1)(x_2-x_3)...(x_2-x_N)} y_2 + ... \\ & + \frac{(x-x_1)(x-x_3)...(x_1-x_{N-1})}{(x_N-x_1)(x_N-x_3)...(x_N-x_{N-1})} y_N \end{aligned} \tag{2.1}$$

Pour construire une méthode itérative plus simple, il existe la méthode de Neville basée sur une structure hiérarchique. Le principe est le suivant : ii on a N points où la fonction a été évaluée, on considère que ces points représentent une approximation d'ordre 0 du polynôme.

L'arborescence de la construction des polynômes d'interpolation par la méthode de Neville a la structure suivante :

$$\begin{array}{ccccccc} P_1 & P_2 & P_3 & P_4 & ... & P_{N-1} & P_N \\ & P_{12} & P_{23} & P_{34} & ... & P_{(N-1)N} & \\ & P_{123} & P_{234} & & ... & ... & \\ & P_{1234} & & & ... & ... & \\ & & & ... & & & \\ & & & P_{1234...(N-1)N} & & & \end{array}$$

A partir des N polynômes constants, on construit les $N-1$ polynômes de degré 1, puis les $N-2$ polynômes de degré 2 et ainsi de suite jusqu'à obtenir le polynôme de degré $N-1$. Le polynôme obtenu à la fin est bien évidemment identique à celui décrit par la formule de Lagrange.

La formule de récurrence pour cette construction est la suivante

$$P_{i(i+1)...(i+m)} = \frac{(x-x_{i+m})P_{i(i+1)...(i+m-1)}}{(x_i-x_{i+m})} + \frac{(x_i-x)P_{(i+1)(i+2)...(i+m)}}{(x_i-x_{i+m})}. \tag{2.2}$$

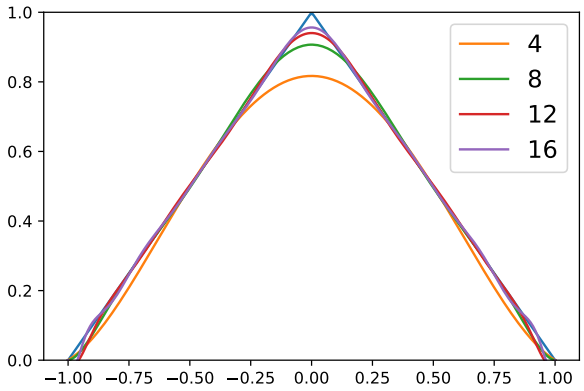


FIGURE 2.2 – Tracé de la fonction "triangle" et des interpolations par polynômes de Tchebyshev de degrés 4, 8, 12, 16.

```
import numpy as np
import matplotlib.pyplot as plt
def triangle(x):
    return(np.heaviside(x,1)*(1-x)+np.heaviside
        ↪ (-x,1)*(1+x))

x2=np.linspace(-1,1,200)
y2=triangle(x2)
plt.plot(x2,y2)
plt.ylim(0,1)

for j in np.arange(4,20,4):
    x=np.linspace(-1,1,j)
    y=triangle(x)
    coeff=np.polynomial.chebyshev.chebfit(x,y,j
        ↪ )
    plt.plot(x2,np.polynomial.chebyshev.chebval
        ↪ (x2,coeff),label=str(j))
plt.legend(fontsize=15)
plt.savefig("chebyshev.pdf")
```

2.2.2 Polynômes de Chebyshev

Dans le cas, où la fonction à interpoler est continue, mais possède des dérivées discontinues, la procédure de Neville ne converge pas uniformément. La figure 2.1 illustre la manière dont des polynômes d'interpolation de degré croissant interpole difficilement la région où la fonction continue a une discontinuité de la dérivée. En augmentant le degré du polynôme, on voit clairement que l'approximation est meilleure au centre mais se détériore gravement sur les bords de l'intervalle. En utilisant un développement sur les polynômes de Chebyshev, on obtient une interpolation avec des polynômes de degrés identiques, c'est-à-dire 4 et 10, les approximations qui apparaissent sur la figure 2.2. Le résultat est spectaculaire. Notons qu'avec cette procédure l'approximation obtenue ne passe pas par tous les points d'un réseau régulier. (voir Fig. 2.2).

2.2.3 Méthodes de lissage ("Spline")

Comme nous l'avons vu ci-dessus, les interpolations de fonction avec l'utilisation de polynômes de dégré élevé peuvent conduire à des défauts très importants situés sur les limites de l'intervalle où la fonction est définie. Inversement, l'approximation locale par une droite joignant deux points conécutifs présente le défaut important que la dérivée de la fonction interpolante a une dérivée constante et discontinue sur chaque intervalle et une dérivée seconde nulle presque partout.

La méthode spline minimise une sorte d'énergie élastique et consiste à imposer que la dérivée première de la fonction interpolant soit continue et que la dérivée seconde le soit aussi sur la totalité de l'intervalle de définition de la fonction.

Si on note $(x_i, y_i = f(x_i))$ la suite de couples abscisse-ordonnée où la fonction a été calculée, l'approximation cubique se construit de la manière suivante

$$y = A(x)y_i + B(x)y_{i+1} + C(x)y''_i + D(x)y''_{i+1} \tag{2.3}$$

où A, B, C and D sont des fonctions de x. Ces fonctions sont determinées par les contraintes suivantes :

- A et B sont déterminées par une interpolation linéaire. Ce qui donne

$$A(x) = \frac{x_{i+1} - x}{x_{i+1} - x_i} \tag{2.4}$$

et

$$B(x) = \frac{x - x_i}{x_{i+1} - x_i} \tag{2.5}$$

- $C(x)$ et $D(x)$ doivent s'annuler sur les limites de l'intervalle $[x_i, x_{i+1}]$ et sont des polynômes de degré 2 en x.
  Il est facile de vérifier que

$$C(x) = \frac{1}{6}(A(x)^3 - A(x))(x_{i+1} - x_i)^2) \tag{2.6}$$

et

$$D(x) = \frac{1}{6}(B(x)^3 - B(x))(x_{i+1} - x_i)^2) \tag{2.7}$$

conviennent

En effet en dérivant deux fois par rapport à x, l'équation (2.3), on obtient

$$\frac{d^2y}{dx^2} = A''(x)y''_i + B''(x)y''_{i+1} \tag{2.8}$$

et donc satisfait bien les contraintes exprimées ci-dessus.

Dans le cas où les dérivées (première et seconde) de la fonction ne sont pas connues, on peut estimer ces dérivées en imposant que la dérivée de l'approximation est continue, c'est-à-dire que

$$\frac{dy}{dx} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{3A(x)^2 - 1}{6}(x_{i+1} - x_i)y''_i + \frac{3B(x)^2 - 1}{6}(x_{i+1} - x_i)y''_{i+1} \tag{2.9}$$

Cela donne un système d'équations pour les dérivées secondes

$$\frac{x_i - x_{i+1}}{6}y''_{i-1} + \frac{x_{i+1} - x_{i-1}}{3}y''_i + \frac{x_{i+1} - x_i}{6}y''_{i+1} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \tag{2.10}$$

On a donc N inconnues pour $N-2$ équations ; pour lever l'indétermination, on peut imposer que la dérivée seconde de la fonction s'annule aux extremités de l'intervalle total.

La figure 2.3 montre la comparaison entre une interpolation linéaire de la fonction $\cos(x^2)$ dont 20 points sont calculés sur l'intervalle $0,5$. Quand x augmente, les variations de la fonction sont de plus en plus importantes et on voit que l'interpolation linéaire est de qualité moindre que l'interpolation cubique

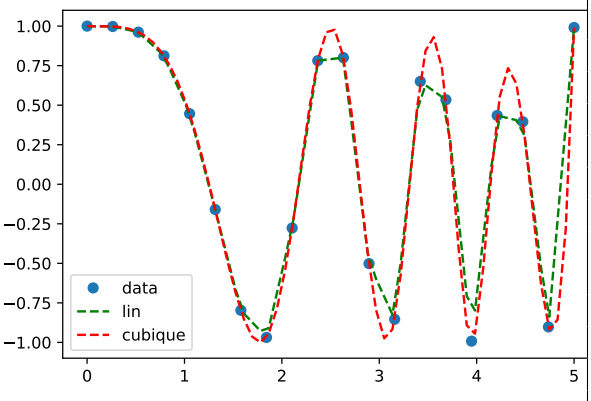


FIGURE 2.3 – Les points représentent les valeurs évaluées de la fonction $\cos(x^2)$, et les approximations linéaires et cubiques sont tracées sur un maillage trois fois plus petit.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

@author: viot
"""
from scipy.interpolate import interp1d
import numpy as np

x = np.linspace(0, 5, num=20)
y = np.cos(-x**2)
f = interp1d(x, y)
f2 = interp1d(x, y, kind='cubic')
xnew = np.linspace(0, 5, num=60)
import matplotlib.pyplot as plt
plt.plot(x, y, 'o', xnew, f(xnew), 'g--', xnew,
    ↪ f2(xnew), 'r--')
plt.legend(['data', 'lin', 'cubique'], loc='
    ↪ best')
plt.savefig("lincubic.pdf")
plt.show()
```

2.2.4 Approximants de Padé

Il est fréquent en Physique d'obtenir un développement perturbatif d'une fonction pour un nombre de dérivées limité. Dans le cas où la fonction inconnue a un développement en série dont le rayon de convergence est fini, toute approximation polynomiale ne permet pas d'obtenir une approche de la fonction au delà du rayon de convergence du développement en série entière.

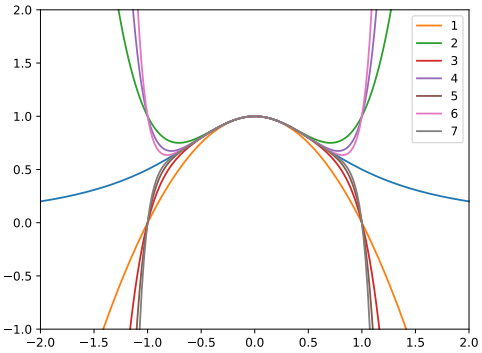


FIGURE 2.4 – Tracé de la fonction donnée par l'équation (2.11) et des développements à l'ordre 1 à 8

```
import numpy as np
import matplotlib.pyplot as plt

def gg(x):
    return(1/(1+x*x))

def ggn(x,n):
    sum=0
    for i in range(n+1):
        sum+=(-1)**i*x**(2*i)
    return(sum)
x=np.linspace(-2,2,200)
plt.plot(x,gg(x))
for j in np.arange(1,8):
    plt.plot(x,ggn(x,j),label=str(j))
plt.legend(fontsize=15)
plt.ylim(-1,2)
plt.xlim(-2,2)
plt.legend()
plt.savefig("devel.pdf")
```

De plus, une approximation de degré de plus en plus élévé donnera une approximation de plus en plus médiocre quand on s'approche du rayon de convergence. Ce phénomène est illustrée sur la Figure 2.4 où l'on considère la fonction

$$f(x) = \frac{1}{1+x^2} \tag{2.11}$$

et où les polynômes sont basés sur le développement en x à l'ordre $4, 6, 18, 20$ et sont tracés conjointement avec la fonction $f(x)$.

Pour remédier à ce type de défaut, les approximants de Padé sont basés sur une interpolation utilisant une fraction rationnelle,

$$Pa[m,n] = \frac{P_m(x)}{Q_n(x)} \tag{2.12}$$

où le polynôme $P_m(x)$ est un polynômes de degré m et $Q_n(x)$ est un polynôme de degré n dont le coefficient constant est égal à 1. Ainsi un approximation de Padé d'ordre $[m,n]$ est une fraction rationnelle dont $m+n+1$ coefficients sont à determiner. Quand on connait la valeur d'une fonction et ses $n+m$ dérivées en un point, on peut calculer $n+m$ approximants différents.

2.3 Fonctions de plusieurs variables

2.3.1 Introduction

Pour des raisons de capacité mémoire sur les ordinateurs, la plupart des fonctions de plusieurs variables que l'on doit interpoler sont des fonctions à deux variables ou à trois variables au maximum. La nécessité de procéder à une interpolation intervient par exemple dans le calcul d'équations intégrales.

2.3.2 Interpolations bilinéaire et bicubiques

Pour une fonctoin tabulée sur un réseau carré régulier, l'approximation bilinéaire consiste à utiliser un polynôme à deux variables qui donne sur chacune des arêtes d'un carré élémentaire la valeur exacte de la fonction.

Soit $x_i = x_0 + hi$ avec i entier et $y_j = y_0 + hj$, pour la cellule élémentaire délimité par les points (x_i, y_j),(x_i, y_{j+1}),(x_{i+1}, y_{j+1}),(x_{i+1}, y_j), on a le polynôme suivant

$$\begin{aligned} P(x,y) =& h^{-2}\Big((x-x_{i+1})(y-y_{i+1})f_{ij} + (x-x_{i+1})(y-y_i)f_{ij+1} \\ &+(x-x_{i+1})(y-y_i)f_{i+1j} + (x-x_i)(y-y_i)f_{i+1j+1}\Big) \end{aligned} \tag{2.13}$$

Avec cette approximation, les derivées partielles secondes de cette approximation sont toutes nulles, ce qui peut être insuffisant pour la qualité de l'approximation. Dans le cas où les dérivées premières ainsi que la (ou les) dérivée(s) croisée(s) seconde(s) est (sont) connue(s), on peut faire une meilleure approximation appelée approximation bicubique. En utilisant le fait que l'approximation doit donner exactement les valeurs de la fonction ainsi que celles des dérivées premières et des dérivées croisées secondes, cela donne seize coefficients à déterminer. En notant

$$\frac{\partial y}{\partial x_1} = y_{,1} \tag{2.14}$$

$$\frac{\partial y}{\partial x_2} = y_{,2} \tag{2.15}$$

$$\frac{\partial^2 y}{\partial x_1 \partial x_2} = y_{,12} \tag{2.16}$$

$$\tag{2.17}$$

on doit résoudre le système d'équations suivant

$$y(x_1, x_2) = \sum_{i=1}^{4}\sum_{j=1}^{4} c_{ij} t^{i-1} u^{j-1} \tag{2.18}$$

$$y_{,1}(x_1, x_2) = \sum_{i=1}^{4}\sum_{j=1}^{4} (i-1) c_{ij} t^{i-2} u^{j-1} \tag{2.19}$$

$$y_{,2}(x_1, x_2) = \sum_{i=1}^{4}\sum_{j=1}^{4} (j-1) c_{ij} t^{i-1} u^{j-2} \tag{2.20}$$

$$y_{,12}(x_1, x_2) = \sum_{i=1}^{4}\sum_{j=1}^{4} (i-1)(j-1) c_{ij} t^{i-2} u^{j-2} \tag{2.21}$$

$$\tag{2.22}$$

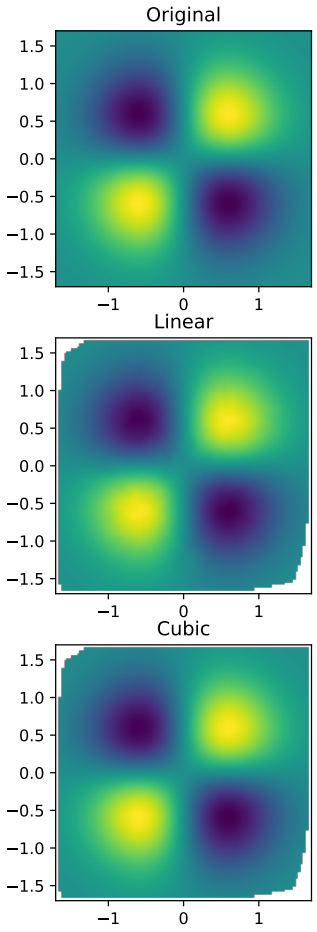
où c_{ij} sont les seize coefficients à déterminer avec

$$t = \frac{x_1 - x_{1,i}}{x_{1,i+1} - x_{1,i}} \tag{2.23}$$

$$u = \frac{x_2 - x_{2,i}}{x_{2,i+1} - x_{2,i}} \tag{2.24}$$

$x_{1,i}$ et $x_{2,i}$ désignent les points du réseau. Ce type de formule se généralise aisément en dimensions 3.

Sur la figure 2.5, on compare la méthode d'interpolaire linéaire avec la méthode bicubique pour la représentation de la fonction $xye^{-x^2-y^2}$. La figure du haut est une image de la fonction évaluée sur une grille de 100×, la figure du milieu est basée sur l'interpolation linéaire sur un ensemble de 800 points choisis aléatoirement dans le plan et la figure du bas correspondant à une approximation bicubique des memes points. Clairement, la meilleure approximation est la bicubique.




```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

@author: viot
"""
from scipy.interpolate import interp1d
import numpy as np

x = np.linspace(0, 5, num=20)
y = np.cos(-x**2)
f = interp1d(x, y)
f2 = interp1d(x, y, kind='cubic')
xnew = np.linspace(0, 5, num=60)
import matplotlib.pyplot as plt
plt.plot(x, y, 'o', xnew, f(xnew), 'g--', xnew,
    ↪ f2(xnew), 'r--')
plt.legend(['data', 'lin', 'cubique'], loc='
    ↪ best')
plt.savefig("lincubic.pdf")
plt.show()
```

FIGURE 2.5 – Interpolation linéaire et bicubique à partir d'un ensemble de points aléatoires

3 Racines d'équations

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 ϵ 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 ϵ_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}.

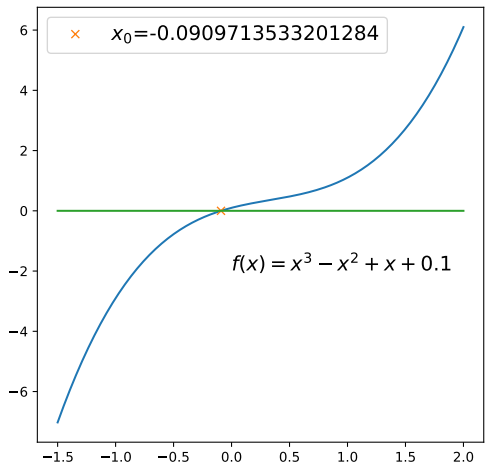


FIGURE 3.1 – Tracé de la fonction $f(x) = x^3 - x^2 + x + 0.1$ et le point racine obtenu par la méthode de dichotomie en utilisant la bibliothèqye scipy

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 16 09:31:44 2020

@author: viot
"""
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
def ff(x):
    return(x**3-x**2+x+0.1)
x=np.linspace(-1.5,2,200)
plt.figure(figsize=(6,6))
plt.plot(x,ff(x),'-')
x0=optimize.bisect(ff,-1,1)
plt.plot(x0,0,'x',label=r'$x_0$={:16.15g}'.
    ↪ format(x0))
plt.plot([-1.5,2],[0,0])
plt.text(0,-2,r'$f(x)=x^3-x^2+x+0.1$',fontsize
    ↪ =15)
plt.legend(fontsize=15)
plt.savefig('dicho.pdf')
plt.show()
```

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

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

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

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

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

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

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 $x_3 = \frac{a_0+b_0}{2}$ 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 $f(a_0)$, $f(x_3)$ et $f(b_0)$, mais à $f(a_0)$, $f(x_3)z$ et $f(b_0)z^2$, on obtient une approximation de la racine, notée x_4 et donnée par

$$x_4 = x_3 + (x_3 - a_0)\frac{sgn(f(a_0) - f(b_0))f(x)}{\sqrt{f(x)^2 - f(a_0)f(b_0)}} \tag{3.21}$$

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 nouvel encadrement, sinon si on considère le produit $f(a_0)f(x_4)$; si celui est négatif, le nouvel encadrement est $[a_0, x_4]$, sinon on prend $[x_4, b]$. 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.

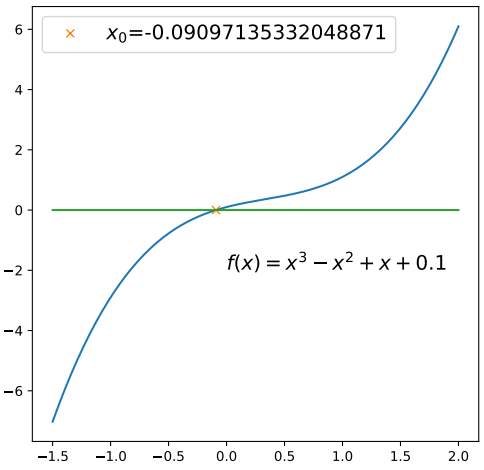


FIGURE 3.2 – Tracé de la fonction $f(x) = x^3 - x^2 + x + 0.1$ et le point racine obtenu par la méthode de Brent en utilisant la bibliothèqye scipy

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 16 09:31:44 2020

@author: viot
"""
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
def ff(x):
    return(x**3-x**2+x+0.1)
x=np.linspace(-1.5,2,200)
plt.figure(figsize=(6,6))
plt.plot(x,ff(x),'-')
x0=optimize.brentq(ff,-1,1)
plt.plot(x0,0,'x',label=r'$x_0$={:17.16g}'.
    ↪ format(x0))
plt.plot([-1.5,2],[0,0])
plt.text(0,-2,r'$f(x)=x^3-x^2+x+0.1$',fontsize
    ↪ =15)
plt.legend(fontsize=15)
plt.savefig('brent.pdf')
plt.show()
```

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

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

Si δ 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 δ 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

$$x - x_1 = a \qquad (3.41)$$
$$x - x_i = b \qquad i \in [2,n] \qquad (3.42)$$

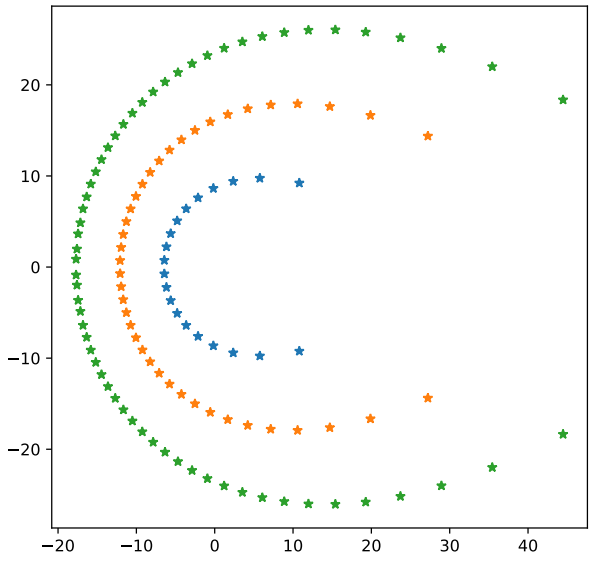
En insérant les équations (3.41) et (3.42) dans les équations (3.38), (3.40), on en déduit respectivement les relations suivantes

$$\frac{1}{a} + \frac{n-1}{b} = G \qquad (3.43)$$
$$\frac{1}{a^2} + \frac{n-1}{b^2} = H \qquad (3.44)$$

Après élimination de b, la valeur de a est

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

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.



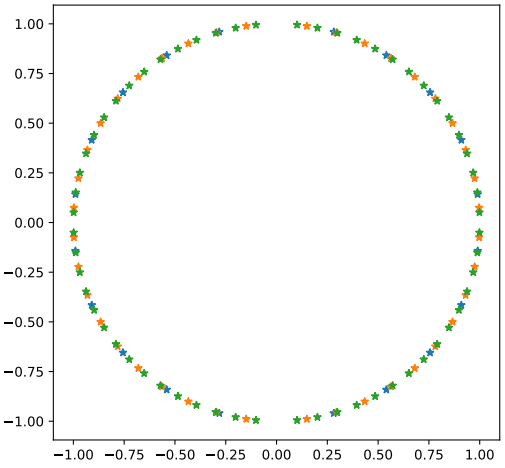

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: viot
"""
import numpy.polynomial.polynomial as
    ↪ npoly
import matplotlib.pyplot as plt
import numpy as np

sizes=[20, 40, 60]
plt.figure(figsize=(6,6))
for j in sizes:
    coeffs=np.ones(j+1)
    for i in range(j):
        coeffs[i+1]=coeffs[i]/(i+1)
    roots=npoly.polyroots(coeffs)
    x=roots.real
    y=roots.imag
    plt.plot(x,y,'*')
plt.savefig('zeroexp.pdf')
```

FIGURE 3.3 – Tracé des zéros de la série tronquée aux ordres 20, 40 et 60 de la fonction exponentielle

On obtient que les zéros s'éloignent de plus en plus du centre du plan quand on augmente le nombre de termes de la zéros. Cela est une conséquence du fait que la fonction exponentielle ne s'annule jamais dans le plan complexe et donc à la limite d'un nombre de termes qui tend vers l'infini, on n'a aucun zéro dans le plan complexe.

Dans le deuxième exemple, on considère le développement en série de la fonction $\frac{1}{1+x^2}$.




```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: viot
"""
import numpy.polynomial.polynomial as
    ↪ npoly
import matplotlib.pyplot as plt
import numpy as np

sizes=[20, 40, 60]
plt.figure(figsize=(6,6))
for j in sizes:
    coeffs=np.zeros(j+1)
    coeffs[0]=1
    for i in np.arange(0,j//2):
        coeffs[2*i+2]=-coeffs[2*i]
    roots=npoly.polyroots(coeffs)
    print(coeffs)
    x=roots.real
    y=roots.imag
    plt.plot(x,y,'*')
plt.savefig('poly2.pdf')
```

FIGURE 3.4 – Tracé des zéros de la série tronquée aux ordres 20, 40 et 60 de la fonction $\frac{1}{1+x^2}$.

Dans ce deuxième exemple, les zéros de la série s'accumule sur le cercle de rayon unité correspondant

au rayon de convergence de la série.

4 Les nombres aléatoires

4.1 Introduction

L'importance de générer des nombres aléatoires est en croissante continue depuis ces dernières décennies. Une raison principale est associée à la l'existence de différentes échelles de temps dans les phénomènes physiques, chimiques et biologiques, voire financiers. En considérant l'échelle de temps la plus grande, on peut justifier (avec un degré d'approximation plus ou moins bien maitrisé) que la dynamique puisse être décrite par une dynamique brownienne.. Un autre domaine faisant intervenir la génération de nombres aléatoires en grande quantité est la simulation Monte Carlo qui permet d'évaluer des intégrales de très grande dimension.

Depuis le début des ordinateurs, la génération de nombres aléatoires est une question essentielle, car on attend de la séquence générée qu'elle satisfasse un nombre de constraintes en nombre quasiment illimité. Un premier groupe de constraintes est que les nombres générés suivent le plus pret possible la distribution de probabilité que l'on souhaite reproduire. Une première manière de vérifier cette propriété est de calculer un nombre de moments de la distribution à partir des nombres générés. Une seconde constrainte non triviale à vérifier est que la correlation entre nombres aléatoires reste égale à zéro. En d'autres termes, quand on a généré un nombre, n'importe quel autre nombre doit pouvoir apparaitre lors d'un tirage suivant. On peut aussi analyser la correlation entre des séquences successives, etc...

Pour commencer, si on est capable de générer une distribution uniforme sur un intervalle donnée ou sur un ensemble discret d'entiers, on peut alors générer une autre séquence de nombres satisfaisant une autre distribution de probabilité et c'est la raison pour laquelle on retrouve dans les bibliothèques de langage les possibilités de générer des nombres à partir d'une distribution gaussienne, poissionnienne, de Cauchy,...

Il existe plusieurs méthodes pour générer des séquences de nombre pseudo-aléatoires, mais la majorité des générateurs reposent des relations de congruence linéaire entre entiers de la forme

$$x_{n+1} = (ax_n + c) \, mod \, m \tag{4.1}$$

où a c sont des entiers premiers et m est une puissance de 2 associée à la capacité de l'ordinateur.

m définit la période maximale les nombres générés sont à nouveau la même suite que l'on a utilisé au commencement de la séquence généré. A l'époque historique, où la représentation d'entiers était faire sur un octet, la séquence de nombres était très peu aléatoire. Même avec des registres 32bits (nos ordinateurs il y a quinze ans !), cela represente quelques milliards de nombres ce qui peut se reveler insuffisant. Aujourd'hui avec des registres de 64 bits ou 128 bits, le problème de la période n'est plus d'actualité. Pour permettre de diminuer les correlations de manière importante, il convient d'aller un peu plus loin en utilisant des combinaisons de congruences ce qui se retrouve dans de nombreux générateurs tel que le Mersenne.

Un dernier élément qu'il convient de connaitre est la notion de graine : quand on démarre un générateur, il doit utiliser une initialisation pour démarrer le calcul de la suite. Si on ne donne pas une valeur initiale il en choisit une par défaut. Mais si on précise rien, quand on recommence une deuxième le calcul on repart avec la même séquence. Si cela reste très utile lors de la vérification d'un programme, c'est évidemment catastrophique quand on peut faire une estimation d'une grandeur à partir de nombres aléatoires si on réutilise toujours les mêmes nombres.

Une fois que l'on a en tête les principes de génération des nombres aléatoires, on doit alors explorer les possibilités offertes par les bibliothèques et non pas réécrire des générateurs dérivés à partir d'une générateur interne mal controlé. Nous allons illustrer quelques distributions classiques à partir du langage Python, mais bien évidemment nous reviendrons en séances d'exercice pour le langage C++.

4.2 Distributions uniformes

4.2.1 Distribution discrete uniforme

Pour générer des nombres aléatoires entre 1 et 6 avec une distribution uniforme, on fait appel à la bibliothèque numpy qui permet de générer en une fois une grande liste de nombres aléatoires. Pour 500 nombres, on compare sur la figure 4.1 le résultat de la simulation avec la probabilité correspondant à la limite d'un nombre de tirages infini.

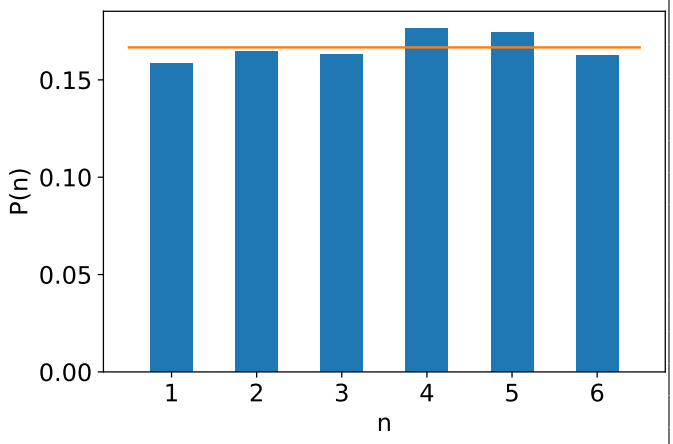


FIGURE 4.1 – Comparaison d'un histogramme de nombres aléatoires issus d'une distribution uniforme et de la probabilité d'un nombre infini de tirages.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: viot
"""
import numpy as np
import matplotlib.pyplot as plt

x=np.random.randint(low=1,high=7,size
    ↪ =5000)
bin = [x1 + 0.5 for x1 in range(0, 7)]
plt.hist(x,bins=bin,density='True',rwidth
    ↪ = 0.5)
plt.xlabel('n',fontsize=15)
plt.ylabel('P(n)',fontsize=15)
plt.tick_params(labelsize=15)
plt.plot([0.5,6.5],[1/6,1/6])
plt.tight_layout()
plt.savefig("discrete.pdf")
plt.show()
```

4.2.2 Distribution continue uniforme

Pour générer des nombres aléatoires dans l'intervalle $[0,1)$ avec une distribution uniforme, on fait appel à la bibliothèque numpy qui permet de générer en une fois une grande liste de nombres aléatoires. Pour 5000 nombres, on compare sur la figure 4.2 le résultat de la simulation avec la probabilité correspondant à la limite d'un nombre de tirages infini.

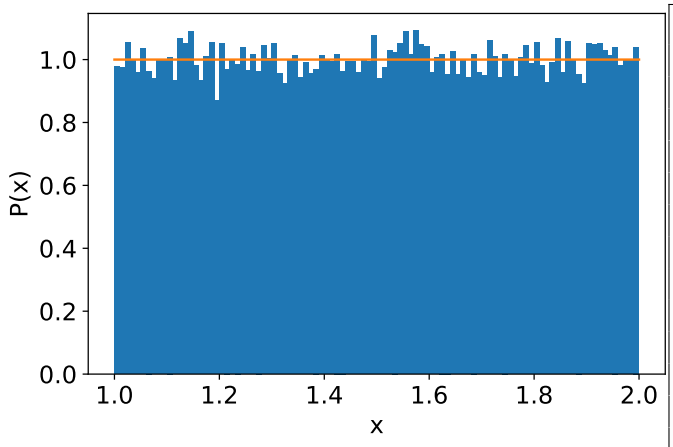


FIGURE 4.2 – Comparaison d'un histogramme de nombres aléatoires d'une distribution uniforme et de la probabilité d'un nombre infini de tirages.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

@author: viot
"""
import numpy as np
import matplotlib.pyplot as plt
x=np.random.uniform(1,2,size=50000)
plt.hist(x,bins=100,density='True')
plt.xlabel('x',fontsize=15)
plt.ylabel('P(x)',fontsize=15)
plt.tick_params(labelsize=15)
plt.plot([1,2],[1,1])
plt.tight_layout()
plt.savefig("uniform.pdf")
plt.show()
```

4.3 Distribution non uniformes

4.3.1 Distribution gaussienne

Pour intégrer une équation différentielle stochastique, il est nécessaire de générer des nombres aléatoires indépendants comme d'habitude et dont la distribution de probabilité est une distribution gaussienne. A nouveau on peut comparer sur la figure suivante l'histogramme d'un nombre fini de points avec la distribution gaussienne exacte.

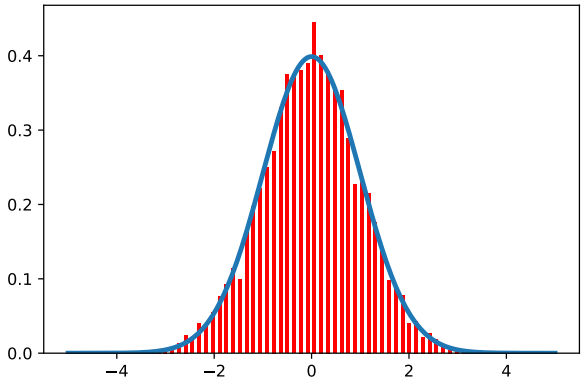


FIGURE 4.3 – Comparaison d'un histogramme de nombres aléatoires d'une distribution gaussienne et de la probabilité d'un nombre infini de tirages.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

@author: viot
"""
import numpy as np
import matplotlib.pyplot as plt
x=np.random.uniform(1,2,size=50000)
plt.hist(x,bins=100,density='True')
plt.xlabel('x',fontsize=15)
plt.ylabel('P(x)',fontsize=15)
plt.tick_params(labelsize=15)
plt.plot([1,2],[1,1])
plt.tight_layout()
plt.savefig("uniform.pdf")
plt.show()
```

4.3.2 Distribution de Cauchy

La distribution de Cauchy est donnée par la loi

$$P(x;x_0,\gamma) = \frac{1}{\pi\gamma\left[1+(\frac{x-x_0}{\gamma})^2\right]} \tag{4.2}$$

Cette loi de probabilité dont le support peut aller sur l'ensemble de la droite réelle ou une demi-droite réelle a une moyenne moyenne ainsi que l'ensemble des moments supérieurs de la distribution qui ne sont pas définis. Cela signifie que pour un nombre fini de valeurs la valeur moyenne est globalement controlé par la plus grande valeur du tirage. Il est possible de générer des valeurs aléatoires simplement à partir d'une instruction d'une bibliothèque.

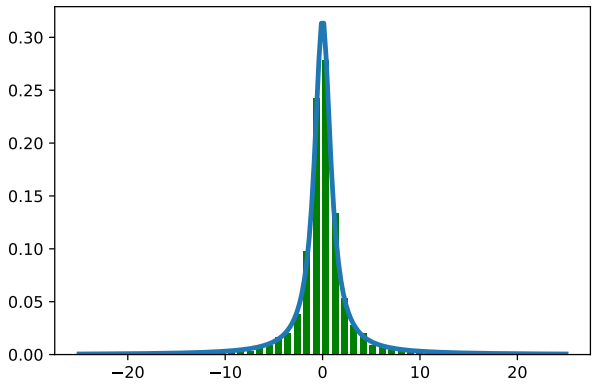


FIGURE 4.4 – Comparaison d'un histogramme de nombres aléatoires d'une distribution de Cauchy centrée sur l'origine et de la probabilité d'un nombre infini de tirages.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 16 09:31:44 2020

@author: viot
"""
import numpy as np
import matplotlib.pyplot as plt
def cauchy(x,mean=0.,x0=0.,gamma=1):
    return(1/(np.pi*gamma*(1+((x-x0)/gamma)
        ↪ **2)))

x1=np.linspace(-25,25,200)
x=np.random.standard_cauchy(5000)
x = x[(x>-25) & (x<25)] #on tronque les
    ↪ valeurs extremes pour tracer l'
    ↪ histogramme
plt.hist(x,bins=50,density=True,color='
    ↪ green',rwidth=0.7)
plt.plot(x1,cauchy(x1),lw=3)
plt.savefig("cauchy.pdf")
plt.show()
```

4.4 Conclusion

La liste de distribution de probabilité disponible dans une bibliothèque dépasse ces premiers exemples. Nous reverrons en séances des propriétés supplémentaires ainsi que quelques méthodes pour générer des distributions non standards. Notons qu'en Python, par défaut, la graine est changées entre deux simulations consécutives, ce qui n'est pas le cas en général pour les langages compilés sauf pour la STL en C++.

5 Equations différentielles

5.1 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 (boost, GSL) et bien entendu python. Il est donc préférable d'utiliser ces bibliothèques plutôt que de réécrire un code peu performant et probablement faux dans un premier temps.

5.2 Définitions

Soit une fonction numérique notée $y(x)$ définie sur un intervalle 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

$$F(x, y, y', y'', \dots y^{(p)}) = 0 \tag{5.1}$$

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 (5.1) toute fonction $y(x)$ de classe C_p qui vérifie l'équation (5.1).

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

$$y^{(p)} = f(x, y, y', y'', \dots y^{(p-1)}) \tag{5.2}$$

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

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ &\dots \\ y_p &= y^{(p-1)} \end{aligned} \tag{5.3}$$

on peut exprimer l'équation (5.2) sous la forme

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ &\dots \\ y_p' &= f(x, y, y_1, y_2, \dots y_p) \end{aligned} \tag{5.4}$$

5.3 Equations différentielles “spéciales”

5.3.1 Introduction

Une classe restreinte, mais importante, d'équations différentielles ont des solutions mathématiques sous la forme de fonctions transcendantes ou de fonctions spéciales en général. Nous allons donc donner un rapide aperçu de ces équations différentielles particulières qui contiennent en autres quelques équations différentielles non-linéaires, dont l'immense majorité ne peut être résolue que numériquement.

5.3.2 Equations du premier ordre

Equation à coefficients constants

Les équations différentielles à coefficients constants de forme générale

$$a\frac{dy}{dx} + by(x) = c(x) \tag{5.5}$$

ont pour solution générale

$$y(x) = e^{-b/ax}\left(\int dt \frac{c(t)}{a} e^{b/at} + cste\right) \tag{5.6}$$

Equation linéaire

Une équation différentielle linéaire a pour forme générale

$$\frac{dy}{dx} + f(x)y(x) = g(x) \tag{5.7}$$

où $f(x)$ et $g(x)$ sont des fonctions arbitraires. La solution de cette équation différentielle est

$$y(x) = e^{-\int^x f(u)du}(\left(\int dt g(t) e^{\int^t f(u)du} + Cste\right) \tag{5.8}$$

Equation de Bernouilli

L'équation de Bernouilli a pour forme générale

$$\frac{dy}{dx} + f(x)y(x) + g(x)y(x)^a = 0 \tag{5.9}$$

où $f(x)$ et $g(x)$ sont des fonctions arbitraires. En introduisant le changement de fonction

$$y(x) = u(x)^{\frac{1}{1-a}} \tag{5.10}$$

l'équation différentielle devient une équation différentielle du premier ordre

$$\frac{du}{dx} = (a-1)\left(u(x)f(x) + g(x)\right) \tag{5.11}$$

qui s'intégre alors comme une équation différentielle linéaire.

Equation de Clairaut

Une équation de Clairaut est une équation de la forme

$$y(x) = x\frac{dy}{dx} + g\left(\frac{dy}{dx}\right) \tag{5.12}$$

Ce type d'équation différentielle admet une solution linéaire de la forme

$$y(x) = Cx + g(C) \tag{5.13}$$

où C est une constante arbitraire. On peut aussi exprimer la solution de l'équation sous forme paramétrique

$$x(s) = -g'(s) \tag{5.14}$$
$$y(s) = g(s) - sg'(s) \tag{5.15}$$

Equation de Riccatti

La forme d'une équation de Riccatti est donnée par l'expression suivante

$$\frac{dy}{dx} - f(x)y(x)^2 - g(x)y(x) - h(x) = 0 \tag{5.16}$$

où $f(x)$, $g(x)$ et $h(x)$ sont des fonctions arbitraires. Il n'y a pas de solutions générales pour l'équation de Riccatti. Quand $h(x) = 0$, on retrouve la forme d'une équation de Bernouilli dont la solution a été donnée plus haut. La forme dite spéciale de l'équation de Riccatti est définie par l'équation différentielle suivante

$$\frac{dy}{dx} - ay(x)^2 - bx^c = 0 \tag{5.17}$$

Si de plus c est de la forme

$$c = \frac{-4 * i}{2i - 1} \tag{5.18}$$

avec i entier relatif, on peut obtenir une expression analytique de l'équation différentielle de Riccati.

5.3.3 Equation différentielles du second ordre

Equations différentielles à coefficients constants

Les équations différentielles à coefficients constants de forme générale

$$a\frac{d^2y}{dx^2} + b\frac{dy}{dx} + cy(x) = d(x) \tag{5.19}$$

ont pour solution complète

$$y(x) = \sum_{i=1}^{2} e^{u_i x}\left(C_i + \int dt \frac{(-1)^{i+1} d(t)}{\sqrt{b^2 - 4ac}} e^{-u_i t}\right) \tag{5.20}$$

où

$$u_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{5.21}$$

sont les solutions de l'équation caractéristique

$$au^2 + bu + c = 0 \tag{5.22}$$

5.3.4 Equation de Bessel

Les équations différentielles de Bessel sont définies par les équations suivantes

$$x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}+(x^2-n^2)y(x)=0 \tag{5.23}$$

et

$$x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}-(x^2-n^2)y(x)=0 \tag{5.24}$$

ce qui donne respectivement pour solutions les équations de Bessel J et Y pour la première et les équations de Bessel modifiées I et K pour la seconde.

5.3.5 Equation différentielle erreur

On appelle equation différentielle erreur l'équation suivante

$$\frac{d^2y}{dx^2}+2x\frac{dy}{dx}-2ny(x)=0 \tag{5.25}$$

où n est un entier. Pour $n=0$ la solution est de la forme

$$y(x)=c+derf(x) \tag{5.26}$$

où c et d sont des constantes arbitraires. Dans le cas où où $n=1$, la solution est de la forme

$$y(x)=cx+d(e^{-x^2}+\sqrt{\pi}erf(x)x) \tag{5.27}$$

De manière générale, la solution est une combinaison générale de fonction de WhittakerM.

5.3.6 Equation différentielle d'Hermite

L'équation différentielle d'Hermite est très proche de l'équation différentielle précédente, puisque on a

$$\frac{d^2y}{dx^2}-2x\frac{dy}{dx}+2ny(x)=0 \tag{5.28}$$

De manière générale, la solution est aussi une combinaison générale de fonction de WhittakerM.

5.4 Méthodes d'intégration à pas séparé

5.4.1 Introduction

Soit l'équation différentielle définie par les équations (5.4). 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 $x_1, x_2, \dots x_N$ 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

$$h=\frac{(b-a)}{N} \tag{5.29}$$

ce qui donne

$$x_k=a+kh \tag{5.30}$$

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

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

$$\begin{aligned} y_{k+1} &= y_k + h\phi(x_k, y_k, h) \\ k &= 0, 1, \ldots N \qquad \text{avec } y_0 \text{ donné} \end{aligned} \tag{5.31}$$

la fonction ϕ 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 $y_k, y_{k-1}, \ldots y_{k-r}$.

5.4.2 Méthode d'Euler

Cette méthode est définie par

$$y_{k+1} = y_k + hf(y_k, x_k) \tag{5.32}$$

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

$$y(x_k + h) = y(x_k) + hy'(x_k) + \frac{h^2}{2}y''(x_k + \theta h) \tag{5.33}$$

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

$$y(x_k + h) = y(x_k) + hf(y_k, x_k) + \frac{h^2}{2}y''(x_k + \theta h) \tag{5.34}$$

D'après la définition de la méthode

$$\frac{1}{h}(y(x_k + h) - y(x_k) - \phi(y_k(x_k), x_k, h)) = \frac{y(x_k + h) - y(x_k)}{h} - f(y(x_k), x_k) \tag{5.35}$$

ou

$$\frac{1}{h}(y(x_k + h) - y(x_k) - \phi(y_k(x_k), x_k, h)) = h\frac{y''(x_k + \theta h)}{2} \tag{5.36}$$

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

$$\max \| \frac{1}{h}(y(x_k + h) - y(x_k) - \phi(y_k(x_k), x_k, h)) \| \leq Kh \tag{5.37}$$

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

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

5.4.4 Méthodes RK implicites à un point

La formule de récurrence est définie par la relation

$$y_{k+1} = y_k + h[(1-\theta)f(x_k, y_k) + \theta f(x_{k+1}, y_{k+1})] \tag{5.38}$$

où θ 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.

5.4.5 Méthodes RK explicites à 2 points intermédiaires

Ces méthodes sont définies par les relations

$$\begin{aligned} y_{k,1} &= y_k + \frac{h}{3} f(x_k, y_k) \\ y_{k,2} &= y_k + \frac{2h}{3} f\left(x_k + \frac{h}{3}, y_{k,1}\right) \\ y_{k+1} &= y_k + \frac{h}{4}\left(f(x_k, y_k) + 3f\left(x_k + \frac{2h}{3}, y_{k,2}\right)\right) \end{aligned} \tag{5.39}$$

ou par

$$\begin{aligned} y_{k,1} &= y_k + \frac{h}{2} f(x_k, y_k) \\ y_{k,2} &= y_k + h\left(f(x_k, y_k) + 2f\left(x_k + \frac{h}{2}, y_{k,1}\right)\right) \\ y_{k+1} &= y_k + \frac{h}{6}\left(f(x_k, y_k) + 4f\left(x_k + \frac{h}{2}, y_{k,1}\right) + f\left(x_{k+1}, y_{k,2}\right)\right) \end{aligned} \tag{5.40}$$

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

5.4.6 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.

$$\begin{aligned} y_{k,1} &= y_k + \frac{h}{2} f(x_k, y_k) \\ y_{k,2} &= y_k + \frac{h}{2} f\left(x_k + \frac{h}{2}, y_{k,1}\right) \\ y_{k,3} &= y_k + \frac{h}{2} f\left(x_k + \frac{h}{2}, y_{k,2}\right) \\ y_{k+1} &= y_k + \frac{h}{6}\left(f(x_k, y_k) + 2f\left(x_k + \frac{h}{2}, y_{k,1}\right) + 2f\left(x_k + \frac{h}{2}, y_{k,2}\right) + f\left(x_{k+1}, y_{k,3}\right)\right) \end{aligned} \tag{5.41}$$

Cette méthode est d'ordre 4.

5.4.7 Formule générale des méthodes RK explicites

Les méthodes de Runge Kutta s'écrivent de manière générale

$$\begin{aligned} K_1 &= h[f(x_k+\theta_1 h, y_k+\alpha_{1,1}K_1+\alpha_{1,2}K_2+\ldots+\alpha_{1,n}K_n)] \\ K_2 &= h[f(x_k+\theta_2 h, y_k+\alpha_{2,1}K_1+\alpha_{2,2}K_2+\ldots+\alpha_{2,n}K_n)] \\ &\ldots\ldots \\ K_n &= h[f(x_k+\theta_n h, y_k+\alpha_{n,1}K_1+\alpha_{n,2}K_2+\ldots+\alpha_{n,n}K_n)] \\ y_{k+1} &= y_k + h[\gamma_1 K_1+\gamma_2 K_2+\ldots\gamma_n K_n] \end{aligned} \tag{5.42}$$

Les coefficients sont déterminés afin que l'ordre soit le plus élevé possible. On note A la matrice de coefficients $(\alpha_{i,j})$, Γ le vecteur des coefficients γ_i et Θ le vecteur des coefficients θ_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 (5.42) donne

	γ_1	γ_2	$\cdots$	γ_n
θ_1	$\alpha_{1,1}$	$\alpha_{1,2}$	$\cdots$	$\alpha_{1,n}$
θ_2	$\alpha_{2,1}$	$\alpha_{2,2}$	$\cdots$	$\alpha_{2,n}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	
θ_n	$\alpha_{n,1}$	$\alpha_{n,2}$	$\cdots$	$\alpha_{n,n}$

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

5.5 Méthodes d'intégration à pas variable

5.5.1 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

$$y(x+2h) = y_1 + (2h)^5\phi + O(h^6) \tag{5.43}$$

$$y(x+2h) = y_2 + 2(h^5)\phi + O(h^6) \tag{5.44}$$

où ϕ 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

$$\Delta = y_2 - y_1 \tag{5.45}$$

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

5.6 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 γ_i (voir section 5.4.7), on peut changer l'ordre de l'évaluation de la solution pour un pas de temps h donné.

$$y_1 = y(x) + \sum_{i=1}^{6} \gamma_i K_i + 0(h^6) \tag{5.46}$$

$$y_2 = y(x) + \sum_{i=1}^{6} \gamma_i^* K_i + 0(h^5) \tag{5.47}$$

ce qui conduit à une estimation de l'erreur

$$\Delta = \sum_{i=1}^{6} (\gamma_i - \gamma_i^*) K_i \tag{5.48}$$

Pour déterminer la valeur du pas la plus adaptée, on note tout d'abord que Δ est calculé à l'ordre h^5. Si on a un pas h_1 qui donne une erreur Δ_1, le pas h_0 donné pour une erreur Δ_0 fixée à l'avance, est donné par la relation

$$h_0 = h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{1/5} \tag{5.49}$$

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 Δ_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.

5.6.1 Exemples

Nous allon tester les méthodes précédentes sur quelques équations différenielles : le premier test consiste à évaluer la différence entre la solution exacte pour une équation différentielle quand elle existe et le second test à contrôler cette différence en testant l'évolution d'une grandeur conservée associée à l'éqution différentielle, ce qui est le cas de l'évolution d'une système Hamiltonien pour lequel l'énergie totale est une grandeur conservée. Les deux méthodes de RK implémentées dans Python sont des méthodes où l'on controle la précision par deux evaluations différentes d'un pas d'intégration. RK23 signifie que l'évaluation se fait avec un développement à l'ordre 2 et un à l'ordre 3. De manière similaire pour la méthode RK45. Dans les deux figures qui suivent, on laisse par défaut les valeurs de tolérances pour faire l'intégration de l'équation différentielle.

La première equation que l'on considère est celle d'un oscillateur harmonique amorti.

$$y'' + 2y + 2y = 0 \tag{5.50}$$

avec les conditions initiales suivantes $y(0) = 1$ et $y'(0) = 0$. Pour être calculée numériquement, on réécrit cette équation sous la forme suivante

$$Y_0' = Y_1 \tag{5.51}$$

$$Y_1' = -2Y_1 - 2Y_0 \tag{5.52}$$

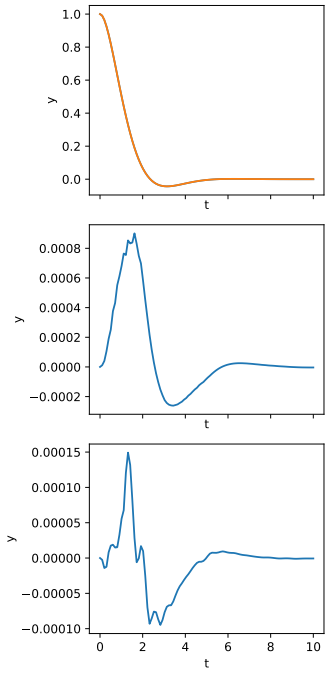


FIGURE 5.1 – (Figure du haut) Résolutions numériques de l'équation différentielle par Runge Kutta d'ordre 2 et d'ordre 4. Figure du milieu : écart entre la solution exacte et la solution RK2.Figure du bas : écart entre la solution exacte et la solution RK4.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: viot
"""
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

def solex(t):
    return(np.exp(-ts)*(np.sin(t)+np.cos(t)))
def dY_dt(t,Y):
    x,v=Y
    return [v, -2*v - 2*x ]


sol = solve_ivp(dY_dt,[0, 10], [1, 0],
    ↪ dense_output=True,method='RK23')
sol2 = solve_ivp(dY_dt,[0, 10], [1, 0],
    ↪ dense_output=True,method='RK45')
ts = np.linspace(0, 10, 100)
ys = sol.sol(ts)
ys4 = sol2.sol(ts)
fig,ax = plt.subplots(3,1,figsize=(4,8),sharex=
    ↪ True)

ax[0].plot(ts, ys.T[:,0])
ax[0].plot(ts, solex(ts))
ax[0].set_xlabel("t")
ax[0].set_ylabel("y")
ax[1].plot(ts, ys.T[:,0]-solex(ts))
ax[1].set_xlabel("t")
ax[1].set_ylabel("y")
ax[2].plot(ts, ys4.T[:,0]-solex(ts))
ax[2].set_xlabel("t")
ax[2].set_ylabel("y")# plt.xlabel("t")

plt.tight_layout()
plt.savefig('RK2345.pdf')
```

On voit que la solution numérique est de meilleure précision avec la méthode Runge-Kutta d'ordre 4 qu'avec celle d'ordre 2.

Dans le deuxième exemple, l'équation différentielle est celle d'un oscillateur anharmonique. L'énergie mécanique doit être conservée au cours du temps. Un calcul rapide montre que l'énergie totale de ce système avec la condition initiale $y(0) = 1$ et $y'(0) = 0$, donne une valeur de 3/4.

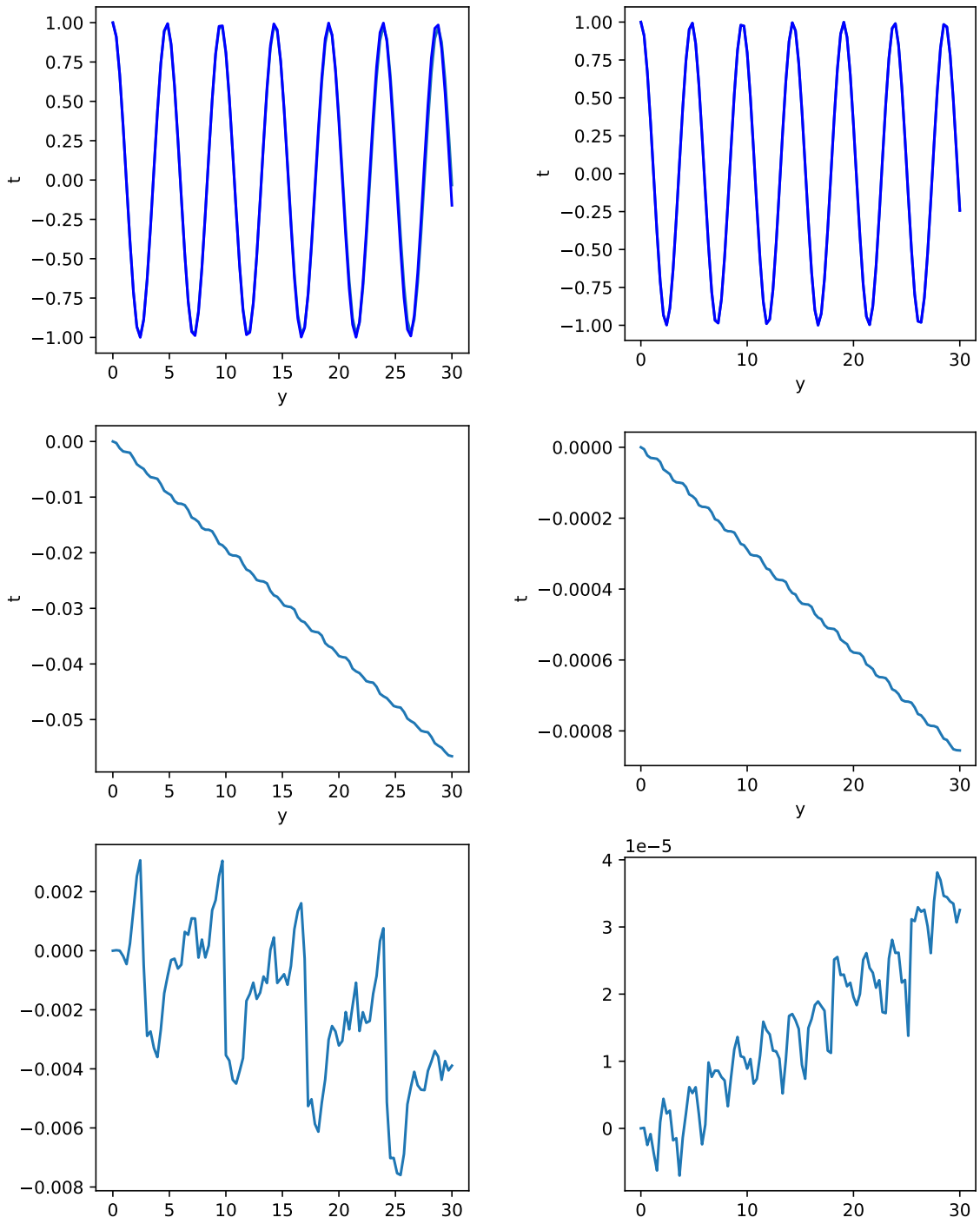


FIGURE 5.2 – figure
(Figure du haut)Résolutions numériques de l'équation différentielle par Runge Kutta d'ordre 2 et d'ordre 4. Figure du milieu : écart entre l'énergie numérique RK2 et la valeur exacte.Figure du bas : écart entre la l'énergie numérique RK4 et la valeur exacte. Les trois figures de gauche correpondent à une précision relative de 10^{-3} et celles de droite à une précision relative de 10^{-5}

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: viot
"""

import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
def solex(t):
    return(np.exp(-ts)*(np.sin(t)+np.cos(t)
        ↪ ))
def dY_dt(t,Y):
    x,v=Y
    return [v, - x -x**3]

sol = solve_ivp(dY_dt,[0, 30], [1, 0],
    ↪ dense_output=True,method='RK23')
sol2 = solve_ivp(dY_dt,[0, 30], [1, 0],
    ↪ dense_output=True,method='RK45')
ts = np.linspace(0, 30, 100)
ys = sol.sol(ts)
ys4 = sol2.sol(ts)
Et=0.5*ys.T[:,0]*ys.T[:,0]+0.25*ys.T
    ↪ [:,0]**4+0.5*ys.T[:,1]**2
Et2=0.5*ys4.T[:,0]*ys4.T[:,0]+0.25*ys4.T
    ↪ [:,0]**4+0.5*ys4.T[:,1]**2
fig,ax= plt.subplots(3,1,figsize=(4,10))

ax[0].plot(ts,ys.T[:,0] )
ax[0].plot(ts,ys4.T[:,0],color='blue')
ax[0].set_xlabel("y")
ax[0].set_ylabel("t")
ax[1].plot(ts, Et-3/4)
ax[1].set_xlabel("y")
ax[1].set_ylabel("t")
ax[2].plot(ts, Et2-3/4)
plt.tight_layout()
plt.savefig('EnergieRK.pdf')
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

@author: viot
"""
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

def solex(t):
    return(np.exp(-ts)*(np.sin(t)+np.cos(t)
        ↪ ))
def dY_dt(t,Y):
    x,v=Y
    return [v, - x -x**3]

sol = solve_ivp(dY_dt,[0, 30], [1, 0],
    ↪ dense_output=True,method='RK23',
    ↪ rtol=1e-5)
sol2 = solve_ivp(dY_dt,[0, 30], [1, 0],
    ↪ dense_output=True,method='RK45',
    ↪ rtol=1e-5)
ts = np.linspace(0, 30, 100)
ys = sol.sol(ts)
ys4 = sol2.sol(ts)
Et=0.5*ys.T[:,0]*ys.T[:,0]+0.25*ys.T
    ↪ [:,0]**4+0.5*ys.T[:,1]**2
Et2=0.5*ys4.T[:,0]*ys4.T[:,0]+0.25*ys4.T
    ↪ [:,0]**4+0.5*ys4.T[:,1]**2
fig,ax= plt.subplots(3,1,figsize=(4,10))
ax[0].plot(ts,ys.T[:,0] )
ax[0].plot(ts,ys4.T[:,0],color='blue')
ax[0].set_xlabel("y")
ax[0].set_ylabel("t")
ax[1].plot(ts, Et-3/4)
ax[1].set_xlabel("y")
ax[1].set_ylabel("t")
ax[2].plot(ts, Et2-3/4)
plt.tight_layout()
plt.savefig('EnergieRKprecis.pdf')
```

Si on reprend le meme exemple et que l'on abaisse par deux ordres de grandeur la valeur de la tolérance relative, on obtient une amélioration de la précision particulièrement dans le cas de la méthode RK45. C'est bien entendu au prix d'un calcul supplémentaire, mais dans ce cas simple, il reste très peu perceptible.

On voit clairement sur cette exemple que si les solutions numériques données par les deux méthodes semblent très proches après un temps de l'ordre de 6 périodes d'oscillation, une analyse plus fine qui calcule l'énergie totale pour chacune de ces méthodes montre dans les deux cas l'apparition d'un biais systèmatique où le système ne conserve pas l'énergie. Dans le cas de la simulation numérique d'un grand nombre de particules et pour des temps considérablement plus long, cela entraine une non conservation de l'énergie mécanique et des résultats qui font évoluer le système vers des régions non physiques. Nous verrons que pour satisfaire cette constrainte essentielle de la conservation de l'énergie, il est nécessaire d'utiliser des algorithmiques symplectiques.

6 Equations différentielles stochastiques

6.1 Introduction

Depuis les observations de Robert Brown sur le déplacement des grains de pollen en solution et les travaux d'Einstein et de Smoluchowski qui ont proposé une description de ce phénomène, on s'est rendu compte que de nombreux situations pouvaient être décrites en faisant intervenir des "forces aléatoires" : la dynamique des particules colloïdales en solution est bien décrite par une dynamique Brownienne, (voir le cours de simulation numérique en Physique Statistique), la cinétique d'une réaction chimique peut être décrite de manière plus réaliste en incorporant des fluctuations liées à l'environnement par l'addition d'un bruit aléatoire, les évolutions des marchés financiers ont été depuis ces trente dernières années l'objet d'une intense recherche par des modélisations faisant intervenir des forces aléatoires,...

Le propos de ce chapitre est de donner des principes de bases pour la résolution des équations différentielles stochastiques. Sur un sujet très vaste et en évolution rapide, nous serons très loin de l'exhaustivité. Nous allons donc rappeler les définitions et propriétés fondamentales concernant les processus stochastiques ainsi que ceux des équations différentielles stochastiques, et nous renvoyons le lecteur à plusieurs ouvrages de base concernant les processus stochastiques.

6.2 Variables aléatoires et processus stochastiques

Pour définir une variable aléatoire X, il est nécessaire d'avoir à la fois un ensemble de valeurs que peut prendre la variable aléatoire X et une loi de probabilité définissant la manière dont les valeurs de cette variable aléatoire peuvent apparaitre.

Compte tenu du fait que la variable aléatoire X prend des valeurs successives à chaque nouveau tirage, on associe à chacun de ces tirages un écoulement du temps, et les instants successifs sont notés $t_1, t_2, ..., t_n, ...$

Un processus stochastique est défini comme une dynamique dont les évévements successifs sont donnés par une loi de probabilité. Trois définitions sont particulièrement importantes pour classifier le type de processus stochastique que l'on étudie. Si on note $x_1, x_2, ...x_n$ la séquence de la variable X, on définit la probabilité jointe de ces n valeurs successives

$$p(x_1, t_1; x_2, t_2; ...x_n, t_n), \tag{6.1}$$

comme la probabilité de voir réaliser la séquence de valeurs de X constituant des valeurs (dans l'ordre) de x_1 à x_n aux instants de t_1 à t_n.

On définit alors la probabilité conditionnelle de la séquence $x_i, x_{i+1}, ...x_n$ arrivant aux instants $t_i, t_{i+1}, ...t_n$ sachant la séquence $x_1, x_2, ..., x_{i-1}$ a eu lieu aux instants $t_1, t_2, ..., t_{i-1}$ comme

$$p(x_i, t_i; x_{i+1}, t_{i+1}; ...x_n, t_n | x_1, t_1; x_2, t_2; ...x_{i-1}, t_{i-1}) = \frac{p(x_1, t_1; x_2, t_2; ...x_n, t_n)}{p(x_1, t_1; x_2, t_2; ...x_{i-1}, t_{i-1}),} \tag{6.2}$$

On définit la probabilité marginale $p(x_2, t_2)$ d'une probabilité jointe $p(x_2, t_2; x_1, t_1)$ comme la somme sur tous les événements x_1 qui sont apparus à l'instant t_1

$$p(x_2, t_2) = \int dx_1 p(x_2, t_2; x_1, t_1) \tag{6.3}$$

de manière similaire, on peut généraliser ce type d'équation pour la probabilité d'avoir un événements x_3 à t_3 sachant que l'on a eu un évévement x_1 à l'instant t_1.

$$p(x_3,t_3|x_1,t_1) = \int dx_2 p(x_3,t_3;x_2,t_2|x_1,t_1) \tag{6.4}$$

$$= \int dx_2 p(x_3,t_3|x_2,t_2,x_1,t_1)p(x_2,t_2|x_1,t_1) \tag{6.5}$$

Une classe particulèrement importante de processus stochastiques a été plus particulièrement étudiée ; il s'agit de dynamiques pour lesquelles la probabilité conditionannelle ne dépend pas de l'histoire du parcours de la "particule", mais uniquement de l'instant présent. En d'autres termes, les processus Markoviens sont définis par une loi de probabilité conditionnelle qui est indépendente des évévements antérieurs à l'instant présent. En termes de probabilités, la probabilité conditionnelle que l'événement x_i apparaisse à l'instant t_i est donnée par l'équation

$$p(x_i,t_i|x_{i-1},t_{i-1};...x_2,t_2;x_1,t_1) = p(x_i,t_i|x_{i-1},t_{i-1}) \tag{6.6}$$

Ainsi un Processus Markovien ne garde la mémoire que de la valeur d'où la variable est partie. Cette classe de processus stochastique n'est pas restrictive car elle contient de très nombreux exemples physiques comme la diffusion de particules, la simulation Monte Carlo, et aussi les modélisations des marchés financiers...

Pour un processus Markovien, l'équation (6.4) se simplifie compte tenu de la propriété, Eq. (6.8), et on obtient l'équation dite de Chapman-Kolmogorov

$$p(x_3,t_3|x_1,t_1) = \int dx_2 p(x_3,t_3|x_2,t_2)p(x_2,t_2|x_1,t_1) \tag{6.7}$$

Elle traduit le fait que les probabilités conditionnelles sont reliées entre elles et ne font pas intervenit des probabilités jointes, contrairement aux processus stochastiques en général.

Le raisonnement précédent correspondant aux processus à événements et temps discrèts peut être généralisé à des espaces d'évévements continus ainsi qu'une évolution à temps continu. L'équation de Chapman Kolmogorov peut-être aussi généralisée et après calculs on obtient (dans le cas où espace et temps sont continus)

$$\frac{\partial p(y,t|x,t')}{\partial t} = \frac{\partial}{\partial y}\left[A(y,t)p(y,t|x,t')\right] + \frac{1}{2}\frac{\partial^2}{\partial y^2}\left[B(y,t)p(y,t|x,t')\right] + \int dz(W(y|z,t),p(y,t|x,t')| - W(z|x,t)p(z,t|x,t')) \tag{6.8}$$

où les fonctions $A(x,t)$, $B(x,t)$ et W$(z|x,t)$ sont définis de manière suivante :

$$A(x,t) = \lim_{\Delta t\to 0}\frac{1}{\Delta t}\int_{|x-y|<\epsilon} dy(y-x)p(y,t+\Delta t|x,t), \tag{6.9}$$

$$B(x,t) = \lim_{\Delta t\to 0}\frac{1}{\Delta t}\int_{|x-y|<\epsilon} dy(y-x)^2 p(y,t+\Delta t|x,t), \tag{6.10}$$

et

$$W(x|z,t) = \lim_{\Delta t\to 0}\frac{1}{\Delta t}p(x,t+\Delta t|z,t). \tag{6.11}$$

Dans le cas où l'espace est discrèt et le temps continu, l'équation de Chapman-Kolmogorov prend alors la forme suivante

$$\frac{\partial p(n,t|n',t')}{\partial t} = \sum_m (W(n|m,t), p(m,t|n',t')| - W(m|n,t)p(n,t|n',t')) \tag{6.12}$$

Cette dernière équation correspond par exemple à la dynamique Monte-Carlo dans le cas d'un modèle d'Ising. Pour les différentes équations de Chapman-Kolmogorov, il existe une infinité de solutions qui satisfont ces équations et il est nécessaire de spécifier plus précisement la dynamique pour obtenir une solution unique. Nous allons voir maitenant des processus Markoviens très utiles.

6.3 Processus de Wiener, bruit blanc

6.3.1 Equation de diffusion

Quand un processus de Markov à espace et temps continus Eq. (6.13), est tel que $W(x|z,t) = 0$ pour tout temps t et x, z ainsi que $A(x,t) = 0$, et $B(x,t) = 1$ on obtient un processus de Wiener avec une équation d'évolution de la probabilité donnée par

$$\frac{\partial p(w,t|w_0,t_0)}{\partial t} = \frac{1}{2}\frac{\partial^2}{\partial w^2} p(w,t|w_0,t_0) \tag{6.13}$$

ce qui correspond à une équation de diffusion. La solution de cette equation est bien entendu exacte et donnée par une Gaussienne

$$p(w,t|w_0,t_0) = \frac{1}{\sqrt{2\pi(t-t_0)}} \exp\left(-\frac{(w-w_0)^2}{2(t-t_0)}\right) \tag{6.14}$$

Cela donne pour la valeur moyenne

$$< W(t) >= w_0 \tag{6.15}$$

et la variance

$$< (W(t) - w_0)^2 >= t - t_0 \tag{6.16}$$

Un processus de Wiener est aussi appelé mouvement Brownien puisque l'équation d'évolution de sa probabilité est identique à celle obtenue par Einstein pour la description du comportement erratique des particules en solution.

Trois propriétés essentielles caractérisent un processus de Wiener : (i) la trajectoire est continue (quand l'intervalle de temps tend vers zéro, les valeurs possibles de la position de la particule à l'instant $t + \Delta t$ reste au voisinage de la position de la particule à l'instant t. (ii) On peut montrer que la valeur de la vitesse en tout point est infinie ce qui correspond au fait que la trajectoire n'est pas différentiable (iii) Partant d'une même condition initiale, si on génère plusieurs dynamiques de Wiener, on obtient des trajectoires dont la valeur moyenne est la même, mais dont la dispersion augmente avec le temps, car la variance croît linéairement avec le temps (voir Fig. (6.2)), (iv) la dernière propriété importante concernant les processus de Wiener est liée à l'indépendance statistique des incréments successifs de la variable W.

6.3.2 Equation de Langevin

L'équation stochastique différentielle que l'on cherche à résoudre est l'équation de type Langevin définie comme suit

$$\frac{dx}{dt} = a(x,t) + b(x,t)\xi(t) \tag{6.17}$$

FIGURE 6.1 – Trajectoires de mouvement Brownien à une dimension en fonction du temps : les cinq trajectoires partent de l'origine.

FIGURE 6.2 – Bruit blanc à une dimension en fonction du temps

où $a(x,t)$ et $b(x,t)$ sont des fonctions continues et $\xi(t)$ est un fonction aléatoire fluctuante variant rapidement. Sans perdre de généralité, on peut toujours imposer (à une redéfinition près de la fonctions a) que la moyenne de ξ est nulle, $<\xi(t)>=0$ et on impose que

$$<\xi(t)\xi(t')>=\delta(t-t') \tag{6.18}$$

qui impose l'abscence de corrélations entre les différents temps du processus.

La notion de bruit blanc provient du fait que si l'on fait la transformée de Fourier de la fonction de corrélation temporelle définie par l'équation (6.18), on obtient une fonction indépendante de la fréquence, ce qui signifie que toutes les fréquences sont également représentées dans le spectre et par analogie avec le spectre lumineux, on qualifie ce processus de bruit blanc.

Soit le processus $u(t)$ défini par l'intégrale suivante

$$u(t)=\int_0^t dt'\xi(t') \tag{6.19}$$

Il est simple de vérifier que $u(t)$ et $u(t')-u(t)$ sont statistiquement indépendant pour $t'>t$. En d'autres termes $u(t)$ est un processus stochastique Markovien. De plus, on vérifie que $<u(t')-u(t)>=0$ et $<(u(t')-u(t))^2>=t'-t$, ce qui montre que le processus u(t) est en fait un processus de Wiener. On a donc ainsi le résultat suivant

$$dW(t)=\xi(t)dt \tag{6.20}$$

Il faut toutefois souligner qu'un processus de Wiener conduit à une trajectoire non différentiable, ce qui signifie que la définition de $\xi(t)$ est en fait mathématiquement délicate et l'équation de Langevin n'est pas a priori intégrable. Il faut donc préciser son sens ce que nous allons voir maintenant avec la procédure introduite par Ito. Pour comprendre simplement cette difficulté, on peut voir que $dW(t)$ est de l'ordre de $\sqrt{dt}$ contrairement à une forme différentielle usuelle.

6.4 Calcul d'Ito et équations différentielles stochastiques

6.4.1 Introduction

Pour résoudre l'équation de Langevin précédemment introduite, nous avons à calculer une intégrale de la forme

$$\int_{t_0}^t f(t)dW(t) \tag{6.21}$$

où $W(t)$ est un processus de Wiener et $f(t)$ une fonction quelconque La manière naturelle de chercher à calculer cette intégrale est de faire une discrétisation du temps avec un intervalle de temps constant et d'évaluer une intégrale de type Riemann. Ainsi on a

$$S_n=\sum_{i=1}^n f(\tau_i)(W(t_i)-W(t_{i-1})) \tag{6.22}$$

où τ_i est un temps intermédiaire entre t_{i-1} et t_i.

Si on choisit maintenant $f(t) = W(t)$, on peut calculer exactement la moyenne de l'intégrale sur les différentes réalisations du processus de Wiener

$$< S_n >= \sum_{i=1}^{n} (\text{Min}(\tau_i, t_i) - \text{Min}(\tau_i, t_{i-1})) \tag{6.23}$$

$$= \sum_{i=1}^{n} (\tau_i - t_{i-1}) \tag{6.24}$$

Si maintenant on choisit de prendre τ_i comme le barycentre des extrèmités de l'intervalle de temps

$$\tau_i = \alpha t_i + (1-\alpha) t_{i-1} \tag{6.25}$$

avec α compris entre 0 et 1, on obtient pour la valeur moyenne de S_n

$$< S_n >= \alpha(t - t_0) \tag{6.26}$$

ce qui conduit à avoir un résultat qui dépend complètement du point intermédiaire, même une fois la moyenne sur les différentes réalisations !

Pour définir de manière la valeur d'une intégrale stochastique, il est nécessaire de choisir le point intermédiaire et le choix d'Ito est de prendre $\alpha = 0$. Ainsi, on a

$$\int_{t_0}^{t} f(t) dW(t) = \lim_{n\to\infty} \sum_{i=1}^{n} f(t_{i-1})(W(t_i) - W(t_{i-1})) \tag{6.27}$$

Ce choix est physiquement raisonnable car il signifie que l'on choisit d'évaluer l'intégrale en prenant une valeur de la fonction f qui est indépendante du comportement du processus de Wiener dans le futur. Un respect de la causalité en quelque sorte !

L'intégrale stochastique d'Ito a des propriétés un peu non intuitives. En effet, on peut montrer en utilisant les propriétés d'un processus de Wiener que

$$\int_{t_0}^{t} f(t)(dW(t))^2 = \int_{t_0}^{t} f(t) dt \tag{6.28}$$

ce qui, une fois de plus, met en évidence que la forme différentielle $dW(t)$ est de l'ordre $\sqrt{dt}$

6.4.2 Calcul différentiel stochastique

Soit le changement de variable $dy = f(x(t), t)$. Si on fait un développement à l'ordre 2 en dx, on obtient

$$dy(t) = f(x(t+dt)) - f(x(t)) + \frac{\partial f}{\partial t} dt \tag{6.29}$$

$$= \frac{\partial f(x,t)}{\partial x} dx(t) + \frac{\partial^2 f(x,t)}{2\partial x^2} dx(t)^2 + \frac{\partial f}{\partial t} dt ... \tag{6.30}$$

Compte tenu de la forme différentielle de l'équation de Langevin, le terme en $dx(t)^2$ contient des termes en dt^2, $dWdt$ qui sont négligeables, mais il reste un terme $dW(t)^2$ qui est égale à dt que l'on doit conserver. Ainsi, on obtient le résultat suivant

$$\begin{aligned} dy(t) =& (\frac{\partial f(x,t)}{\partial t} + a(x,t)\frac{\partial f(x,t)}{\partial x} + \frac{\partial^2 f(x,t)}{2\partial x^2} b^2(x,t)) dt \\ &+ b(x,t) f'(x).dW(t) \end{aligned} \tag{6.31}$$

Une fois ces règles parfaitement définies,, il est possible de donner un sens à l'équation de Langevin ; sa solution est une intégrale stochastique intégrale donnée par

$$x(t) - x(t_0) = \int_{t_0}^{t} dt' a(x(t'), t') + \int_{t_0}^{t} dW(t') b(x(t'), t') \tag{6.32}$$

Nous allons voir maintenant quelques exemples très connus d'équations de Langevin avant d'aborder les méthodes de résolution numérique.

6.4.3 Processus d'Orstein-Uhlenbeck

Un procesus d'Ornstein-Uhlenbeck satisfait l'équation de Langevin suivante

$$\frac{dx}{dt} = \theta(\mu - x(t)) + \sigma\eta(t) \tag{6.33}$$

ou de manière équivalente (et mathématiquement plus correcte), on peut écrire sous la forme différentielle suivante

$$dx = \theta(\mu - x(t))dt + \sigma dW(t) \tag{6.34}$$

où $dW(t)$ est la forme différentielle d'un processus de Wiener.

On utilise le changement de variable suivant

$$y(t) = x(t)e^{\theta t} \tag{6.35}$$

En appliquant les règles d'Ito, on a

$$dy(t) = \theta x(t)e^{\theta t}dt + e^{\theta t}dx(t) \tag{6.36}$$

$$= \theta\mu e^{\theta t}dt + \sigma e^{\theta t}dW(t) \tag{6.37}$$

On peut alors faire l'intégrale stochastique, ce qui donne

$$y(t) = y(t_0) + \mu(e^{\theta t} - 1) + \int_{t_0}^{t} \sigma e^{\theta s} dW(s) \tag{6.38}$$

Revenant à la variable initiale $x(t)$, on obtient

$$x(t) = x(t_0)e^{-\theta t} + \mu(1 - e^{-\theta t}) + \int_{t_0}^{t} \sigma e^{\theta(s-t)} dW(s) \tag{6.39}$$

La valeur moyenne à l'instant t est donnée par

$$< x(t) >= x(t_0)e^{-\theta t} + \mu(1 - e^{-\theta t}) \tag{6.40}$$

tandis que la variance est donnée par

$$< x(t)^2 > - < x(t) >^2 = \frac{\sigma^2}{\theta}(1 - e^{-2\theta t}) \tag{6.41}$$

Le processus d'Ornstein-Uhlenbeck est un processus de Markov qui conduit donc une valeur moyenne qui devient indépendante de la condition initiale et dont la variance est finie quand le temps devient très grand.

On considère est le processus d'Ornstein-Uhlenbeck suivant

$$\frac{dx}{dt} = -5(x - 1) + \eta(t) \tag{6.42}$$

où $\eta(t)$ est un bruit blanc gaussien.

A ce jour, le module de la résolution des équations différentielles stochastiques en Python est dans une phase de construction. Dans la figure **??**, on illustre la résolution de l'équation différentielle stochastique par la méthode d'Euler en utilisant la convention d'Ito. Une dizaine de trajectoires indépendantes sont calculées et tracées ainsi que la valeur moyenne qui est obtenu exactement dans ce processus.

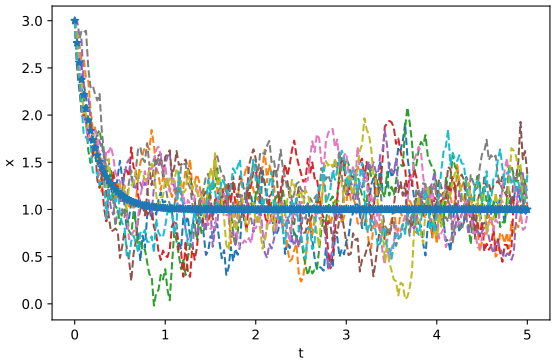


FIGURE 6.3 – 10 trajectoires correspondant au processus d'Ornstein-Uehlenbeck ainsi que la solution exacte pour la valeur moyenne

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: viot
"""
import numpy as np
import sdeint
import matplotlib.pyplot as plt

def f(x, t):
    return -5*(x-1)
def g(x,t):
    return 1
x0 = 3
def moyexact(t,x0):
    return((x0-1)*np.exp(-5*t)+1)
tspan = np.linspace(0.0, 5.0, 201)
for rep in range(10):
    result = sdeint.itoint(f, g, x0, tspan)
    plt.plot(tspan,result,'--')
plt.plot(tspan,moyexact(tspan,x0),'*',lw
    ↪ =4)
plt.xlabel('t')
plt.ylabel('x')
plt.tight_layout()
plt.savefig("ou.pdf")
```

6.4.4 Modèle de Black-Scholes

Le modèle de Black-Scholes est un processus Markovien stochastique satisfaisant l'équation de Langevin suivante

$$\frac{dS(t)}{dt} = S(t)(\mu + \sigma\eta(t)) \tag{6.43}$$

où $S(t)$ est le prix de l'actif sous-jacent à l'instant t. Pour les lecteurs peu familiers au modèles financiers, je vous renvoie au cours de N. Sator, et bien sur aussi à des ouvrages plus spécialisés.

On applique à nouveau la formule d'Ito en utilisant le changement de variable

$$y(t) = \ln(S(t)) \tag{6.44}$$

ce qui donne

$$dy(t) = \frac{dS(t)}{S(t)} - \frac{\sigma^2 dt}{2} \tag{6.45}$$

$$= (\mu - \frac{\sigma^2}{2})dt + \sigma dW(t) \tag{6.46}$$

La solution est donnée par l'intégrale stochastique suivante

$$y(t) = y(t_0) + (\mu - \frac{\sigma^2}{2})t + \sigma W(t) \tag{6.47}$$

Revenant à la variable $S(t)$, on obtient

$$S(t) = S(t_0)\exp((\mu - \frac{\sigma^2}{2})t + \sigma W(t)) \tag{6.48}$$

Compte tenu de l'expression de la solution, on parle souvent de modèle Brownien géométrique. La distribution de la probabilité de S est une loi log-normale, c'est-à-dire une une distribution Gaussienne avec le logarithme de la variable.

Ce type de modèle peut être aisément généralisé pour tenir compte de comportements plus réalistes, en particulier, la volatilité symbolisée par σ n'est généralement pas constante en fonction du temps.

6.4.5 Transformée de Lamperti

Avant d'aborder les méthodes de résolution numérique, nous allons introduire la transformée de Lamperti qui est très utile pour traiter de nombreux processus stochastiques Markoviens. Sous la forme différentielle (autonome) suivante

$$dX(t) = a(X,t)dt + b(X)dW(t) \tag{6.49}$$

On appelle la transformée de Lamperti de X, la fonction Y définie de la manière suivante

$$Y(t) = F(X(t)) = \int_0^{X(t)} \frac{du}{b(u)} \tag{6.50}$$

Avec ce changement de variable, l'équation stochastique devient

$$dY(t) = a_Y(Y,t)dt + dW(t) \tag{6.51}$$

avec

$$a_Y(y,t) = \frac{a(t,F^{-1}(y)}{b(F^{-1}(y))} - \frac{1}{2}\frac{\partial b}{\partial x}(F^{-1}(y)) \tag{6.52}$$

La démonstration de cette propriété est un simple exercice du calcul différentiel stochastique exposé ci-dessus. L'intérêt de cette transformation est qu'elle supprime le bruit multiplicatif de l'équation différentielle stochastique initiale au profit d'une équation de Langevin non linéaire avec un bruit blanc simple. Nous allons voir par la suite que sur le plan numérique ce changement de variable est très utile et peut améliorer la précision numérique.

6.5 Méthodes numériques

6.5.1 Introdution

De manière similaire aux équations différentielles ordinaires où la résolution numérique passe par une discrétisation du temps et un schéma d'approximation concernant l'intervalle de temps élémentaire sur lequel l'intégration est faite, il est nécessaire de procéder de manière similaire avec les équations différentielles stochastiques, à quelques différences près : (i) pour une équation ordinaire, la trajectoire étant déterministe (en tout cas pour les exemples simples à une particule, oscillateur harmonique ou anharmonique), on peut contrôler avec la solution exacte la qualité de l'approximation. Avec un processus de Wiener, nous avons vu que deux trajectoires sont très différentes, cette différence s'accroissant avec le temps. Si par contre, on cherche à résoudre un processus d'Ornstein-Uhlenbeck, on sait que le système évolue vers une distribution stationnaire que la variance des trajectoires est finie... (ii) Les schémas d'approximation des méthodes de résolution des équatioins différentielles sont basés sur le calcul différentiel usuel. Dans le cas des équations différentielles stochastiques, ces schémas reposent sur le calcul

différentiel stochastique de nature assez différente. (iii) Des questions fondamentales sur l'existence et l'unicité d'une solution pour une équation existent de manière similaire aux équations ordinaires.

Sans bien évidemment rentrer dans le détail qui relève de travaux très techniques, il y a deux crières pour prouver l'existence ete l'unicité d'une solution sur un intervalle donné

1. Une condition de Lipschitz qui exprime le fait que pour tout couple (x, y) et tout t appartenant à l'intervalle où l'intégration numérique doit être faite, il existe une constante K telle que

$$|a(x,t) - a(y,t)| + |b(x,t) - b(y,t)| \leq K|x - y| \tag{6.53}$$

2. Une condition de croissance qui exprime le fait que les "variations" de $a(x,t)$ et de $b(x,t)$ ne sont pas trop rapides, c'est-à-dire qu'il existe une constante K telle que dans l'intervalle de temps de l'intégration on ait la condition

$$|a(x,t)|^2 + |b(x,t)|^2 \leq K^2(1 + x^2) \tag{6.54}$$

Sur le plan numérique, on parle d'ordre fort de convergence γ si une approximation discrétisée en temps X_δ d'un processus continu X, δ réprésentant la borme supérieure des incréments de temps utilisés vérifie la propriété

$$\langle |X_\delta(T) - X(T)| \rangle \leq C\delta^\gamma \tag{6.55}$$

les crochets symbolisant la valeur moyenne sur les trajectoires et C étant une constante.

6.5.2 Schéma d'Euler

Soit à nouveau la forme différentielle de l'équation stochastique

$$dX(t) = a(X,t)dt + b(X,t)dW(t)$$

avec une valeur initiale X_0.

L'approximation d'Euler consiste à utiliser une discrétisation régulièrement espacée du temps $t_i = i\Delta t$ avec $\Delta t = T/N$ et i varie de 0 à N.

Le schéma d'Euler est bien évidemment analogue à celui que l'on a pour les équations différentielles ordinaires

$$X_{i+1} = X_i + a(X_i, t_i)(t_{i+1} - t_i) + b(X_i, t_i)(W_{i+1} - W_i) \tag{6.56}$$

On peut montrer que l'ordre fort de convergence de la méthode d'Euler est égal à $\gamma = 1/2$. Facile à mettre en oeuvre, l'algorithme d'Euler souffre de défaults similaires à ceux rencontrés lors des équations différentielles ordinaires, car la précision n'est pas très grande et il convient de diminuer fortement le pas de temps pour éviter les problèmes numériques.

6.5.3 Schéma de Milstein

Il est bien évidemment raisonnable de chercher d'avoir une méthode de résolution allant au dela de cette approximation et dans cet état d'esprit, Milstein a proposé une approximation du second ordre qui utilise à nouveau le calcul stochastique différentiel. En utilisant à nouveau un pas de discrétisation tempotrel constant, on a le schéma itératif suivant

$$\begin{aligned} X_{i+1} = X_i + a(X_i, t_i)(t_{i+1} - t_i) + b(X_i, t_i)(W_{i+1} - W_i) + \\ \frac{1}{2} b(X_i, t_i) \frac{\partial b}{\partial x}(X_i, t_i)[(W_{i+1} - W_i)^2 - (t_{i+1} - t_i)] \end{aligned} \tag{6.57}$$

Cette approximation a un ordre fort de convergence égal à 1. Cette méthode améliore donc les instabilités numériques par rapport à la méthode d'Euler. Toutefois, il y a un lien entre les deux méthodes dans

le cas où on peut réaliser une transformée de Lamperti de l''équation différentielle stochastique de départ. En effet, dans le cas où l'équation stochastique de départ n'a pas de bruit multiplicatif, comme par exemple dans l'exemple du processus d'Ornstein-Uhlenbeck, la méthode d'Euler a un ordre de convergence fort qui devient égal à 1. Or, avec une transformée de Lamperti (si $b(x,t) = b(x)$ est indépendant du temps), on peut transformer l'équation stochastique en une autre équation sans bruit multiplicatif. Ainsi, on peut montrer que le schéma d'Euler de l'équation transformée est identique au schéma de Milstein sur l'équation originale. Dans le cas où la transformée de Lamperti est difficile à obtenir analytiquement, il est utile d'utiliser le schéma de Milstein qui est plus précis.

6.5.4 Runge-Kutta

En continuant le parallèlisme avec les équations différentielles ordinaires, on peut aller aller au dela et des méthodes plus précises ont été developpés. Jusqu'à une période récente (1996), on pensait que les méthodes de type Runge-Kutta ne pouvaient pas donner un ordre de convergence supérieur à 1.5. Cette limite est aujourd'hui dépassée et il est possible de contruire de manière systématique des schémas d'approximation qui augmentent progressivement l'ordre fort de convergence. Pour information, nous donnons ci-dessous une méthode de Runge-Kutta explicite d'ordre 1.5

$$\begin{aligned} X_{i+1} = X_i + a(X_i,t_i)(t_{i+1}-t_i) + b(X_i,t_i)(W_{i+1}-W_i) + \\ \frac{1}{2}(b(X_i',t_i) - b(X_i,t_i))\frac{[(W_{i+1}-W_i)^2 - (t_{i+1}-t_i)]}{\sqrt{(t_{i+1}-t_i)}} \end{aligned} \tag{6.58}$$

avec

$$X_i' = X_i + a(X_i)(t_{i+1}-t_i) + b(X_i)\sqrt{(t_{i+1}-t_i)} \tag{6.59}$$

Cette méthode permet de réaliser des simulations en n'utilisant qu'un nombre aléatoire issu d'une distribution gaussienne. Il existe bien entendu d'autres méthodes d'ordre plus élevé, mais faisant faisant intervenir des nombres aléatoires qui ne sont pas des variables aléatoires gaussiennes, ce qui complique de beaucoup la resolution de l'équation différentielle stochastique. Toutefois, pour des équations de Langevin sans bruit multiplicatif, il a été développé des algorithmes d'ordre élevé (jusqu'à 4) et ne faisant intervenir que des variables aléatoires gaussiennes.

7 Fonctions spéciales et évaluation de fonctions

7.1 Introduction

Les fonctions spéciales sont définies de manière diverse 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 disponible numériquement en Python et en C++ pour un grand nombre d'entre elles.

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 ainsi que les fonctions similaires hyperboliques.

7.2 Fonction Gamma

7.2.1 Définition et propriétés

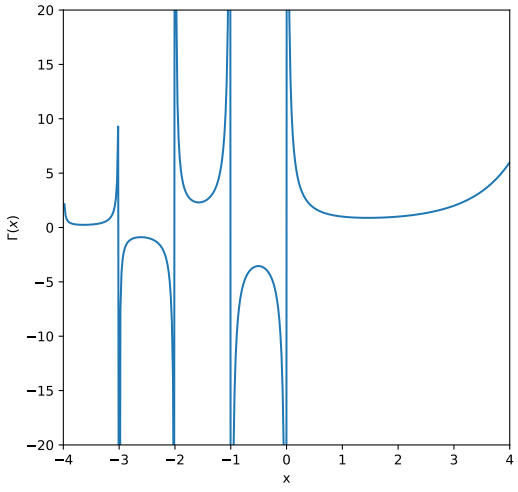


Figure 7.1 – Tracé de la fonciton Γ

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 1 19:36:43 2020

@author: viot
"""
from scipy.special import gamma
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-4,4,400)
plt.figure(figsize=(6,6))
plt.plot(x, gamma(x))
plt.xlim(-4.0,4.0)
plt.ylim(-20.0,20)
plt.xlabel('x')
plt.ylabel(r'$\Gamma(x)$')
plt.savefig('gamma.pdf')
```

La fonction Gamma est généralement définie par l'intégrale suivante

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt \tag{7.1}$$

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

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

$$\Gamma(z) = \lim_{n\to\infty} \frac{n! n^z}{z(z+1)\dots(z+n)} \tag{7.2}$$

En intégrant par parties l'équation (7.1), on peut facilement montrer que

$$\Gamma(z+1) = z\Gamma(z) \tag{7.3}$$

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

$$\Gamma(n+1) = n! \tag{7.4}$$

Avec cette définition, la fonction Γ apparaît comme un prolongement analytique de la fonction factorielle définie sur $\mathbb{N}$. D'après l'équation (7.2), la fonction Γ a un pôle en 0 et pour toutes les valeurs entières négatives (voir Fig. (7.1)).

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

$$\Gamma(1-z) = \frac{\pi}{\Gamma(z)\sin(\pi z)} \tag{7.5}$$

Pour calculer numériquement la fonction Γ 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 Γ. La formule qui s'inspire de la formule de Stirling bien connue pour la fonction factorielle n'est valable que pour $Re(z) > 0$ et est donnée par

$$\begin{aligned}\Gamma(z+1) = &\left(z+\gamma+\frac{1}{2}\right)^{z+\frac{1}{2}} e^{-(z+\gamma+\frac{1}{2})} \\ &\times \sqrt{2\pi}\left[c_0 + \frac{c_1}{z+1} + \frac{c_2}{z+2} + \dots + \frac{c_N}{z+N} + \epsilon\right]\end{aligned} \tag{7.6}$$

où ϵ 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 Γ pour des valeurs de z un peu importantes. Cela résulte de la croissance très rapide de la fonction Γ. On peut montrer que la fonction Γ 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 Γ a une croissance super-exponentielle.

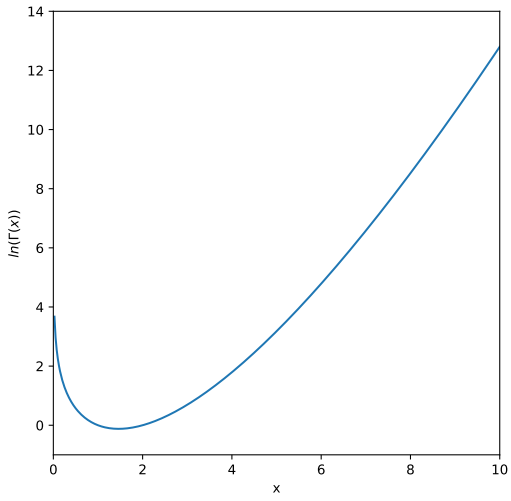


FIGURE 7.2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

@author: viot
"""
from scipy.special import loggamma
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,10,400)
plt.figure(figsize=(6,6))
plt.plot(x, loggamma(x))
plt.xlim(0.0,10.0)
plt.ylim(-1,14)
plt.xlabel('x')
plt.ylabel(r'$ln(\Gamma(x))$')
plt.savefig('lgamma.pdf')
```

Dans de nombreuses formules, la fonction Γ 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. 7.2). 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.

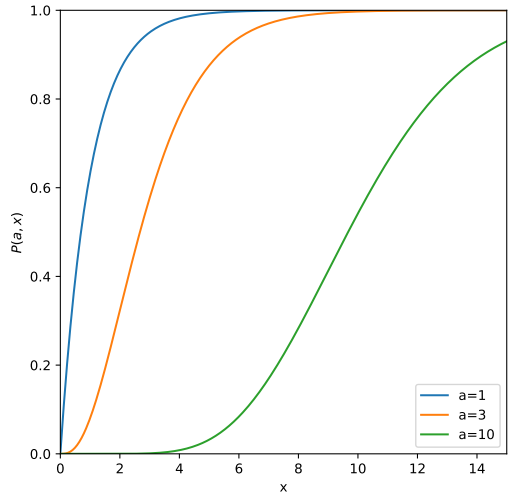


FIGURE 7.3

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 1 19:36:43 2020

@author: viot
"""
import scipy.special as sc
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,15,400)
plt.figure(figsize=(6,6))
a=[1,3,10]
for ia in a:
    plt.plot(x, sc.gammainc(ia,x),label=r'a='+
        ↪ str(ia))
plt.xlim(0.0,15.0)
plt.ylim(0,1)
plt.xlabel('x')
plt.ylabel(r'$P(a,x)$')
plt.legend()
plt.savefig('incg.pdf')
```

On définit la fonction γ incomplète comme

$$\gamma(a,x) = \int_0^x t^{a-1} e^{-t} dt \tag{7.7}$$

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

$$P(a,x) = \frac{\gamma(a,x)}{\Gamma(a)} \tag{7.8}$$

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 7.3).

7.2.2 Fonctions reliées : Ψ, B

A partir de la fonction Γ, on définit des fonctions dérivées. En raison de leur grande fréquence d'utilisation, elles ont "reçu" un nom. Ainsi, la fonction Ψ, appelée aussi fonction Digamma est définie comme la dérivée logarithmique de la fonction Gamma [1] :

$$\Psi(x) = \frac{d\ln(\Gamma(x))}{dx} \tag{7.10}$$

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

$$\Psi(n) = -\gamma + \sum_{i=1}^{n-1} \frac{1}{i} \tag{7.11}$$

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 :

$$B(z,w) = \frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)} \tag{7.12}$$

7.3 Fonctions de Bessel

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

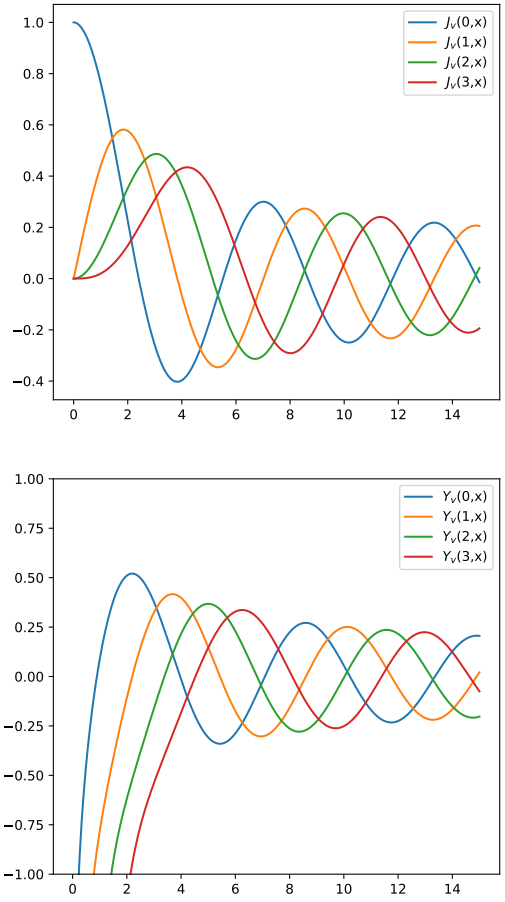
$$x^2y'' + xy' + (x^2 - \nu^2)y = 0 \tag{7.13}$$

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 ν n'est pas un entier, ces fonctions sont reliées par la relation suivante :

$$Y_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)} \tag{7.14}$$

1. On définit aussi les fonctions polygamma comme une généralisation de la fonction Digamma

$$\Psi(n,x) = \frac{d^n\Psi(x)}{dx^n} \tag{7.9}$$




```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: viot
"""
import numpy as np
from scipy.special import jv,yv
import matplotlib.pyplot as plt
x = np.linspace(0,15,400)
fig,ax=plt.subplots(2,1,figsize=(6,12))
orders=np.arange(4)
for i in orders:
    ax[0].plot(x,jv(i,x),label=r"$J_v$("+str(i)
        ↪ +",x)")
    ax[1].plot(x,yv(i,x),label=r"$Y_v$("+str(i)
        ↪ +",x)")
ax[1].set_ylim(-1,1)
ax[1].legend()
ax[0].legend()
plt.savefig("bessel1.pdf")
```

FIGURE 7.4 – Les quatre premières fonctions de Bessel entières de première espèce et de deuxième espèce

La figure 7.4 représente graphiquement les fonctions de Bessel de première et de seconde espèces pour les quatre premières valeurs entières de ν

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

$$J_\nu(x) \simeq \sqrt{\frac{2}{\pi x}}(\cos(x - \nu\pi/2 - \pi/4)) \tag{7.15}$$

$$Y_\nu(x) \simeq \sqrt{\frac{2}{\pi x}}(\sin(x - \nu\pi/2 - \pi/4)) \tag{7.16}$$

Soit l'équation différentielle du second ordre

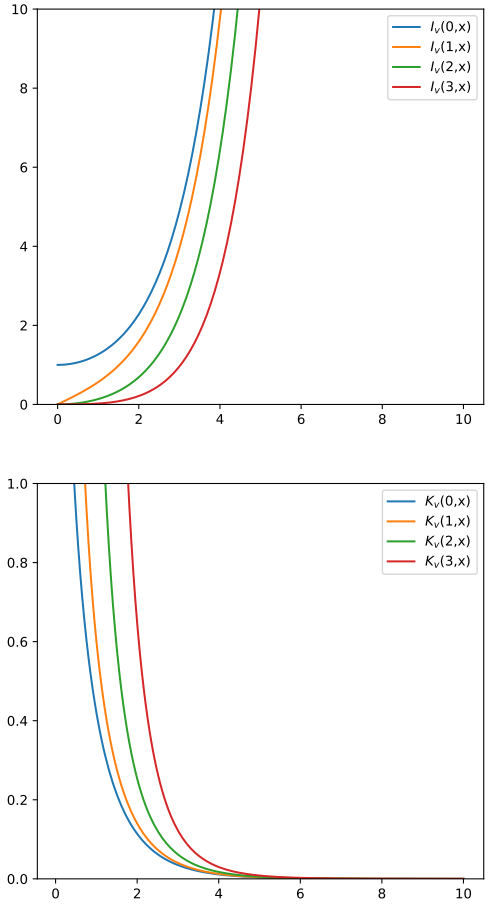
$$x^2y'' + xy' - (x^2 - \nu^2)y = 0 \tag{7.17}$$

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 :

$$K_\nu(x) = \frac{\pi(I_\nu(-x) - I_\nu(x))}{2\sin(\nu\pi)} \tag{7.18}$$

La figure **??** représente graphiquement les fonctions de Bessel modifiées de première et de deuxième espèce pour les quatre premières valeurs entières de ν.




```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Jul 30 17:13:58 2020

@author: viot
"""


import numpy as np
from scipy.special import iv,kv
import matplotlib.pyplot as plt
x = np.linspace(0,10,200)
plt.figure(figsize=(6,12))
plt.subplot(2,1,1)
orders=range(4)
for i in orders:
    plt.plot(x,iv(i,x),label=r"$I_v$("+str(i)+"
        ↪ ,x)")
    plt.ylim(0,10)
plt.legend()
plt.subplot(2,1,2)
for i in orders:
    plt.plot(x,kv(i,x),label=r"$K_v$("+str(i)+"
        ↪ ,x)")
    plt.ylim(0,1)
plt.legend()
#plt.show()
plt.savefig("besselm1.pdf")
```

FIGURE 7.5 – Les quatre premières fonctions de Bessel entières modifiées de première et de deuxième espèce.

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

$$I_\nu(x) \simeq \frac{e^z}{\sqrt{2\pi x}} \tag{7.19}$$

$$K_\nu(x) \simeq \sqrt{\frac{\pi}{2x}} e^{-z} \tag{7.20}$$

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

définies par la relation

$$H_{1,\nu}(x) = J_\nu(x) + iY_\nu(x) \tag{7.21}$$
$$H_{2,\nu}(x) = J_\nu(x) - iY_\nu(x) \tag{7.22}$$

7.4 Fonctions Hypergéométriques

7.4.1 Fonction Hypergéométrique Gaussienne

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

$$x(1-x)y'' + [c-(a+b+1)x]y' - aby = 0 \tag{7.23}$$

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

$$y = F(a,b;c;x) + Bx^{1-c}F(a-c+1, b-c+1; 2-c; x) \tag{7.24}$$

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

$$\begin{aligned} F(a,b;c;x) &\equiv {}_2F_1(a,b,c;x) \\ &= \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!} \end{aligned} \tag{7.25}$$

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, ${}_2F_1(1,1,2;x) = -x^{-1}\ln(1-x)$

7.4.2 Fonctions Hypergéométriques généralisées

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

$${}_pF_q\left[\begin{array}{c} a_1, a_2, \ldots, a_p \\ b1, b_2, \ldots, b_q \end{array}; z\right] = \sum_{k=0}^{\infty} \frac{(a_1)_k (a_2)_k \ldots (a_p)_k}{(b_1)_k (b_2)_k \ldots (b_q)_k} \frac{x^k}{k!} \tag{7.26}$$

où on a utilisé la notation de Pochhammer

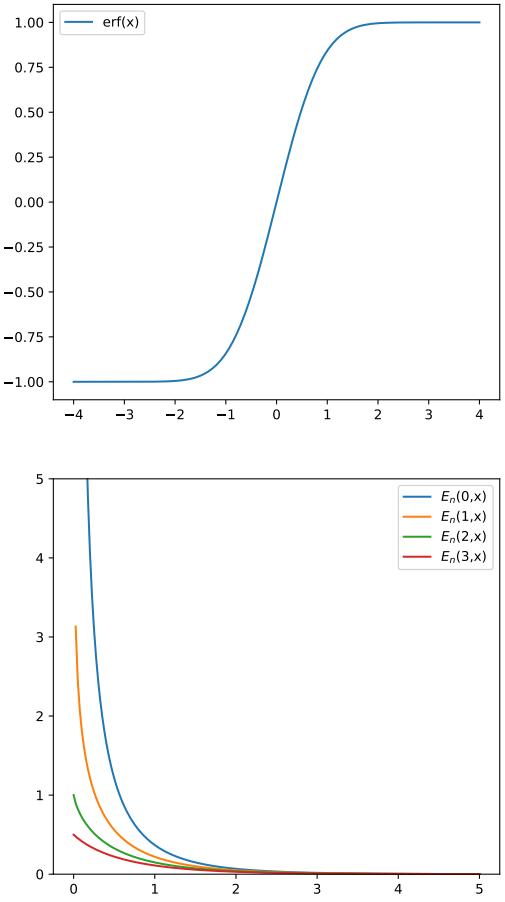
$$(a)_k = \frac{\Gamma(a+k)}{\Gamma(a)} \tag{7.27}$$

7.5 Fonction erreur, exponentielle intégrale

La fonction erreur et la fonction erreur complémentaire sont définies comme

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{7.28}$$
$$\begin{aligned} erfc(x) &= 1 - erf(x) \\ &= \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt \end{aligned} \tag{7.29}$$

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




```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Jul 30 17:13:58 2020

@author: viot
"""
import numpy as np
from scipy.special import expn,erf
import matplotlib.pyplot as plt
x = np.linspace(0,5,200)
x2 = np.linspace(-4,4,200)
fig,ax= plt.subplots(2,1,figsize=(6,12))
orders=np.arange(4)
ax[0].plot(x2,erf(x2),label=r"erf(x)")
ax[0].legend()
for i in orders:
    ax[1].plot(x,expn(i,x),label=r"$E_n$("+str(
        ↪ i)+",x)")
ax[1].set_ylim(0,5)
ax[1].legend()
plt.savefig("expint.pdf")
```

FIGURE 7.6 – La fonction erreur et les quatre premières fonctions exponentielles intégrales (fonctions E_n

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

$$Ei(x) = -\int_{-\infty}^{x} \frac{e^t}{t} dt \tag{7.30}$$

Le développement en série de cette fonction donne

$$Ei(x) = \gamma + \ln(x) + \sum_{n=1}^{\infty} \frac{x^n}{n\, n!} \tag{7.31}$$

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

$$Ei(x) \simeq \frac{e^x}{x}\left(1 + \frac{1}{x} + \ldots\right) \tag{7.32}$$

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

$$E_n(z) = \int_1^\infty \frac{e^{-zt}}{t^n} dt \tag{7.33}$$

La Figure 7.6 représente les quatre premières exponentielles intégrales. Le développement en série de cette fonction donne

$$E_1(x) = -(\gamma + \ln(x)) + \sum_{n=1}^{\infty} (-1)^n \frac{x^n}{n\, n!} \tag{7.34}$$

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

$$E_i(x) = -E_i(1, -x) \tag{7.35}$$

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

$$E_n(x) = x^{n-1} \gamma(1 - n, x) \tag{7.36}$$

8 Transformée de Fourier rapide et algorithmes de tri

8.1 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.

Profitant du fait que l'une des opérations essentielles de la transformée de Fourier est un réarrangement des éléments d'un tableau, nous allons voir les principaux algorithmes de tri que l'on peut utiliser pour réordonner les éléments d'un tableau.

8.2 Propriétés

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

$$\hat{f}(\nu) = \int_{-\infty}^{+\infty} f(t) e^{2\pi i \nu t} dt \tag{8.1}$$

La transformée de Fourier inverse est définie comme

$$f_1(t) = \int_{-\infty}^{+\infty} \hat{f}(\nu) e^{-2\pi i \nu t} d\nu \tag{8.2}$$

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.

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(t) e^{i\omega t} dt \tag{8.3}$$

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

$$f_1(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \hat{f}(\omega)e^{-i\omega t}d\omega \tag{8.6}$$

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

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-i\omega t}dt \tag{8.7}$$

avec bien entendu la transformée de Fourier inverse correspondante

$$f_1(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \hat{f}(\omega)e^{i\omega t}d\omega \tag{8.8}$$

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

- 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 $\hat{f}(\nu)e^{2\pi i\nu t_0}$.

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

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} f(t)e^{i\omega t}dt \tag{8.4}$$

et la transformée de Fourier inverse comme

$$f_1(t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} \hat{f}(\omega)e^{-i\omega t}d\omega \tag{8.5}$$

- 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 δ : soit f une fonction continue définie sur $\mathbb{R}$, La distribution δ est définie à partir de la relation suivante

$$\int_{-\infty}^{+\infty} dx f(x)\delta(x-x_0) = f(x_0) \tag{8.9}$$

Ainsi, on en déduit facilement que

$$\int_{-\infty}^{+\infty} dx \delta(x-x_0)e^{2\pi kix} = e^{2\pi kix_0} \tag{8.10}$$

et donc que

$$\int_{-\infty}^{+\infty} dx \delta(x)e^{2\pi ikx} = 1 \tag{8.11}$$

Par inversion de cette relation, on a la représentation intégrale de la distribution δ

$$\delta(x) = \int_{-\infty}^{+\infty} dk e^{-2\pi ikx} \tag{8.12}$$

De manière équivalente, on montre que

$$\delta(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} dk e^{-ikx} \tag{8.13}$$

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

$$(f*g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau \tag{8.14}$$

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.

$$\widehat{(f*g)}(\nu) = \hat{f}(\nu)\hat{g}(\nu) \tag{8.15}$$

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

$$C_f(t) = \int_{-\infty}^{+\infty} du f(u)f(t+u) \tag{8.16}$$

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

$$\hat{C}_f(\nu) = \hat{f}(\nu)\hat{f}(-\nu) \tag{8.17}$$

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

$$\widehat{C_f}(\nu) = |\hat{f}(\nu)|^2 \tag{8.18}$$

Cette relation est appelée théorème de Wiener-Khinchin.

Le théorème de Parseval donne que

$$\int_{-\infty}^{+\infty} |f(t)|^2 dt = \int_{-\infty}^{+\infty} |\hat{f}(\nu)|^2 d\nu \tag{8.19}$$

8.3 Discrétisation de la transformée de Fourier

8.3.1 Échantillonage

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

$$f_n = f(n\Delta) \qquad \text{avec } n = \ldots, -2, -1, 0, 1, 2, 3, \ldots \tag{8.20}$$

Pour le pas de l'échantillonnage Δ, il existe une fréquence critique, appelée fréquence de Nyquist,

$$\nu_c = \frac{1}{2\Delta} \tag{8.21}$$

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

$$f(t) = \Delta \sum_{n=-\infty}^{+\infty} f_n \frac{\sin(2\pi\nu_c(t - n\Delta))}{\pi(t - n\Delta)} \tag{8.22}$$

Si la fonction $f(t)$ est multipliée par la fonction $e^{2\pi\nu_1 t}$ avec ν_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.

8.3.2 Transformée de Fourier discrète

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

$$f_k = f(t_k) \qquad \text{avec } k = 0, 1, 2, \ldots, N-1 \tag{8.23}$$

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

$$\nu_n = \frac{n}{N\Delta} \qquad \text{avec } n = -N/2, \ldots, 0, \ldots, N/2 \tag{8.24}$$

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

$$\hat{h}_n = \frac{1}{N} \sum_{k=0}^{N-1} h_k e^{2\pi i k n/N} \tag{8.25}$$

La transformée de Fourier discrète inverse est donnée par

$$h_k = \sum_{n=0}^{N-1} \hat{h}_n e^{-2\pi i k n/N} \tag{8.26}$$

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

$$\hat{f}(\nu_n) \simeq \Delta \hat{f}_n \tag{8.27}$$

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

$$\sum_{k=0}^{N-1} |\hat{h}_k|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |h_k|^2 \tag{8.30}$$

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.

2. La transformée de Fourier continue est approchée de la manière suivante

$$\hat{f}(\nu_n) = \int_{-\infty}^{+\infty} f(t) e^{2\pi i \nu_n t} dt \tag{8.28}$$

$$\approx \Delta \sum_{k=0}^{N-1} f_k e^{2\pi i \nu_n t_k} \tag{8.29}$$

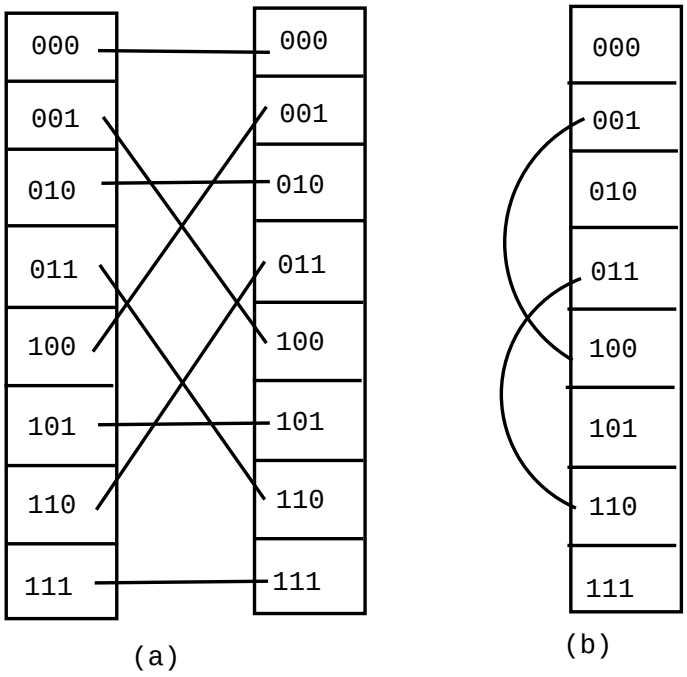


FIGURE 8.1 – Schéma illustrant le principe de la mise en ordre du tableau par renversement de bits. (a) Sur un tableau de 8 éléments étiquettés en binaire, on note les opérations de déplacements. (b) Bilan des déplacements à effectuer : deux opérations d'échange d'éléments.

8.4 Transformée de Fourier rapide

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

$$W = e^{2\pi i/N} \tag{8.31}$$

La composante de Fourier de h s'exprime alors comme

$$\hat{h}_n = \sum_{k=0}^{N-1} W^{nk} h_k \tag{8.32}$$

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

$$F_k = \sum_{j=0}^{N-1} e^{2\pi ijk/N} f_j \tag{8.33}$$

$$= \sum_{j=0}^{N/2-1} e^{2\pi i(2j)k/N} f_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi i(2j+1)k/N} f_{2j+1} \tag{8.34}$$

$$= \sum_{j=0}^{N/2-1} e^{2\pi ijk/(N/2)} f_{2j} + W^k \sum_{j=0}^{N/2-1} e^{2\pi ijk/(N/2)} f_{2j+1} \tag{8.35}$$

$$= F_k^p + W^k F_k^i \tag{8.36}$$

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 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 $\ln_2(N)$

$$F_k^{iipip\ldots ppi} = f_n \tag{8.37}$$

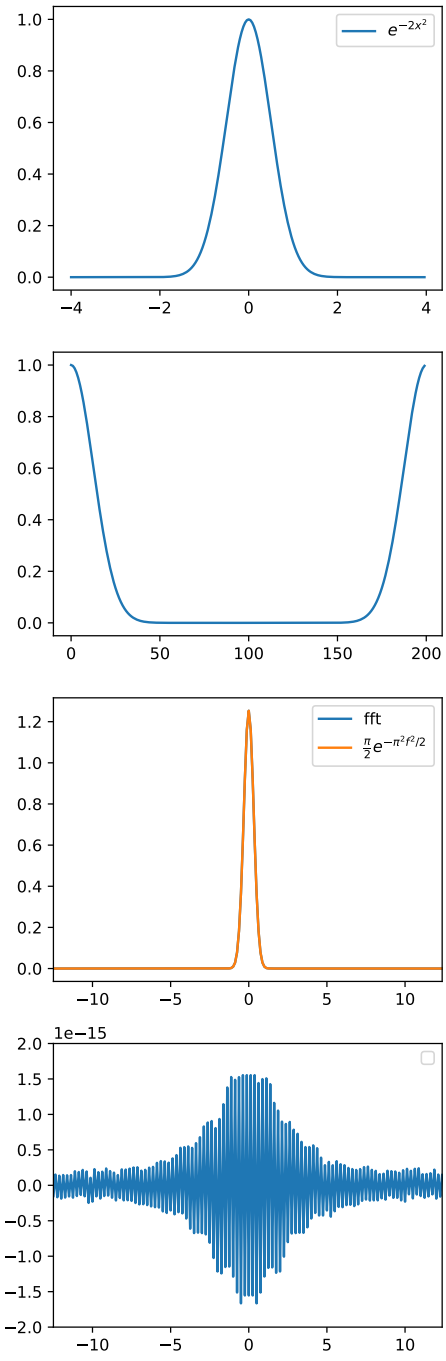
On remarque donc que pour chaque valeur de n on a un alphabet fini d'indice $iipip\ldots ppi$ constitué de i et de p. Cette correspondance est biunivoque. Si on pose que $i = 1$ et $p = 0$, on a l'expression inversée en binaire de chaque valeur de n.

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 8.1 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.

La figure 8.2 illustre le calcul de la transformée de fourier de la Gaussienne en fréquences. Le résultat exact de la transformée de Fourier de la fonction $e^{-\alpha x^2}$ est égale à

$$\int_{-\infty}^{\infty} dx e^{-2i\pi kx} e^{-\alpha x^2} = \sqrt{\frac{\pi}{\alpha}} e^{-\frac{\pi^2}{\alpha}k^2} \tag{8.38}$$




```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: viot
"""
import numpy as np
import matplotlib.pyplot as plt
def gauss(x,alpha=2):
    return(np.exp(-alpha * x**2))

nc = 200
dt = 0.04
tmax = (nc/2-1) * dt
tmin = -nc/2 * dt
t = np.linspace(tmin, tmax, nc)
# Trace la gaussienne
fig,ax= plt.subplots(4,1,figsize=(4,12))
x=gauss(t,2)
ax[0].plot(t,x,label=r"$e^{-2x^2}$")
plt.legend()
# Decalage pour positionner le temgps zero
    ↪ comme premier element
a = np.fft.ifftshift(x)
ax[1].plot(a)
#Transformee de Fourier
A = np.fft.fft(a)
# Decalage inverse pour positionner la
    ↪ frequence zero au centre
X = dt*np.fft.fftshift(A)
# calcul des frequences avec fftfreq
n = t.size
freq = np.fft.fftfreq(n, d=dt)
f = np.fft.fftshift(freq)
# comparaison avec la solution exacte
ax[2].plot(f, np.real(X), label="fft")
ax[2].plot(f, np.sqrt(np.pi/2)*gauss(f,np.pi
    ↪ **2/2), label=r"$\frac{\pi}{2}e^{-\pi^2f
    ↪ ^2/2}$")
ax[2].set_xlim(min(f),max(f))
ax[2].legend()
# comparaison avec la solution exacte

ax[3].plot(f, np.real(X)-np.sqrt(np.pi/2)*gauss
    ↪ (f,np.pi**2/2))
ax[3].set_xlim(min(f),max(f))
ax[3].set_ylim(-2e-15,2e-15)
ax[3].legend()
plt.tight_layout()
plt.savefig("fftg.pdf")
```

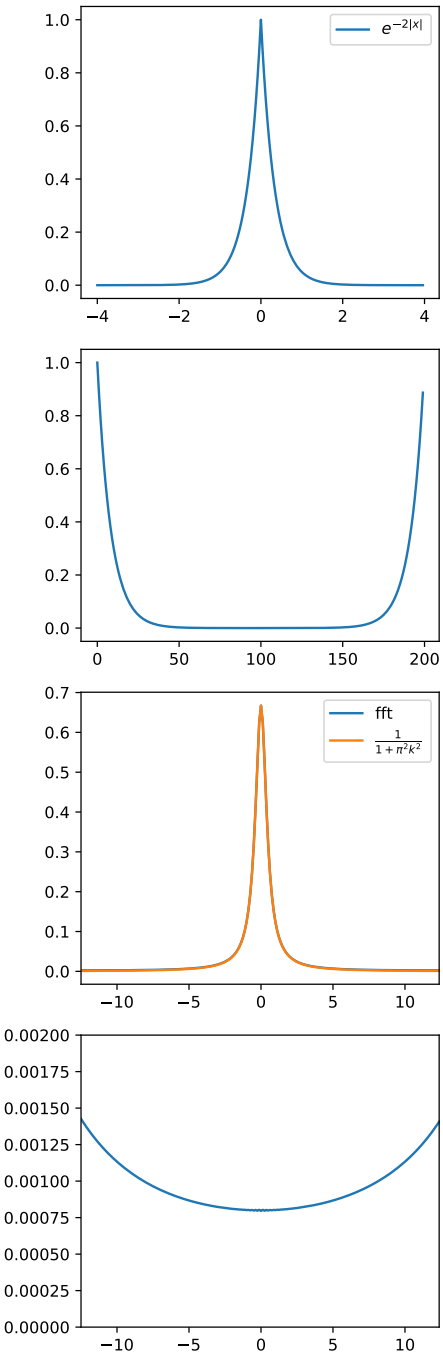
FIGURE 8.2 – Tracée de la gaussienn (haut gauche), tracée de la gaussienn déplacée (haut droit), tracée de la solution exacte et de la transformée de Fourier numérique (bas gauche), différence entre la valeur exacte et la transformée de Fourier numérique

On voit que la précision numérique de la transformée numérique est extremement bonne dans le cas de la gaussienne. Cela est liée en particulièrement à la décroissance extremement rapide de la fonction et que la fonction gaussienne est infiniment dérivable. Ainsi les valeurs numériques de la fonction sont

quasi-nulles aux bornes du support.

Dans le deuxième exemple, on considère la fonction $e^{-\alpha * x}$ dont la transformée de Fourier est aussi connue

$$\int_{-\infty}^{\infty} dx e^{-2i\pi kx} e^{-\alpha|x|} = \frac{2\alpha}{\alpha^2 + 4\pi^2 k^2} \tag{8.39}$$




```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 3 16:43:16 2020

@author: viot
"""
import numpy as np
import matplotlib.pyplot as plt
def expp(x,alpha=2):
    return(np.exp(-alpha * abs(x)))
def loren(x,alpha=2):
   return(2*alpha/(alpha**2+4*np.pi**2*x**2))
nc = 200
dt = 0.04
tmax = (nc/2-1) * dt
tmin = -nc/2 * dt
t = np.linspace(tmin, tmax, nc)
# Trace la gaussienne
fig,ax= plt.subplots(4,1,figsize=(4,12))
x=expp(t,3)
ax[0].plot(t,x,label=r"$e^{-2|x|}$")
ax[0].legend()
# Decalage pour positionner le temgps zero
    ↪ comme premier element
a = np.fft.ifftshift(x)
ax[1].plot(a)
#Transformee de Fourier
A = np.fft.fft(a)
# Decalage inverse pour positionner la
    ↪ frequence zero au centre
X = dt*np.fft.fftshift(A)
# calcul des frequences avec fftfreq
n = t.size
freq = np.fft.fftfreq(n, d=dt)
f = np.fft.fftshift(freq)
# comparaison avec la solution exacte
ax[2].plot(f, np.real(X), label="fft")
ax[2].plot(f, loren(f,3), label=r"$\frac{1}{1+\
    ↪ pi^2k^2}$")
ax[2].set_xlim(min(f),max(f))
ax[2].legend()
# comparaison avec la solution exacte

ax[3].plot(f, np.real(X)-loren(f,3))
ax[3].set_xlim(min(f),max(f))
ax[3].set_ylim(0,2e-3)
plt.tight_layout()
plt.savefig("fftexp.pdf")
```

FIGURE 8.3 – De haut en bas : Tracé de la fonction exponentielle, tracée de la fonction exponentielle déplacée , tracée de la solution exacte et de la transformée de Fourier numérique, différence entre la valeur exacte et la transformée de Fourier numérique

On voit que la précision de la transformée de Fourier a chuté considérablement et cela est liée à l'existence de la discontinuité de la dérivée à l'origine. Dans le dernier exemple, on considère une fonction qui est nulle pour $|x| > a\alpha$ et qui vaut 1 dans l'intervalle $|x| < \alpha$. Le code est similaire aux deux

précédents exemples et je laisse le lecteur le déterminer. La figure illustre les résultats numériques de cette transformée de Fourier numérique.

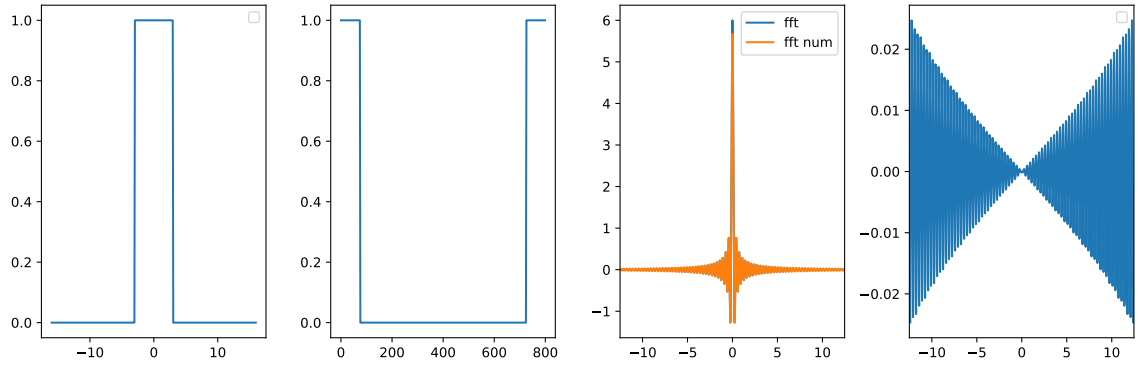


FIGURE 8.4 – De gauche à droite : Tracée de la fonction porte , tracée de la fonction porte déplacée , tracée de la solution exacte et de la transformée de Fourier numérique , différence entre la valeur exacte et la transformée de Fourier numérique

La discontinuité de la fonction aux bornes de l'intervalle a cette fois un impact plus important sur la précision numérique, en particulier vers les hautes fréquences.

8.5 Algorithmes de tri

8.5.1 Introduction

Un tableau est une suite d'élements repérés par un entier appelé indice. Une opération très classique consiste à changer l'ordre initial (qui peut être aléatoire) en respectant un critère défini par l'utilisateur. Une opération de base consiste à trier des nombres selon un ordre croissant ou décroissant. Une opération voisine est la recherche du plus petit ou du grand élément de ce tableau.

Les opérations précédentes sont destructrices dans le sens où l'ordre initial du tableau est perdu une fois l'opération effectuée. On peut en créant un tableau supplémentaire construire une correspondance entre l'ordre obtenu par un tri et l'indice occupé par l'élément du tableau correspondant dans le tableau initial.

Si, au lieu d'utiliser un tableau de nombres, on dispose d'un tableau d'objets, c'est-à-dire un tableau dont l'élément de base est une structure comprenant des champs, qui sont eux-mêmes des nombres et des chaînes de caractères, les opérations de tri peuvent aussi réalisées sur plusieurs champs de manière séquentielle ; cette opération est appelée opération de tri multiple.

8.5.2 Méthode d'insertion

Supposons que l'on veuille trier des éléments d'un tableau de nombre selon un ordre croissant.L'algorithme de la méthode d'insertion est le suivant : on place un à un les éléments du tableau à la bonne place du nouveau tableau rempli des éléments triés. On procède donc de manière itérative jusqu'au dernier élément du tableau à trier. On peut choisir de construire un nouveau tableau ou de réutiliser le tableau existant. Pratiquement, on procède de la manière suivante. On choisit le premier élément : il est automatiquement bien placé dans un tableau à un élément. On tire le second élément : on augmente d'un élément la taille du tableau à trier ; soit le nouvel élément est plus grand que le précédent et on place cet élément à la deuxième place, sinon, on déplace l'élément initialement placé à la seconde place et l'on place le nouvel élément à la première place. On choisit alors un troisième élément ; on teste si celui-ci est plus grand que le deuxième. Si le test est positif, cet élément est placé en troisième place, sinon on déplace l'élément placé en numéro deux en numéro trois et on teste si le nouvel élément est plus grand

que l'élément un. Si le test est positif, on place le nouvel élément en numéro deux, sinon on déplace le premier élément en deux et on place le nouvel élément en un. On continue cet algorithme jusqu'au dernier élément du tableau à trier. Il est facile de voir que le nombre d'opérations en moyenne évolue comme N^2, où N est le nombre d'éléments à trier. Comme nous allons le voir ci-dessous les algorithmes de type "tri rapide" sont beaucoup plus efficaces car ceux-ci dépendent du nombre d'éléments comme $N\log_2(N)$. Néanmoins, un algorithme de type insertion reste suffisamment efficace si $N < 20$.

8.5.3 Tri à bulles

Le tri à bulles es un algorithme en N^2 qui opère de la manière suivante : On compare les deux premiers éléments du tableau ; si l'ordre est correct, on ne fait rien sinon on échange les deux éléments : on passe alors au couple suivant constitué des éléments 2 et 3, et on effectue un test à nouveau. Après $N-1$ opérations de ce type, on a placé le plus grand élément du tableau en dernière position ; on peut alors recommencer sur le tableau restant des $N-1$ éléments ; cet algorithme est bien évidemment un algorithme en N^2 et ne peut servir que pour des valeurs de N très modestes.

8.5.4 Tri rapide

Connu par son nom anglais "quicksort", le tri dit rapide procède de la manière suivante : on prend le premier élément du tableau à trier, noté a_1. On construit avec les autres éléments deux sous tableaux : le premier contient les éléments plus petits que a_1 et le second les éléments plus grands. On peut placer a_1 à la bonne place dans le tableau final et on applique l'algorithme du tri rapide sur chacun des deux sous-tableaux. On peut montrer qu'en moyenne l'algorithme est en $N\log_2(N)$, mais il est possible d'obtenir des situations où l'algorithme est en N^2.

Des variantes de cet algorithme puissant existent pour par exemple éviter la récursivité qui pénalise la rapidité d'exécution du programme. A nouveau, il est inutile, hormis pour des raisons pédagogiques, de chercher à réécrire un programme pour ces algorithmes, que cela soit en programmation où de nombreuses bibliothèques sont disponibles, ou dans les logiciels scientifiques où les instructions de tri sont disponibles, mais il convient de tester la méthode la plus adaptée car le "désordre initial" du tableau à trier influe de manière significative sur la performance des algorithmes.

Sur la figure 8.5, on mesure le temps de calcul pour le tri d'un tableau constitué de valeurs aléatoires par la méthode quicksort. Cette méthode peut avoir des performances de l'ordre de $n\ln(n)$) en moyenne, mais les performances peuvent décroitre pour être en n^2. Pour ce type de tableau, on trouve des performances se situant croissant plutôt linéairement.

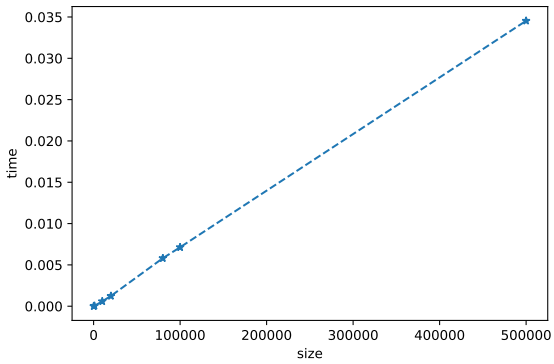


FIGURE 8.5 – Temps de calcul en fonction de la taille du tableau par la méthode de tri rapide

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 4 16:30:02 2020

@author: viot
"""
import numpy as np
import time
import matplotlib.pyplot as plt
perf=[]
sizes
    ↪ =[100,1000,10000,20000,80000,100000,500000]
    ↪
for i in sizes:
    tab=np.random.uniform(size=i)
    start_time = time.time()
    tab.sort(kind='quicksort')
    perf.append(time.time() - start_time )

plt.plot(sizes,perf,'*--')
plt.xlabel('size')
plt.ylabel('time')
plt.tight_layout()
plt.savefig('sort.pdf')
```

9

Algèbre linéaire

9.1 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

$$A.x = b \tag{9.1}$$

où A est une matrice $M \times N$

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \ddots & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{pmatrix} \tag{9.2}$$

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

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_N \end{pmatrix} \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_M \end{pmatrix} \tag{9.3}$$

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

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.

- 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.

9.2 Élimination de Gauss-Jordan

9.2.1 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 4×4, on a

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{132} & a_{133} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} . \left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \sqcup \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{pmatrix} \sqcup \begin{pmatrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \\ y_{31} & y_{132} & y_{133} & y_{34} \\ y_{41} & y_{42} & y_{43} & y_{44} \end{pmatrix} \right) =$$
$$= \left(\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} \sqcup \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} \sqcup \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \right) \tag{9.4}$$

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.

9.2.2 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

$$\begin{pmatrix} 1 & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{pmatrix} \tag{9.5}$$

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

$$\begin{pmatrix} 1 & 0 & a''_{13} & a''_{14} \\ 0 & 1 & a''_{23} & a''_{24} \\ 0 & 0 & a'_{33} & a''_{34} \\ 0 & 0 & a''_{43} & a''_{44} \end{pmatrix} \tag{9.6}$$

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.

9.2.3 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.

9.3 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

$$\begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{133} & a'_{34} \\ 0 & 0 & 0 & a'_{44} \end{pmatrix} . \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{pmatrix} \tag{9.7}$$

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

$$x_4 = \frac{b'_4}{a'_{44}} \tag{9.8}$$

$$x_3 = \frac{1}{a'_{33}}(b'_3 - x_4 a'_{34}) \tag{9.9}$$

et de manière générale

$$x_i = \frac{1}{a'_{ii}}(b_i - \sum_{j=i+1}^{N} a'_{ij}x_j) \tag{9.10}$$

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.

9.4 Décomposition LU

Les méthodes précédentes nécessitent de connaître à l'avance le membre de gauche de l'équation (9.1). Les méthodes qui suivent consiste à réécrire la matrice A afin que la résolution du système d'équations soit exécutée plus facilement.

9.4.1 Principe

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

$$A = L.U \tag{9.11}$$

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

$$L = \begin{pmatrix} \alpha_{11} & 0 & 0 & \dots & 0 \\ \alpha_{21} & \alpha_{22} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \dots & \vdots \\ \vdots & \vdots & \vdots & \alpha_{(N-1)(N-1)} & 0 \\ \alpha_{N1} & \alpha_{N2} & \dots & \dots & \alpha_{NN} \end{pmatrix} \tag{9.12}$$

et

$$U = \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \dots & \beta_{1N} \\ 0 & \beta_{22} & \beta_{23} & \dots & \beta_{2N} \\ \vdots & \vdots & \ddots & \dots & \vdots \\ \vdots & \vdots & \vdots & \beta_{(N-1)(N-1)} & \beta_{(N-1)N} \\ 0 & 0 & \dots & 0 & \beta_{NN} \end{pmatrix} \tag{9.13}$$

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

$$a_{ij} = \sum_{l=1}^{i} \alpha_{il}\beta_{lj} \qquad i \leq j \tag{9.14}$$

$$a_{ij} = \sum_{l=1}^{j} \alpha_{il}\beta_{lj} \qquad i > j \tag{9.15}$$

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

$$\alpha_{ii} = 1 \tag{9.16}$$

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

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

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj} \qquad i \leq j \tag{9.17}$$

$$\alpha_{ij} = \frac{1}{\beta_{jj}}(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj}) \qquad i \geq j+1 \tag{9.18}$$

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]

9.4.2 Résolution d'un système linéaire

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

$$A.x = L.U.x \tag{9.20}$$

$$= L.(Ux) = b \tag{9.21}$$

Soit

$$L.y = b \tag{9.22}$$

$$U.x = y \tag{9.23}$$

Chaque système peut être résolu par une procédure de substitution.

$$y_1 = \frac{b_1}{\alpha_{11}} \tag{9.24}$$

$$y_i = \frac{1}{\alpha_{ii}}(b_i - \sum_{j=1}^{i-1} \alpha_{ij}y_j) \tag{9.25}$$

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

$$x_N = \frac{y_N}{\beta_{NN}} \tag{9.26}$$

$$x_i = \frac{1}{\beta_{ii}}(y_i - \sum_{j=i+1}^{N} \beta_{ij}x_j) \tag{9.27}$$

car la matrice est triangulaire supérieure.

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

$$\begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \cdots & \beta_{1N} \\ \alpha_{21} & \beta_{22} & \beta_{23} & \cdots\cdots\cdots & \beta_{2N} \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \beta_{(N-1)(N-1)} & 0 \\ \alpha_{N1} & \alpha_{N2} & \cdots & \cdots & \beta_{NN} \end{pmatrix} \tag{9.19}$$

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

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

$$det(A) = \prod_{j=1}^{N} \beta_{jj} \tag{9.28}$$

9.5 Matrices creuses

9.5.1 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.

9.5.2 Matrices tridiagonales

Soit le système suivant

$$\begin{pmatrix} b_1 & c_1 & 0 & \dots & 0 \\ a_2 & b_2 & c_2 & \dots & 0 \\ 0 & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \vdots & b_{N-1} & c_{N-1} \\ 0 & 0 & \dots & \dots & b_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_N \end{pmatrix} \tag{9.29}$$

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.

9.5.3 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 ?

$$B = A + u \otimes v \tag{9.30}$$

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

$$B^{-1} = (1 + A^{-1}.u \otimes v)^{-1}.A^{-1} \tag{9.31}$$
$$= (1 - A^{-1}.u \otimes v + A^{-1}.u \otimes v.A^{-1}.u \otimes v - \ldots).A^{-1} \tag{9.32}$$
$$= A^{-1} - A^{-1}.u \otimes vA^{-1}(1 - \lambda + \lambda^2 + \ldots) \tag{9.33}$$
$$= A^{-1} - \frac{(A^{-1}.u) \otimes (v.A^{-1})}{1 + \lambda} \tag{9.34}$$

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

$$B^{-1} = A^{-1} - \frac{z \otimes w}{1 + \lambda} \tag{9.35}$$

9.6 Décomposition de Choleski

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

$$x.A.x > 0 \tag{9.36}$$

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

$$A = L.L^T \tag{9.37}$$

Les éléments de matrice sont déterminés par les relations

$$L_{ii} = (a_{ii} - \sum_{k=1}^{i-1} L_{ik}^2)^{1/2} \tag{9.38}$$

$$L_{ji} = \frac{1}{L_{ii}}(a_{ij} - \sum_{k=1}^{i-1} L_{ik}L_{jk}) \tag{9.39}$$

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 9.4.2.

9.7 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.

10 Analyse spectrale

10.1 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 λ, un vecteur qui satisfait la relation

$$A.x = \lambda x \tag{10.1}$$

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

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

$$det(A - \lambda.1) = 0 \tag{10.2}$$

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

$$A = A^T \tag{10.3}$$

$$a_{ij} = a_{ji} \quad \forall i, j \tag{10.4}$$

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

$$A = A^\dagger \tag{10.5}$$

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

$$A.A^T = A^T.A = 1 \tag{10.6}$$

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

$$A.A^\dagger = A^\dagger.A = 1 \tag{10.7}$$

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.

10.2 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

$$A.x_i^R = \lambda_i x_i^R \tag{10.8}$$

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

$$x_i^L.A = \lambda_i x_i^L \tag{10.9}$$

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

$$A.X_R = X_R.diag(\lambda_1,\ldots,\lambda_n) \tag{10.10}$$

de même on a

$$X_L.A = diag(\lambda_1,\ldots,\lambda_n).X_L \tag{10.11}$$

En multipliant l'équation (10.10) par X_L et l'équation (10.11) par X_R, on obtient

$$X_L.X_R.diag(\lambda_1,\ldots,\lambda_n) = diag(\lambda_1,\ldots,\lambda_n)X_L.X_R \tag{10.12}$$

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 (10.10) par X_R^{-1} que

$$X_R^{-1}.A.X_R = diag(\lambda_1,\ldots,\lambda_n) \tag{10.13}$$

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.

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

Nous avons alors construit une matrice de transformation similaire de A

Rappelons la propriété suivante : soit B une matrice telle que

$$B = P^{-1}.A.P \tag{10.14}$$

où P est une matrice inversible. On a

$$\begin{aligned} det(B - \lambda 1) &= det(P^{-1}.A.P - \lambda.1) && (10.15)\\ &= det(P^{-1}.(A - \lambda.1)P) && (10.16)\\ &= det(A - \lambda.1) && (10.17) \end{aligned}$$

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.

10.3 Méthodes directes

10.3.1 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

$$P_{pq} = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & c & \dots & \dots & s & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & \dots & \dots & 1 & \dots & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & -s & \dots & \dots & c & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 1 & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & \dots & 1 \end{pmatrix} \tag{10.18}$$

avec la condition que

$$c^2 + s^2 = 1. \tag{10.19}$$

Soit la matrice A' telle que

$$A' = P_{pq}^T.A.P_{pq} \tag{10.20}$$

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

$$a'_{rp} = ca_{rp} - sa_{rq} \tag{10.21}$$
$$a'_{rq} = ca_{rq} + sa_{rp} \tag{10.22}$$
$$a'_{pp} = c^2 a_{pp} + s^2 a_{qq} - 2csa_{pq} \tag{10.23}$$
$$a'_{qq} = s^2 a_{pp} + c^2 a_{qq} + 2csa_{pq} \tag{10.24}$$
$$a'_{pq} = (c^2 - s^2)a_{pq} + cs(a_{pp} - a_{qq}) \tag{10.25}$$

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

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

$$\theta = \frac{c^2 - s^2}{2sc} \tag{10.26}$$
$$= \cot(2\phi) \tag{10.27}$$
$$= \frac{a_{qq} - a_{pp}}{a_{pq}} \tag{10.28}$$

Si on appelle $t = s/c$, on obtient en utilisant l'équation (10.26)

$$t^2 + 2t\theta - 1 = 0 \tag{10.29}$$

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 [4]

$$t = \frac{sgn(\theta)}{|\theta| + \sqrt{\theta^2 + 1}} \tag{10.30}$$

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

$$c = \frac{1}{\sqrt{1 + t^2}} \tag{10.31}$$

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

$$a'_{pp} = a_{pp} - ta_{pq} \tag{10.32}$$
$$a'_{qq} = a_{qq} + ta_{pq} \tag{10.33}$$
$$a'_{rp} = a_{rp} - s(a_{rq} + \tau a_{rp}) \tag{10.34}$$
$$a'_{rq} = a_{rq} + s(a_{rp} - \tau a_{rq}) \tag{10.35}$$

avec τ défini par

$$\tau = \frac{s}{1 + c} \tag{10.36}$$

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

4. Noter que la fonction *sgn* que l'on définit dans l'équation (10.29) vaut soit +1 soit −1. Cette fonction est généralement intrinsique à beaucoup de langages, mais produit aussi la valeur 0 dans certains langages. Il faut donc s'assurer que cette fonction est correctement implémentée dans le langage informatique que l'on utilise.

En calculant la somme S

$$S = \sum_{r \neq s} |a_{rs}|^2 \tag{10.37}$$

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

$$S' = S - 2|a_{pq}|^2 \tag{10.38}$$

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

$$D = V^T.A.V \tag{10.39}$$

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

10.3.2 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

$$P = 1 - 2w.w^T \tag{10.40}$$

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

Vérifions que la matrice P est une matrice orthogonale

$$P^2 = (1 - 2w.w^T).(1 - 2w.w^T) \tag{10.41}$$
$$= 1 - 4w.w^T + 4w.(w^T.w).w^T \tag{10.42}$$
$$= 1 \tag{10.43}$$

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

$$P = 1 - \frac{u.u^T}{H} \tag{10.44}$$

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

$$u = x \mp |x|e_1 \tag{10.45}$$

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

$$H = 2(|x^2| \pm |x|x_1) \tag{10.46}$$

En appliquant P à x

$$P.x = x - \frac{u}{H}.(x \mp |x|e_1)^T.x \tag{10.47}$$

$$= x - \frac{2u.(|x|^2 \mp |x|x_1)}{2|x|^2 \mp 2|x|x_1} \tag{10.48}$$

$$= x - u \tag{10.49}$$

$$= \pm |x|e_1 \tag{10.50}$$

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

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & & & & & \\ 0 & & & & & \\ 0 & & & & & \\ \vdots & & & P_1^{(n-1)} & & \\ 0 & & & & & \\ 0 & & & & & \end{pmatrix} \tag{10.51}$$

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

$$A' = P.A.P \tag{10.52}$$

$$= \begin{pmatrix} a_{11} & k & 0 & \dots & \dots & 0 \\ k & & & & & \\ 0 & & & & & \\ 0 & & & & & \\ \vdots & & & & & \\ 0 & & & & & \\ 0 & & & & & \end{pmatrix} \tag{10.53}$$

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

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & 0 \\ 0 & 0 & & & & \\ 0 & 0 & & & & \\ \vdots & \vdots & & P_2^{(n-2)} & & \\ 0 & 0 & & & & \\ 0 & 0 & & & & \end{pmatrix} \tag{10.54}$$

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

$$p = \frac{A.u}{H} \tag{10.55}$$

En conséquence, la première multiplication matricielle peut être écrite comme

$$A.P = A.(1 - \frac{u.u^T}{H}) \tag{10.56}$$
$$= A - p.u^T \tag{10.57}$$

de même pour la seconde

$$A' = P.A.P = A - p.u^T - u.p^T + 2Ku.u^T \tag{10.58}$$

avec

$$K = \frac{u^T.p}{2H} \tag{10.59}$$

En posant

$$q = p - Ku \tag{10.60}$$

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

$$A' = A - q.u^T - u.q^T \tag{10.61}$$

10.3.3 Algorithme QL

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

$$A = Q.R \tag{10.62}$$

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

$$A = Q.L \tag{10.63}$$

. 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

$$A' = L.Q \tag{10.64}$$

Puisque Q est orthogonal, on en déduit de l'équation (10.63) que

$$L = Q^{-1}.A \tag{10.65}$$

ce qui donne

$$A' = Q^T.A.Q \tag{10.66}$$

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

$$A_s = Q_s.L_s \tag{10.67}$$
$$A_{s+1} = L_s.Q_s \tag{10.68}$$

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.

10.3.4 Factorisation de Schur

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

- Si la matrice A est complexe
$$A = ZTZ^{\dagger} \tag{10.69}$$
où Z est unitaire et T est une matrice triangulaire supérieure.
- Si la matrice A est réelle
$$A = ZTZ^{T} \tag{10.70}$$
où Z est orthogonale et T est une matrice quasi-triangulaire supérieure, ce qui signifie que la diagonale est constituée soit de blocs 1×1 soit de blocs 2×2.

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
$$A = QHQ^{\dagger} \tag{10.71}$$
où Q est unitaire et H est une matrice de Hessenberg.
- Si la matrice A est réelle
$$A = QTQ^{T} \tag{10.72}$$
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.

10.4 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.

10.4.1 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

$$x_1 = A.x_0 \tag{10.73}$$

Si on désigne α comme l'angle entre ces deux vecteurs, on a

$$\cos(\alpha) = \frac{x_1.x_0}{|x_1|.|x_0|} \tag{10.74}$$

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 ϵ 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 (λ_1/λ_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.

10.4.2 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

$$\beta_2 u_1 = A.u_0 - \alpha_1 u_0 \tag{10.75}$$

où β_2 et α_1 sont des constantes déterminées de la manière suivante : α_1 est choisi de manière à ce que le vecteur u_1 soit orthogonal au vecteur u_0

$$\alpha_1 = u_0^T.A.u_0 \tag{10.76}$$

et β_2 est déterminé de manière à ce que le vecteur u_1 soit normalisé.

$$\beta_2 = \sqrt{u_0^T.A^2.u_0 - \alpha_1^2} \tag{10.77}$$

Pour le vecteur suivant u_2, on utilise la relation :

$$\beta_3 u_2 = A.u_1 - \alpha_2 u_1 - \beta_2 u_0 \tag{10.78}$$

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

$$\alpha_2 = u_1^T.A.u_1 \tag{10.79}$$

et la normalisation de u_2 est imposée en choisissant β_3 comme

$$\beta_3 = \sqrt{u_1^T.A^2.u_1 - \alpha_2^2 - \beta_2} \tag{10.80}$$

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

$$\beta_{i+1} u_i = A.u_{i-1} - \alpha_i u_{i-1} - \beta_i u_{i-2} \tag{10.81}$$

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 α_i et β_{i+1} sont déterminées par les relations suivantes

$$\alpha_i = u_{i-1}^T.A.u_{i+1} \tag{10.82}$$

$$\beta_{i+1} = \sqrt{u_{i-1}^T.A^2.u_{i-1} - \alpha_i^2 - \beta_i} \tag{10.83}$$

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

$$A' = \begin{pmatrix} \alpha_1 & \beta_2 & 0 & \dots & \dots & \dots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & 0 & \dots & \dots & 0 \\ 0 & \beta_3 & \alpha_3 & \beta_4 & 0 & \dots & 0 \\ 0 & \dots & & \dots & & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & \dots & 0 & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ 0 & \dots & \dots & \dots & 0 & \beta_n & \alpha_n \end{pmatrix} \tag{10.84}$$

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 Schrödinger), le procédé itératif conduit progressivement à des valeurs de β_i de plus en en plus petites. On peut considérer que la base de Krylov est complète quand β_n est devenu inférieure en valeur absolue à une valeur ϵ choisie à l'avance (généralement 10^{-12}).

Table des matières