Méthodes numeriques pour la physique

Licences de Physique

Faculté des Sciences de Luminy

Hubert Klein (klein@gpec.univ-mrs.fr)
GPEC UMR CNRS 6631
case 901 faculté des Sciences de Luminy 13288 Marseille cedex 9

Table des matières

Chapitre 1

Petite introduction historique

Ce document n'a pas pour but d'être un cours d'informatique, encore moins un cours d'analyse numérique. L'idée en est plutôt d'utiliser la puissance de calcul d'un ordinateur pour traiter des problèmes physique très variés. Pourquoi utiliser un ordinateur ? Pour gagner du temps par exemple lors d'un calcul fastidieux que l'on pourrait réaliser à la main, mais aussi dans des cas où le problème ne peut pas être traité avec un crayon et une feuille de papier (certaines équations n'ont par exemple pas de solutions analytiques).

Les calculs numériques sont apparus bien avant l'apparition des ordinateurs. Au XVII siècle par exemple, John Napier (1550-1616) introduisit les logarithmes (*Mirifici logarithmorum canoni descriptio*, Edimbourg,1614). Les multiplications pouvait être transformée en addition, un calcul de racine carrée en division par deux. C'était déjà un gain de temps appréciable . . .De nombreux contemporains de Napier réalisèrent des prouesses en calculs numériques (calculs des orbites planétaires par exemple), et tout ceci à la main. Outre le temps requis par de tels calculs, les erreurs étaient monnaie courante.

Aussi, l'idée d'automatiser ces calculs fastidieux apparut rapidement. Le baron Gaspard de Prony était chargé pendant le premier empire de mettre en place des tables pour le calcul des impôts fonciers (dont les règles étaient déjà complexes à l'époque). Il eut l'idée de fractionner ce travail énorme en plusieurs partie. Des mathématiciens décomposèrent tous les calculs nécessaires en opérations élémentaires, des gestionnaires organisèrent le travail et collectèrent les résultats. Enfin, les calculs élémentaires furent assurés par des opérateurs pour lesquels la qualification requise était de savoir faire des additions. Cette approche est exactement celle d'un programmeur devant résoudre un problème à l'aide d'un ordinateur : sa tâche est d'écrire un *algorithme*, c'est à dire de décomposer un problème en une série de tâches élémentaires compréhensibles par un ordinateur (additions, comparaisons . . .), et de définir l'ordre d'exécution de ces tâches. Dans ce cas, les opérateurs humains sont remplacés par un calculateur beaucoup plus rapide.

La mécanisation du calcul suivit, notamment grâce à un anglais, Charles Babbage au début du XIX siècle. Il eut l'idée d'associer cette idée de décomposition d'un problème en tâches simples à une machine à calculer mécanique ressemblant à la machine de Pascal et à un système de lecture de cartes perforées utilisées sur les métiers à tisser Jacquard. Sa machine à différences était l'analogue mécanique de nos calculateurs électroniques. Cependant, son idée n'aboutit pas, car il se heurta a de très importantes difficultés techniques. Mais l'idée était lancée . . .

Cette idée fut concrétisée en 1890 par Herman Hollerith qui répondit à un appel d'offre du gouvernement américain pour traiter les données du recensement national. Le traitement des données du recensement précédent avait nécessité sept ans de travail. Hollerith proposa une machine qui lisait les données sur des cartes perforées, et le résultat fut obtenu en six semaines. Un gain de temps appréciable. Hollerith fonda alors une société, la *Tabulating Machine Company* qui devint en 1924 la *International Business Machines*, plus connue sous le sigle IBM ...

Au cours de la deuxième guerre mondiale, les efforts de recherche dans ce domaine se sont intensifiés : les armées avaient besoin de gros volumes de calcul (établissement de tables de portée de tir pour l'artillerie, décryptage des communications ennemies ...).

Les premiers calculateurs électroniques ont vu le jour très peu de temps après la guerre (l'ENIAC en 1946 aux Etats-Unis). L'invention du transistor en 1951 par John Bardeen, William Shockley et Walter Brattain a ensuite ouvert la voie aux calculateurs électroniques modernes. Leurs performances n'ont depuis jamais cessé de croître.

Chapitre 2

Représentation des nombres dans un ordinateur

On parle souvent de *précision* du résultat au cours de calculs numériques. Cette précision n'est jamais une précision absolue. En effet, outre le fait qu'un algorithme puisse introduire une imprécision dans les résultats, les nombres eux-mêmes sont représentés avec une certaine précision dans la mémoire d'un ordinateur.

2.1 Représentation des nombres entiers

Dans la mémoire d'un ordinateur, les nombres ont une représentation binaire, c'est à dire une suite de 0 et de 1. Aisi pour les nombres entiers on obtient la correspondance suivante entre représentation décimale (première ligne) et binaire (deuxième ligne)

0	1	2	3	4	5	6	7	8	9	10	...
0	1	10	11	100	101	110	111	1000	1001	1010	...

Une case contenant em mémoire un 0 ou un 1, est appelée un *bit*. Dans la majorité des ordinateurs, ces bits sont regroupés par 8, les *octets* ou *bytes*. Ces octets constituent alors un emplacement correspondant à une adresse dans la mémoire. Le tableau précédent s'écrirait en fait

0	1	2	3	4	5	...
00000000	00000001	00000010	00000011	00000100	00000101	...

Avec un octet on peut représenter les nombres entiers $0 \dots 2^8 - 1$, c'st à dire de 0 à 255.

Afin d'avoir une dynamique de représentation plus importante, on groupe des octets en *mots*, la taille de ces mots dépendant du nombre de bits qu'un ordinateur sait manipuler en une opération. Sur les ordinateurs personnels actuels, la taille de ces mots est de 32 bits soit 4 octets. Dans un mot de quatre octets, on peut coder les entiers allant de 0 à $2^{32} - 1$ soit de 0 à 4 294 967 296.

Pour pouvoir représenter les nombres négatifs, on change de convention de codage : un nombre négatif est le complément à deux de sa valeur positive. Par exemple 00000001 = 1 et 11111110=-1 (au lieu de 254). Aisi de 00000000=0 à 01111111=127 on code des nombres positifs, et de 11111110=-1 à 11111111=-128 des nombres négatifs. Sur 32 bits, on peut coder

des nombres entiers signés allant de -2^{31} à $2^{31}-1$

2.2 Représentation des nombres réels

Parler de nombre réels est abusif, on ne peut en effet traiter que des nombres rationels, par exemple $\sqrt{2} = 1.4142213562\ldots$. La restriction est encore plus grande puisque l'on ne peut représenter qu'un nombre fini de chiffres : $1/3$ devient par exemple 0.33333333. Si l'on supprime la virgule, ce nombre peut être représenté comme un entier, il suffit ensuite d'indiquer la position de la virgule par une puissance de 10. $\sqrt{2} = 1.4142213562$ peut s'écrire 14142213562.10^{-9} ou encore $0,14142213562.10^{1}$.

L'avantage de la deuxième écriture est que l'expression des chiffres composant le nombre est comprise entre 0 et 1, chaque chiffre correspondant à une puissance négative de 10. On pourra donc représenter un nombre avec 3 papramètres : le signe, la position de la virgule (ou l'*exposant*) et les chiffres (ou la *mantisse*). La décomposition que nous avons fait en base 10 peut sans problème être transposée en base 2.

Aisi un nombre réel codé sur 4 octets se répartit de la manière suivante

- 1 bit de signe
- 8 bits pour l'exposant
- 23 bits pour la mantisse, comme le premier chiffre sera toujours zéro, on peut l'omettre et gagner ainsi un bit significatif

Cette convention appelée *notation en virgule flottante* permet de coder les nombres réels de $\pm 1,175494.10^{-38}$ à $\pm 3,402823.10^{38}$ avec 7 chiffres significatifs. Il est aussi possible de représenter un nombre réels sur 64 bits (8 octets) en *double précision* avec 15 chiffres significatifs de $\pm 2,225074.10^{-308}$ à $1,797693.10^{308}$.

Il convient cependant de se rappeler que cette représentation des nombres réels n'est pas exacte, c'st une approximation.

2.3 Conséquences

La conséquence de ces représentations, est que les calculs se feront toujours avec une certaine précision intrinsèque (par exemple 7 chiffres significatifs). Imaginons que nous voulions réaliser le calcul suivant avec des nombres réels codés sur 4 octets

$$s_{i+1} = s_i + 10^{-9}$$

avec $s_0 = 1$, si l'on répète le calcul 10^9 fois, le résultat donnera 1... La raison en est que pour réaliser l'addition la machine dispose d'une précision de 10^{-7}, donc $1+10^{-9} = 1$. Le même calcul conduit en double précision donnerait un résultat correct (la précision est alors de 15 chiffres).

Il faut donc extrêmement prudent lorsque l'on réalise des opérations entre des nombres qui peuvent prendre des valeurs très différentes. Il est aussi prudent de se rappeler que lorsque l'on fait des calculs itératifs (ce qui est très fréquent en calcul numérique), les erreurs peuvent s'ajouter les unes aux autres pour finalement conduire à des résultats dont l'erreur n'est pas négligeable, voire à des résultats aberrants.

Un deuxième cas classqiue d'erreur est le suivant : on compare un nombre réel `if(a == 0.0) then ...`. Le résultat d'une telle comparaison est aléatoire à cause des arrondis ! Imaginons que a soit le résultat d'un long calcul, va-t-il valoir 0 ou $0,1234567.10^{-35}$? Il est prudent de remplacer la comparaison précédente par `if(`$|a| < \epsilon$`)`, qui se traduirait par : si $|a|$ est inférieur à la précision alors ...

Il faut cependant se poser la question de la précision adaptée à un calcul : si l'on est capable représenter un nombre avec une précision de 10^{-7}, cela ne représente pas une précision absolue mais relative. On pourra par exemple par exemple représenter $x \approx 1$ avec cette précision, mais $x \approx 10^{20}$ sera représenté avec une précision de $10^{-7} \times 10^{20} = 10^{13}$ ...

D'une manière générale, lorsque l'on fait des calculs numériques, il est utile de vérifier le calcul sur des cas où l'on connait la solution afin de vérifier le fonctionnement correct d'un programme.

Chapitre 3

Recherche des racines d'une fonction

Nous nous intéressons ici à un des problèmes les plus basiques : résoudre numériquement une équation du type

$$f(x) = 0$$

Dans le cadre de ce document, nous n'allons nous intéresser qu'au cas où il y a une seule variable indépendante x, c'est à dire au problème à une dimension. Excepté le cas des problèmes linéaires, la recherche des racines de la fonction $f(x)$ va se faire par itérations. Il faut pour cela partir d'une solution approchée, même grossière, et bâtir un algorithme qui va "améliorer" la solution jusqu'à satisfaire un critère de convergence. Nous allons nous intéresser à deux types d'algorithmes :

- méthode dichotomique
- méthode utilisant la dérivée

Mais avant d'aborder ces algorithmes, nous allons étudier des algorithmes simples d'encadrement permettant de définir des solutions approchées de $f(x) = 0$.

3.1 Encadrement

On dira qu'une racine d'une fonction $f(x)$ est encadrée par l'intervalle (a, b) si $f(a)$ et $f(b)$ ont des signes opposés. Si la fonction est continue, alors il existe au moins une racine de f dans cet intervalle (thèorème de la valeur intermédiaire). Si la fonction n'est pas continue, on peut à la place d'une racine, trouver une discontinuité qui traverse le zéro dans l'intervalle (a, b).

D'un point de vue numérique, cela va être équivalent à une racine car le comportement va être le même, c.a.d. que l'on va traverser le zéro entre deux nombres flottants de la représentation en précision finie de la machine. Seules les fonctions du type

$$f(x) = \frac{1}{x - c}$$

vont présenter des singularités mais pas de racines. Dans ce cas, certains algorithmes pourront converger vers c, mais dans ce cas il n'y a (heureusement) pas de confusion possible avec une racine, puisqu'une évaluation de $|f(x)|$ va donner un résultat très grand plutôt que proche de zéro. Il est donc important d'avoir une idée de la variation de $f(x)$ dans l'intervalle étudié pour

choisir un algorithme adapté, d'où la nécessité d'avoir des algorithmes simples permettant de déterminer les intervalles contenant une racine pour une fonction quelconque.

L'algorithme 1 permet, partant d'un intervalle de départ (x_1, x_2), de l'élargir jusqu'à ce qu'il contienne une racine, les valeurs renvoyées sont alors le nouvel intervalle (x_1, x_2) contenant une racine. Ce type de procédure est une bonne manière de commencer une recherche de racine, et a de bonnes chances de fonctionner si la fonction étudiée a des signes opposés à la limite $x \to \pm\infty$.

Algorithme 1 Encadrement d'une fonction

Pré-requis : un intervalle (x_1, x_2), un nombre d'essais N

Pré-requis : $f(x)$ définie dans l'intervalle (x_1, x_2) étudié, $x_1 \neq x_2$

Fournit : l'intervalle x_1, x_2 contenant une racine

$f_1 = f(x_1)$

$f_2 = f(x_2)$

$\alpha \leftarrow 1.6$ (par exemple)

pour $i = 1$ à N **faire**

  si $f_1 < f_2$ **alors**

    Renvoyer x_1 et x_2

  fin si

  si $|f_1| < |f_2|$ **alors**

    $f_1 = f(x_1 + \alpha(x_1 - x_2))$

  sinon

    $f_2 = f(x_2 + \alpha(x_2 - x_1)$

  fin si

fin pour

L'algorithme 2 permet lui, à l'inverse, de réduire l'intervalle de départ. On va simplement subdiviser l'intervalle de départ (x_1, x_2) en n intervalles, et renvoyer les sous-intervalles contenant une racine sous forme de deux tableaux.

A l'aide de ces algorithmes, nous sommes maintenant capables de trouver des intervalles contenant les racines d'une fonction, c'est un pré-requis pour estimer plus précisemment la valeur d'une racine particulière d'une fonction.

Algorithme 2 Réduction de l'encadrement d'une fonction

Pré-requis : L'intervalle (x_1, x_2) fourni contient une racine de f

Pré-requis : nombre maximum de racines recherchées N

Fournit : nr le nombre de racines trouvés, $x_1[]$ et $x_2[]$ deux tableaux contenant les bornes des intervales contenant les racines

$nr \leftarrow 0$

$dx \leftarrow (x_2 - x_1)/n$

$x \leftarrow x_1$

$fp = f(x)$

pour $i = 0$ à N **faire**

$x \leftarrow x + dx$

$fc = f(x)$

si $fp * fc \leq 0$ **alors**

$nr \leftarrow nr + 1$

$x_1[nr] \leftarrow x - dx$

$x_2[nr] \leftarrow x$

fin si

$fp = fc$

fin pour

renvoyer nr

3.2 Dichotomie

Le principe est extrêmement simple, il est schématisé sur la figure 3.1. On choisit un intervalle (a, b) encadrant une racine de la fonction f étudiée. On divise cet intervalle en 2 par un point c. On évalue $f(a).f(c)$, si $f(a).f(c) < 0$, alors le nouvel intervalle devient (a, c), sinon (c, b). Et l'on recommence ainsi jusqu'à ce que l'intervalle satisfasse le critère de convergence. Cette méthode a de bonnes chances de fonctionner si la fonction étudiée a des signes opposés à la limite $x \to \pm\infty$.

L'algorithme 3.2 est un algorithme simple permettant de déterminer une valeur approchée de la racine d'une fonction dans le cas où $f(x)$ traverse le zéro dans l'intervalle (x_1, x_2) étudié.

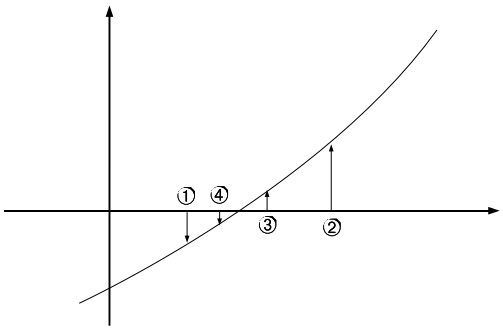

FIG. 3.1 – principe de la méthode de recherche de racines par dichotomie. L'intervalle initial (① ,②) est divisé en 2 par ③, puis l'intervalle devient (①,③)

Algorithme 3 Recherche d'une racine par dichotomie

Pré-requis : l'intervalle d'étude (x_1, x_2) et la précision désirée ϵ

Fournit : l'intervalle encadrant la racine à la précision près

$f1 \leftarrow f(x_1)$
$f2 \leftarrow f(x_2)$
repeter
  $ftemp \leftarrow f((x_1 + x_2)/2)$
  si $f1 * ftemp < 0$ **alors**
    $x_2 \leftarrow (x_1 + x_2)/2$
  sinon
    $x_1 \leftarrow (x_1 + x_2)/2$
  fin si
tant que $x_2 - x_1 < \epsilon$
renvoyer x_1 et x_2

3.2.1 Convergence de la méthode

Dans cette algorithme, après chaque itération, l'intervalle (x_1, x_2) a diminué d'un facteur 2. Après n itérations, la racine est contenu dans un intervalle de taille ϵ_n, avec

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

Donc, pour parvenir à la précision ϵ désirée en partant de $\epsilon_0 = x_2 - x_1$ il faudra

$$n = \log_2 \frac{\epsilon_0}{\epsilon}$$

itérations. Le cas étudié ici où l'on divise l'intervalle par 2 à chaque itération présente une **convergence linéaire**
On voit en effet que pour $\epsilon_{n+1} = \alpha \times \epsilon_n^m$ avec $\alpha < 1$

- la convergence est linéaire si $m = 1$
- la convergence est superlinéaire si $m > 1$

Pour obtenir des résultats cohérents avec un algorithme de ce type, il faut faire attention au critère de convergence ϵ. Il est en effet nécessaire de se rappeler qu'un ordinateur utilise un nombre fini de bits pour représenter un nombre en virgule flottante. Si, analytiquement la fonction f passe par zéro, il est possible que la valeur *calculée* ne vale jamais zéro. Si la racine recherchée est proche de 1, demander une précision absolue de 10^{-8} est raisonnable, mais ne l'est plus si la racine est proche de 10^{20} par exemple. On pourrait fournir une précision *relative* (fractionnaire), mais dans ce cas le calcul devient problématique si la racine est proche de zéro...D'une manière générale, il est préférable de fournir une tolérance absolue pour que les itérations se poursuivent jusqu'à ce que l'intervalle de recherche deviennent plus petit que cette tolérance en unités absolues. La valeur $\epsilon_m(|x_1| + |x_2|)/2$ où ϵ_m est la précision de représentation de la machine, et (x_1, x_2) l'intervalle initial est un bon choix. Il faut cependant toujours se poser la question de ce que représente une tolérance raisonnable lorsque la racine est proche de zéro.

3.3 Méthode de Newton - Raphson

Cette méthode est la plus utilisée pour la recherche de racines dans les problèmes à une dimension. Elle requiert cependant l'évaluation de $f(x)$ et de $f'(x)$.

Le principe est le suivant : on prend la tangente de $f(x)$ au point x_i (solution approchée), et on la prolonge jusqu'à l'axe des abscisses (fig. 3.2). On utilise alors cette nouvelle abscisse comme point x_{i+1}, et l'on recommence tant que $x_{i+1} - x_i$ ne satisfait pas le critère de convergence.

Cela peut se traduire algébriquement grâce aux développements de Taylor

$$f(x + \delta) \approx f(x) + \delta f'(x) + \frac{\delta^2}{2} f''(x) \ldots$$

Si on néglige les termes d'ordre supérieurs à 1, écrire $f(x+\delta) = 0$ revient à écrire $f(x)+\delta f'(x) \approx 0$ d'où

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

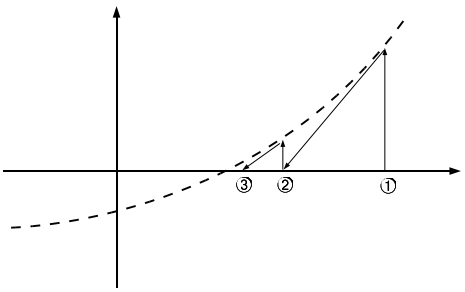

FIG. 3.2 – Principe de la méthode de Newton-Raphson : on part de la solution approchée ◯ 1 , l'extrapolation de la tangente à la courbe pour cette abscisse fournit la nouvelle solution approchée ②.

Lorsque l'on se trouve loin d'une racine de la fonction étudiée, les termes du développement d'ordre supérieur à 1 *ne peuvent pas* être négligés, aussi la méthode peut donner des résultats aberrants. Par exemple si la solution approchée de départ est loin d'une racine, l'intervalle iniial de recherche peut être proche d'un extrémum local (fig.3.3), ce qui signifie que $f'(x) \to 0$. Cela sera fatal dans le cadre de cette méthode...

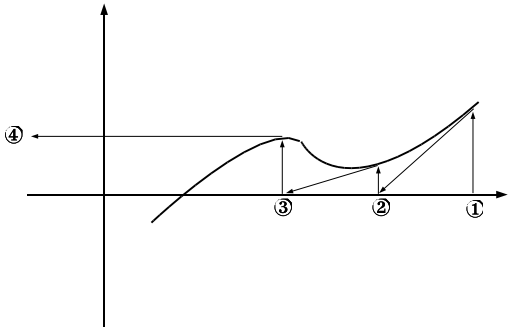

FIG. 3.3 – Cas particulier où l'on trouve un extrémum local. Dans ce cas la méthode de Newton-Raphson diverge

L'algorithme 3.3, présente une implémentation de cette méthode.

3.3.1 Convergence de la méthode

Pour ϵ petit

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x) + \epsilon^2 \frac{f''(x)}{2} + \ldots$$

$$f'(x+\epsilon) \approx f'(x) + \epsilon f''(x) + \ldots$$

Algorithme 4 Recherche de racines par la méthode de Newton-Raphson

Pré-requis : valeur de départ x, précision désirée ϵ

Fournit : valeur x à la précision désirée

si $f'(x) = 0$ **alors**

Sortie du programme

fin si

repeter

$x_n \leftarrow x$

$x \leftarrow x - f(x)/f'(x)$

tant que $|x - x_n| < \epsilon$ OU $f'(x) = 0$

renvoyer x

d'après la formule de Newton-Paphson

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

Quand une solution approchée x_i diffère de la solution vraie α par une quantité $\epsilon_i = \alpha - x_i$, on a

$$f(\alpha) = f(x_i + \epsilon_i) = f(x_i) + \epsilon_i f'(x_i) + \epsilon_i^2 \frac{f''(x_i)}{2} + \dots$$

$$f(\alpha) = 0 \Leftrightarrow f(x_i + \epsilon_i) = f(x_i) + (\alpha - x_i) f'(x_i) + \epsilon_i^2 \frac{f''(x_i)}{2} + \dots = 0$$

donc

$$\underbrace{\frac{f(x_i)}{f'(x_i)} - x_i}_{=-x_{i+1}} + \alpha + \epsilon_i^2 \frac{f''(x_i)}{2f'(x_i)} = 0$$

d'où

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

on voit donc que la convergence est quadratique, la méthode est plus efficace que la dichotomie où la convergence n'est que linéaire. Ceci suppose bien sûr que l'on sache évaluer $f'(x)$ en tout point, et que $f'(\alpha) \neq 0$.

Si la forme analytique de $f'(x)$ n'est pas connue, il faut calculer numériquement cette quantité par

$$f'(x) \approx \frac{f(x + dx) - f(x)}{dx}$$

on doit donc évaluer $f(x)$ 2 fois à chaque itération, l'ordre de convergence va donc passer de 2 à $\sqrt{2}$. De plus, si l'intervalle dx est trop petit, il y a un risque que les erreurs d'arrondi fassent diverger le calcul. A l'opposé, si l'intervalle dx est grand, la convergence ne sera que linéaire.

En conclusion on peut donc dire que la méthode de Newton-Raphson n'est pas une méthode intéressante si l'on ne peut déterminer de forme analytique de $f'(x)$.

Chapitre 4

Minimisation ou maximisation d'une fonction

Le problème peut-être posé simplement : considérons une fonction f dépendant d'une ou plusieurs variables indépendantes. Nous voulons trouver les valeurs de ces variables pour lesquelles f prend une valeur maximale ou minimale. La recherche d'un maximum ou d'un minimum peuvent être réalisée par un programme identique en travaillant simplement sur f ou $1/f$.

Un extrémum d'une fonction peut être soit *local* (minimum dans un intervalle donné)soit *global* (minimum réel) comme on le voit sur la figure 4.1

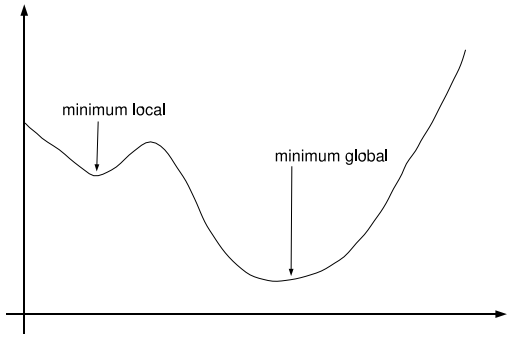

FIG. 4.1 – Les extrémums d'une fonction peuvent être locaux ou globaux

C'est une des raisons pour lesquelles la recherche d'un extrémum est souvent une tâche délicate. Ce chapitre va être extrêmement succint, puisque nous n'aborderons qu'une méthode de minimisation d'une fonction présentant une variable indépendante (probléme 1D).

4.1 Recherche par la section d'or (golden section search)

Nous avons déjà vu dans le chapitre 3 que pour obtenir une valeur approchée d'une racine d'une fonction il était suffisant de l'encadrer dans un intervalle (a, b). Ceci est insuffisant pour

caractériser un minimum. Nous aurons besoins dans ce cas de trois points (a, b, c). Un intervalle tel que $a < b < c$ et $f(b) < f(a), f(b) < f(c)$ caractérise un minimum dans l'intervalle (a, c). En d'autres termes pour encadrer un minimum nous avons besoin d'un triplet de point tel que le point central présente une valeur de f inférieure à celles des bornes de l'intervalle.

4.1.1 Principe

Le principe de cette recherche est résumé sur la figure 4.2. Le minimum est encadré à l'origine par (①,③,②). La fonction est évaluée en ④, qui remplace ②. Ensuite en ⑤qui remplace ①puis en ⑥qui remplace ④. La règle est de garder un point central pour lequel la valeur de f est inférieure à celles des bornes. Après cette série d'itérations, le minimum est alors encadré par (⑤,③,⑥). Le principe est analogue à celui utilisé lors d'une recherche de racine par dichotomie (cf. chapitre 3.2)

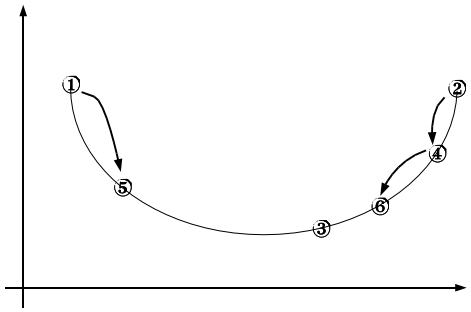

FIG. 4.2 – Encadrements succesif d'un minimum d'une fonction f. Le minimum est encadré à l'origine par (①,③,②). Après 3 évaluations de f le minimum est finalement encadré par (⑤,③ ,⑥).

Le seul point délicat, est de définir une méthode pour choisir un nouveau point d'encadrement dans l'intervalle (a, b, c) initial. Supposons que b soit une fraction ω du segment $[ac]$, alors

$$\frac{b-a}{c-a} = \omega \Leftrightarrow \frac{c-b}{c-a} = 1-\omega$$

Si l'on choisit un nouveau point x eloigné d'une fraction z par rapport à b

$$\frac{x-b}{c-a} = z$$

le nouveau segment d'encadrement aura alors une longueur $\omega + z$ par rapport au segment initial, ou de longueur $1-\omega$. Si nous voulons minimiser le pire cas (attitude plutôt saine...) nous allons choisir z pour que ces deux longueurs soient équivalentes.

$$1-\omega = \omega + z \Leftrightarrow z = 1-2\omega$$

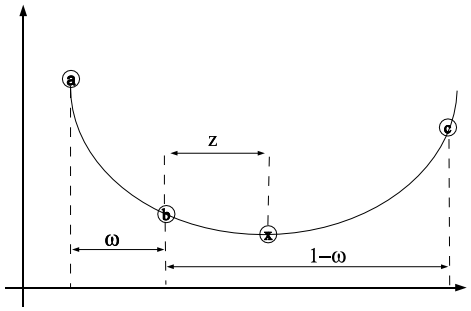

FIG. 4.3 – Comment choisir un nouveau point pour réduire l'encadrement du minimum. La position optimale est $\omega = (b-a)/(c-a) = (3-\sqrt{5})/2$. Le nouveau point x doit être choisi pour respecter cette proportion par rapport à (a,b) ou (b,c).

On montre alors simplement que le nouveau point x est le symétrique de b dans son intervalle d'origine, c.a.d. que $|b-a| = |x-c|$. Cela signifie que le point x se trouve dans le plus grand des segments $[ab], [bc]$. Reste maintenant à définir la position du point x à l'intérieur de ce segment. La valeur ω provient d'une étape précédente de calcul, et si l'on suppose qu'elle est optimale, alors z doit être choisie de la même manière. Cette similarité d'échelle implique que x doit être situé à la même fraction du segment $[bc]$ (si $[bc]$ est le segment le plus long) que b l'était du segment $[ac]$. Cela conduit à la relation

$$\frac{z}{1-\omega} = \omega$$

en combinant cette équation avec la définition de z on arrive à l'équation quadratique

$$\omega^2 - 3\omega = 0 \Leftrightarrow \omega = \frac{3-\sqrt{5}}{2} \approx 0.38197$$

En d'autres termes, ce résultat signifie que l'intervalle d'encadrement (a,b,c) a son point central b est situé a une distance telle que

$$\frac{b-a}{c-a} = \omega \approx 0.38197$$

ou

$$\frac{c-b}{c-a} = 1-\omega \approx 0.61803$$

ces fractions sont celles de la *moyenne d'or* ou de *la section d'or* supposées avoir certaines propriétés esthétiques par les pythagoriciens du monde antique. C'est la raison pour laquelle cette méthode optimale de détermination du minimum d'une fonction est appelée recherche par section d'or (golden search section). L'algorithme 5 décrit une impémentation de cette méthode.

Algorithme 5 Recherche d'un minimum par la methode golden section search

Pré-requis : un intervalle de départ (a, b, c), une tolérance ϵ

Fournit : la meilleure estimation du minimum

$R \leftarrow 0.38197$

$C \leftarrow 1 - R$

$x_0 \leftarrow a$

$x_3 \leftarrow c$

si $(c - b) > (b - a)$ **alors**

$x_1 \leftarrow b$

$x_2 = (c - b) * R + b$

sinon

$x_2 \leftarrow b$

$x_1 \leftarrow a + (b - a) * R$

fin si

$f_1 \leftarrow f(x_1)$

$f_2 \leftarrow f(x_2)$

tant que $|x_3 - x_0| > \epsilon * (|x_1| + |x_2|)$ **faire**

si $f_2 < f_1$ **alors**

$x_0 \leftarrow x_1; x_1 \leftarrow x_2; x_2 \leftarrow C * x_1 + R * x_3$

$f_1 \leftarrow f_2; f_2 \leftarrow f(x_2)$

sinon

$x_3 \leftarrow x_2; x_2 \leftarrow x_1; x_1 \leftarrow C * x_2 + R * x_0$

$f_2 \leftarrow f_1; f_1 \leftarrow f(x_1)$

fin si

fin tant que

si $f_1 < f_2$ **alors**

renvoyer x_1

sinon

renvoyer x_2

fin si

4.1.2 Convergence

Cette méthode garantit que chaque nouvelle évaluation de la fonction va encadrer le minimum dans un intervalle qui vaut 0.61803 fois le précédent, en d'autres termes

$$\epsilon_{i+1} = 0.61803\epsilon_i$$

Cela signifie donc que la convergence est linéaire, mais moins rapide que dans le cas de la méthode de recherche de racines par dichotomie où le nouvel intervalle valait la moitié du précédent.

La méthode abordée dans ce chapitre n'est bien évidemment pas la seule. Il existe des méthodes pouvant être plus performantes, ou mieux adaptées à des cas particuliers, et également un très vaste champ d'investigation concernant les problèmes à N dimensions que nous n'avons pas abordés. Je ne saurais trop vous conseiller la lecture du chapitre 10 des *Numerical Recipes* [1] pour vous cultiver sur ce sujet...

[1] *Numerical Recipes in C*, W.H. Press et al., Cambridge University Press,2002. Le même livre existe pour d'autres language de programmation : Fortran et C++.

Chapitre 5

Interpolation

L'interpolation est un concept fondamental en calcul numérique : on évalue une fonction f en $x_0, x_1 \dots x_n$. Que faire si l'on veut connaître la valeur de $f(x)$ si $x_0 < x < x_1$? Il n'y a pas d'évaluation de f pour cette valeur, on a donc recours a l'interpolation qui consiste en une estimation d'une valeur inconnue s'appuyant sur des valeurs connues qui l'encadrent.

Ce problème se pose fréquemment pour l'expérimentateur scientifique qui réalise des mesures d'une grandeur en un certain nombre de points, et qui a parfois besoin de connaître une approximation de cette grandeur en dehors des valeurs mesurées. Une autre utilisation courante de l'interpolation est le lissage de courbe. Cela rend le résultat graphiquement plus satisfaisant, mais la courbe obtenue n'est pas plus précise que la série $f(x_0), f(x_1) \dots f(x_n)$, mais peut permettre d'avoir une estimation d'une valeur x non comprise dans la série. Ceci à la condition que l'on soit capable d'estimer l'erreur commise lors du processus d'interpolation. De plus, l'interpolation est à la base de nombreuses méthodes numériques d'intégration de fonctions.

Ces techniques sont basées sur les polynômes de Lagrange. On peut en effet montrer que pour une fonction f définie sur un intervalle $[a, b]$ pour $n + 1$ valeurs distinctes, il existe un polynôme d'ordre n tel que $P_n(x) = f(x_i)$ pour $i = 0 \dots n$. Nous détaillerons dans ce chapitre les calculs pour des polynômes d'ordre 1 et 2, puis nous généraliserons à l'ordre n.

5.1 Interpolation linéaire

Il est bien connu qu'il n'y a qu'une droite passant par deux points (x_i, f_i) et (x_{i+1}, f_{i+1}), comme cela est schématisé sur la figure 5.1. Cette droite peut-être décrite par un polynôme $P_i(x)$ de premier degré en x :

$$P_i(x) = f_i + \frac{f_{i+1} - f_i}{x_{i+1} - x_i}(x - x_i)$$

si l'on suppose que f a une dérivée seconde, l'écart entre f et P_i (et donc l'erreur commise lors d'une interpolation peut-être estimée par

$$\max |f(x) - P_i(x)| \leq \frac{(x_{i+1} - x_i)^2}{2} \max |f''(x)|$$

pour $x_i \leq x_{i+1}$. Si l'écart entre deux évaluations successives de la fonction (le pas d'échantillongage) vaut $x_{i+1} - x_i = h$ alors l'erreur s'exprime

$$\max |f(x) - P_i(x)| \leq \frac{h^2}{2} \max |f''(x)|$$

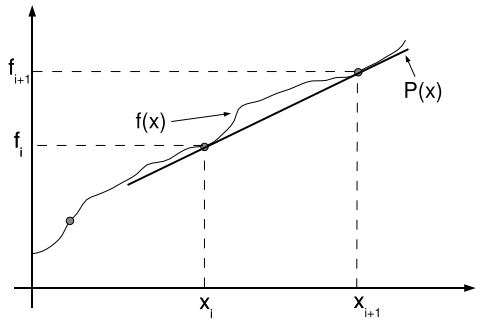

FIG. 5.1 – La fonction f est définie aux points x_i, entre ces points, il est possible de l'interpoler par un polynôme d'ordre 1 passant par deux points consécutifs.

exemple : soit la fonction $f(x) = \sin(x)$ évaluée (tabulée) en $n+1$ points dans l'intervalle $[0; \frac{\pi}{2}]$ avec une précision désirée de 10^{-m}. Dans ce cas le pas d'échantillonage $h = \pi/2n$, et l'on souhaite

$$\max |f(x) - P(x)| \leq 10^{-m} \Leftrightarrow \left(\frac{\pi}{2n}\right)^2 \times \frac{1}{2} \max |\frac{d^2 \sin(x)}{dx^2}| \leq 10^{-m}$$

$$\frac{d^2 \sin(x)}{dx^2} = -\sin(x)$$

donc

$$\max |\frac{d^2 \sin(x)}{dx^2}| = 1 \Leftrightarrow \left(\frac{\pi}{2n}\right)^2 \times \frac{1}{2} \leq 10^{-m}$$

d'où

$$\frac{\pi^2}{8n^2} \leq 10^{-m} \Leftrightarrow \frac{8n^2}{\pi^2} \geq 10^m \Leftrightarrow n^2 \geq \frac{\pi^2}{8}.10^m$$

ce qui conduit finalement à

$$n \geq \frac{\pi}{2\sqrt{2}}.10^{m/2}$$

Concrètement, cela signifie que si l'on souhaite pouvoir interpoler un point de la fonction f avec une précision de 10^{-m}, il va être nécessaire d'évaluer la fonction f en n points. Le tableau 5.1 résume ceci pour quelques valeurs de m. Ceci vous montre qu'augmenter la précision peut couter extrêmement cher en temps de calcul.

Il pourrait être astucieux de faire varier le pas d'échantillonage h de la fonction f, de façon à suivre le rayon de courbure local de la fonction : là où il est grand choisir h n'a pas besoin d'être grand, et inversement. On pourrait construire un algorithme réalisant cette opération en

précision	n
10^{-3}	~ 35
10^{-6}	~ 1000
10^{-8}	~ 11000

TAB. 5.1 – Nombre d'évaluations n nécessaires pour parvenir à une précision donnée dans le cas d'une interpolation linéaire.

évaluant la dérivée seconde de la fonction en chaque point. Néanmoins, le calcul d'une dérivée seconde n'est pas toujours facile à réaliser, il vaut donc mieux utiliser une méthode plus précise, c.a.d. utiliser un polynôme d'ordre plus élevé.

5.2 Interpolation quadratique

Le principe est identique à l'interpolation linéaire : il existe un polynôme unique d'ordre 2 P_{i+1} qui passe par 3 points décrivant notre fonction f (voir figure 5.2)

$$\begin{aligned} P_{i+1}(x) &= f_i \frac{(x-x_{i+1})(x-x_{i+2})}{(x_i-x_{i+1})(x_i-x_{i+2})} + f_{i+1}\frac{(x-x_{i+2})(x-x_i)}{(x_{i+1}-x_{i+2})(x_{i+1}-x_i)} \\ &\quad + f_{i+2}\frac{(x-x_i)(x-x_{i+1})}{(x_{i+2}-x_i)(x_{i+2}-x_{i+1})} \end{aligned}$$

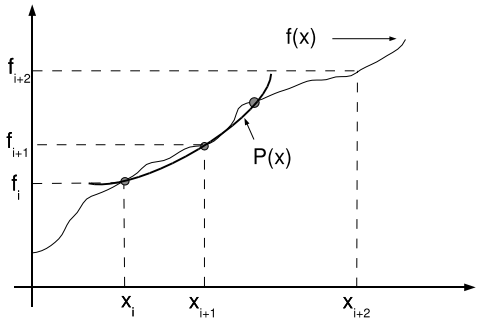

FIG. 5.2 – La fonction f est définie aux points x_i, entre ces points, il est possible de l'interpoler par un polynôme d'ordre 2 passant par trois points consécutifs.

Cette expression est simple lorsque le pas d'échantillonage est constant. Si ce pas vaut h alors $x = x_{i+1} + u.h$. Si $u = -1$ $x = x_i$, si $u = 0$ $x = x_{i+1}$ et si $u = 1$ $x = x_{i+2}$. L'équation précédente peut alors s'écrire plus simplement

$$P_{i+1}(u) = f_{i+1} + \frac{f_{i+2} - f_i}{2}u + \frac{f_{i+2} - 2f_{i+1} + f_i}{2}u^2$$

L'erreur est maintenant majorée par le terme d'ordre 3 du développement en série de Taylor de $f(x)$ au point x_{i+1} :

$$\max|f(x) - P_{i+1}(x)| \leq \frac{h^3}{6}|f^{(3)}(x)| \max$$

Exemple : Si l'on reprend l'exemple précédent de $f(x) = \sin(x)$ évaluée en $n+1$ points, avec une précision de 10^{-m}, on obtient l'inégalité

$$n \geq \frac{\pi}{2\sqrt[3]{6}}.10^{m/3}$$

Le tableau 5.2 présente le nombre d'évaluations nécessaires de la fonction pour une précision donnée. Il est clair que cette méthode est plus performante dans le cas de la fonction étudiée que l'interpolation d'ordre 1.

précision	n
10^{-3}	~ 9
10^{-6}	~ 90
10^{-8}	~ 400

TAB. 5.2 – Nombre d'évaluations n nécessaires pour parvenir à une précision donnée dans le cas d'une interpolation quadratique.

5.3 Généralisation à l'ordre n

Un polynôme de Lagrange d'ordre n s'exprime

$$P_n(x) = \sum_{i=0}^{n} f_i \frac{(x-x_0)\dots(x-x_i)\dots(x-x_n)}{(x_i-x_0)\dots(x_i-x_i)\dots(x_i-x_n)}$$

ou plus simplement

$$P_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

avec

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)}$$

L'erreur est dans ce cas majorée par

$$\max|f(x) - P_n(x)| \leq \max|f^{(n+1)}(x)|.\frac{(x_n-x_0)^{n+1}}{(n+1)!}$$

Ceci suppose que la dérivée d'ordre $(n+1)$ de f existe et est continue. L'algorithme 6 permet de calculer la valeur d'un polynôme de Lagrange d'ordre n en un point, connaissant la fonction f en $n+1$ points.

Exemple : si l'on reprend l'exemple de la fonction $f(x) = \sin(x)$ définie en $n+1$ points sur l'intervalle $[0; \pi/2]$, et que l'on veuille interpoler une valeur avec une précision de 10^{-m} alors

$$10^{-m} \leq \left(\frac{\pi}{2n}\right)^{n+1} \cdot \frac{1}{(n+1)!}$$

5.3.1 Du bon usage de l'interpolation

Il convient de faire attention à l'ordre du polynôme utilisé. En effet si l'on considère la fonction $f(x) = \sin(10x)$, l'erreur

$$|f^{(n+1)}|_{\max} \approx 10^n$$

peut augmenter très rapidement avec l'ordre du polynôme. On peut établir la règle de prudence suivante :

- si $f(x)$ varie lentement, elle pourra être approximée correctement par un polynôme de degré élevé, c.a.d. que l'erreur diminuera avec le degré du polynôme
- si la dérivée de f varie brutalement ou est discontinue, la fonction sera mieux approximée par un polynôme de degré faible, c.a.d. que l'erreur peut augmenter rapidement avec l'ordre du polynôme.

D'une manière générale, l'utilisation de polynôme d'ordre $n > 4,5$ donne rarement de bons résultats numériques.

Algorithme 6 Interpolation par un polynôme de Lagrange d'ordre n

Pré-requis : une fonction f définie en $x_0 \ldots x_n$, l'ordre du polynôme n, la valeur $x_i nter$ où l'on souhaite approximer la fonction

Fournit : l'approximation de f au point souhaité

```
lagr ← 0
pour i = 0 ... n faire
  li ← 1
  pour j = 0 ... n faire
    si j ≠ i alors
      li ← li * (x_inter − x_j)/(x_i − x_j)
    fin si
  fin pour
  lagr ← lagr + li * f(x_i)
fin pour
renvoyer lagr
```

Chapitre 6

Intégration de fonctions

Dans ce chapitre, nous n'allons pas aborder la détermination des primitives des fonctions (ceci concerne le calcul formel, et des logiciels comme MAPLE, MATHEMATHICA...), mais le calcul numérique d'une intégrale

$$\int_a^b f(x)dx$$

qui revient à calculer l'aire sous la courbe $f(x)$ entre a et b. Nous allons aborder 3 méthodes :

- intégration par la méthode des rectanges
- intégration par la méthode des trapèzes
- intégration par la méthode de Simpson

ces méthodes sont basées sur l'interpolation d'ordre 0,1,2 respectivement. Ces méthodes sont généralisables à l'ordre n. Nous terminerons ce chapitre par une discussion sur la recherche d'un pas optimal d'intégration.

6.1 Intégration par la méthode des rectanges

Soit une fonction $f(x)$ dépendant d'une variable indépendante. Nous souhaitons calculer

$$I = \int_{x_0}^{x_n} f(x)dx$$

numériquement, cela revient à calculer la somme

$$I = \sum_{i=0}^{n-1} f_i(x_{i+1} - x_i)$$

Pour cela, nous allons tabuler $f(x)$ en $n+1$ points $f_0, f_1 \ldots f_n$ dans l'intervalle d'intégration $[x_0, x_n]$. On peut approximer l'aire sous la courbe $f(x)$ par une série de rectangles comme représenté sur la figure 6.1, ce qui mathématiquement correspond à l'interpolation de f par un pôlynome d'ordre 0 (une constante). Si la distribution des points est uniforme, ce qui revient à $x_{i+1} - x_i = h$, l'intégration revient à calculer

$$I = h \sum_{i=0}^{n-1} f_i$$

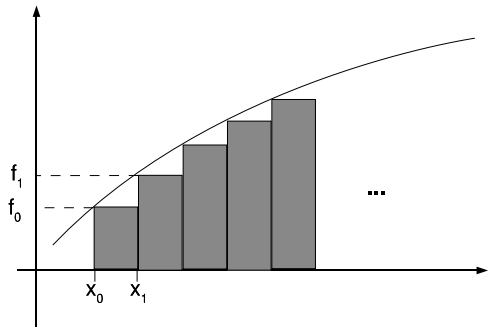

FIG. 6.1 – Découpage de l'aire sous la courbe f par une série de rectangle.

Il y a cependant 2 possibilités pour réaliser ce calcul , comme cela est montré à la figure 6.2. On peut en effet majorer ou minorer la fonction selon que l'on calcule respectivement

$$I = h \sum_{i=0}^{n-1} f_{i+1}$$

ou

$$I = h \sum_{i=0}^{n-1} f_i$$

ceci dans le cas où $f'(x) > 0$. Ceci pose deux problèmes.

Tout d'abord les résultats ne seront pas identiques suivant que l'on majore ou minore f. On peut éventuellement résoudre le problème en réalisant les calculs dans les deux cas, puis en faisant une moyenne des deux résultats.

Ensuite, pour les mêmes raisons, on n'obtiendra pas les mêmes résultats selon que l'on intègre de $a \to b$ ou de $b \to a$. Cette limitation doit être prise en compte lors de calculs utilisant cette méthode.

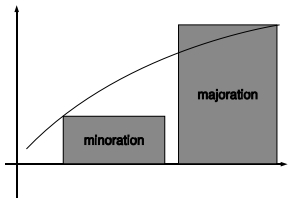

FIG. 6.2 – Il y a deux manières de délimiter l'aire sous la courbe par une série de rectangle : $f_i(x_{i+1} - x_i)$ (minoration) ou $f_{i+1}(x_{i+1} - x_i)$ (majoration)

6.1.1 Incertitude de la méthode

Nous avons vu que cette méthode revient à approximer la fonction f par un polynôme d'ordre 0. Dans ce cas, l'incertitude sur chaque point f_i s'exprime $\epsilon \leq h|f'(x)|_{\max}$, donc pour chaque intervalle h d'intégration

$$\epsilon \leq \frac{h^2}{2}|f'(x)|_{\max}$$

L'incertitude totale pour n pas d'intégration sera donc

$$\epsilon \leq n.\frac{h^2}{2}|f'(x)|_{\max}$$

Cette méthode est extrêment simple à mettre en œuvre, et ne nécessite pas de gros calculs. Cependant si l'on souhaite conserver une incertitude acceptable, il est généralement nécessaire d'utiliser un pas d'échantillonage petit.

6.2 Intégration par la méthode des trapèzes

Le principe de cette méthode est analogue à celui de l'intégration par une série de rectangles. Dans cette méthode on cherche à augmenter la précision du calcul tout en évitant la disymétrie qui apparaît dans la définition de l'intégration par rectangles. Nous allons donc ici aussi tabuler $f(x)$ en $n+1$ points $f_0, f_1 \ldots f_n$ dans l'intervalle d'intégration $[x_0, x_n]$. On va ensuite interpoler linéairement (polynôme d'ordre 1, c.a.d. une droite) pour approximer $f(x)$ entre deux points tabulés. On va donc remplacer les arcs x_i, x_{i+1} par leurs cordes (figure 6.3). L'intégration va

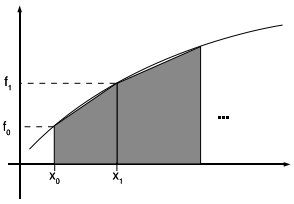

FIG. 6.3 – Découpage de l'aire sous la courbe f par une série de trapèzes.

alors se faire en sommant l'aire des trapèzes, soit

$$I = \sum_{i=0}^{n-1} \frac{f_{i+1} + f_i}{2}(x_{i+1} - x_i)$$

si l'échantillonage des points est régulier, c.a.d. si $x_{i+1} - x_i = h$ alors l'intégrale s'exprimera

$$I = \frac{h}{2}(f_0 + f_n) + h.\sum_{i=1}^{n-1} f_i$$

en effet chaque trapèze aura chacun de ces côtés en commun avec les trapèzes adjacents, à l'exception du premier et du dernier qui n'ont qu'un côté en commun.

6.2.1 Incertitude de la méthode

Nous avons vu que cette méthode revient à approximer la fonction f par un polynôme d'ordre 1. Dans ce cas, l'incertitude sur chaque point f_i s'exprime

$$\epsilon \leq \frac{h^2}{2}|f''(x)|_{\max}$$

donc pour chaque intervalle h d'intégration

$$\epsilon \leq \frac{h^3}{4}|f''(x)|_{\max}$$

L'incertitude totale pour n pas d'intégration sera donc

$$\epsilon \leq n.\frac{h^3}{4}|f''(x)|_{\max}$$

La précision est supérieure à celle de l'intégration par rectangles, le terme d'erreur étant maintenant le terme du deuxième ordre du développement de f en série de Taylor.

6.3 Intégration par la méthode de Simpson

Comme les autres méthodes, cette méthode est basée sur une distribution f_i de la fonction $f(x)$ que l'on interpole. La méthode de Simpson correspond au cas de l'interpolation parabolique (polynôme d'ordre 2 passant par 3 points f_i). Le polynôme d'interpolation $P_i(x)$d'ordre 2 a été défini au chapitre 5.2, et l'on peut donc exprimer

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx = \int_{x_{i-1}}^{x_{i+1}} P_i(x)dx = h\int_{-1}^{1} P_i(u)du$$

en posant $x = x_{i+1} + u.h$, donc si $u = -1$ $x = x_i$, si $u = 0$ $x = x_{i+1}$, et si $u = 1$ $x = x_{i+2}$. On peut alors calculer

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx = \frac{h}{3}(f_{i-1} + 4f_i + f_{i+1})$$

l'expression de l'intégrale sur un intervalle de calcul de $2n+1$ points s'exprime alors

$$I = \frac{h}{3}\sum_{i=1}^{2n}(f_{i-1} + 4f_i + f_{i+1})$$

que l'on peut réécrire plus commodemment

$$I = \frac{h}{3}\left[(f_1 + f_n) + \sum_{\substack{i=2 \\ i\ pair}}^{n} 4.f_i + \sum_{\substack{i=2 \\ i\ impair}}^{n} 2.f_i\right]$$

6.3.1 Incertitude de la méthode

Nous avons vu que cette méthode revient à approximer la fonction f par un polynôme d'ordre 2. Dans ce cas, l'incertitude sur chaque point f_i est majorée par le terme d'ordre 3 du développement en série de Taylor de f

$$\epsilon \leq \frac{h^3}{6}|f^{(3)}(x)|_{\max}$$

donc pour chaque intervalle h d'intégration

$$\epsilon \leq \frac{h^4}{12}|f^{(3)}(x)|_{\max}$$

L'incertitude totale pour n pas d'intégration sera donc

$$\epsilon \leq n.\frac{h^4}{12}|f^{(3)}(x)|_{\max}$$

6.4 Détermination d'un pas optimal d'intégration

La question ici est : lorsque l'on cherche à calculer

$$I_n = \int_b^a f(x)dx$$

quel est le pas optimal $h = x_{i+1} - x_i$ permettant de calculer I_n avec une précision ϵ donnée. Nous avons jusqu'à présent considéré le pas h comme une constante. Reprenons le cas de l'intégration par trapèze. Si l'on souhaite déterminer I_n à la précision ϵ, il suffit de choisir un pas h_0, de calculer I_0, puis de diminuer le pas en h_1, calculer I_1... Ceci tant que $|I_{n+1} - I_n| \geq \epsilon$. La méthode est simple, mais les calculs sont répétitifs et potentiellement longs. Il existe un moyen de réduire le temps de calcul en introduisant une relation de récurrence permettant d'exprimer I_{n+1} en fonction de I_n. Nous allons détailler ce calcul, toujours dans le cas d'une intégration par trapèzes.

①On pose $h_0 = b - a$, et on calcule

$$I_0 = \frac{h_0}{2}\left[f(a) + f(b)\right]$$

②On pose $h_1 = h_0/2$ et on calcule

$$\begin{aligned} I_1 &= \frac{h_1}{2}[f(a) + f(a+h_1) + f(a+h_1) + f(b)] = \overbrace{\frac{h_1}{2}[f(a) + f(b)]}^{\frac{I_0}{2}} + h_1 f(a+h_1) \\ &= \frac{1}{2}I_0 + h_1 f(a+h_1) \\ &= \frac{1}{2}[I_0 + h_0(f(a+h_1))] \end{aligned}$$

③On pose $h_2 = h_1/2 = h_0/4$ et on calcule

$$\begin{aligned} I_2 &= \frac{h_2}{2}[f(a)+f(a+h_2)+f(a+h_2)+f(a+2h_2)+f(a+2h_2)+f(a+3h_2)+f(a+3h_2)+f(b)] \\ &= \underbrace{\underbrace{\frac{h_2}{2}[f(a)+f(b)]}_{\frac{h_1}{4}[f(a)+f(b)]} + h_2[f(a+h_2)+\underbrace{f(a+2h_2)}_{f(a+h_1)}+f(a+3h_2)]}_{\frac{I_1}{2}} \\ &= \frac{1}{2}I_1 + h_2[f(a+h_2)+f(a+3h_2)] \\ &= \frac{1}{2}\{I_1 + h_1[f(a+h_2)+f(a+3h_2)]\} \end{aligned}$$

④On pose $h_3 = h_2/2 = h_0/8$

$$\begin{aligned} I_3 &= \frac{h_3}{2}[f(a)+f(b)+2f(a+h_3)+2f(a+2h_3)+2f(a+3h_3)+2f(a+4h_3) \\ &\quad +2f(a+5h_3)+2f(a+6h_3)+2f(a+7h_3)] \\ &= \frac{1}{2}I_2 + h_3[f(a+h_3)+f(a+3h_3)+f(a+5h_3)+f(a+7h_3)] \\ &= \frac{1}{2}\{I_2 + h_2[\ldots]\} \end{aligned}$$

On peut alors généraliser en

$$I_n = \frac{1}{2}\left[I_{n-1} + h_{n-1}\sum_{i=1 impair}^{2n-1} f(a+(2i-1)h_n)\right]$$

Cette relation permet de calculer économiquement le terme n connaissant le terme précédent. D'une manière générale, un programme réalisant une intégration sera divisé de la manière suivante :

- une fonction permettant de réaliser un pas d'intégration
- une fonction réalisant le calcul de l'intégrale en utilisant des relations de récurrence du type de celle que nous venons de détailler pour fournir un résultat avec une précision connue.

Chapitre 7

Résolution d'équations différentielles linéaires

C'est un problème très fréquent en physique (phénomènes transitoires dans un circuit RLC par exemple), chimie (cinétique des réactions chimiques), astronomie ...Le problème à l'ordre 1 peut-être résolu par les méthodes d'intégration du chapitre 6

$$\frac{dy}{dx} = r(x) \Leftrightarrow y(x) = \int r(x)dx$$

Dans ce chapitre, nous considèrerons 3 types de bases d'équations différentielles respectivement décroissantes, croissantes ou oscillantes :

$$\frac{dy}{dx} + \alpha y \Leftrightarrow y = y_0 \exp(-\alpha x)$$

$$\frac{dy}{dx} - \alpha y \Leftrightarrow y = y_0 \exp(\alpha x)$$

$$\frac{dy}{dx} \pm i\omega y \Leftrightarrow y = y_0 \exp(\pm i\omega x)$$

et nous aborderons plusieurs méthodes numériques de résolution de ce type d'équation, en mettant l'accent sur le fait que toutes les méthodes ne sont pas adaptées à tous les types d'équation.

Les problèmes physiques sont souvent d'ordre supérieur, comme dans l'équation

$$\frac{d^2y}{dx} + \alpha\frac{dy}{dx} = r(x)$$

on peut facilement se ramener au problème à l'ordre 1 en réécrivant l'équation du deuxième ordre précédente comme un système d'équation du premier ordre

$$\begin{aligned} \frac{dy}{dx} &= z(x) \\ \frac{dz}{dx} &= r(x) - \alpha z(x) \end{aligned}$$

système que nous pouvons résoudre par intégration. On peut généraliser à une équation différentielle d'ordre N, que l'on récrit comme un système de N équations f_i d'ordre 1

$$\frac{dy_i(x)}{dx} = f(x, y_1 \dots y_N)$$

où les fonctions f_i sont connues. Pour résoudre le système, il suffit d'intégrer les fonctions f_i.

Pour résoudre numériquement ce système, on part d'un point donné (x_0, y_0) représentant les conditions initiales du problème. On choisit ensuite un pas d'intégration fini $h = \Delta x$ comme pour les méthodes du chapitre 6, et l'on intègre pas à pas avec

$$x_n = x_{n-1} + h$$

$$y_n = y_{n-1} + \Delta y$$

dans ce chapitre, nous allons détailler différentes méthodes pour évaluer les pas finis Δy.

7.1 Méthode d'Euler

Soit

$$\frac{dy_i}{dx} = f_i(x, y) \tag{1}$$

La formule la plus simple que nous puissions utiliser est la formule d'Euler. C'est à dire que dy et dx sont remplacés par des accroissements finis Δy et Δx, et l'on réécrit ensuite les équations en multipliant par Δx, ela revient à écrire, pour un accroissement h

$$y(x + h) = y(x) + hy(x) + O(h^2)$$

le terme $O(h^2)$ étant le terme d'erreur (à l'ordre 2)

On peut alors réécrire l'équation (1) en négligeant les termes du deuxième ordre

$$y_{i+1} = y_i + hf(x_i, y_i)$$

dans le cas où l'on ne considère qu'un échantillonage de x $x_0 \dots x_n$, avec $h = x_{i+1} - x_i$. Si l'on connait les valeurs initiales $\{x_0, y_0\}$, cette méthode permet de calculer toutes les valeurs successives y_i, comme on le voit sur la figure 7.1

Avec une méthode de ce type, on fait avancer la solution de x_n à $x_n + h = x_{n+1}$ à chaque pas d'intégration. Nous avons cependant vu au chapitre 5 que cette méthode est peu performante, et surtout asymétrique. Cela peut poser quelques problèmes : si par exemple vous tracez une trajectoire dépendant d'une équation différentielle entre $t = 0$ et t, vous obtiendrez des résultats différents en réalisant les mêmes calculs entre t et $t = 0$...

Cette méthode est donc à éviter dans la majorité des cas. Il est utile de l'introduire, pour permettre la compréhension des méthodes d'intégration des équations différentielles, et elle reste cependant utilisable dans certains cas simples.

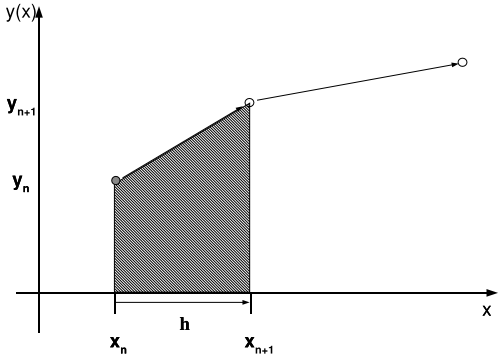

FIG. 7.1 – Méthode d'Euler. Dans cette méthode d'intégration d'une équation différentielle, la dérivée au point de départ x_n de chaque intervalle est extrapolée pour trouver la valeur suivante en x_{n+1} de la fonction.

7.1.1 Précision et stabilité de la méthode

Quelle est la précision de la méthode d'Euler ? Pour y répondre, nous allons considérer le dévelopement en série de Taylor de la fonction $y(x)$ autour de la valeur x_n

$$y_{n+1} = y_n + h\left(\frac{dy}{dx}\right)_n + \frac{h^2}{2}\left(\frac{d^2y}{dx^2}\right)_n + \dots$$

si l'on soustrait cette expression à l'expression de y_{n+1} défini au chapitre 7.1 on obtient

$$y_{n+1} = y_n + h\left(\frac{dy}{dx}\right)_n + \frac{h^2}{2}\left(\frac{d^2y}{dx^2}\right)_n + \dots \approx y_n - hf(y_n, x_n) = y_n + h\left(\frac{dy}{dx}\right)_n$$

Nous voyons donc que le terme en h est correctement exprimé dans l'approximation de la méthode d'Euler, mais que le terme d'ordre supérieur est faux. Ceci nous permet de décrire la méthode d'Euler comme une méthode du premier ordre.

Le terme d'erreur parfois appelé erreur d'arrondi est ici le terme en h^2, on peut réécrire l'équation de la manière suivante

$$y(x+h) = y(x) + hy(x) + O(h^2)$$

le terme $O(h^2)$ étant le terme d'erreur (à l'ordre 2)

Nous venons de voir que la méthode d'Euler est une méthode du premier ordre. Il reste à se poser une question importante : quelle est sa stabilité ? Supposons qu'à un moment du calcul, la solution numérique diffère de la vraie solution d'une quantité δy. Nous écrivons

$$y_{n+1} + \delta y_{n+1} = y_n + \delta y_n - h\left[f(y_n, x_n) + \left(\frac{\partial f}{\partial y}\right)_n \delta y_n\right]$$

où le terme entre crochets est le développement de f en série de taylor, en soustrayant ceci à l'expression de y_{n+1}, on obtient

$$\delta y_{n+1} = \left[1 - h\left(\frac{\partial f}{\partial y}\right)_n\right]\delta y_n = g\delta y_n$$

si g a une magnitude supérieure à 1, alors δy_n va avoir tendance à augnmenter quand n augmente. Il y a alors un risque que cette erreur devienne prédominante devant la solution recherchée. On dira que la méthode d'Euler est stable si $|g| \leq 1$ ou

$$-1 \leq 1 - h\frac{\partial f}{\partial x} \leq 1$$

h étant positif par définition, la deuxième inégalité impose donc que la dérivée de f par rapport à y soit elle aussi positive. La première inégalité introduit une restriction sur δx

$$h \leq 2/\frac{\partial f}{\partial y}$$

Si cette dérivée est complexe, il faut faire attention au calcul de $|g|$. Dans ces cas il peut être plus facile de rechercher des solutions à la condition $|g^2| \leq 1$.

Dans le cas d'une équation d'oscillation du type

$$\frac{dy}{dx} \pm i\omega y = 0 \Leftrightarrow y = y_o \exp(\pm i\omega x)$$

cette inégalité conduira à

$$1 + h^2\omega^2 \leq 1$$

qui est impossible à satisfaire pour des ω et h réels. De même pour une équation du type

$$\frac{dy}{dx} - \alpha y = 0 \Leftrightarrow y = y_o \exp(\alpha x)$$

cette analyse n'a pas de sens. Dans ce cas il serait plus judicieux de s'intéresser à l'erreur relative, et imposer que cette erreur dans la solution n'augmente pas.

décroissance	croissance	oscillation
$h \leq 2/\alpha$	instable	instable

7.1.2 Exemple d'application : la loi de décroissance radioactive

Pour illustrer ce propos, voici un exemple dans un cas extrêmement simple : la loi de décroissance radioactive, où nous sommes capables de donner une solution analytique exacte très facilement.

Une substance radioactive est caractérisée par une constante de décroissance radioactive k. On peut écrire la loi

$$N(t) = N_0 \exp(-k.t)$$

permettant d'exprimer le nombre de noyaux radioactifs à un instant t, $N(t)$ en fonction du nombre de noyaux N_0 à l'instant $t = 0$ et de la constante de décroissance radioactive. Cette loi correspond à l'équation différentielle

$$\frac{dN}{dt} + k.N = 0 \Leftrightarrow \frac{dN}{dt} = -k.N = f(t)$$

Nous allons résoudre numériquement cette équation avec $k = 100s^{-1}$ et $N_0 = 1$ en utilisant la méthode d'Euler.

Nous avons montré précédemment que cette méthode était stable si le pas d'intégration h (ici un temps) respectait le critère $h \leq 2/\alpha$, dans ce cas précis $\alpha = k$. Il est donc nécessaire dans ce cas de choisir un pas $h \leq 2.10^{-2}$s pour satisfaire le critère de stabilité. La figure 7.2 représente les résultats de la simulation pour des pas d'intégration inférieurs ou supérieurs à ce critère, ainsi que la solution exacte en trait plein.

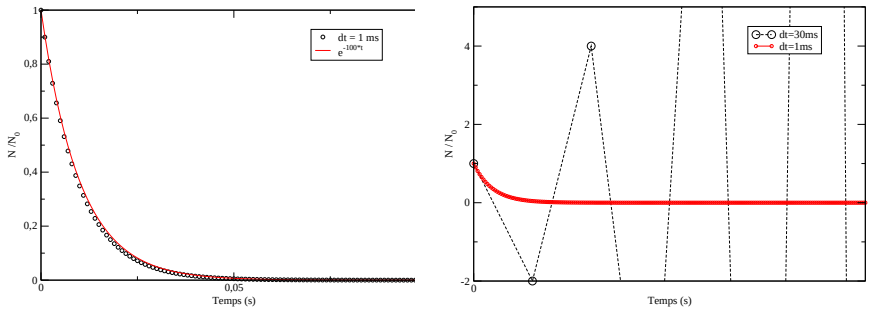

FIG. 7.2 – Application de la méthode d'Euler : simulation de l'évolution temporelle du nombre de noyaux d'une source radioactive (constante de décroissance radioactive $100s^{-1}$. Le critère de stabilité de la méthode impose un pas d'intégration inférieur à 0.02 s. Courbe de gauche : le critère est respecté, le calcul numérique donne des résultats corrects (la solution exacte est tracée en trait plein). Lorsque ce critère n'est pas respecté (courbe de droite) le résultat diverge.

Un simple examen de ce graphe permet de comprendre qu'il est important de choisir un pas d'intégration satisfaisant le critère de stabilité de la méthode, sous peine de voir le calcul diverger rapidement ...On voit donc qu'une méthode peut donner de bons résultats dans certains cas, et pas dans d'autres. Il convient donc de se poser ces questions avant de se lancer dans des calculs complexes...

7.2 Méthode du saut de grenouille

L'idée ici est d'améliorer la méthode d'Euler en symétrisant le problème. On va simplement écrire

$$y_{n+1} = y_{n-1} + 2h.f(x_n, y_n)$$

ici, le terme d'erreur est en $h^3/6$, on parlera donc d'une méthode d'ordre2.

7.2.1 Stabilité

Si l'on tient le même raisonnement que pour la méthode d'Euler, on peut exprimer la variation de la solution par rapport à la solution vraie pour une équation décroissante :

$$\delta y_{n+1} = \delta y_{n-1} - 2h\frac{\partial f}{\partial y}\delta y_n$$

on va écrire $\delta y_n = g\delta y_{n-1}$ et $\delta y_{n+1} = g^2\delta y_{n-1}$ ce qui conduit à

$$g^2 = 1 - 2h.\frac{\partial f}{\partial y}g$$

dont les solutions s'écrivent

$$g = h.\frac{\partial f}{\partial y} \pm \sqrt{1 + \left(h.\frac{\partial f}{\partial y}\right)^2}$$

on a toujours, pour les deux racines $g_1.g_2 = -1$ et $g_1 \neq g_2$ ce qui implique que l'une des deux racines est supérieure à 1. La méthode est donc instable dans ce cas, ainsi que dans le cas d'une équation croissante. Il y a une exception dans le cas ou $\frac{\partial f}{\partial y}$ est purement imaginaire, le terme sous la racine dans l'expression de g peut alors être négatif, il est alors possible que $g_1.g_2 < 1$. C'est le cas pour une équation oscillante où $\frac{\partial f}{\partial y} = \pm i\omega$. Dans ce cas, l'algorithme est stable si le terme sous la racine dans l'expression de g est positif, c'est à dire si $h \leq 1/\omega$

décroissance instable	croissance instable	oscillation $h \leq 1/\omega$

7.3 Méthode de Runge Kutta à l'ordre 2

Nous avons vu deux méthodes : l'une est stable pour des équations décroissantes, l'autre pour les équations oscillantes en respectant certaines conditions. Est-il possible de combiner ces deux conditions de stabilité dans une seule méthode ? La réponse est oui, avec la méthode de Runge Kutta à l'ordre 2.

Le principe est schématisé sur la figure 7.3. Les valeurs des dérivées des y_i au point initial sont utilisées pour estimer les y_i au centre du pas d'intégration, puis les valeurs de ces dérivées sont utilisées pour faire avancer la solution jusqu'au point x_{n+1}.

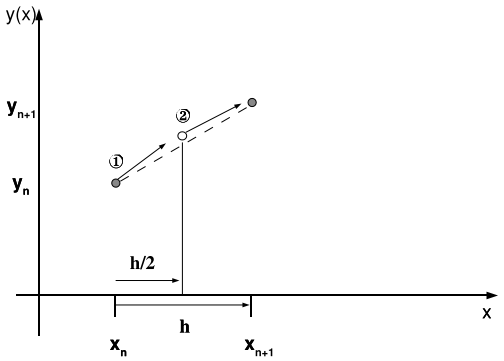

FIG. 7.3 – Méthode de Runge Kutta d'ordre 2. Dans cette méthode d'intégration d'une équation différentielle, on obtient une précision au deuxième ordre en utilisant la dérivée au point de départ x_n pour trouver un point intermédiaire. On utilise ensuite la dérivée en ce point pour trouver la valeur suivante de la fonction.

On calcule la dérivée de $y(x)$ au point x_n

$$k_1 = \left(\frac{dy}{dx}\right)_{x=x_n} = h.f(x_n, y_n)$$

puis on calcule la dérivée au point $x_n + h/2$

$$k_2 = \left(\frac{dy}{dx}\right)_{x=x_n+\frac{h}{2}} = h.f(x_n + \frac{h}{2}, y_n + \frac{k_1}{2})$$

on écrit alors que

$$y_{n+1} = y_n + k_2 + O(h^3)$$

c'est une méthode du deuxième ordre, l'erreur correspond donc au terme suivant d'ordre 3.

On voit que cette méthode requiert deux évaluations de $f(x)$ à chaque pas d'intégration. Contrairement à la méthode d'Euler, cette méthode est symétrique, c.a.d que l'intégration de l'équation différentielle de $x_0 \to x_n$ ou de $x_n \to x_0$ conduira au même résultat.

7.3.1 Stabilité de la méthode

On exprime comme précédemment l'écart à la vrai solution

$$\delta y_{n+1} = \delta y_n \left[1 - h\frac{\partial f}{\partial y} + \frac{1}{2}\left(h\frac{\partial f}{\partial y}\right)^2\right]$$

ce qui conduit en posant que le terme entre crochets est inférieur à 1 aux conditions de stabilité suivantes

décroissance	croissance	oscillation
$h \leq 2/\alpha$	instable	$1 + \frac{1}{4}(h\omega)^4 \leq 1$

nous avons ici combiné les critères de stabilité des méthodes d'Euler et de la grenouille au sein d'une seule méthode d'ordre 2. Dans le cas d'une équation oscillante, on peut considérer la méthode comme instable, mais en pratique si $h\omega < 1$ les erreurs générées sont négligeables devant le vrai résultat, et la méthode est utilisable.

Exemple d'application

Reprenons l'exemple présenté pour la méthode d'Euler,

$$\frac{dN}{dt} + k.N = 0 \Leftrightarrow \frac{dN}{dt} = -k.N = f(t)$$

Nous allons résoudre numériquement cette équation avec $k = 100s^{-1}$ en utilisant la méthode RK2.

Nous avons montré précédemment que cette méthode était stable si le pas d'intégration h (ici un temps) respectait le critère $h \leq 2/\alpha$, dans ce cas précis $\alpha = k$. Il est donc nécessaire dans ce cas de choisir un pas $h \leq 2.10^{-2}$s pour satisfaire le critère de stabilité. La figure 7.4

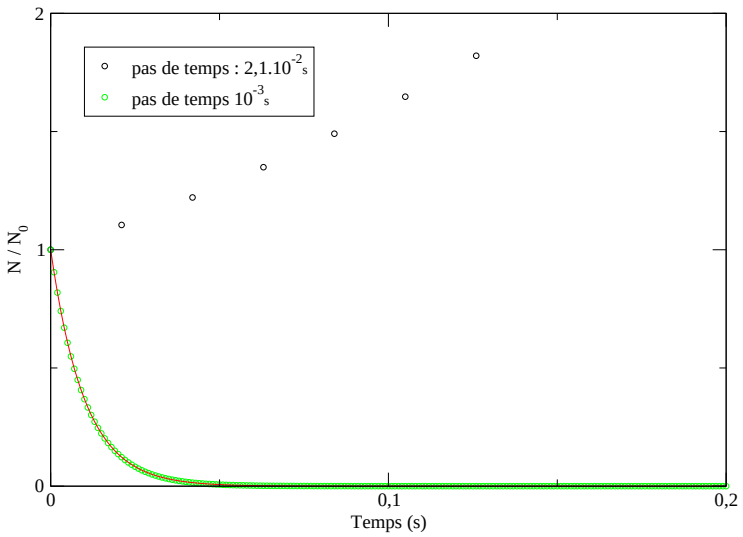

FIG. 7.4 – Application de la méthode Runge Kutta d'ordre 2 : simulation de l'évolution temporelle du nombre de noyaux d'une source radioactive (constante de décroissance radioactive 100s^{-1}. Le critère de stabilité de la méthode impose un pas d'intégration inférieur à 0.02 s. Lorsque ce critère n'est pas respecté, le résultat diverge par rapport à la solution exacte tracée en trait plein.

représente les résultats de la simulation pour des pas d'intégration inférieurs ou supérieurs à ce critère, ainsi que la solution exacte en trait plein.

A nouveau,un simple examen du graphe permet de comprendre qu'il est important de choisir un pas d'intégration satisfaisant le critère de stabilité de la méthode, sous peine de voir le calcul diverger rapidement. On notera cependant que le comportement est différent de celui observé avec la méthode d'Euler.

7.4 Méthode Runge Kutta à l'ordre 4

Dans la méthode RK2, on utilise l'estimation des y_i au demi-pas d'intégration pour le calcul des dérivées. La méthode de Runge-Kutta d'ordre 4 tente de fournir une meilleure estimation à l'aide d'une moyenne de 4 estimations, ce principe est schématisé sur la figure 7.5.

La première estimation est celle d'Euler : on fait un demi-pas de longueur $h/2$ par la méthode d'Euler pour évaluer les valeurs y_i au demi-pas. Cela permet de calculer de nouvelles valeurs des dérivées. On revient ensuite au point d'origine et l'on utilise ces dérivées pour refaire un demi-pas suivant la méthode d'Euler et ainsi obtenir de nouvelles valeurs des y_i : on ne trouvera pas les mêmes valeurs qu'à la première étape puisque l'on n'utilise pas les mêmes valeurs des dérivées (n'oublions pas qu'il ne s'agit que d'*estimations*). Cela permet d'obtenir une troisième estimation des dérivées. Cette estimation est utilisée pour faire un pas h complet suivant la méthode d'Euler, toujours à partir du même point de départ, cela conduit à une quatrième

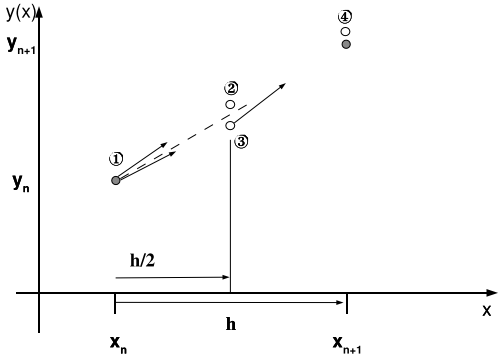

FIG. 7.5 – Méthode Runge Kutta d'ordre 4. Dans cette méthode d'intégration d'une équation différentielle, on obtient une précision au quatrième ordre en évaluant 4 fois la dérivée : une fois au point de départ, deux fois au milieu de l'intervalle, et une fois en un point final estimé.

estimation des dérivées des y_i. Il ne reste plus alors qu'à faire une moyenne pondérée de ces quatres estimations en affectant un poids double aux deux estimations intermédiaires, et faire un pas complet de longueur h pour faire avancer la solution jusqu'au point x_{n+1}. L'algorithme 7 propose une implémentation de cette méthode.

Cette méthode requiert 4 évaluations de $f(x, y)$ par pas d'intégration h : une au point de départ de l'intervalle, 2 au demi-intervalle, et une à la fin de l'intervalle :

$$\begin{aligned} k_1 &= h.f()x_n, y_n) \\ k_2 &= h.f(x_n + \frac{h}{2}, y_n + \frac{k1}{2}) \\ k_3 &= h.f(x_n + \frac{h}{2}), y_n + \frac{k2}{2}) \\ k_4 &= h.f(x_n + h, y_n + k_3) \end{aligned}$$

et l'on calcule alors la valeur de y au point x_{n+1} en sommant ces quatres termes

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)$$

le terme d'erreur est alors d'ordre 5. Cette méthode peut paraitre ressembler à un bricolage, mais c'est une des méthodes les plus utilisées en physique : elle est en effet stable et efficace dans beaucoup de cas. Pour des raisons de lourdeur des calculs, nous ne détaillerons pas les critères de stabilité de cette méthode, qui est cependant la plus robuste de toute celles que nous venons d'aborder.

Algorithme 7 Résolution d'un système d'équations différentielles du premier ordre par la méthode Runge-Kutta d'ordre 4

Pré-requis : un pas d'intégration h, les valeurs n valeurs y_i du début d'intervalle x dans un tableau, une fonction deriv permettant le calcul des dérivées

Fournit : les valeurs f_i au point $x + h$

$dh \leftarrow h/2$

$d_1 \leftarrow \text{deriv}(x, y[], n)$

pour $i = 1$ à n **faire**

$y_{temp}[i] = y[i] + d_1[i] * dh$

fin pour

$d_2 \leftarrow \text{deriv}(x + dh, y[], n)$

pour $i = 1$ à n **faire**

$y_{temp}[i] = y[i] + d_2[i] * dh$

fin pour

$d_3 \leftarrow \text{deriv}(x + dh, y[], n)$

pour $i = 1$ à n **faire**

$y_{temp}[i] = y[i] + d_3[i] * dh$

fin pour

$d_4 \leftarrow \text{deriv}(x, y[], n)$

pour $i = 1$ à n **faire**

$y[i] \leftarrow y[i] + h\left(\frac{d_1[i]}{6} + \frac{d_2[i]}{3} + \frac{d_3[i]}{3} + \frac{d_4[i]}{6}\right)$

fin pour

renvoyer y

Annexe A

Références bibliographiques et informatiques

La littérature couvrant ce domaine est importante, je ne citerai ici que quelques références qui m'ont servi à la réalisation de ce document. Libre à vous d'en utiliser d'autres.

1. Numerical Recipes in C, second edition, Cambridge University Press, 1992
   W.H. Press, S.A. Teukolski, W.T. Vetterling, B.P. Flannery
   La bible pour l'utilisateur ! Le livre peut être téléchargé au format PDF à l'adresse http ://lib-www..lanl.gov/numerical/bookcpdf.html
2. Physique numérique, Ed. Vuibert, 1998
   P. Depondt
   Malheureusement épuisé
3. Méthodes de calcul numérique, Ed. Hermès, 2001
   J.P. Nougier

Vous trouverez ici quelques références d'outils informatiques libres. Cela signifie que ces outils sont gratuits, et que vous disposez de leurs codes sources. Libre à vous de les utiliser, de lire les sources pour les comprendre . . .Ces outils développés sur platte-forme UNIX, fonctionnent aussi sous Windows (sauf Linux bien sûr).

1. Un système d'exploitation **stable**, fonctionnant sur architecture PC : Linux. Voir par exemple le site de la distribution Mandrake
   http ://www.linux-mandrake.com
   ou
   http ://www.linux.org
2. La référence des compilateurs C : gcc, il compile aussi le C++, l'ObjC . . .
   http ://www.gnu.org/software/gcc
3. Besoin d'une librairie de calcul numérique ? Pensez à la GNU Scientific Library
   http ://www.gnu.org/software/gsl
4. Un logiciel de tracé de courbe très complet : Gnuplot, il fonctionne même sous Windows
   http ://www.gnuplot.info

Annexe B

Le header float.h de la librairie C standard

La lecture des fichiers .h des librairies sont souvent instructifs. Dans float.h, on trouve notamment le codage des nombres à virgule flottante `float` et `double`, ainsi que leur dynamique de représentation (valeurs minimales et maximales), leurs précisions ...

```
/*
 * Copyright (c) 1989 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 * This product includes software developed by the University of
 * California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```

```
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * from: @(#)float.h 7.1 (Berkeley) 5/8/90
 * $FreeBSD: src/sys/i386/include/float.h,v 1.8 1999/08/28 00:44:11 peter Exp $
 */

#ifndef _MACHINE_FLOAT_H_
#define _MACHINE_FLOAT_H_ 1

#define FLT_RADIX 2 /* b */
#define FLT_ROUNDS 1 /* FP addition rounds to nearest */

#define FLT_MANT_DIG 24 /* p */
#define FLT_EPSILON 1.19209290E-07F /* b**(1-p) */
#define FLT_DIG 6 /* floor((p-1)*log10(b))+(b == 10) */
#define FLT_MIN_EXP (-125) /* emin */
#define FLT_MIN 1.17549435E-38F /* b**(emin-1) */
#define FLT_MIN_10_EXP (-37) /* ceil(log10(b**(emin-1))) */
#define FLT_MAX_EXP 128 /* emax */
#define FLT_MAX 3.40282347E+38F /* (1-b**(-p))*b**emax */
#define FLT_MAX_10_EXP 38 /* floor(log10((1-b**(-p))*b**emax)) */

#define DBL_MANT_DIG 53
#define DBL_EPSILON 2.2204460492503131E-16
#define DBL_DIG 15
#define DBL_MIN_EXP (-1021)
#define DBL_MIN 2.2250738585072014E-308
#define DBL_MIN_10_EXP (-307)
#define DBL_MAX_EXP 1024
#define DBL_MAX 1.7976931348623157E+308
#define DBL_MAX_10_EXP 308

#define LDBL_MANT_DIG DBL_MANT_DIG
#define LDBL_EPSILON DBL_EPSILON
#define LDBL_DIG DBL_DIG
#define LDBL_MIN_EXP DBL_MIN_EXP
#define LDBL_MIN DBL_MIN
#define LDBL_MIN_10_EXP DBL_MIN_10_EXP
#define LDBL_MAX_EXP DBL_MAX_EXP
```

```
#define LDBL_MAX DBL_MAX
#define LDBL_MAX_10_EXP DBL_MAX_10_EXP
#endif /* _MACHINE_FLOAT_H_ */
```

Liste des Algorithmes