Analyse Numérique

Michael Tolley

Décembre 2010

Chapitre 1

Introduction

1.1 Vocabulaire

1.1.1 Algorithme

La description précise des opérations successives à effectuer pour obtenir un certain résultat s'appelle un algorithme. Un algorithme peut donc être assimilé à un protocole ou une recette. Le mot, d'origine arabe, vient du nom Al Khuwarizmi d'un mathématicien perse qui vécut au IXème siècle. Les premiers algorithmes remontent à Euclide et même aux Babyloniens.

1.1.2 Calcul numérique

On appellera calcul numérique tout calcul (évaluation d'un nombre, d'une fonction, d'une matrice,...) qui est effectué au moyen d'une machine et/ou de tables de valeurs numériques.

1.1.3 Analyse numérique

On peut définir l'analyse numérique comme l'étude théorique des méthodes constructives de l'algèbre et de l'analyse mathématique. Par méthode constructive il faut comprendre une méthode qui fournit le moyen d'obtenir la solution d'un problème, ou, à tout le moins, une approximation de celle-ci. Ainsi, un théorème qui établit l'existence et l'unicité de la solution d'un problème de Cauchy (cf. cours d'analyse) ne sera d'aucune utilité au numéricien s'il n'indique pas comment construire effectivement la solution du problème.

1.1.4 Exemples d'applications

Le calcul numérique est devenu indispensable dans de très nombreux domaines. La liste des disciplines qui ont recours au calcul et à l'analyse numériques est bien trop longue pour être dressée ici. Elle va de la physique à la démographie en passant par la chimie, la biologie, la médecine, le génie civil, l'archéologie, la gestion des stocks, la psychologie expérimentale, les jeux vidéos, etc.

Les sciences de l'ingénieur sont probablement celles qui font le plus appel au calcul numérique intensif. Les moyens informatiques modernes et les performances extraordinaires des microprocesseurs et des mémoires actuels ont permis un développement considérable des capacités de calcul des ordinateurs. A côté de ces matériels très performants, les chercheurs disposent de logiciels très élaborés (développés et mis au point par d'autres chercheurs). Il est devenu possible, par exemple, (ce n'était pas le cas il y a à peine dix ans) de simuler numériquement des « crash-tests » incluant à la fois les déformations des véhicules et les mouvements et chocs subis par les passagers (virtuels heureusement !). De telles simulations ont permis des avancées importantes en matière de sécurité automobile passive et active.

Un autre domaine qui a bénéficié des progrès du matériel et des logiciels informatiques est la météorologie. On peut, actuellement, lorsque les conditions de stabilité sont bonnes, prévoir l'évolution du temps sur une période de cinq à six jours. Ces prévisions sont effectuées dans des centres de calculs spécialisés (p.ex. Reading en Angleterre) et nécessitent de résoudre des millions de fois des systèmes de plusieurs millions d'équations à autant d'inconnues. Les ordinateurs utilisés dans ces centres peuvent effectuer plusieurs centaines de milliards d'opérations arithmétiques par seconde.

Une autre discipline dont le développement spectaculaire est lié au progrès des calculateurs numériques et de la modélisation mathématique est la bio-informatique. Certaines études, dans ce domaine, menées notamment dans notre faculté, s'intéressent au modifications conformationnelles de protéines qui sont à l'origine de maladies diverses telles la maladie d'Alzheimer et les encéphalopathies spongiformes. Ces études devraient permettre de mieux comprendre l'origine de ces maladies et par là, contribuer à les juguler.

1.2 Arithmétique entière ou en virgule flottante

Dans la vie courante, les opérations arithmétiques que nous effectuons sont la plupart du temps exactes. Pensons par exemple à la note que nous payons au bistrot, au supermarché ou au guichet d'une salle de spectacle. Nous sommes habitués à manipuler des nombres entiers et des réels ayant peu de chiffres décimaux. Généralement on se limite à faire des additions et à calculer des pourcentages et il est donc facile de faire du calcul exact.

A y réfléchir de plus près, on se rend compte qu'on est souvent (inconsciemment ?) confronté à des problèmes de calcul approché : hier (19/09/2008), j'ai fait le plein de diesel routier à 1.009 euro le litre, ce qui m'a coûté 42.09 euro, alors que mon réservoir a accepté 41.71 litres de carburant. J'aurais dû, en réalité, débourser 42.08539 euro... Mais, ai-je bien reçu la quantité affichée au compteur de la pompe ? C'est peu probable. La mesure du volume fourni est évidemment entachée d'une certaine erreur (qui varie d'ailleurs en fonction de la température ambiante, de la pression atmosphérique et du débit de fluide) et j'ai peut-être payé mon carburant à 1.024 euro ou 0.994 euro le litre !

Cet exemple de la vie courante permet de mettre en évidence deux sources d'erreurs différentes. L'une est directement liée à la manière d'exprimer les nombres : on utilise deux chiffres décimaux pour estimer le volume de carburant fourni, trois pour le prix au litre et deux pour le prix total à payer. L'autre source d'erreur est directement liée

aux données qui peuvent elles-mêmes n'être qu'approchées comme c'est le cas ici pour le volume de diesel effectivement délivré par la pompe.

1.2.1 Arithmétique entière

Lorsqu'on fait des calculs qui n'utilisent que des nombres entiers, si toutes les données sont connues exactement, les résultats obtenus seront exacts également (NB : on exclut les opérations dont le résultat ne serait pas un entier). Toutefois, nos calculatrices ne peuvent afficher et manipuler qu'un nombre limité d'entiers. L'ensemble des entiers sur lesquels nous pouvons opérer est donc borné. Si le résultat d'un calcul est, par exemple 7435900897765123145322 il ne sera pas accessible à la majorité des calculettes ni à de nombreux ordinateurs.

Conclusion : les calculs seront exacts si les données le sont, pour autant que les données et les résultats soient représentables, le domaine des nombres représentables étant borné.

1.2.2 Arithmétique en virgule flottante

Un nombre réel peut être écrit sous la forme standardisée [1]

$X = \pm.a_1 a_2 \dots a_m b^M$.

Il est constitué d'un signe $+$ *ou* $-$, du point décimal ., d'une mantisse $a_1 a_2 \dots a_m$, d'une base (b) et d'un exposant (M), soumis à certaines contraintes :

a) la base b doit être un entier ≥ 2
b) les a_i doivent être des entiers tels que $0 \leq a_i < b$
c) l'exposant M doit respecter les conditions $M_{min} \leq M \leq M_{max}$

Un nombre réel non nul est dit normalisé si $a_1 \neq 0$.

$X = -.0231 \times 10^2$ n'est pas normalisé, mais, $X = -.2310 \times 10^1$ l'est.

Le nombre de réels représentables est limité et vaut, pour m, b, M_{max} et M_{min} fixés :

$2(b-1)b^{(m-1)}(M_{max} - M_{min+1}) + 1$

Le nombre de réels normalisés représentables est $2(b-1)b^{(m-1)}(M_{max} - M_{min+1}) + 2$ car deux représentation de zéro sont prises en compte $+.0000...$ et $-.0000...$

Exemple 1

$b = 10, \quad m = 3, \quad M_{min} = -15, \quad M_{max} = 16$

$x = +.125 \times 10^6, \quad y = -.128 \times 10^6, \quad z = +.437 \times 10^{12}, \quad u = +.215 \times 10^{-10}$

$$\begin{aligned} x + y &= (+.125 \times 10^6) + (-.128 \times 10^6) \\ &= (+.125 - .128) \times 10^6 \\ &= -.003 \times 10^6 \\ &= -.300 \times 10^4 (\text{normalisation}) \end{aligned}$$

1. Certains exemples et définitions de cette sous-section sont empruntés à [1]

$$\begin{aligned} x + z &= (+.125 \times 10^6) + (+.437 \times 10^{12}) \\ &= (+.000000125 + .437) \times 10^{12} \quad \text{(alignement)} \\ &= +.437000125 \times 10^{12} = +.437 \times 10^{12} \quad \text{(standardisation)} \end{aligned}$$

Conclusion : $x + z = z$, avec $x \neq 0$! ! !

$$\begin{aligned} x \times y &= (+.125 \times 10^6) \times (-.128 \times 10^6) \\ &= (+.125) \times (-.128) \times 10^{6+6} \\ &= -.016000 \times 10^{12} = -.160 \times 10^{11} \quad \text{(normalisation)} \end{aligned}$$

$$\begin{aligned} x \times z =& (+.125 \times 10^6) \times (+.437 \times 10^{12}) \\ &= (+.125) \times (.437) \times 10^{6+12} \\ &= +.054625 \times 10^{18} = +.546 \times 10^{17} \quad \text{(normalisation)} \end{aligned}$$

Conclusion : l'exposant est supérieur à M_{max} et le produit $x \times z$ est donc supérieur au plus grand nombre représentable ($+.999 \times 10^{16}$) on dit qu'il y a surdépassement. Ceci se traduira par un message d'erreur affiché par le calculateur indiquant qu'il y a *overflow*.

$$\begin{aligned} x \div z &= (+.125 \times 10^6) \div (+.437 \times 10^{12}) \\ &= (+.125) \div (+.437) \times 10^{6-12} \\ &= +.286041189... \times 10^{-6} = +.286 \times 10^{-6} \quad \text{(normalisation)} \end{aligned}$$

$$\begin{aligned} u \div z &= (+.215 \times 10^{-10}) \div (+.437 \times 10^{12}) \\ &= (+.215) \div (+.437) \times 10^{-10-12} \\ &= +.491990846... \times 10^{-22} = +.491 \times 10^{-22} \quad \text{(normalisation)} \end{aligned}$$

Conclusion : l'exposant est inférieur à M_{min} et le quotient $u \div z$ est donc plus petit que le plus petit nombre positif représentable ($+.100 \times 10^{-15}$) ceci se traduira par un message d'erreur affiché par le calculateur indiquant qu'il y a *underflow*.

Dans pareil cas, selon les logiciels et matériels utilisés, le résultat est remplacé par zéro ou par $\epsilon(0)$. « L'$\epsilon(0)$ machine » dépend du calculateur et du logiciel employés et correspond au plus petit nombre distinguable de zéro (dans la version 7.0.1 de Matlab, implémentée sur PC portable, l'$\epsilon(0)$ machine vaut habituellement $4.940656458412465 \times 10^{-324}$) [2].

Exemple 2

Soient $b = 10$, $m = 3$, $x = y = +.400 \times 10^0$ et $z = +.100 \times 10^3$

On a :

$y + z = +.100 \times 10^3, \Rightarrow x + (y + z) = +.100 \times 10^3 = z$

$x + y = +.800 \times 10^0, \Rightarrow (x + y) + z = .101 \times 10^3 \neq x + (y + z)$.

Conclusion : l'ordre dans lequel sont effectuées les opérations peut avoir une incidence sur le résultat final. Lorsqu'on additionne des nombres positifs, il faut commencer par les plus petits (quand on peut... !).

2. Le plus petit nombre distinguable de 1, $\epsilon(1)$ vaut $2.220446049250313 \times 10^{-16}$

Exemple 3

La moyenne μ et la variance σ de n nombres s'obtiennent à l'aide des formules suivantes :

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1.1}$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \mu^2 \tag{1.2}$$

Si l'on choisit $b = 10, m = 3$ et $n = 2$, les résultats obtenus pour la variance des deux nombres $x_1 = +.310 \times 10^2$ et $x_2 = +.320 \times 10^2$, sont $\sigma^2 = +.250 \times 10^0$ ou $\sigma^2 = -.200 \times 10^1$ selon la formule utilisée.

Conclusion : deux formules théoriquement équivalentes peuvent donner des résultats numériques très différents !

L'erreur est due à la soustraction de deux nombres à peu près égaux. On parle de cancellation [3] ou d'annulation numérique. Elle se traduit par la perte de chiffres significatifs et peut conduire, comme dans cet exemple, à des résultats aberrants : $\sigma^2 < 0$!!

Exemple 4

$A = +.4025 \times 10^1, B = +.4019 \times 10^1 \Rightarrow A - B = C = +.6000 \times 10^{-2}$

Questions : les zéros sont-ils significatifs ? Le six est-il significatif ?

Si A et B sont des valeurs approchées et qu'on a en réalité :

$A = +.4024786 \times 10^1$ et $B = +.4019432 \times 10^1$ on obtient alors $C = +.5354 \times 10^{-2}$

Conclusion : la soustraction de deux nombres proches peut se traduire par une perte de chiffres significatifs, c'est la cancellation.

Exemple 5

Les racines du trinôme du second degré $ax^2 + bx + c$ sont données par les formules équivalentes :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}} \tag{1.3}$$

$$\text{Si} \quad b^2 \gg ac \Rightarrow x = \frac{-b \pm |b|}{2a} \quad \text{et} \quad x = \frac{2c}{-b \mp |b|} \tag{1.4}$$

Conclusion : on obtient alors une racine «nulle» et l'autre «infinie» ce qui est évidemment aberrant et résulte à nouveau d'une cancellation.

3. Il ne s'agit pas d'un angliscisme. L'origine du mot est francophone mais il nous est revenu via un détour par l'anglais...

Exemple 6

$\sum_{n=1}^{\infty} \frac{1}{n}$ est une série divergente mais $\lim_{n\to\infty} \frac{1}{n} \to 0$.

En base 10, le calcul avec un seul chiffre, $(m = 1)$ donne une somme qui vaut 2. Avec deux chiffres $(m = 2)$ on obtient 3.9

Conclusion : une série divergente peut sembler converger lorsqu'on calcule numériquement la somme de ses termes. Dans l'exemple précédent, en travaillant avec un seul chiffre la série semble converger vers 2. Toutefois, si on augmente le nombre de chiffres des mantisses utilisées, on obtient des sommes différentes. Tout se passe donc comme si la série admettait plusieurs limites distinctes... Contrairement aux apparences il n'y a donc pas convergence.

1.2.3 Erreur absolue et erreur relative

Soit r la valeur exacte d'un nombre réel et x une approximation de ce nombre.

Définition

$r - x$ est l'erreur absolue commise sur r

Définition

$(r - x)/r$ est l'erreur relative commise sur r

Exemple 7

$x = 22/7$ est une approximation de π. L'erreur absolue vaut : $\pi - x = -.001264...$ et l'erreur relative : $(\pi - x)/\pi = -.00040...$

Intuitivement, $22/7$ est précis à trois chiffres, puisque la quatrième position est erronée :

$22/7 = 3.142857142...$; $\pi = 3.141592653...$

On dira que $22/7$ donne π avec une précision de trois chiffres ou avec trois chiffres exacts.

Définition

Lorsqu'on travaille dans le système décimal l'approximation x de r est précise (exacte) à

$$N(x, r) = -log_{10}|\frac{r - x}{r}| \quad \text{chiffres.} \tag{1.5}$$

On représente ce nombre de chiffres précis (exacts) par $N(x, r)$.

Valeurs approchées

Soit le réel $r = 421.57892 = .42157892 \times 10^3$

$x1 = 421.57$ est une valeur de r par défaut

$x2 = 421.58$ est une valeur de r par excès

$x3 = 421.57$ est une valeur tronquée de r.

D'habitude, on arrondit les nombres «au plus près».

Propagation des erreurs

Soient deux nombres a et u (valeurs exactes) et leurs approximations $\hat{a}$ et $\hat{u}$.

Les erreurs absolues commises en remplaçant a et u par leurs approximations sont : $e_a = a - \hat{a}$ et $e_u = u - \hat{u}$.

L'erreur absolue sur la somme $a + u$ est la somme des erreurs absolues sur a et u :

$(a + u) - (\hat{a} + \hat{u}) = e_a + e_u$

L'erreur absolue commise sur le produit $a.u$ vaut :

$(a.u) - (\hat{a}.\hat{u}) = \hat{a}.e_u + \hat{u}.e_a + e_a.e_u$

Conclusion : lorsqu'on effectue un produit de deux nombres il y a un risque d'amplification des erreurs absolues. Le risque est d'autant plus élevé que les nombres sont grands.

Conséquences des erreurs d'arrondi et de leur propagation

Voir, par exemple, le site `http://www.ima.umn.edu/~arnold/disasters`

1.2.4 Processus itératif - Récurrence

Itérations d'un réel

$x_0 = a$ valeur de départ ou initiale.
$x_n = \Phi(x_{n-1})$ $n = 1, 2, 3, \ldots$ suite récurrente de réels.

Itérations d'un vecteur

$V_0 = a$ vecteur de départ ou initial.
$V_n = MV_{n-1}$ $n = 1, 2, 3, \ldots$ suite récurrente de vecteurs. [4]

Itérations d'une matrice

$A_0 = M$ matrice de départ ou initiale.
$A_n = TA_{n-1}$ $n = 1, 2, 3, \ldots$ suite récurrente de matrices.

4. Le produit matriciel doit évidemment être licite.

Itérations de fonctions - Récurrence à trois termes

$T_0(x) = 1$ et $T_1(x) = x$ fonctions de départ (2 précurseurs).
$T_{n+1}(x) = 2x.T_n - T_{n-1}(x)$ $n = 1, 2, 3, \ldots$ suite récurrente de fonctions [5].

1.2.5 Stabilité d'un processus itératif

Si au cours des calculs le processus itératif se fait sans amplification des erreurs on le qualifiera de stable. A une faible erreur initiale correspndra une faible erreur finale.
S'il y a propagation des erreurs avec amplification on dira que le processus, ou la méthode itérative, est instable. A une faible erreur initiale correspondra une forte erreur finale.

Exemple 8

Générer la suite $1/3^n$ peut se faire de multiples façons, par exemple [6] :

a) Poser $r_0 = 1$ et faire $r_n = r_{n-1}/3$ pour $n = 1, 2, 3, \ldots$

b) Poser $p_0 = 1$ et $p_1 = 1/3$ puis faire $p_n = (4p_{n-1} - p_{n-2})/3$ pour $n = 2, 3, 4, \ldots$

c) Poser $q_0 = 1$ et $q_1 = 1/3$ puis faire $q_n = (10q_{n-1} - 3q_{n-2})/3$ pour $n = 2, 3, 4, \ldots$

On obtient les résultats suivants :

n	a)	b)	c)
1	1.00000000000000	1.00000000000000	1.00000000000000
2	0.33333333333333	0.33333333333333	0.33333333333333
3	0.11111111111111	0.11111111111111	0.11111111111111
4	0.03703703703704	0.03703703703704	0.03703703703704
5	0.01234567901235	0.01234567901235	0.01234567901234
6	0.00411522633745	0.00411522633745	0.00411522633745
7	0.00137174211248	0.00137174211248	0.00137174211247
8	0.00045724737083	0.00045724737083	0.00045724737080
9	0.00015241579028	0.00015241579028	0.00015241579019
10	0.00005080526343	0.00005080526343	0.00005080526316
11	0.00001693508781	0.00001693508781	0.00001693508701
12	0.00000564502927	0.00000564502927	0.00000564502687
13	0.00000188167642	0.00000188167642	0.00000188166924
14	0.00000062722547	0.00000062722547	0.00000062720392
15	0.00000020907516	0.00000020907516	0.00000020901050
16	0.00000006969172	0.00000006969172	0.00000006949774
17	0.00000002323057	0.00000002323057	0.00000002264865
18	0.00000000774352	0.00000000774352	0.00000000599775
19	0.00000000258117	0.00000000258117	-0.00000000265616
20	0.00000000086039	0.00000000086039	-0.00000001485161

5. Il s'agit de la suite des polynômes de Chebychev
6. Cet exemple est emprunté à [2]

21	0.00000000028680	0.00000000028680	-0.00000004684920
22	0.00000000009560	0.00000000009560	-0.00000014131239
23	0.00000000003187	0.00000000003187	-0.00000042419211
24	0.00000000001062	0.00000000001062	-0.00000127266131
25	0.00000000000354	0.00000000000354	-0.00000381801225

Si on perturbe les données de départ en posant

$r_0 = .99996, p_1 = q_1 = .33332$

Voici les résultats qu'on obtient :

n	a)	b)	c)
1	0.99996000000000	1.00000000000000	1.00000000000000
2	0.33332000000000	0.33332000000000	0.33332000000000
3	0.11110666666667	0.11109333333333	0.11106666666667
4	0.03703555555556	0.03701777777778	0.03690222222222
5	0.01234518518519	0.01232592592593	0.01194074074074
6	0.00411506172840	0.00409530864198	0.00290024691358
7	0.00137168724280	0.00135176954733	-0.00227325102880
8	0.00045722908093	0.00043725651578	-0.01047775034292
9	0.00015240969364	0.00013241883859	-0.03265258344760
10	0.00005080323121	0.00003080627953	-0.09836419448242
11	0.00001693441040	-0.00000306457349	-0.29522806482714
12	0.00000564480347	-0.00001435485783	-0.88572935494139
13	0.00000188160116	-0.00001811828594	-2.65720311831082
14	0.00000062720039	-0.00001937276198	-7.97161437276133
15	0.00000020906680	-0.00001979092066	-23.9148447908936
16	0.00000006968893	-0.00001993030689	-71.7445349302174
17	0.00000002322964	-0.00001997676896	-215.233604976497
18	0.00000000774321	-0.00001999225632	-645.700814991441
19	0.00000000258107	-0.00001999741877	-1937.10244499497
20	0.00000000086036	-0.00001999913959	-5811.30733499180
21	0.00000000028679	-0.00001999971320	-17433.9220049777
22	0.00000000009560	-0.00001999990440	-52301.7660149339
23	0.00000000003187	-0.00001999996813	-156905.298044802
24	0.00000000001062	-0.00001999998938	-470715.894134406
25	0.00000000000354	-0.00001999999646	-1412147.68240321

Exemple 9

Drôle de deal...

Un banquier immaginaire propose à ses clients de réaliser un placement de longue durée particulièrement rentable. Le capital de départ est fixe et vaut (en unités arbitraires) : $C_0 = e - 1 = 2.718281828 \cdots - 1, = 1.718281828....$

L'évolution annuelle du capital se fait selon la loi : $C_n = n.C_{n-1} - 1$ pour $n = 1, 2, \ldots$

Question : à combien s'élèvera le capital après 25 ans ?

Réponse n°1 : $-140.302.545.600.000$ (perte de l'ordre de 1.410^{15})
Calcul effectué sur une calculette Casio FX102P.

Réponse n°2 : $+4.645.987.753$ (gain de l'ordre de 4.610^{10})
Calcul effectué sur une station de travail SUN Sparc.

Réponse n°3 : ça dépend de la représentation des réels utilisée pour effectuer les calculs...

La fonction qui suit permet de le montrer

```
function Cap=bizbank(N,Y)
%  N : nombre de chiffres de la mantisse
%  Y : durée en années
digits(N)
ee=vpa(exp(1)); a=ee-1; % vpa : variable precision arithmetics
for n=1:Y
a=a*n-1; b(n)=a;
end
Cap=b(Y);
```

Voici les résultats obtenus, en base 10, selon le nombre N de chiffres de mantisse retenu :

```
 N    Capital

 1   .2e25
 2  -.30e24
 3   .268e23
 4  -.4375e22
 5   .28188e21
 6  -.283618e20
 7   .2660808e19
 8  -.44143420e18
 9   .239020730e17
10  -.7120347065e16
11   .63525795660e15
12   .148095548685e14
13  -701655174804.2
14  -701655174804.68
15    73905327361.8675
16    11860487188.54402
17     4104882166.8785182
18     4570218468.17844785
19     4647774518.395102780
20     4646223397.3907696828
21     4645913173.18990306312
```

```
22      4645990729.240119718052
23      4645987626.9981110518545
24      4645987782.11021148516450
25      4645987751.087791398502530
26      4645987752.6389124028356298
27      4645987753.41447290500217898
28      4645987753.383450484915516998
29      4645987753.3912060899371824905
30      4645987753.39105097783674918075
```

1.3 Choisir un algorithme

Lorsqu'on cherche à résoudre numériquement un problème donné, la question du choix d'un "bon algorithme" s'impose d'elle-même. Cette question est délicate et loin d'être triviale. Elle est fortement liée au problème spécifique que l'on traite et à l'expérience que l'on a acquise du calcul numérique.

Il existe actuellement de nombreux logiciels sensés résoudre la plupart des problèmes numériques classiques. Certains sont généralistes et permettent d'aborder beaucoup de sujets différents :

Résolution d'une équation non linéaire à une inconnue
Résolution d'un système algébrique linéaire à autant d'équations que d'inconnues
Résolution d'un système incompatible au sens des moindres carrés
Résolution d'un système d'équations non linéaires
Interpolation et approximation de fonctions d'une ou plusieurs variables
Evaluation de fonctions spéciales
Calcul d'intégrales d'une fonction d'une ou de plusieurs variables
Calcul des dérivées d'une fonction d'une ou de plusieurs variables
Calcul de transformées de Fourier numériques (FFT [7])
Calcul des valeurs propres et des vecteurs propres d'une matrice
Résolution de problèmes aux valeurs propres généralisés
Résolution d'une équation différentielle ordinaire du premier ordre (problème de Cauchy)
Résolution d'un système d'équations différentielles ordinaires du premier ordre
Résolution d'équations différentielles du second ordre (problème aux limites)
Résolution d'équations et de systèmes d'équations aux dérivées partielles
Etc...

D'autres sont plus spécialisés et permettent de résoudre des problèmes spécifiques en faisant appel à une méthode numérique particulière. On peut citer, parmi les méthodes classiques :

La méthode des différences finies
La méthodes des éléments finis
La méthode des éléments de frontière

7. Fast Fourier Transform

Ce sont les méthodes les plus utilisées pour résoudre des systèmes d'équations aux dérivées partielles de manière approchée.

Qu'ils soient généralistes ou spécialisés, les modes d'emploi de ces logiciels sont toujours très volumineux et il est fréquent qu'ils comptent plusieurs milliers de pages.

Le profane croit souvent que certaines méthodes sont quasiment universelles. C'est le cas par exemple de la méthode de Runge-Kutta à quatre paramètres pour la résolution de problèmes de Cauchy pour un système d'équations différentielles ordinaires du premier ordre, ou de la méthodes des différences finies pour la résolution d'équations aux dérivées partielles. Il a alors tendance à utiliser des "boîtes noires" pour résoudre les problèmes auxquels il est confrontés, faisant entière confiance aux concepteurs du logiciel. Les choses sont cependant moins prosaïques et pour mener un calcul numérique de manière efficace il est toujours utile et souvent indispensable, de comprendre :

comment fonctionne la méthode utilisée,
quel est son domaine d'application,
quelle est sa limite de validité,
quelle est la précision des résultats qu'elle fournit.

Cette compréhension, qui permettra de choisir un algorithme de résolution du problème posé en toute connaissance de cause, est un des objectifs du cours "Eléments d'analyse numérique" et passe principalement par l'expérience acquise lors des séances d'exercices.

1.4 Organisation des calculs

Lorsqu'on applique un algorithme de calcul il est bon de prendre certaines précautions et de respecter quelques principes. C'est particulièrement important si l'on est amené à répéter de très nombreuses fois les mêmes calculs ou les mêmes procédures. Il est habituel, dans les problèmes pratiques, d'avoir à résoudre des dizaines, voire des centaines de miliers de fois des systèmes d'équations comportant chacun de nombreux miliers d'équations...

1.4.1 Minimiser le nombre de calculs

Un défaut classique est l'écriture de programmes qui impliquent le calcul de grandeurs déjà connues. Voici un exemple simple où cette erreur est commise. On se propose de calculer N valeurs du polynôme en $\sin(x\times\pi/N)$ suivant : $P(x) = 4*\sin((x\times\pi)/N)^2-\sin((x\times\pi)/N)^7$.

Les cinq programmes suivants effectuent ce calcul pour $N = 100.000$ et en affichent la durée.

```
% Programme P1
tic % Déclenche le chronomètre
N=100000;
for i=1:N
    y(i)=4*sin((i*pi)/N)^2-sin((i*pi)/N)^7;
```

```
end
toc % Affiche le temps écoulé entre tic et toc


% Programme P2
tic % Déclenche le chronomètre
N=100000;
for i=1:N
    S=sin((i*pi)/N);
    y(i)=4*S^2-S^7;
end
toc % Affiche le temps écoulé entre tic et toc


% Programme P3
tic % Déclenche le chronomètre
N=100000;
for i=1:N
    S=sin((i*pi)/N);
    y(i)=S*S*(4-S^5);
end
toc % Affiche le temps écoulé entre tic et toc


% Programme P4
tic % Déclenche le chronomètre
N=100000;
for i=1:N
    S=sin((i*pi)/N);
    S2=S*S;
    S4=S2*S2;
    y(i)=S2*(4-S4*S);
end
toc % Affiche le temps écoulé entre tic et toc


% Programme P5
tic % Déclenche le chronomètre
N=1:100000;
S=sin((N*pi)/N(end));
S2=S.*S;
S4=S2.*S2;
y=S2.*(4-S4.*S);
toc % Affiche le temps écoulé entre tic et toc
```

La durée d'exécution des cinq programmes est reprise dans le tableau ci-dessous :

```
Programme        Durée (s)      Durée (s) si y(i) est remplacé par y avec N = 10^5

    P1         102.337000                          7.961000
    P2         102.288000                          6.249000
    P3         102.217000                          6.239000
    P4         102.067000                          1.892000
    P5           0.281000                          2.504000
```

Le programme le plus rapide nécessite donc un temps de calcul 364 fois moindre que celui du programme le plus lent, lorsqu'on stocke les résultats dans un vecteur y. Sans stocker les résultats (sauf pour P5...) cette réduction du temps de calcul n'est que d'environ 4.2.

1.4.2 Eviter les affichages inutiles

Reprenons le programme P1 et modifions le légèrement pour obtenir les variantes P1V1 et P1V2

```
% Programme P1V1
tic % Déclenche le chronomètre
N=10000;
for i=1:N
    y(i)=4*sin((i*pi)/N)^2-sin((i*pi)/N)^7;
end
toc % Affiche le temps écoulé entre tic et toc


% Programme P1V2
tic % Déclenche le chronomètre
N=10000;
for i=1:N
    y(i)=4*sin((i*pi)/N)^2-sin((i*pi)/N)^7
end
toc % Affiche le temps écoulé entre tic et toc
```

Les durées d'exécution respectives de ces programmes sont :

```
Programme       Durée en secondes

  P1V1              0.240000
  P1V2            931.315000
```

Le simple fait d'oublier un ; provoque l'affichage de y à chaque itération et multiplie le temps d'exécution par plus de 3880 ! (NB : ici $N = 10000$ au lieu de 100000 dans les programmes P1 à P5).

1.4.3 Contrôler l'erreur

Le programme suivant, P6, applique la méthode du point fixe (cf. page 33) pour essayer d'obtenir une solution d'une équation de la forme $x = g(x)$. Cette méthode, lorsqu'elle fonctionne, génère une suite x_n de valeurs convergeant vers un zéro de l'équation. Il est donc naturel de suivre l'évolution des itérations en testant l'écart qui sépare les valeurs successives x_{n+1} et x_n. On peut en effet s'attendre à ce que cet écart diminue lorsque n augmente. Il est donc raisonnable, lorsque le zéro r cherché est de l'ordre de l'unité, de considérer que si $|x_{n+1} - x_n| \leq 10^{-14}$ la convergence numérique est atteinte et que x_{n+1} constitue une très bonne approximation de r. Cette idée est mise en application dans le programme P7 au moyen de la boucle `while... end`.

```
% Programme P6
% La fonction dont on cherche les zéros est  y(x) = 1 - x^3/3
X(1) = 0.2;   Y(1) = 0;
X(2) = 0.3;
n = 1;
while abs(X(n+1)-X(n)) >= 1e-14
    n=n+1;
    X(n+1)  = X(n) - X(n)^3/3 + 1;
end
format long
X'
plot(X,'-o','LineWidth',1.3)
xlabel('\it n','FontSize',14)
ylabel('\it X_n','FontSize',14)
title('Tentative de résolution de \it 1 - x^3/3 = 0','FontSize',14)
```

L'exécution du programme ne fournit aucun résultat et il faut forcer son interruption car il ne s'arrête pas de lui-même. Le test de convergence qui compare deux approximations successives x_{n+1} et x_n n'est jamais vérifié.

Si on interrompt les calculs après soixante secondes, on constate que

```
n=98301, X(98301)=1.62924292865886 et X(98300)=1.18767112407051.
```

1.4.4 Limiter le nombre d'itérations

Pour éviter le problème rencontré par le programme P6, il suffit d'y inclure un test destiné à limiter le nombre d'itérations. Le programme ainsi modifié, P7, est reproduit ci-dessous.

```
% Programme P7
% La fonction dont on cherche les zéros est  y(x) = 1 - x^3/3
X(1) = 0.2;   Y(1) = 0;
X(2) = 0.3;
n = 1; Nitermax = 50;
```

```
while abs(X(n+1)-X(n)) > 1e-14
    n=n+1;
    X(n+1)  = X(n) - X(n)^3/3 + 1;
    if n > Nitermax
        break
    end
end
format long
X'
plot(X,'-o','LineWidth',1.3)
xlabel('\it n','FontSize',14)
ylabel('\it X_n','FontSize',14)
title('Tentative de résolution de \it 1 - x^3/3 = 0','FontSize',14)
```

Le programme modifié, P7, s'interrompt lorsque `n=51`.
Les derniers itérés sont `X(51)=1.18769854917539` et `X(50)=1.62923166804048`.
Le graphique produit par le programme illustre ce qui s'est passé : le processus itératif conduit à des valeurs successives `X(n)` et `X(n+1)` qui tendent respectivement vers `1.62924292865886` et `1.18767112407051`. On dit que le processus itératif cycle.

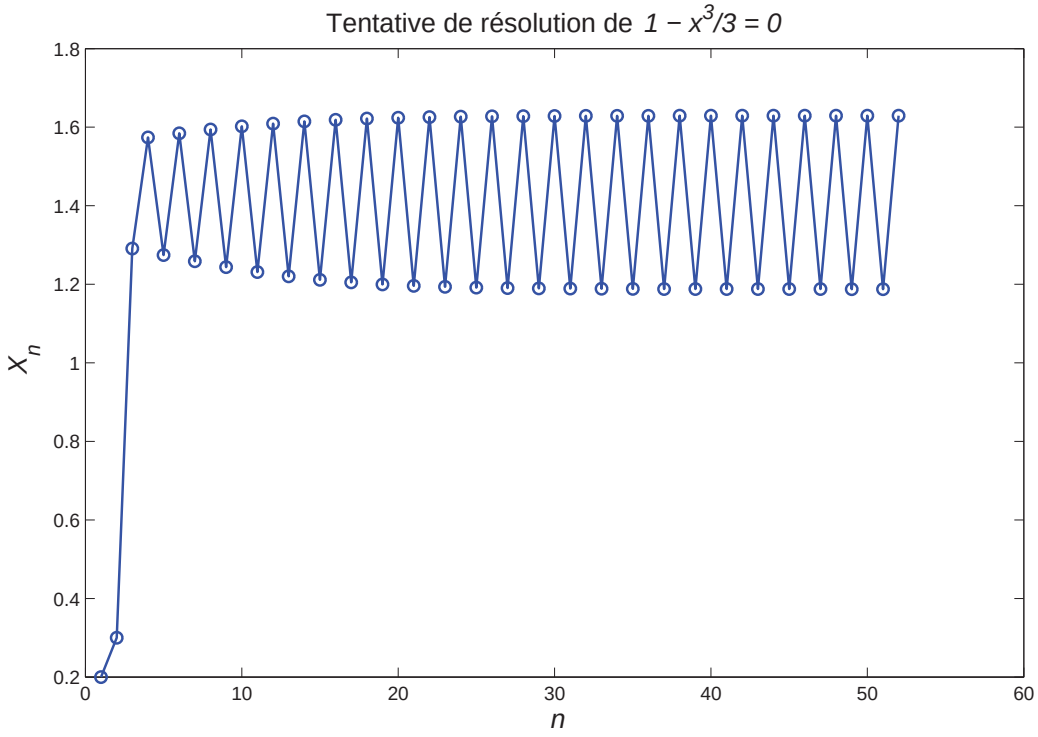


FIGURE 1.1 – La figure produite par le programme P7 illustre une situation dans laquelle la méthode du point fixe cycle .

Chapitre 2

Résolution d'une équation $F(x) = 0$

2.1 Préliminaires

Dans ce chapitre nous étudierons quelques méthodes de résolution d'équations de la forme $F(x) = 0$. D'une manière générale, x est un réel [1] et $F(.)$ une fonction réelle.

On connaît des formules permettant de calculer explicitement les racines de tout polynôme de degré < 5 mais de telles formules n'existent pas pour les polynômes de degré supérieur à 4 ni, en général, pour les équations transcendantes.

L'équation $x - 0.2\sin(x) - 0.5 = 0$ (cet exemple est emprunté à Dion et Gaudet [1]) admet une racine réelle au voisinage de 0.615.

Comme on ne dispose pas d'une formule qui permet de calculer cette racine explicitement, pour la connaître il faudra procéder numériquement.

Une bonne pratique, si l'on dispose d'un ordinateur et d'un logiciel de calcul numérique avec des moyens graphiques (c'est le cas de Matlab et d'Octave [2]), est de commencer par faire un graphe de la fonction. On peut, évidemment, aussi étudier cette fonction par les moyens analytiques classiques pour localiser son (ses) zéro(s) éventuel(s).

Voici un programme « écrit en Matlab » pour le cas de la fonction ci-dessus :

```
x = -1 :0.001 :1;           % On construit un vecteur x à 2001 composantes
f = x - 0.2*sin(x) - 0.5;   % Le vecteur f contient les 2001 images de x
plot(x,f)                   % On représente f en fonction de x
grid on                     % On place une grille sur le graphe
zoom on                     % On active le zoom
```

La fonction zoom permet de cliquer sur le graphe de manière à faire un agrandissement, on peut ainsi «aller voir» le(s) zéro(s) de plus près.

Les méthodes qui permettent de calculer des valeurs approchées des zéros d'une fonction sont toutes basées sur la notion d'itération. Elles engendrent une suite de valeurs qui, quand tout va bien, converge vers le zéro cherché.

1. Nous verrons que certaines méthodes restent valables si x est complexe.
2. Octave est un logiciel libre, quasi-clone de Matlab

Les éléments de la suite sont appelés approximations successives.

Exemple 1

Pour résoudre l'équation $F(x) = x - 0.2\sin(x) - 0.5 = 0$ on peut procéder comme suit :

On remplace l'équation donnée par une équation équivalente en écrivant par exemple,

$x = g(x)$, avec $g(x) = 0.2\sin(x) + 0.5$ ($g : \mathbb{R} \to \mathbb{R}$, on travaille avec des réels).

On se donne alors une valeur x_0 « proche » de la racine cherchée et on calcule les approximations successives engendrées par la récurrence $x_{n+1} = g(x_n),\ n = 0, 1, 2, \ldots$

Dans le cas présent, voici ce qu'on obtient pour les 20 premiers termes de la suite en prenant $x_0 = 0.6$

```
0.61292849467901
0.61505306701027
0.61540037303958
0.61545709802029
0.61546636150471
0.61546787424461
0.61546812127620
0.61546816161663
0.61546816820424
0.61546816928001
0.61546816945568
0.61546816948437
0.61546816948905
0.61546816948982
0.61546816948994
0.61546816948996
0.61546816948996
0.61546816948997
0.61546816948997
0.61546816948997
```

Ces résultats ont été obtenus à l'aide du programme que voici [3] :

```
x0 = 0.6;
for n = 1:20
    x(n) = 0.2*sin(x0) + 0.5;
    x0 = x(n);
end
format long
x'
```

3. Il est instructif d'éxécuter successivement le programme précédent et celui-ci.

Un programme équivalent au précédent serait :

```
x(1) = 0.6;
for n = 1:20
    x(n+1) = 0.2*sin(x(n)) + 0.5;
end
format long
x'
```

Dans le cas de cet exemple, nous obtenons une suite de valeurs qui semble converger, d'ailleurs : $F(0.61546816948997) = 3.774758283725532\ 10^{-15}$

Voyons comment caractériser :

la vitesse de convergence d'une suite,

la précision ajoutée par une itération,

la notion de nombre de chiffres précis ou de «chiffres exacts».

Soit $x_0, x_1, x_2, x_3, x_4...$ une suite de réels qui converge vers le réel r.

Nous écrirons $\{x_n\} \to r$ ou $\mathbf{x} \to r$

Définitions

1. Si $K_1(\mathbf{x}) = \lim\limits_{n\to\infty} \dfrac{r - x_{n+1}}{r - x_n}\ \exists$ et si $K_1(\mathbf{x}) \in (-1, 1)\backslash\{0\}$

on dit que x_n converge linéairement vers r.

2. Si $K_1(\mathbf{x}) = 0$ et si $K_2(\mathbf{x}) = \lim\limits_{n\to\infty} \dfrac{r - x_{n+1}}{(r - x_n)^2} \neq 0\ \exists$

on dit que x_n converge quadratiquement vers r.

3. Si $K_1(\mathbf{x}) = K_2(\mathbf{x}) = 0$ et si $K_3(\mathbf{x}) = \lim\limits_{n\to\infty} \dfrac{r - x_{n+1}}{(r - x_n)^3} \neq 0\ \exists$

on dit que x_n converge cubiquement vers r.

4. Si $K_1(\mathbf{x}) = K_2(\mathbf{x}) = K_3(\mathbf{x}) = \cdots = K_{p-1}(\mathbf{x}) = 0$ et si $K_p(\mathbf{x}) = \lim\limits_{n\to\infty} \dfrac{r - x_{n+1}}{(r - x_n)^p} \neq 0\ \exists$

on dit que x_n converge vers r à la puissance, ou à l'ordre, p.

$K_1, K_2, K_3, \ldots, K_p$ caractérisent la vitesse de convergence de la suite $\{x_n\}$.

Dans le cas de la convergence linéaire, plus $K_1(\mathbf{x})$ est proche de zéro, plus la convergence est rapide. Il en va de même pour $K_p(\mathbf{x})$ dans le cas de la convergence d'ordre p.

Théorème

Soit $\{x_n\} \to r$

1. Si la convergence est linéaire, alors $N(x_{n+1}, r) \simeq N(x_n, r) - log_{10}|K_1(\mathbf{x})|$.

2. Si la convergence est quadratique, alors $N(x_{n+1}, r) \simeq 2.N(x_n, r) - log_{10}|r.K_2(\mathbf{x})|$.

3. Si la convergence est cubique, alors $N(x_{n+1}, r) \simeq 3.N(x_n, r) - log_{10}|r^2.K_3(\mathbf{x})|$.

4. Si la convergence est d'ordre p alors $N(x_{n+1}, r) \simeq p.N(x_n, r) - log_{10}|r^{p-1}.K_p(\mathbf{x})|$.

Démonstration

Il suffit d'appliquer la définition de $N(x, r)$ pour obtenir

la précision ajoutée par une itération :

$N(x_{n+1}, r) - N(x_n, r) = -log_{10}|\frac{r-x_{n+1}}{r}| + log_{10}|\frac{r-x_n}{r}| = -log_{10}|\frac{r-x_{n+1}}{r-x_n}| \rightarrow -log_{10}|K_1(\mathbf{x})|$

On a aussi :

$N(x_{n+1}, r) - 2N(x_n, r) = -log_{10}|\frac{r-x_{n+1}}{r}| + 2log_{10}|\frac{r-x_n}{r}| = -log_{10}|r\frac{r-x_{n+1}}{(r-x_n)^2}| \rightarrow -log_{10}|rK_2(\mathbf{x})|$

$N(x_{n+1}, r) - 3N(x_n, r) = \cdots \rightarrow -log_{10}|r^2 K_3(\mathbf{x})|$

et ainsi de suite...

Ce théorème peut s'interpréter de la façon suivante :

Si la convergence est linéaire, on « gagne la même quantité de précision » à chaque itération. Si elle est quadratique, on « gagne le double de précision » à chaque itération, si elle est cubique, on en « gagne le triple », etc.

Exemple 2

Les suites $\{x_n = 1 + 0.8^n\}$ et $\{y_n = 1 + 0.01^n\}$ convergent linéairement vers $r = 1$. Leurs vitesses de convergence sont caractérisées par $K_1(\mathbf{x}) = 0.8$ et $K_1(\mathbf{y}) = 0.01$.

Comme $K_1(\mathbf{y}) \ll K_1(\mathbf{x})$, $\mathbf{y}$ converge plus rapidement que $\mathbf{x}$.

De plus, puisque $N(\mathbf{x}, r) = -log_{10}|0.8| = 0.096910013...$ il faut à peu près 10 itérations pour gagner un chiffre en précision en ce qui concerne la suite $\mathbf{x}$.

Par contre, avec $N(\mathbf{x}, r) = -log_{10}|0.01| = 2$ on gagne deux chiffres en précision à chaque itération de la suite $\mathbf{y}$.

Ceci se vérifie aisément à l'aide du programme :

```
for n = 1:40
    iter(n) = n;
    x(n) = 1 + 0.80^n;
    y(n) = 1 + 0.01^n
end
format long
[iter' x' y']
```

qui produit les résultats suivants [4] :

4. Le format d'affichage de la première colonne a été modifié.

1	1.80000000000000	1.01000000000000
2	1.64000000000000	1.00010000000000
3	1.51200000000000	1.00000100000000
4	1.40960000000000	1.00000001000000
5	1.32768000000000	1.00000000010000
6	1.26214400000000	1.00000000000100
7	1.20971520000000	1.00000000000001
8	1.16777216000000	1.00000000000000
9	1.13421772800000	1.00000000000000
10	1.10737418240000	1.00000000000000
11	1.08589934592000	1.00000000000000
12	1.06871947673600	1.00000000000000
13	1.05497558138880	1.00000000000000
14	1.04398046511104	1.00000000000000
15	1.03518437208883	1.00000000000000
16	1.02814749767107	1.00000000000000
17	1.02251799813685	1.00000000000000
18	1.01801439850948	1.00000000000000
19	1.01441151880759	1.00000000000000
20	1.01152921504607	1.00000000000000
21	1.00922337203685	1.00000000000000
22	1.00737869762948	1.00000000000000
23	1.00590295810359	1.00000000000000
24	1.00472236648287	1.00000000000000
25	1.00377789318630	1.00000000000000
26	1.00302231454904	1.00000000000000
27	1.00241785163923	1.00000000000000
28	1.00193428131138	1.00000000000000
29	1.00154742504911	1.00000000000000
30	1.00123794003929	1.00000000000000
31	1.00099035203143	1.00000000000000
32	1.00079228162514	1.00000000000000
33	1.00063382530011	1.00000000000000
34	1.00050706024009	1.00000000000000
35	1.00040564819207	1.00000000000000
36	1.00032451855366	1.00000000000000
37	1.00025961484293	1.00000000000000
38	1.00020769187434	1.00000000000000
39	1.00016615349947	1.00000000000000
40	1.00013292279958	1.00000000000000

Les deux suites convergent linéairement, mais, on ajoute 2/0.096910013... soit environ 20.63... fois plus de chiffres précis par itération de $\{y_n\}$ que par itération de $\{x_n\}$.

Dans ce qui précède les valeurs de K_1 supposent la connaissance a priori de la limite r que l'on cherche. Il s'agit donc de valeurs théoriques.

Dans la pratique, pour avoir une idée de la vitesse de convergence d'un calcul itératif, on utilisera une mesure approximative $K_1(\mathbf{x}, n)$ de $K_1(\mathbf{x})$ dont le calcul ne nécessite pas la connaissance de la valeur exacte r.

Théorème

Soit x_n une suite de réels qui converge linéairement vers r, posons :

$K_1(\mathbf{x}, n) = \dfrac{x_{n+2} - x_{n+1}}{(x_{n+1} - x_n)}$ nous avons $K_1(\mathbf{x}) = \lim\limits_{n\to\infty} K_1(\mathbf{x}, n) \Rightarrow K_1(\mathbf{x}, n) \cong K_1(\mathbf{x})$

Démonstration

$$\lim_{n\to\infty} K_1(\mathbf{x}, n) = \lim_{n\to\infty} \frac{x_{n+2} - x_{n+1}}{x_{n+1} - x_n} = \lim_{n\to\infty} \frac{\frac{x_{n+2}-r}{x_{n+1}-r} - 1}{1 - \frac{x_n - r}{x_{n+1}-r}} = \frac{K_1(\mathbf{x}) - 1}{1 - \frac{1}{K_1(\mathbf{x})}} = K_1(\mathbf{x}),$$

puisque $K_1(\mathbf{x}) \neq 0$ et $K_1(\mathbf{x}) \neq 1$.

$K_1(\mathbf{x}, n) = \dfrac{x_{n+2} - x_{n+1}}{(x_{n+1} - x_n)}$ servira d'indicateur de convergence effective lors des calculs.

Exemple 3

Soient trois approximations consécutives d'une suite qui converge linéairement vers le nombre inconnu r :

```
x1 = 3.074088000
x2 = 3.031116960
x3 = 3.013069123
```

On a $K_1(\mathbf{x}, 1) \cong 0.4200000047$ d'où l'on tire :

$N(x_{n+1}, r) - N(x_n, r) \cong -\log_{10} K_1(\mathbf{x}, 1) \cong -\log_{10} K_1(\mathbf{x}) \cong -\log_{10} 0.420... \cong 0.377$

On ajoute donc environ 0.377 chiffre décimal précis par itération et il faut 2.65 itérations pour gagner une décimale en précision [5]. Pour les trois valeurs approchées de r les deux premiers chiffres sont identiques 3.0... si on admet qu'ils sont exacts il faudra environ 8×2.65 itérations pour obtenir 10 chiffres exacts en tout.

Ce diagnostic est plausible...

Mais :

ce n'est pas parce que les deux premiers chiffres sont les mêmes pour les trois valeurs successives qu'ils correspondent aux deux premiers chiffres de r. Une stabilité apparente des premiers chiffres peut simplement être la traduction d'une convergence trop lente pour affecter et corriger en seulement deux itérations un chiffre incorrect en deuxième position.

De plus, rien ne garantit que le chiffre en première position soit exact, d'ailleurs, le quatrième terme de la suite est : $x_4 = 2.991254806$.

Il faut donc faire preuve d'une grande prudence dans l'utilisation des vitesses de convergence approchées et ne pas se contenter d'une seule valeur estimée à partir de trois termes successifs de la suite étudiée.

5. On ne peut évidemment pas fractionner un nombre d'itérations... !

Exemple 4

La série de Leibnitz fournit des approximations de π : $\pi = 4 * [1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \dots]$

Soit S_n la somme des n premiers termes de cette série, on obtient les valeurs suivantes de quelques sommes partielles :

```
n = 1   4.00000000000000    n = 7        3.28373848373848
n = 2   2.66666666666667    n = 8        3.01707181707182
n = 3   3.46666666666667    n = 9        3.25236593471888
n = 4   2.89523809523810    n = 10       3.04183961892940
n = 5   3.33968253968254    n = 1000     3.14059265383979
n = 6   2.97604617604618    n = 100000   3.14158265358972

n = 1000000    3.14159165358977
```

On constate que la série de Leibnitz converge lentement puisqu'en sommant un million de termes on n'obtient que six chiffres exacts.

La convergence est linéaire. $K_1(\mathbf{x}, 8) \cong -0.89$, mais, comme nous l'avons signalé ci-dessus, $K_1(\mathbf{x}, n)$ peut varier en fonction de n.

A titre d'exemple on a :

$$K_1(\mathbf{x}, 10) \cong \frac{3.05840276592733 - 3.23231580940559}{3.23231580940559 - 3.04183961892940} \cong -0.913043$$

et

$$K_1(\mathbf{x}, 630) \cong \frac{3.14001037609913 - 3.14317743864861}{3.14317743864861 - 3.14000535300230} \cong -0.99841646872447.$$

Ainsi, si on somme 630 termes on obtient π avec trois chiffres exacts (3.14) et d'après la valeur de $K_1(\mathbf{x}, 630)$ il faudrait encore ajouter $1/(-\log_{10} 0.99841646872447) \cong 1453$ termes supplémentaires pour obtenir un quatrième chiffre exact !

En pratique, on constate que pour avoir deux approximations successives qui présentent toutes deux quatre chiffres exacts il faut sommer un total de 2454 termes de la série :

$S_{2453} = 3.14200031765765$
$S_{2454} = 3.14118515564420$
$S_{2455} = 3.14199998554846$

Notre prévision était donc trop optimiste. On peut d'ailleurs constater que plus on prend de termes de la série, plus il en faudra pour gagner un chiffre précis. Ainsi, $K_1(\mathbf{x}, 100000) = -0.99998999997138$ ce qui suggère que pour obtenir un chiffre précis en plus il faut prendre $-1/\log_{10}(0.99998999997138) = 230.256,6989...$ termes supplémentaires de la série de Leibnitz. Alors que $K_1(\mathbf{x}, 1000000) = -0.99999899991060$ ce qui impliquerait qu'il faut plus de 2 millions de termes en plus pour gagner un chiffre puisque $-1/\log_{10}(0.0.99999899991060) = 2.302.378,1078....$

Pour se faire une idée de la dépendance de $K_1(\mathbf{x}, n)$ en fonction de n, pour la série de Leibnitz, nous avons représenté cette fonction à la figure 2.1.

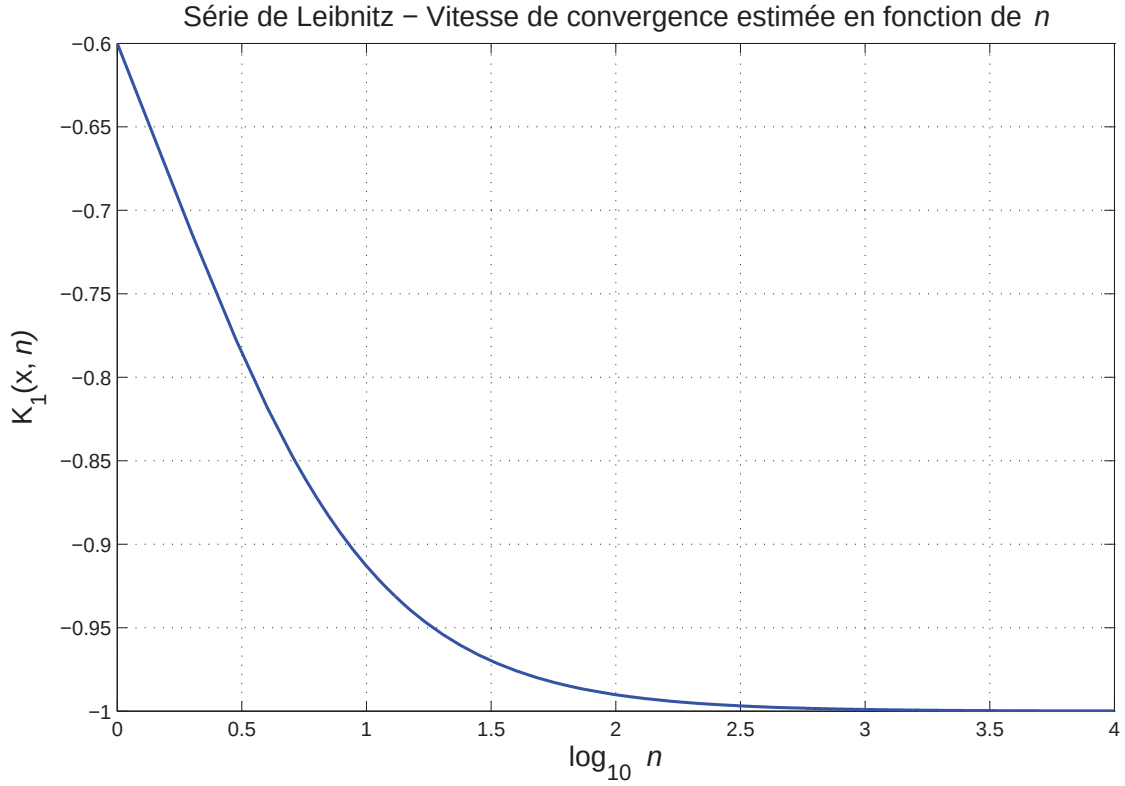


FIGURE 2.1 – Evolution de l'estimateur de la vitesse de convergence en fonction du nombre de termes conservés dans la somme partielle qui définit la série de leibnitz.

On constate que la valeur asymptotique $K_1(\mathbf{x}, n)$ tend vers -1 et qu'elle différe fortement des valeurs obtenues pour n "petit" . Ceci confirme qu'il faut être prudent lorsqu'on essaye d'estimer $K_1(\mathbf{x})$ à partir de $K_1(\mathbf{x}, n)$ et tient au fait que $K_1(\mathbf{x}) = \lim_{n\to\infty} K_1(\mathbf{x}, n)$.

Une autre illustration de l'influence de n sur l'estimateur $K_1(\mathbf{x}, n)$ de la vitesse de convergence est donnée à la figure 2.2 où l'on a représenté le nombre de termes, qu'en principe, il faudrait ajouter à la somme partielle S_n pour gagner un chiffre précis en plus à n donné.

Les résultats précédents ont été obtenus à l'aide de la fonction et du programme suivants :

```
% Calcul d'une valeur approchée de par sommation
% des N premiers termes de la série de Leibnitz
%
function piapp=apppi(N)
piapp=0; fac=1;
for n=1:N
piapp=piapp+fac/(2*n-1);
fac=-fac;
end
piapp=4*piapp;
```

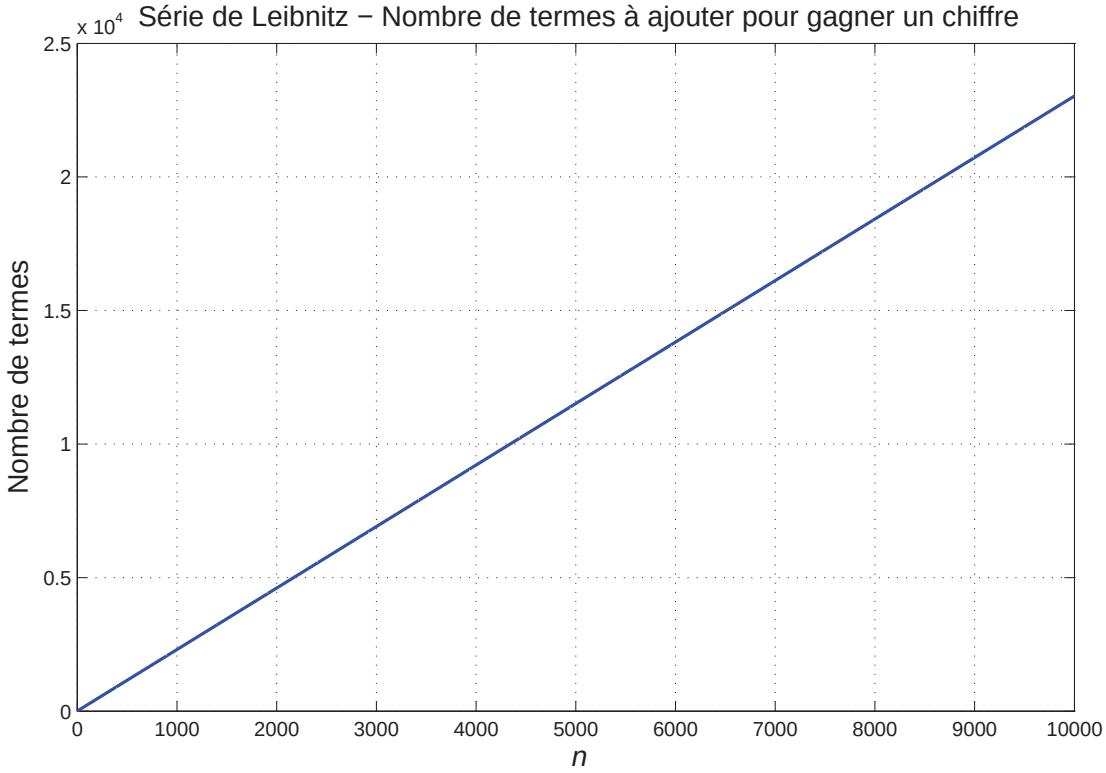


FIGURE 2.2 – Evolution du nombre de termes qu'il faudrait ajouter à la somme partielle qui définit la série de Leibnitz pour obtenir un chiffre précis supplémentaire, en fonction du nombre de termes n de cette somme. (Notez l'échelle des ordonnées : 10^4).

```
% Programme permettant de réaliser les figure 2.1 et 2.2 du chapitre 2
clc;clear
Nmax=10000;
x=1:Nmax;
log10x=log10(x);
piapp=0; fac=1;
for n=1:Nmax+2
piapp=piapp+fac/(2*n-1);
fac=-fac;
Terme(n)=piapp;
end
Terme=4*Terme;
for k=1:Nmax
    t1=Terme(k);
    t2=Terme(k+1);
    t3=Terme(k+2);
    K1(k)=(t3-t2)/(t2-t1);
end
```

```
Ntermes=-1./log10(abs(K1));
figure(1)
plot(log10x,K1,'LineWidth',1.5)
xlabel('log_1_0 \it n','FontSize',14)
ylabel('K_1(x,\it n)','FontSize',14)
titre1='Série de Leibnitz - Vitesse de convergence en fonction de \it n';
title(titre1,'FontSize',14)
grid on
figure(2)
plot(x,Ntermes,'LineWidth',1.5)
xlabel('\it n','FontSize',14)
ylabel('Nombre de termes','FontSize',14)
titre2='Série de Leibnitz - Nombre de termes à ajouter pour gagner un chiffre';
title(titre2,'FontSize',14)
grid on
```

2.2 Accélération de la convergence

Il arrive fréquemment qu'une suite convergeant linéairement soit de peu d'intérêt pratique en raison d'une convergence trop lente. C'est le cas de la suite des sommes partielles tirées de la série de Leibnitz.

Pour améliorer la convergence de la suite on peut la transformer en une nouvelle suite au moyen d'un algorithme d'accélération de convergence. Dans certains cas les effets d'accélération sont très spectaculaires.

Algorithme d'Aitken-Shanks

Soit r le réel vers lequel la suite $x_1, x_2, x_3, \ldots$ converge linéairement.

Pour n suffisamment grand, on a : $\dfrac{r - x_{n+1}}{r - x_n} \cong \dfrac{x_{n+2} - x_{n+1}}{x_{n+1} - x_n}$

d'où l'on tire une nouvelle approximation de r : $r \cong x_{n+2} - \dfrac{(x_{n+2} - x_{n+1})^2}{(x_{n+2} - 2x_{n+1} + x_n)}$

Cette formule, connue sous le nom de formule *d'Aitken-Shanks* associe à la suite $\{x_n\}$ une nouvelle suite $\{y_n\}$

$$y_n = x_{n+2} - \frac{(x_{n+2} - x_{n+1})^2}{(x_{n+2} - 2x_{n+1} + x_n)} \tag{2.1}$$

La nouvelle suite **y** converge généralement plus vite que la suite d'origine.

L'application répétée de la formule *d'Aitken-Shanks* aux neuf premiers termes de la suite considérée dans l'exemple 4 fournit la première colonne du tableau ci-dessous. La deuxième colonne de ce tableau s'obtient en faisant opérer le même algorithme sur la première colonne et ainsi de suite. La valeur qui figure dans la dernière colonne est une meilleure approximation de π que celle obtenue en sommant un million de termes de la série de Leibnitz !

```
3.16666666666667
3.13333333333334
3.14523809523810 3.14210526315790
3.13968253968254 3.14145021645022
3.14271284271284 3.14164332399627 3.14159935731901
3.14088134088134 3.14157129020143 3.14159086039588
3.14207181707182 3.14160284160284 3.14159323124376 3.14159271403378
```

2.3 Méthode de dichotomie (bissection)

2.3.1 Principe de la méthode de dichotomie

La méthode de dichotomie, basée sur le théorème de la valeur intermédiaire pour les fonctions continues est, en quelque sorte, ce nous pourrions appeler une méthode de calcul par tâtonnements organisés.

Si la fonction $f(x)$ est continue sur l'intervalle $[a, b]$ et telle que $f(a).f(b) < 0$ il y a au moins une racine dans (a, b).

La moitié de l'intervalle qui contient la racine sert de nouvel intervalle pour l'itération suivante.

La méthode de dichotomie génère donc une suite d'intervalles emboîtés $\{[a_n, b_n]\}$ telle que pour tout n : $f(a_n).f(b_n) < 0$.

2.3.2 Algorithme de la méthode de dichotomie

a) $[a_0, b_0] = [a, b]$
b) $[a_n, b_n]$ étant connu, on pose $w = (a_n + b_n)/2$ et on teste le signe de $f(a_n).f(w)$.
c) Si ce signe est positif, le changement de signe de $f(x)$ a lieu dans la deuxième moitié de l'intervalle et on pose $[a_{n+1}, b_{n+1}] = [w, b_n]$. S'il est négatif, le changement de signe a lieu dans la première moitié de l'intervalle et on pose $[a_{n+1}, b_{n+1}] = [a_n, w]$. Si le produit $f(a_n).f(w)$ est nul, on a trouvé le zéro cherché (ou un autre !).

Retour à l'exemple 1

Proposons nous de calculer une approximation du zéro de $f(x) = x - 0.2sin(x) - 0.5$ qui se trouve dans l'intervalle $[a, b] \equiv [0, 1]$.

Voici les résultats obtenus pour les 7 premières approximations (itérations) :

a_n	b_n	$f(a_n)$	$f(b_n)$	w	$f(w)$
0.0	1.0	$-$	$+$	0.5	$-$
0.5	1.0	$-$	$+$	0.75	$+$
0.5	0.75	$-$	$+$	0.625	$+$
0.5	0.625	$-$	$+$	0.5625	$-$
0.5625	0.625	$-$	$+$	0.59375	$-$
0.59375	0.625	$-$	$+$	0.609375	$-$
0.609375	0.625	$-$	$+$	0.6171875	$+$
0.609375	0.6171875	$-$	$+$		

NB : pour obtenir dix chiffres exacts il faut faire trente quatre itérations.

Comme la méthode de dichotomie génère une suite d'intervalles emboîtés, plutôt qu'une suite de réels, notre définition des $K_i(\mathbf{x})$ n'est pas valable. Nous admettrons toutefois que la convergence est linéaire et que $K_1(\mathbf{x}) = 0.5$.

2.3.3 Evaluation du coût de la méthode de dichotomie

Le coût d'une méthode numérique est le nombre d'opérations, d'évaluations de fonctions, d'assignations et de comparaisons à effectuer pour mettre la méthode en oeuvre. Chaque itération a un coût et le coût total dépend évidemment du nombre d'itérations à effectuer. Il s'agit donc d'une notion relativement théorique. En effet, le nombre d'itérations à effectuer pour atteindre une précisison fixée peut varier fortement d'une méthode à l'autre. De plus, pour résoudre une équation donnée on est conduit à évaluer une ou plusieurs fonctions à chaque itération et ces fonctions dépendent de la méthode de résolution choisie. Il y a même des cas où un algorithme donné peut être mis en oeuvre en faisant appel à différentes fonctions [6]. Ce n'est donc qu'à titre indicatif que l'on évaluera le coût d'une méthode de résolution d'une équation non linéaire à une inconnue.

En ce qui concerne le coût de la méthode de dichotomie, il se monte à :

une assignation, une addition, une division, une évaluation de fonction et un test de signe par itération.

2.3.4 Avantages de la méthode de dichotomie

Les principaux avantages de la méthode de dichotomie sont :

a) la convergence certaine de l'algorithme vers la racine cherchée si $f(x)$ est continue sur l'intervalle $[a, b]$ dans lequel on a réussi à isoler une racine unique ;
b) la possibilté de déterminer a priori le nombre d'itérations à effectuer pour obtenir la racine voulue avec une précision donnée.

En effet, on part d'un intervalle connu $[a, b]$ et on génère des intervalles dont la longueur vaut la moitié de celle de l'intervalle précédent. Ainsi, après n itérations, l'intervalle qui encadre la racine cherchée est de longueur $\dfrac{(b-a)}{2^n}$ et si l'on veut que l'erreur absolue

6. C'est le cas de la méthode d'itération, ou du point fixe.

commise sur le résultat soit inférieure à une quantité ϵ prédéfinie, il suffira de faire n itérations avec $n > \dfrac{\ln \frac{b-a}{\epsilon}}{\ln 2}$.

2.3.5 Inconvénients de la méthode de dichotomie

Les principaux désavantages de la méthode de dichotomie sont :

a) les contraintes imposées par les hypothèses de départ dont, principalement, le fait qu'il faille déterminer un intervalle qui encadre une seule racine et dans lequel la fonction étudiée change de signe [7].
b) la relative lenteur de la convergence. En effet, la convergence est linéaire et sa vitesse est caractérisée par $K_1(\mathbf{x}) = 0.5$.

2.4 Méthode regula falsi (fausse position)

On peut substituer à la méthode de dichotomie une méthode plus efficace qui en découle directement. L'idée est de tenir compte des valeurs de la fonction aux extrémités de l'intervalle de travail et non seulement des signes qu'elle y prend.

Supposons que les conditions d'application de la méthode de dichotomie soient remplies. Au lieu de diviser l'intervalle de travail en deux parties égales, on procède différemment.

Sur l'intervalle en question on assimile le graphe de $f(x)$ à la droite qui joint les points $(a, f(a))$ et $(b, f(b))$. Une approximation de la racine cherchée est donc l'abscisse du point où cette droite coupe l'axe des x. On prend pour w l'abscisse de ce point [8] et il est facile de montrer que $w = b - f(b)\dfrac{b-a}{f(b)-f(a)} = a - f(a)\dfrac{b-a}{f(b)-f(a)}$.

Le nouvel intervalle de travail sera soit $[a, w]$ soit $[w, b]$ selon les signes pris par $f(a)$, $f(b)$ et $f(w)$.

2.5 Méthode de la sécante

2.5.1 Principe de la méthode de la sécante

La méthode de la sécante est apparentée à la méthode Regula Falsi. Ici aussi on assimile le graphe de $f(x)$ à une droite. Toutefois, on se libère des contraintes des deux méthodes précédentes en n'exigeant plus la détermination d'un intervalle qui encadre la racine cherchée et sur lequel $f(x)$ change de signe.

Soient x_0 et x_1 deux approximations de la racine r d'une équation $f(x) = 0$. Si, dans un voisinage de l'intervalle $[x_0, x_1]$ on remplace le graphe de $f(x)$ par une droite qui joint les points $(x_0, f(x_0))$ et $(x_1, f(x_1))$ on peut espérer trouver une nouvelle approximation de la

7. Ce qui exclut que cette méthode puisse être appliquée aux racines de multiplicité paire.
8. L'abscisse w est une "fausse position" du zéro cherché.

racine cherchée en assimilant celle-ci au point x_2 où la droite coupe l'axe des x, comme on le fait dans la méthode de la fausse position [9].

Il est facile, à partir de l'équation de la droite, d'obtenir x_2. On a :

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

Ensuite, partant de x_1 et x_2, on calcule x_3, etc.

2.5.2 Algorithme de la méthode de la sécante

Eu égard à ce qui précède, l'algorithme de la méthode de la sécante est le suivant :

a) Fixer x_0 et x_1

b) Calculer $x_{n+2} = x_{n+1} - f(x_{n+1})\frac{x_{n+1} - x_n}{f(x_{n+1}) - f(x_n)}$, pour $n = 0, 1, 2, \ldots$

Nouveau retour à l'exemple 1

On se propose de résoudre l'équation $x - 0.2sin(x) - 0.5 = 0$ par la méthode de la sécante à l'aide du programme rudimentaire, écrit en Matlab, qui suit :

```
% Méthode de la sécante
% Résolution de l'équation  x - 0.2sin(x) - 0.5 = 0
f=inline('x-0.2*sin(x)-0.5'); % Fonction dont on cherche un zéro
x(1)=0; x(2)=1;               % Valeurs initiales : 2 précurseurs de la suite
% Boucle calculant 8 valeurs de la suite : (3), x(4), ... , x(11)
for n=1:8
    F=f(x(n+1));                                % Pour ne pas évaluer 2 fois f(x(n+1))
    x(n+2)=x(n+1)-F*(x(n+1)-x(n))/(F-f(x(n)));
end
format long
y=f(x);                       % Evaluation des images par f de la suite des x
[x' y']                       % Affichage de la suite et de son image
```

Les résultats produits par ce programme sont les suivants :

```
Warning: Divide by zero.
> In secante_EX1 at 7

ans =

                   0  -0.50000000000000
   1.00000000000000   0.33170580303842
   0.60117411490142  -0.01194810966334
```

9. Dans le cas de la méthode de la sécante, x_2 peut être à l'extérieur de l'intervalle x_0 et x_1.

```
0.61504043574228  -0.00035787398859
0.61546858886061   0.00000035088715
0.61546816947759  -0.00000000001035
0.61546816948997                  0
0.61546816948997                  0
             NaN                NaN
             NaN                NaN
```

On a onze chiffres exacts [10] après 4 itérations seulement. Pour atteindre le même résultat, la méthode Regula Falsi nécessiterait 7 itérations, tandis que la méthode de dichotomie en demanderait 34 !

Remarques

Le message d'erreur :

```
Warning: Divide by zero.
> In secante_EX1 at 7
```

Indique à l'utilisateur que le programme effectue une division par zéro. Ce message est un avertissement `"Warning"` il ne provoque pas l'interruption du programme.

La liste des valeurs des itérés et de leurs images comprend des `NaN` [11] qui indiquent que x(10), x(11) et leurs images ne sont pas des nombres. Ceci provient du fait, que x(10) résulte d'une division de zéro par zéro. Comme x(10) n'est pas un nombre, f(x(10)), x(11) et f(x(11)) n'en sont pas non plus puisqu'ils découlent de calculs impliquant x(10).

2.5.3 Evaluation du coût de la méthode de la sécante

Le coût de la méthode de la sécante, par itération, se monte à deux assignations, trois soustractions, un produit, une division et deux évaluations de fonction. Il convient néanmoins de se souvenir des précautions énoncées précédemment à la page 28.

2.5.4 Avantages de la méthode de la sécante

Les principaux avantages de la méthode de la sécante sont :

a) les hyptothèses de départ sont peu contraignantes si la fonction présente un seul zéro
b) la convergence est rapide dans le cas des racines simples, même si elles sont nombreuses, pour autant qu'on puisse trouver de bons précurseusr x_1 et x_2.

10. Dont dix après le point décimal
11. NaN signifie Not a Number

2.5.5 Inconvénients de la méthode de la sécante

Le principal désavantage de la méthode de la sécante est la lenteur de sa convergence et son manque de précision relatif lorsqu'on a affaire à une racine multiple.

Le programme qui suit et les résultats qu'il produit illustrent ce désavantage.

Le programme :

```
% Méthode de la sécante
% Résolution d'une équation dont pi/12 est une racine double
f=inline('16*(sin(x).*cos(x)).^2-4*sin(2*x)+1'); % Fonction f
fp=inline('4*sin(x).*cos(x)-1');                 % Dérivée de f
x(1)=pi/11; x(2)=pi/13;    % Valeurs initiales : 2 précurseurs de la suite
% Boucle calculant 39 valeurs de la suite : (3), x(4), ... , x(41)
for n=1:39
    F=f(x(n+1));
    x(n+2)=x(n+1)-F*(x(n+1)-x(n))/(F-f(x(n)));
end
format long
y=f(x);                 % Evaluation des images par f de la suite des x
z=fp(x);                % Evaluation des images par fp de la suite des x
[x' y' z']              % Affichage de la suite et de ses images
r=pi/12;                % r=pi/12 est racine double
[r f(r) fp(r)]          % Affichage de pi/12 et de ses images
```

Les résulats du programme :

```
Warning: Divide by zero.
> In secante_EX2_rac_double at 8

ans =

   0.28559933214453   0.00660670417384   0.08128163491120
   0.24166097335306   0.00497781836261  -0.07055365591246
   0.10738687770536   0.32918587883926  -0.57374722556127
   0.24372258791612   0.00400178979700  -0.06325970120863
   0.24540036635754   0.00328735955913  -0.05733549999027
   0.25312044972630   0.00091287720551  -0.03021385783884
   0.25608846004702   0.00039394439753  -0.01984803258590
   0.25834160538625   0.00014404624185  -0.01200192658909
   0.25964036294440   0.00005607585010  -0.00748838100671
   0.26046824301520   0.00002129600284  -0.00461475923963
   0.26097516117979   0.00000815994743  -0.00285656216898
   0.26129005217571   0.00000311490341  -0.00176490889615
   0.26148447169612   0.00000119049845  -0.00109109965331
   0.26160474582933   0.00000045472812  -0.00067433531598
```

```
   0.26167907884839   0.00000017371505  -0.00041679137510
   0.26172502960872   0.00000006635538  -0.00025759538490
   0.26175343022065   0.00000002534653  -0.00015920594590
   0.26177098392135   0.00000000968168  -0.00009839553245
   0.26178183300694   0.00000000369812  -0.00006081220037
   0.26178853825933   0.00000000141257  -0.00003758414384
   0.26179268238307   0.00000000053956  -0.00002322833259
   0.26179524361093   0.00000000020609  -0.00001435592347
   0.26179682654628   0.00000000007872  -0.00000887245333
   0.26179780485636   0.00000000003007  -0.00000548347969
   0.26179840948386   0.00000000001149  -0.00000338898548
   0.26179878315836   0.00000000000439  -0.00000209453787
   0.26179901411495   0.00000000000168  -0.00000129448032
   0.26179915684805   0.00000000000064  -0.00000080003817
   0.26179924506164   0.00000000000024  -0.00000049445728
   0.26179929949920   0.00000000000009  -0.00000030588002
   0.26179933318719   0.00000000000004  -0.00000018918137
   0.26179935408803   0.00000000000001  -0.00000011677875
   0.26179936667013   0.00000000000001  -0.00000007319309
   0.26179937553076   0.00000000000000  -0.00000004249896
   0.26179937970047   0.00000000000000  -0.00000002805466
   0.26179938506152   0.00000000000000  -0.00000000948343
   0.26179938774205  -0.00000000000000  -0.00000000019781
   0.26179938666984                  0  -0.00000000391206
   0.26179938666984                  0  -0.00000000391206
                NaN                NaN                NaN
                NaN                NaN                NaN

ans =

   0.26179938779915                  0  -0.00000000000000
```

La meilleure approximation de la racine est `x(37) = 0.26179938774205` avec une erreur relative d'environ -2.10^{-10}. On constate que les résulats se dégradent pour `n > 37` et que la méthode ne permet pas d'obtenir la "pleine précision", c'est-à-dire d'obtenir 14 chiffres précis.

2.6 Méthode d'itération (ou du point fixe)

Les méthodes de dichotomie et Regula Falsi fournissent une suite d'intervalles emboîtés encadrant la racine cherchée de l'équation $F(x) = 0$ que l'on veut résoudre.
La méthode de la sécante produit une suite de valeurs sensées tendre vers la valeur du zéro cherché et chaque nombre de la suite dépend de deux précurseurs. On peut schématiser son algorithme de la façon suivante :

a) Choisir deux valeurs de départ : x_1 et x_2
b) Calculer les itérés successifs : $x_{n+2} = g(x_n, x_{n+1})$ pour $n = 1, 2, ...$

2.6.1 Principe et algorithme de la méthode d'itération

Pour résoudre l'équation $F(x) = 0$, la méthode d'itération, aussi appelée méthode du point fixe [12], procède comme suit :

a) Choisir une valeur de départ x_1
b) Itérer, autant de fois qu'il faut : $x_{n+1} = g(x_n)$ pour $n = 1, 2, ...$

Cette méthode est donc, en principe, plus simple à programmer que les méthodes précédentes. De plus elle nécessite moins d'opérations par itération.
Le point délicat est ici de faire un "choix convenable" de la fonction d'itération $g(.)$.
En effet, on peut mettre l'équation à résoudre $F(x) = 0$ sous la forme équivalente $x = g(x)$ d'une infinité de manières.

Proposons-nous de calculer le zéro proche de 0.6 de l'équation $x - 0.2\sin(x) - 0.5 = 0$ en la mettant sous la forme évidente : $x = 0.2\sin(x) + 0.5$. Comme nous l'avons vu précédemment (cf. page 19), la méthode est efficace et fournit une valeur approchée du zéro cherché avec au moins 14 chiffres significatifs exacts : 0.61546816948997 [13].

2.6.2 Interprétation graphique de la méthode d'itération

La méthode converge

Si l'on fait un graphe de $f(x) = 1 - x^3/4$ on constate que cette expression ne s'annule que pour une seule valeur réelle de x, proche de 1.59, qui vaut d'ailleurs $\sqrt[3]{4} = 1.58740105196819\ldots$.

Il est facile d'obtenir une approxiamtion de ce zéro avec une erreur absolue de l'ordre de 10^{-5}. Il suffit pour cela d'appliquer la méthode du point fixe en choisissant $g(x) = x + f(x) = x + 1 - x^3/4$ et d'éxécuter le programme suivant :

```
x(1)=1.59;
for n=1:50
x(n+1)=x(n) + 1 - x(n)^3/4;
end
x'
```

pour trouver $x(51) = 1.587408675995064$. Si l'on veut une précision supérieure, il suffit de réexécuter le programme en augmentant le nombre d'itérations à faire ou d'en écrire une variante qui teste l'écart entre deux itérés successifs, par exemple :

12. Une autre dénomination est : méthode des approximations successives.
13. Le quinzième chiffre est un 7 mais il résulte d'un arrondi car les chiffres en positions 15 et 16 sont respectivement 6 et 5, le 17ème valant 3 ou 4.

```
x(1)=1.59;
n=1;
f=1;
while abs(f) > 1e-14
f = 1 - x(n)^3/4;
x(n+1)=x(n) + f;
n=n+1;
end
x'
length(x)
```

Après 233 itérations, on obtient comme approximation de $\sqrt[3]{4}$ la valeur 1.587401051968204 qui est exacte à moins de 5.10^{-15} près. Si l'on part de $x_1 = 1.59$, à partir du 244ème itéré, les valeurs successives de x_n oscillent entre 1.587401051968199 et 1.587401051968200.

Le programme suivant, un peu plus élaboré que les précédents, permet d'illustrer graphiquement comment fonctionne la méthode du point fixe pour résoudre notre problème. Il fournit un diagramme qui s'affiche étape par étape grâce aux instructions `pause` qu'il contient. Le nombre d'itérations a été volontairement limité à 10. On constate que la suite des x_n oscille autour du zéro cherché dont l'abscisse est celle du point pour lequel on a $x = g(x)$ [14], ce qui se traduit part un graphe en forme de "spirale rectangulaire" s'enroulant autour du point fixe $(r, g(r))$.

```
% Ce programme illustre graphiquement le "fonctionnement"
% de la méthode d'itération lorsqu'elle converge.
% La fonction dont on cherche un zéro est  y(x) = 1 - x^3/4
a=0:0.001:2;
b = a - a.^3/4 + 1;
plot(a,b,'LineWidth',1.3)
axis equal
xlabel('\it x','FontSize',14); ylabel('\it g(x) = x - x^3/4 + 1','FontSize',14)
title('Méthode du point fixe','FontSize',14)
grid on ; hold on ; pause
x0=0.2;
NiterMax = 10;
plot(a,a,'LineWidth',1.3);
hold on
X(1) = x0;   Y(1) = 0;
for n = 1:1:NiterMax
    g  = x0 - x0^3/4 + 1;
    x(n)=g;
    X(2) = x0;   Y(2) = g;
    X(3) = g;    Y(3) = g;
    plot(X,Y,'rO-','LineWidth',1.3)
```

14. Dans le système d'axes (x, y) c'est l'abscisse du point d'intersection de la bissectrice $y = x$ avec le graphe de la courbe $y = g(x)$.

```
    X(1) = g;     Y(1) = g;
    pause
    x0 = g;
end
pause ; format long e; x'
```

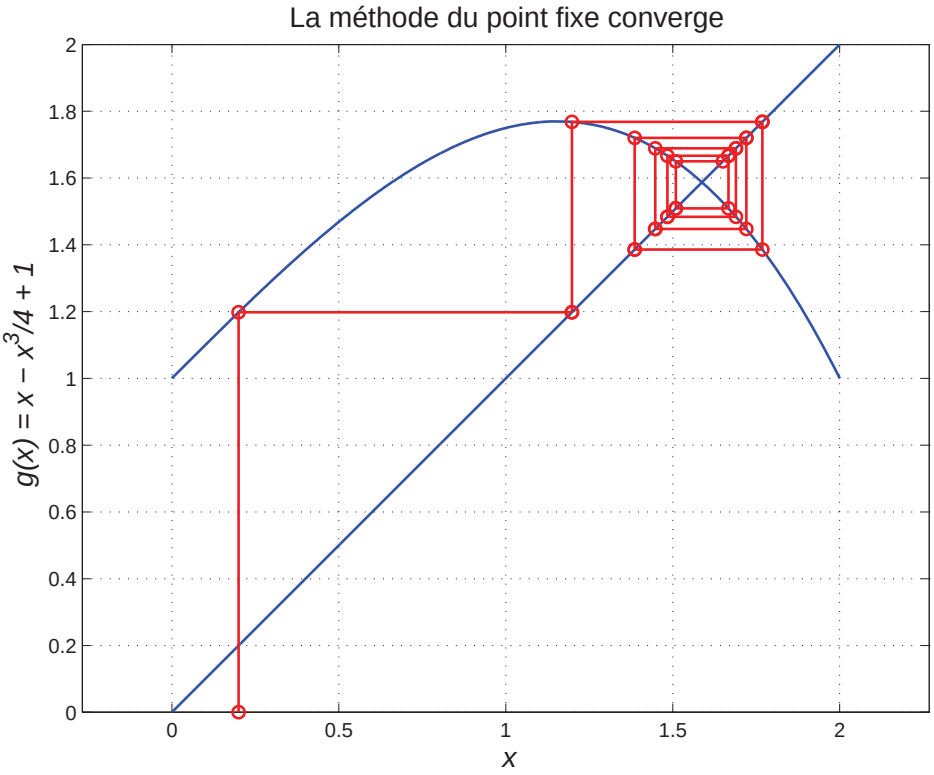


FIGURE 2.3 – La figure produite par le programme ci-dessus illustre une situation dans laquelle la méthode du point fixe converge.

Remarque

Lorsque la méthode du point fixe converge, l'illustration graphique de la convergence peut conduire à des diagrammes différents de celui que montre la figure 2.3. Au lieu d'avoir une suite oscillante qui se traduit par un diagramme en "spirale", la convergence vers le zéro cherché peut être monotone. La suite peut alors être croissante ou décroissante selon les cas et présente un diagramme en escalier.

La méthode cycle

Si l'on essaye de trouver le zéro de $f(x) = 1 - x^3/3$ plutôt que celui de $f(x) = 1 - x^3/4$ la méthode est prise en défaut, elle cycle (voir page 15). Après un certain nombre d'itérations, les valeurs des itérés successifs se répètent en alternance, on a un cycle d'ordre deux avec $x_{2n} = 1.62924292865886$ et $x_{2n+1} = 1.18767112407051$ pour tout n suffisamment grand. Ces valeurs encadrent $\sqrt[3]{3} = 1.442249570307408\ldots$ le zéro cherché.

```
% Ce programme illustre graphiquement le "fonctionnement"
% de la méthode d'itération lorsqu'elle cycle.
% La fonction dont on cherche les zéros est  y(x) = 1 - x^3/3
a=0:0.001:2; b = a - a.^3/3 + 1; plot(a,b)
xlabel('x'); ylabel('g = x - x^3/3 + 1')
title('Méthode du point fixe')
grid on ; hold on ; axis equal; pause
x0=0.2;
NiterMax = 10;
plot(a,a); hold on
X(1) = x0;   Y(1) = 0;
for n = 1:1:NiterMax
    g  = x0 - x0^3/3 + 1;
    x(n)=g;
    X(2) = x0;   Y(2) = g;
    X(3) = g;    Y(3) = g;
    plot(X,Y,'rO-')
    X(1) = g;    Y(1) = g;
    pause
    x0 = g;
end
pause ; format long ; x'
```

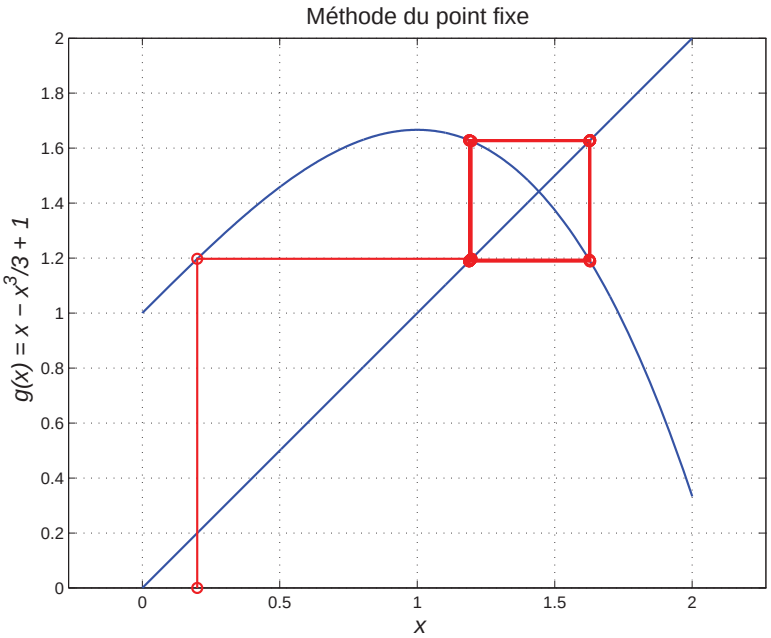


FIGURE 2.4 – La figure produite par le programme ci-dessus illustre une situation dans laquelle la méthode du point fixe cycle.

La méthode diverge

```
% Point fixe - Diverge
% Ce programme illustre graphiquement le "fonctionnement"
% de la méthode d'itération lorsqu'elle diverge.
% La fonction dont on cherche un zéro est  y(x) =  x - sin(x) - 1
clc;clear
a=0:0.01:20;
b = 2*a - sin(a) - 1;
plot(a,b,'LineWidth',1.3)
axis([0 20 0 20]); axis equal
xlabel('\it x','FontSize',14);
ylabel('\it g = 2x - sin x - 1','FontSize',14)
title('La méthode du point fixe diverge','FontSize',14)
grid on
hold on
pause
x0=1.935;
NiterMax = 12;
plot(a,a,'LineWidth',1.3);
pause
X(1) = x0;   Y(1) = 0;
for n = 1:1:NiterMax
    g  = 2*x0 - sin(x0) - 1;
    x(n)=g;
    X(2) = x0;   Y(2) = g;
    X(3) = g;    Y(3) = g;
    plot(X,Y,'rO-','LineWidth',1.3)
    X(1) = g;    Y(1) = g;
    pause
    x0 = g;
end
pause
zoom on
format long
x'
```

Le zéro que l'on cherche vaut 1.934563210752024 à moins de 10^{-14} près (voir ci-dessous). Bien que la valeur initiale, 1.9350, soit très proche de ce zéro, le processus itératif diverge. Il en va de même si l'on prend comme valeur de départ 1.934563210752023 ou 1.934563210752025 !

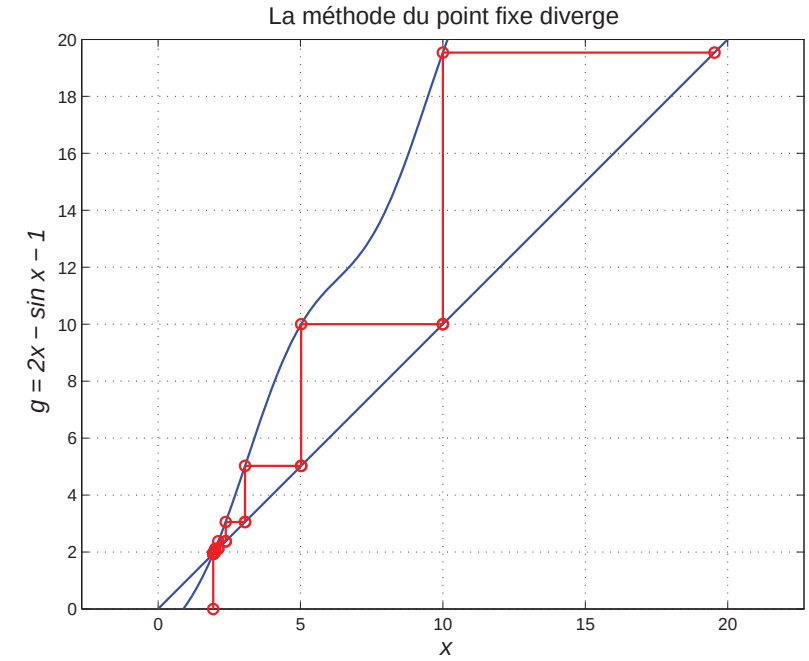


FIGURE 2.5 – La figure produite par le programme ci-dessus illustre une situation dans laquelle la méthode du point fixe diverge.

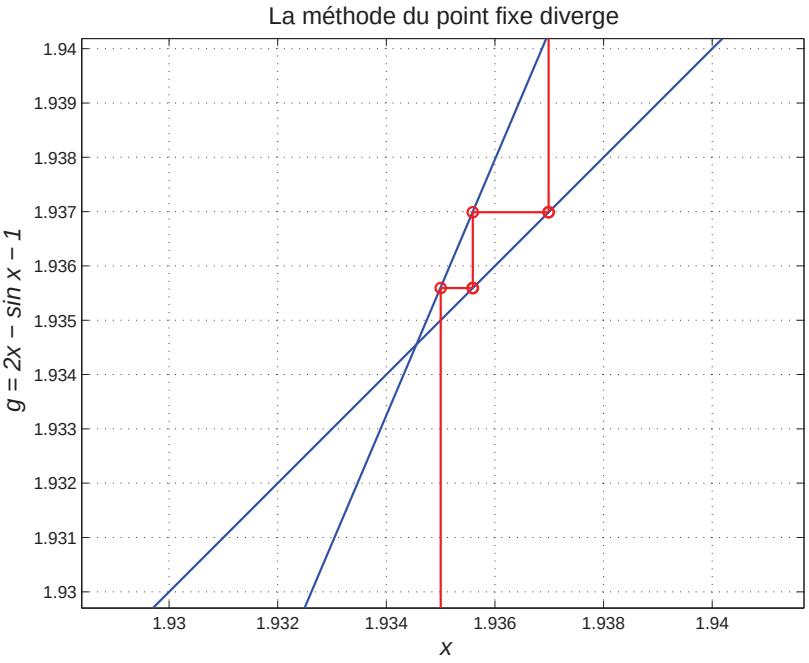


FIGURE 2.6 – Un zoom sur la figure précédente permet de voir que dès la première itération les valeurs de la suite x_n s'écartent de la position du zéro.

Pour trouver le zéro de $x - \sin x - 1$ il suffit de changer de fonction d'itération g et d'écrire $x = 1 + \sin x$ au lieu de $x = 2x - 1 - \sin x$, ce que fait le programme suivant. Ce programme calcule également un estimateur de la vitesse de convergence linéaire ainsi qu'une estimation du nombre d'itérations à effectuer pour gagner 14 chiffres exacts supplémentaires. Si on exécute ce programme on constate que la vitesse de convergence est peu dépendante de n et qu'il faut faire 32 itérations en tout, en partant de $x_1 = 2$, pour obtenir un résultat précis à moins de 10^{-14} près. Le programme donne d'ailleurs $x_{33} = 1.934563210752025$.

```
x(1)=2;
for n=1:32
    x(n+1)=1+sin(x(n));
end
format long e
% Affichage de la suite des x
x'
% Estimation de la vitesse de convergence linéaire K1
for n=1:31
    K1(n)=(x(n+2)-x(n+1))/(x(n+1)-x(n));
end
% Estimation du nombre d'itération à faire pour gagner 14 chiffres
format short
NB_IT_14=round(1+14./-log10(abs(K1)));
% Affichage des K1(n) et du nombre d'itérations à faire pour gagner 14 chiffres
[K1' NB_IT_14']
% Valeurs moyennes
[mean(K1) mean(NB_IT_14)]
```

2.6.3 Justification de la méthode d'itération

La justification théorique de la méthode du point fixe repose sur le théorème de Banach.

Point fixe

Soit r un réel et g une fonction de $\mathbb{R}$ dans $\mathbb{R}$. Si l'on a $r = g(r)$ on dit que r est un point fixe de g.

Théorème du point fixe

Soit I un intervalle fermé et borné, et une fonction d'itération $g \in C^1(I)$ telle que $g(I) \subseteq I$

1. La fonction d'itération g possède au moins un point fixe dans I.
2. Si $\max_{x \in I} |g'(x)| = K < 1$, alors ce point fixe est unique.
3. Si $\max_{x \in I} |g'(x)| = K < 1$ et si, de plus, $x_0 \in I$ et $\{x_n\}$ est définie par l'itération $x_{n+1} = g(x_n)$, pour $n = 0, 1, 2, \ldots$

alors

a) $|r - x_{n+1}| \leq K|r - x_n|$ (contraction)
b) $|r - x_n| \leq K^n|r - x_0|$ et $x_n \to r$
c) $\exists K_1(\mathbf{x}) = \lim_{n\to\infty} \dfrac{r - x_{n+1}}{r - x_n} = g'(r)$ et $|K_1(\mathbf{x})| \leq K$

Pour la démonstration de ce théorème nous renvoyons à la page 61 du livre de Dion et Gaudet[1].

Assouplissement des hypothèses

Les hyptothèses $g \in C^1(I)$ et $\max_{x\in I} |g'(x)| = K < 1$ peuvent être remplacées par les conditions plus souples $g \in C^0(I)$ et $|g(x) - g(y)| \leq K|x - y|$ pour $x, y \in I$ avec $K < 1$ permettant une généralisation du théorème ci-dessus.

2.6.4 Convergence de la méthode d'itération

Une étude détaillée de la convergence de la méthode du point fixe peut être trouvée dans Dion et Gaudet [1] pp.59 et seq.

Nous reprenons ici les point essentiels :

Soit r un point fixe de g et x_1 une valeur "proche" de r.
La méthode diverge si $|g'(r)| > 1$.
La méthode converge si $|g'(r)| < 1$ ($\neq 0$) et la convergence est linéaire avec $K_1(\mathbf{x}) = g'(r)$.

Si $g'(r) = 0$, la convergence est d'ordre P avec $K_P(\mathbf{x}) = \dfrac{g^{(P)}(r)}{(-1)^P P!}$ si $g''(r) = \cdots = g^{(P-1)}(r) = 0$, et $g^{(P)}(r) \neq 0$.

2.6.5 Du choix de la fonction d'itération g

Comme nous l'avons vu précédemment, plus $K_1(\mathbf{x})$ est proche de zéro (sans s'annuler) plus la vitesse de convergence linéaire est élevée. Il est donc être judicieux de se poser la question :

comment choisir la fonction d'itération g de façon à avoir $g'(r)$ aussi proche de zéro que possible ?

Une démarche simple permet de faire ce choix. En effet, si nous cherchons une racine r de $f(x) = 0$ en écrivant cette équation sous la forme $x = g(x)$ nous pourrons, en principe, trouver une fonction g pour laquelle $g'(r) \simeq 0$. Il suffit pour cela de choisir g comme suit :

$$x = g(x) = x + C.f(x) \tag{2.2}$$

Cette équation a les mêmes racines que $f(x) = 0$ et la constante C est un réel que l'on peut choisir de façon à annuler $g'(r)$ en posant :

$$C = \frac{-1}{f'(r)} \quad \text{avec} \quad f'(r) \neq 0 \tag{2.3}$$

Ceci est évidemment théorique[15]. En pratique, les choses sont un peu moins évidentes puisque la valeur de r est inconnue. Toutefois, on pourra remplacer la valeur théorique de C par une valeur approchée obtenue à partir d'une approximation de $g'(r)$ ou de $f'(r)$.

Exemple

Soit à calculer une approximation de la plus petite racine positive, r, de $f(x) = 1 + x \sin x$ par la méthode d'itération. On constate aisément que $f(0) > 0$ et $f(3\pi/2) < 0$ ce qui implique que la fonction $f(x)$ possède au moins un zéro dans l'intervalle $[0, 3\pi/2]$, puisque f est continue dans cet intervalle. Un graphe de l'image de cet intervalle permet de déterminer une valeur approchée du plus petit zéro positif de $f(x)$. On obtient $r \simeq 3.4368$. Ecrivons à présent l'équation sous la forme équivalente $x = x + \dfrac{-1}{f'(3.4368)}(1 + x \sin x)$ et appliquons la méthode itérative via le programme suivant :

```
% Recherche du plus petit zéro > 0 de f(x)=1+x.sin(x)
% par la méthode d'itération
f=inline('1+x.*sin(x)');
fp=inline('sin(x)+x.*cos(x)');
x(1)=3.4368;                    % Valeur de départ
C=-1/fp(x(1));                  % Constante C
for n=1:4
    x(n+1)=x(n)+C*f(x(n));
end
format long e
format compact
% Itérés successifs
x=x'
% Estimateur de la vitesse de convergence linéaire
K1=(x(4)-x(3))/(x(3)-x(2))
% Nombre de chiffres précis gagnés par itération
NB_cpgi=-log10(abs(K1))
% Sous la forme standard de g(x) : g(x)=x+f(x)
% le processus serait divergent car |g'(r)| > 1
gprime_standard = 1+fp(x(end))
```

15. Cette démarche revient à appliquer la méthode de Newton, cf. 2.7.

Les résultats obtenus sont :

```
x =
    3.436800000000000e+000
    3.436828912433360e+000
    3.436828912326676e+000
    3.436828912326677e+000
    3.436828912326677e+000
K1 =
   -8.325285557294615e-006
NB_cpgi =
    5.079600861265435e+000
gprime_standard =
   -2.579094994011832e+000
```

La vitesse de convergence est très élevée et l'on gagne environ cinq chiffres précis par itération. A la deuxième itération, le zéro est approché avec une erreur de l'ordre de 10^{-15}.

2.6.6 Evaluation du coût de la méthode d'itération

Le coût de la méthode d'itération du point fixe, par itération, est d'une assignation et d'une évaluation de fonction. Notons cependant que ceci ne nous donne pas une information pertinente quant au coût total, en temps machine, que nécessite l'obtention d'une racine particulière d'une équation donnée. Tout dépend évidemment de l'équation traitée, du choix qui est fait de la fonction d'itération ainsi que de la précision souhaitée.

2.6.7 Avantages de la méthode d'itération

Le principal avantage de la méthode d'itération réside dans sa simplicité de mise en oeuvre. De plus, on peut, la plupart du temps, trouver sans trop de difficulté une fonction d'itération qui permet une convergence très rapide.

2.6.8 Inconvénients de la méthode d'itération

Le principal inconvénient de la méthode d'itération est la lenteur de sa convergence, voire même sa divergence, si l'on utilise une fonction d'itération qui n'est pas adéquate.

2.7 Méthode de Newton (ou de la tangente)

2.7.1 Principe et algorithme de la méthode de Newton

Calculer une valeur approchée d'une racine r de l'équation $F(x) = 0$ par la méthode de Newton [16] revient à assimiler, en un point donné $P(x_1, F(x_1))$, "proche" du zéro cherché, le graphe de la fonction à celui de sa tangente en ce point. Si la fonction est "douce" au voisinage du point P son graphe diffère peu de celui de sa tangente et l'intersection de cette dernière avec l'axe des x fournit une approximation de la racine cherchée, comme on peut le voir à la figure 2.7.

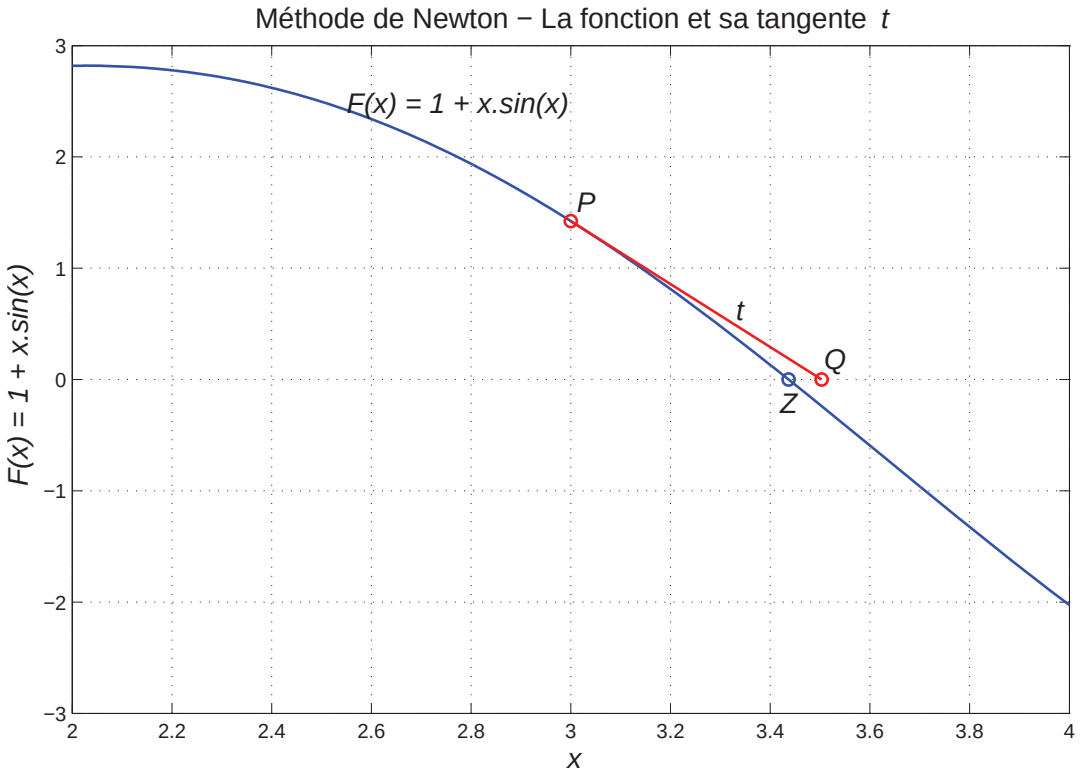


FIGURE 2.7 – Méthode de Newton ou de la tangente. Au voisinage de P, on assimile la fonction F à sa tangente. Son intersection Q avec l'axe des x fournit une approximation du zéro Z.

La traduction analytique du graphe de la figure 2.7 est la suivante :

Soit $F(x) = 0$ l'équation à résoudre. Si, dans un voisinage $[a, b]$ du zéro r que l'on cherche, la fonction F est de classe C^1 on peut écrire une équation de la tangente en P au graphe de F : $y - F(x_1) = F'(x_1).(x - x_1)$. Soit x_2 l'abscisse du point Q d'intersection de cette

16. On l'appelle aussi méthode de Newton-Raphson, notamment pour le calcul des solutions de systèmes d'équations non linéaires.

tangente avec l'axe des x. Elle s'obtient évidemment pour $y = 0$ et est donc fournie par l'équation :

$$x_2 = x_1 - F(x_1)/F'(x_1) \tag{2.4}$$

Ce qui permet d'induire l'algorithme suivant :

a) Choisir une valeur de départ x_1 "proche" du zéro cherché
b) Faire : $x_{n+1} = x_n - F(x_n)/F'(x_n)$ avec $n = 1, 2, \ldots$

Une manière alternative d'établir la formule de Newton est de remplacer $F(x)$ par son développement en série de Taylor au voisinage de x_n, "proche" de r :

$$\begin{aligned} F(x_n + \delta) &= F(x_n) + \frac{\delta}{1!}.F'(x_n) + \frac{\delta^2}{2!}.F''(x_n) + \frac{\delta^3}{3!}.F'''(x_n) + \ldots \\ &\simeq F(x_n) + \delta.F'(x_n) \end{aligned} \tag{2.5}$$

d'où l'on tire l'équation 2.4 en remplaçant δ par $x_{n+1} - x_n$ et en assimilant $F(x_n + \delta) \simeq F(r)$ à zéro.

2.7.2 Interprétation géométrique de la méthode de Newton

L'application de l'algorithme de Newton pour résoudre $F(x) = 0$ engendre une suite de tangentes au graphe de la fonction F ainsi qu'une suite de valeurs x_n. Divers cas peuvent se présenter dont quelques uns sont envisagés ci-après.

La méthode converge

Le programme suivant va nous permettre d'illustrer différentes situations que l'on peut rencontrer lorsqu'on applique la méthode de Newton à une équation qui possède de nombreux zéros simples [17].

On peut d'ailleurs facilement le modifier pour traiter n'importe quelle équation à une inconnue pour autant, bien entendu, que la fonction qui caractérise l'équation soit de classe au moins C^1 dans un voisinage de la racine cherchée.

```
% Méthode de Newton racine simple.
% Ce programme illustre graphiquement le "fonctionnement"
% de la méthode de Newton.
% Il représente une partie de la courbe d'équation  y = 5 sin x - log x - 1
% Il détermine une racine de l'équation  5 sin x - log x - 1 = 0
% A l'aide du "zoom" on peut se rendre compte de la manière dont la
% tangente à la courbe, près du point solution, se confond avec la courbe.
% NB: la présence de "pause" dans le programme nécéssite une intervention
% de l'opérateur pour que le programme continue.
```

17. Attention à la confusion : multiples zéros et zéro multiple...

```
%
clc;clear;clf
a=0.001:0.001:5;
b = 5*sin(a)-log(a)-1;
plot(a,b,'LineWidth',1.3)
xlabel('\it x','FontSize',14); ylabel('\it f(x) = 5 sin x - log x - 1','FontSize',14)
title('Méthode de Newton ou de la tangente','FontSize',14)
grid on
hold on
pause
x0=4.1;
NiterMax = 4;
for n = 1:1:NiterMax
    f  = 5*sin(x0) - log(x0) - 1;
    fp = 5*cos(x0) - 1/x0;
    x(n) = x0 - f/fp;
    X(1) = x0;   Y(1) = 0;
    X(2) = x0;   Y(2) = f;
    X(3) = x(n); Y(3) = 0;
    plot(X,Y,'rO-','LineWidth',1.3)
    pause
%     textx=['\it x' num2str(n)];
%     gtext(textx,'FontSize',12)
%     textP=['\it P' num2str(n)];
%     gtext(textP,'FontSize',12)
    x0 = x(n);
end
pause
format long
x'
```

Le premier graphe (figure 2.8), construit à partir de ce programme, illustre la convergence de la méthode de Newton vers un zéro proche de 2.73, à partir de la valeur initiale 4.1 en seulement 4 itérations. Ce graphe permet de constater que la courbe et sa tangente sont quasi confondues après seulement trois itérations ($x_4 = 2.738115\ldots$) bien que l'abscisse de départ vaille 4.1.
Un zoom sur cette figure confirme d'ailleurs cette constatation comme on peut le voir sur la figure 2.9.

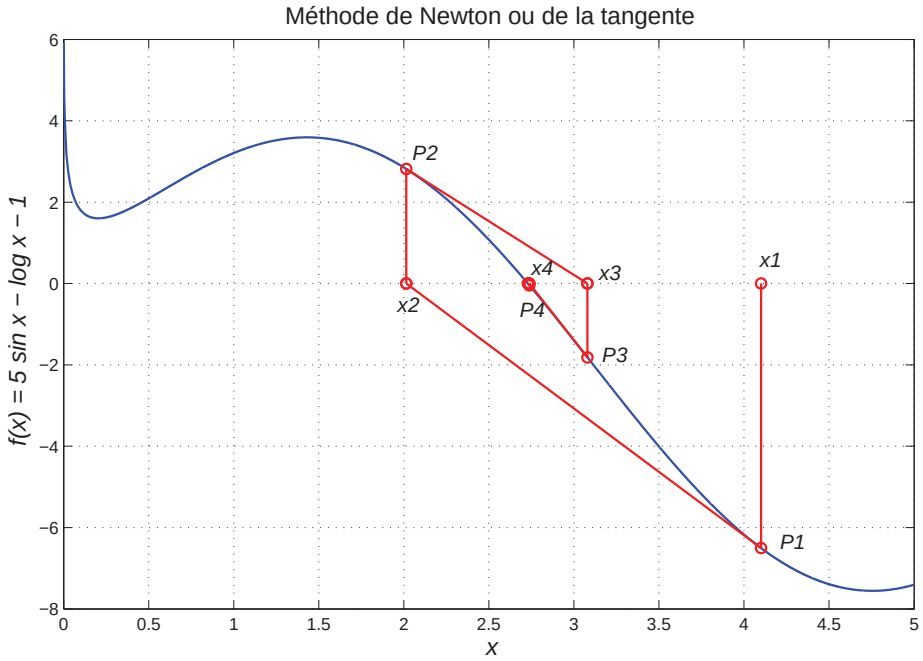


FIGURE 2.8 – La méthode de Newton converge vers un zéro de $f(x) = 5\sin x - \log x - 1$ proche de 2.73.

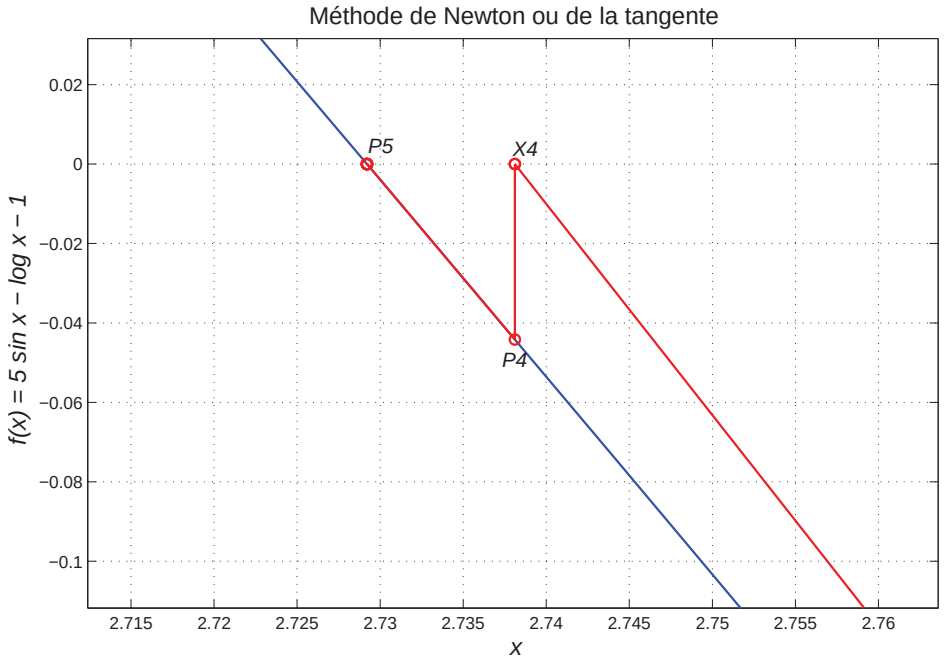


FIGURE 2.9 – Zoom de la figure 2.8. Au voisinage de P_4 la tangente est quasi confondue avec le graphe de $f(x)$.

L'application du programme précédent en prenant comme valeur de départ $x_0 = 4.2$ conduit au graphe de la figure 2.10. Après 8 itérations le processus semble converger vers $x_9 = 64.28649...$

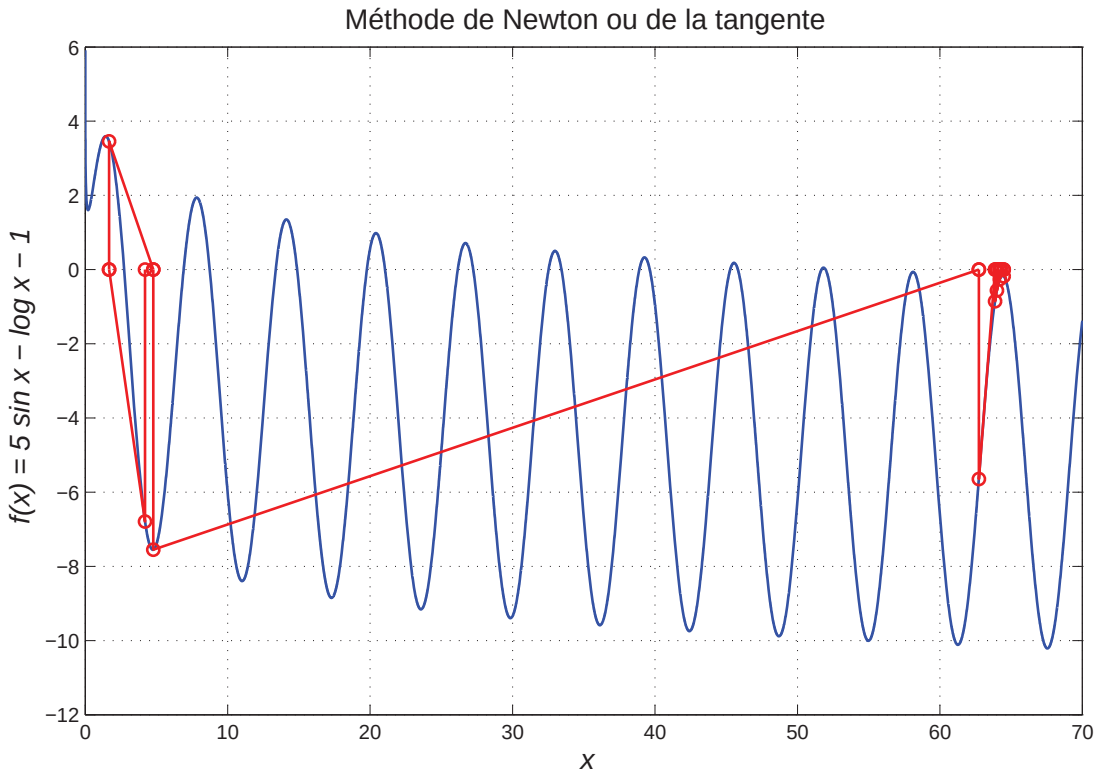


FIGURE 2.10 – Si on prend comme valeur de départ $x = 4.2$, après 8 itérations le processus semble converger vers $x_9 = 64.28649...$

En réalité, la fonction ne possède pas de zéro au voisinage de $x = 64$ et si, pour la même valeur initiale du processus itératif, l'on effectue 54 itérations au lieu de 8, on obtient un nouveau graphe, représenté à la figure 2.11. Le zéro trouvé est bien loin du point de départ puisqu'il vaut $x_{55} = 51.6881272625610\ldots$, tous les chiffres affichés étant exacts.

Des agrandissements de la figure 2.11 dans les domaines $[63.9, 64.8]$ (cf. figure 2.12) et $[51.2, 52.1]$ (cf. figure 2.13) permettent de visualiser la manière dont la suite des x_n a évolué.

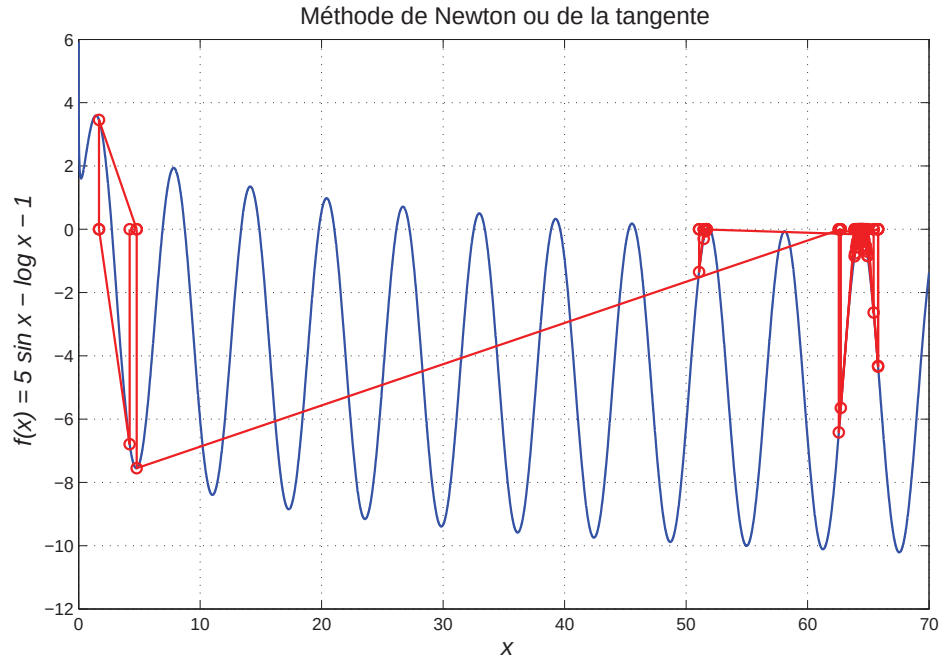


FIGURE 2.11 – Si on prend comme valeur de départ $x = 4.2$, après 54 itérations le processus a convergé vers 51.6881272625610.

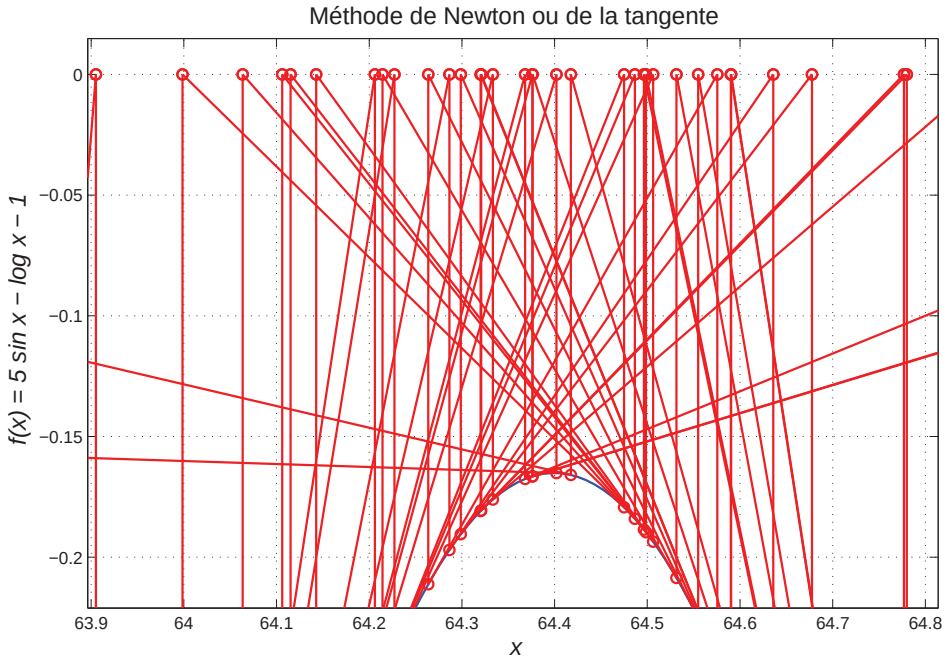


FIGURE 2.12 – Un zoom sur l'intervalle $[63.9, 64.8]$ montre que la fonction ne s'annule pas dans cette zone.

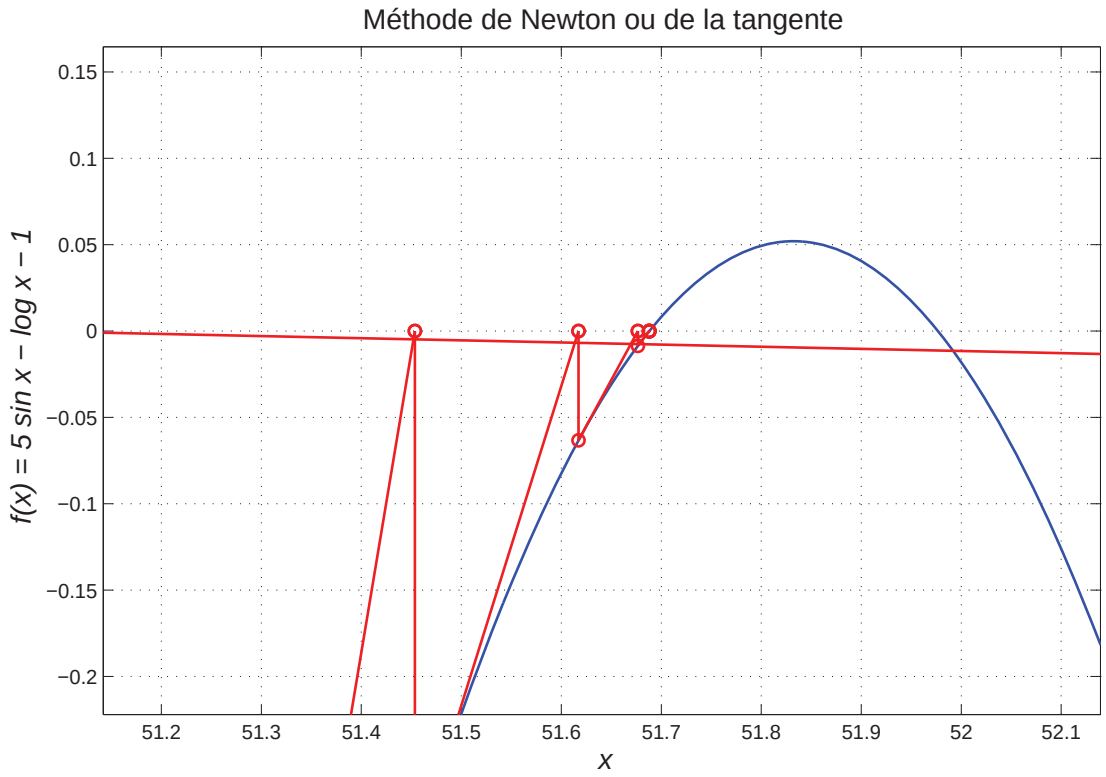


FIGURE 2.13 – L'inspection de l'intervalle $[51.2, 52.1]$ montre où aboutit la suite des itérés de Newton : en $x_{55} = 51.6881272625610\ldots$

2.7.3 Newton avec ou sans complexes ?

L'équation $5\sin x - \log x - 1 = 0$ possède 17 zéros réels simples. Elle possède également des racines complexes [18] dont une dizaine d'entre-elles est donnée ci-dessous. La méthode de Newton fournit aisément ces racines si on utilise des valeurs négatives de x_1 pour initier le processus itératif. Notons également qu'à chaque racine complexe correspond son complexe conjugué. Si l'on classe les racines complexes dans l'ordre croissant de leur module il semble, à première vue, qu'à partir d'un certain rang, les parties réelles de deux racines complexes consécutives soient séparées d'à peu près 2π.

```
 -0.02397168807685 - 0.32338608069695i
 -5.80860767554988 - 0.63769089201116i
-11.97306948958522 - 0.68832405385070i
-18.18895563945838 - 0.72144521189010i
-24.42583111971792 - 0.74688651791948i
-30.67410589366283 - 0.76778144177470i
-36.92951394177416 - 0.78560135983540i
-43.18978758629538 - 0.80117623285305i
-49.45357967168683 - 0.81502806443427i
-55.72002616003292 - 0.82750987350185i
```

2.7.4 Racines multiples

Lorsque la fonction F admet une racine multiple r, sa dérivée s'annule en r. Dans ces conditions, F et F' prennent des valeurs très petites au voisinage de r. Il en résulte que la convergence de la méthode de Newton est beaucoup plus lente que dans le cas d'une racine simple. Elle perd son caractère quadratique [19] et devient linéaire (cf.page 54). Il y a cependant moyen d'accélérer la convergence en introduisant un coefficient multiplicateur dans la formule de Newton [20].

Théorème

Soit L un coefficient réel et x_n une suite générée par la méthode de Newton pondérée à partir de x_1 choisi

$$x_{n+1} = x_n - L.F(x_n)/F'(x_n) \quad n = 1, 2, \ldots \tag{2.6}$$

Supposons que la suite x_n converge vers la racine r de multiplicité M et que $K_1(\mathbf{x}) = \lim_{n\to\infty} \dfrac{r - x_{n+1}}{r - x_n}$ existe, alors $K_1(\mathbf{x}) = 1 - L/M$ si $F \in C^{M+2}$ dans un certain voisinage de r.

Ce théorème, permet d'accélérer la convergence de la méthode de Newton dans le cas des racines multiples au moyen d'une stratégie adaptée.

18. Il y en a vraisemblablement une double infinité.
19. Ou d'ordre plus élevé que 2.
20. Un tel coefficient est souvent appelé coefficient ou paramètre de relaxation.

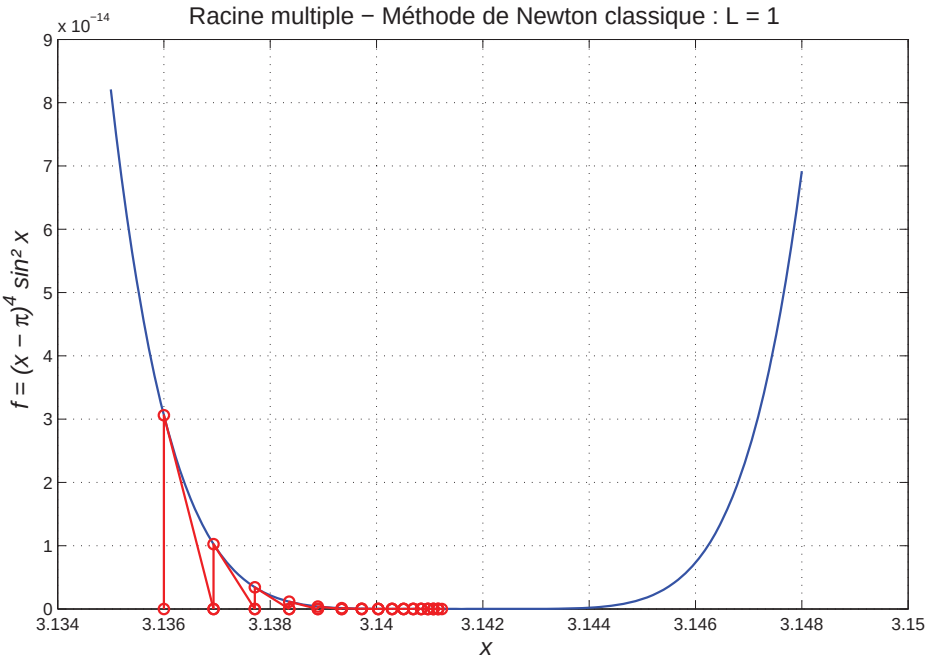


FIGURE 2.14 – Recherche du zéro multiple de $f(x) = (x-\pi)^4 \sin^2 x$. Interprétation graphique. Notez l'échelle des ordonnées : 10^{-14} !

On commence par appliquer la méthode sous sa forme standard, c'est-à-dire en faisant $L = 1$ dans la formule (2.6). On calcule ensuite l'estimateur $K_1(\mathbf{x}, n) = 1 - L/M^* \simeq K_1(\mathbf{x}) = 1 - L/M$. On en déduit l'entier M^{**} le plus proche de M^* que l'on assimile à L et on poursuit les calculs en appliquant l'algorithme pondéré (2.6) avec $L = M^{**}$.

Exemple 6

La fonction $f(x) = (x-\pi)^4 \sin^2 x$ admet la racine sextuple $x = \pi$.
Avec $x_1 = 3.136$, la méthode de Newton classique (non pondérée) donne le graphe ci-dessous et les 15 premières approximations du zéro multiple π sont :

1	3.13693211217102	6	3.13971968900098	11	3.14083995128939
2	3.13770887094878	7	3.14003184988746	12	3.14096540168069
3	3.13835616914047	8	3.14029198390826	13	3.14106994367011
4	3.13889558384317	9	3.14050876222926	14	3.14115706199270
5	3.13934509583092	10	3.14068941081293	15	3.14122966059375

On constate que la convergence est très lente. On a $K_1(\mathbf{x}, 13) = 0.83333332060902$ ce qui montre qu'il faut faire environ 13 itérations pour gagner un chiffre et donne également $M^* = 5.99999954192482$ d'où l'on déduit $L = M^{**} = 6$.

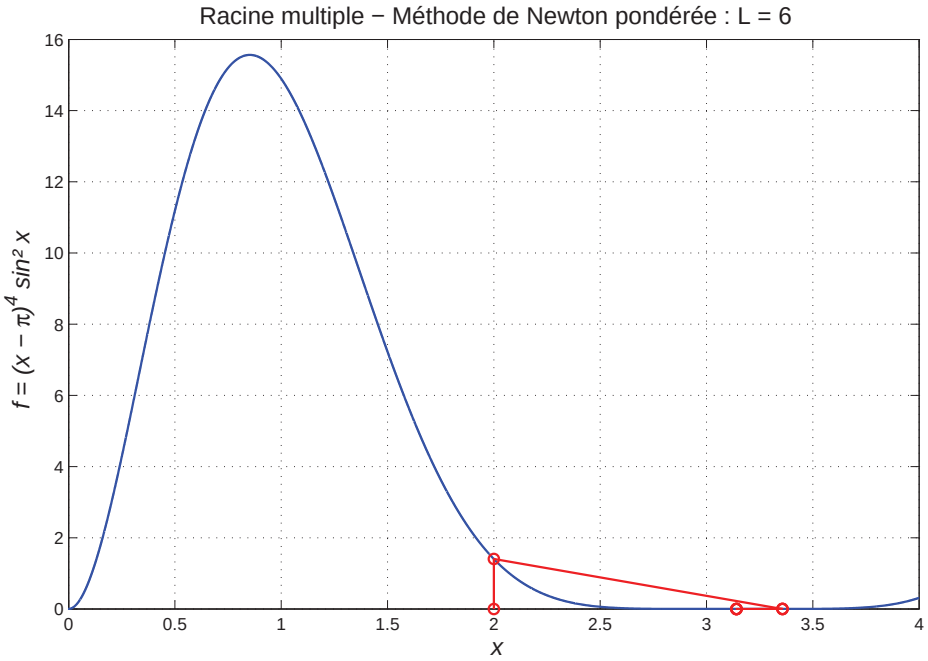


FIGURE 2.15 – Recherche du zéro multiple de $f(x) = (x - \pi)^4 \sin^2 x$. Interprétation graphique : sous l'effet du coefficent de relaxation L, la "tangente" n'est plus tangente !

Le même problème résolu à l'aide de l'algorithme de Newton pondéré avec $L = 6$ produit les approximations suivantes :

```
3.14159267302610
3.14159265358979
             NaN
```

NB : Le calcul montre que $K_1(\mathbf{x}, 1) = 0.83333231657359$, il n'était donc pas nécessaire de faire 15 itérations pour déterminer que π est racine sextuple. En effet, lorsque la méthode classique converge quadratiquement, $K_1(\mathbf{x}, n)$ est proche de zéro.

Avec $x_1 = 2$, les approximations obtenues sont :

```
3.35771429268940
3.14046162027139
3.14159265375056
3.14159265358979
             NaN
```

2.7.5 Convergence de la méthode de Newton

Les théorèmes de convergence de la méthode d'itération du point fixe peuvent être appliqués à la méthode de Newton. En effet, les algorithmes :

$$x_1 = a \tag{2.7}$$

$$x_{n+1} = x_n - L.F(x_n)/F'(x_n); \quad n = 1, 2, \ldots \tag{2.8}$$

et

$$x_1 = a \tag{2.9}$$

$$x_{n+1} = g(x_n); \quad n = 1, 2, \ldots \tag{2.10}$$

sont équivalents, à condition de choisir :

$$g(x) = x - L.F(x)/F'(x) \tag{2.11}$$

Racine simple

Si la racine cherchée est simple la convergence est ordre P, avec $f'(r) \neq 0, f^{(i)}(r) = 0,$ pour $2 \leq i \leq P-1$ et $f^{(P)}(r) = 0$.

Racine multiple

Si la racine cherchée est multiple, la convergence est linéaire, avec $K_1(\mathbf{x}) = 1 - 1/M$ où M est la multiplicité de la racine.

Il en résulte que plus la multiplicité de la racine est élevée plus la convergence est lente, au point de rendre la méthode inutilisable en partique, du moins sous sa forme standard.

2.7.6 Approximation de la dérivée

Pour calculer un zéro de $F(x)$ par la méthode de Newton il faut connaître la dérivée $F'(x)$. Il peut arriver qu'on ne connaisse pas cette dérivée ou qu'il soit difficle de la calculer.

C'est le cas par exemple pour $F(x) = 1 + \log|(\sin(x^{\arctan(\sinh(x^3 - 7\sec(\pi - 3x^{2/3})))}))|$.

Un moyen de se tirer d'affaire est de remplacer $F'(x)$ par une approximation basée sur la définition de la dérivée :

$$F'(x) = \lim_{h \to 0} \frac{F(x+h) - F(x)}{h} \tag{2.12}$$

en choisissant $x = x_n$ et h "petit".

On peut prendre, par exemple, $h = \omega(x_n - x_{n-1})$ $(n > 1)$ où ω est un coefficient de relaxation.

L'algorithme de Newton est alors remplacé par une version approchée :

$$x_1 = a \tag{2.13}$$

$$x_{n+1} = x_n - h.F(x_n)/(F(x_n + h) - F(x_n)); \quad n = 1, 2, \ldots \tag{2.14}$$

2.7.7 Evaluation du coût de la méthode de Newton

Chaque itération de la méthode de Newton requiert une assignation, une soustraction, une division et deux évaluations de fonction. Ceci est évidemment indicatif et les considérations énoncées précédemment restent en vigueur 2.3.3.

2.7.8 Avantages de la méthode de Newton

Le principal avantage de la méthode résulte de sa convergence au moins quadratique dans le cas des racines simples.
Les hypothèses de départ sont généralement assez peu contraignantes pour une fonction qui admet peu de zéros simples.
La méthode est peu sujette aux instabilités numériques.
L'algorithme reste d'application pour la détermination de zéros complexes de fonctions réelles.

2.7.9 Inconvénients de la méthode de Newton

Le prinipal inconvénient de la méthode de Newton est la nécessité de calculer la dérivée de la fonction dont on cherche un zéro.
La méthode de Newton peut s'avérer globalement plus lente que la méthode de la sécante en raison des évaluations de fonctions.
Lorsqu'il y a des racines multiples la convergence peut devenir très lente (sauf « aménagements »).

3. RESOLUTION DE SYSTEMES ALGEBRIQUES LINEAIRES. METHODES DIRECTES.

Méthode directe ≡ Méthode exacte «en algèbre»

3.1. SYSTÈME ALGÉBRIQUE LINÉAIRE

$$
\begin{array}{l}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1j}x_j + \ldots + a_{1n}x_n = b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2j}x_j + \ldots + a_{2n}x_n = b_2 \\
\quad . \quad . \quad . \quad \ldots \quad . \quad \ldots \quad . \quad . \\
\quad . \quad . \quad . \quad \ldots \quad . \quad \ldots \quad . \quad . \\
\quad . \quad . \quad . \quad \ldots \quad . \quad \ldots \quad . \quad . \\
a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + \ldots + a_{ij}x_j + \ldots + a_{in}x_n = b_i \\
\quad . \quad . \quad . \quad \ldots \quad . \quad \ldots \quad . \quad . \\
\quad . \quad . \quad . \quad \ldots \quad . \quad \ldots \quad . \quad . \\
\quad . \quad . \quad . \quad \ldots \quad . \quad \ldots \quad . \quad . \\
a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nj}x_j + \ldots + a_{nn}x_n = b_n
\end{array}
$$

On se limite ici aux systèmes carrés réels : n équations à n inconnues.

Données (a_{ij}) , $i,j=1,2,\ldots,n$ et b_i , $i=1,2,\ldots,n$.

Inconnues x_j , $j=1,2,\ldots,n$.

3.2. FORME CONDENSÉE – FORME MATRICIELLE

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1,2,...,n. \quad \text{ou} \quad \mathbf{Ax = b}$$

A : matrice carrée des a_{ij} $(n \times n)$
x et **b** : vecteurs colonne x_j et b_j $(n \times 1)$

3.3. MÉTHODE DU DÉTERMINANT

$$\mathbf{Ax = b} \quad \Rightarrow \quad \mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b} = (\det \mathbf{A})^{-1}(\mathbf{A}^{*})^{t}\,\mathbf{b}$$

Avec $\mathbf{A}^{*} = (a^{*}_{ij})$ $i,j = 1,2,\ldots,n$ la matrice des cofacteurs des a_{ij} : $a^{*}_{ij} = (-1)^{i+j}$ dét $\mathbf{A}_{ij}$, où

les sous-matrices $\mathbf{A}_{ij}$ de la matrice **A** sont les mineurs des a_{ij} .

$\Rightarrow$ $n^2\, n\,! + (n+1)\,! + n^3 - 1 + n(4n-1)$

$\cong$ $(n+2)\,!$ opérations arithmétiques $\Rightarrow$ impraticable

Question :

un ordinateur peut effectuer 10 milliards d'opérations par seconde, combien de temps lui faudrait-il pour résoudre un système de 20 équations à 20 inconnues ?

3.4. MÉTHODE DE CRAMER

⇒ *Encore pire…*

3.5. SYSTÈMES TRIANGULAIRES

Système à matrice triangulaire inférieure :

$a_{11}x_1 = b_1$
$a_{21}x_1 + a_{22}x_2 = b_2$
$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$
.
.
.
$a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + \ldots + a_{ii}x_i = b_i$
.
.
.
$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{ni}x_i + \ldots + a_{nn}x_n = b_n$

$\Rightarrow$ $x_1 = b_1/a_{11}$, $x_2 = (b_2 - a_{21}x_1)/a_{22}$, … $\Rightarrow$ $x_i = \dfrac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j}{a_{ii}}$, … $x_n = \dfrac{b_n - \sum_{j=1}^{n-1} a_{nj}x_j}{a_{nn}}$

⇒ élimination par descente n'est possible que si dét $\mathbf{A} \neq 0$ (a_{ii} non nuls).

Coût de la méthode

Nombre d'opérations à effectuer pour le calcul de x_i (il y en a n en tout) :

(i – 1) multiplications
(i – 1) additions
1 division

$\Rightarrow$ $\sum_{i=1}^{n}(2i-1) = 2\sum_{i=1}^{n} i - n = 2\dfrac{n(n+1)}{2} - n = n^2$ → *n^2 opérations arithmétiques en tout.*

Système à matrice triangulaire supérieure :

$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1i}x_i + \ldots + a_{1n}x_n = b_1$
$a_{22}x_2 + a_{23}x_3 + \ldots + a_{2i}x_i + \ldots + a_{2n}x_n = b_2$
…
$a_{ii}x_i + \ldots + a_{in}x_n = b_i$
$a_{nn}x_n = b_n$

⇒ élimination en remontant de x_n à x_1 , dét $\mathbf{A} \neq 0$!
⇒ *n^2 opérations arithmétiques en tout.*

ALGORITHMES

Système à matrice triangulaire inférieure

Pour k = 1, 2, 3, …, n : calculer

$$x_k = \frac{b_k - \sum_{j=1}^{k-1} a_{kj} x_j}{a_{kk}} \quad , \quad a_{kk} \neq 0 \ , \quad (\text{dét } \mathbf{A} \neq 0)$$

Système à matrice triangulaire supérieure

Pour k = n, n-1, n-2 , …, 1 : calculer

$$x_k = \frac{b_k - \sum_{j=k+1}^{n} a_{kj} x_j}{a_{kk}} \quad , \quad a_{kk} \neq 0 \ , \quad (\text{dét } \mathbf{A} \neq 0)$$

- Facile à programmer
- Peu d'accumulation d'erreurs d'arrondi
- n^2 opérations arithmétiques en tout

3.6. MÉTHODE DE GAUSS

Par combinaisons linéaires successives des lignes du système, on triangularise supérieurement sa matrice. On travaille sur une matrice augmentée d'une colonne, le vecteur **b.**

Exemple

$$\begin{pmatrix} 3 & -1 & 0 \\ -2 & 1 & 1 \\ 2 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \\ 15 \end{pmatrix} \Rightarrow \left(\begin{array}{ccc|c} 3 & -1 & 0 & 5 \\ -2 & 1 & 1 & 0 \\ 2 & -1 & 4 & 15 \end{array}\right) \left(\begin{array}{ccc|c} 3 & -1 & 0 & 5 \\ -2 & 1 & 1 & 0 \\ 2 & -1 & 4 & 15 \end{array}\right) \begin{array}{l} \\ +(2/3)\ \textit{ligne}\ 1 \\ -(2/3)\ \textit{ligne}\ 1 \end{array}$$

$$\Rightarrow \left(\begin{array}{ccc|c} 3 & -1 & 0 & 5 \\ 0 & 1/3 & 1 & 10/3 \\ 0 & -1/3 & 4 & 35/3 \end{array}\right) \begin{array}{l} \\ \\ +\ \textit{ligne}\ 2 \end{array} \qquad \Rightarrow \left(\begin{array}{ccc|c} 3 & -1 & 0 & 5 \\ 0 & 1/3 & 1 & 10/3 \\ 0 & 0 & 5 & 15 \end{array}\right)$$

Elimination ascendante (back - substitution)

$x_3 = 15/5 \qquad = 3$
$x_2 = (10/3 - 1.x_3)/(1/3) \qquad = 1$
$x_1 = (5 - (-1.x_2 + 0.x_3))/3 \qquad = 2$

CAS GÉNÉRAL - SYSTÈME N×N

Soit **A** la matrice :

$$\begin{matrix} a_{11} & a_{12}\dots & a_{1j}\dots & a_{1n} \\ a_{21} & a_{22}\dots & a_{2j}\dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{i1} & a_{i2}\dots & a_{ij}\dots & a_{in} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2}\dots & a_{nj}\dots & a_{nn} \end{matrix}$$

pour annuler l'élément en position (i,1) il faut soustraire de la i^{ème} ligne, la première ligne multipliée par $m_{i1} = a_{i1}/a_{11}$

Annuler tous les éléments de la première colonne de **A**, sous a_{11}, revient à prémultiplier la matrice **A** par la matrice identité dans laquelle les éléments de la première colonne, à partir du deuxième, sont remplacés par $-m_{i1}$,

soit $\mathbf{T}_1$ cette matrice :

$$\mathbf{T}_1 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ -m_{21} & 1 & 0 & \cdots & 0 \\ -m_{31} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -m_{n1} & 0 & 0 & \cdots & 1 \end{pmatrix}$$

a_{11} est appelé « élément pivot de la transformation $\mathbf{T}_1$

On obtient un nouveau système équivalent au système de départ : $\mathbf{T}_1\mathbf{A}\boldsymbol{x} = \mathbf{T}_1\boldsymbol{b}$ car $\mathbf{T}_1$ est inversible.

$$T_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{21} & 1 & 0 & \cdots & 0 \\ m_{31} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & 0 & 0 & \cdots & 1 \end{pmatrix}$$

$$\mathbf{T}_1\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}, \qquad \mathbf{T}_1\boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ b_3^{(1)} \\ \vdots \\ b_n^{(1)} \end{pmatrix}$$

Les seuls éléments à calculer sont les

$$a_{ij}^{(1)}, \quad i, j = 2,3,...,n; \quad et \quad b_i^{(1)}, \quad i = 2,3,...,n.$$

on a $$\left\{ \begin{array}{l} m_{i1} = a_{i1} / a_{11} \ , \quad i = 2,3,...,n \\ b_i^{(1)} = b_i - m_{i1} b_1 \ , \ i = 2,3,...,n \\ a_{ij}^{(1)} = a_{ij} - m_{i1} a_{1j} \ , i, j = 2,3,...,n \end{array} \right\}$$

Pour annuler les éléments de la deuxième colonne de $\mathbf{T}_1\mathbf{A}$ qui sont sous la diagonale, il suffit d'appliquer à la sous-matrice $\mathbf{A}^{(1)}$ extraite de $\mathbf{T}_1\mathbf{A}$, le procédé utilisé pour obtenir $\mathbf{T}_1\mathbf{A}$.

Le pivot de cette nouvelle transformation est l'élément $a_{22}^{(1)}$.

De chaque ligne i (i=3,4,…,n) il faut soustraire la deuxième ligne multipliée par

$$m_{i2} = a_{i2}^{(1)} / a_{22}^{(1)}$$

La nouvelle matrice de transformation s'écrit :

$$\mathbf{T}_2 \quad = \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -m_{32} & 1 & 0 & 0 & 0 \\ 0 & -m_{42} & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -m_{n2} & 0 & 0 & 0 & 1 \end{pmatrix}$$

On obtient le nouveau système équivalent : $\mathbf{T}_2\mathbf{T}_1\mathbf{A}\boldsymbol{x} = \mathbf{T}_2\mathbf{T}_1\boldsymbol{b}$

avec

$$\mathbf{T}_2\mathbf{T}_1\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{32}^{(2)} & a_{34}^{(2)} & \cdots & a_{3n}^{(2)} \\ 0 & 0 & a_{43}^{(2)} & a_{44}^{(2)} & \cdots & a_{4n}^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(2)} & a_{n4}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix} \qquad \text{et} \qquad \mathbf{T}_2\mathbf{T}_1\boldsymbol{b} \quad = \quad \begin{pmatrix} b_1 \\ b_2^{(1)} \\ b_3^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix}$$

Les seuls éléments à calculer sont cette fois :

$a_{ij}^{(2)}, \quad i, j = 3,4,...,n; \quad et \quad b_i^{(2)}, \quad i = 3,4,...,n.$

selon les formules

$$\left\{ \begin{array}{l} m_{i2} = a_{i2}^{(1)} / a_{22}^{(1)} \ , \quad i = 3,4,...,n \\ b_i^{(2)} = b_i^{(1)} - m_{i2} b_2^{(1)} \ , \ i = 3,4,...,n \\ a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i2} a_{2j}^{(1)} \ , i, \ j = 3,4,...,n \end{array} \right\}$$

Par application de n-1 transformations du type de $\mathbf{T}_1$ et $\mathbf{T}_2$ on obtient finalement, pour autant que les pivots successifs soient non nuls, le système :

$\mathbf{T}_{n-1}\mathbf{T}_{n-2}\ldots\mathbf{T}_2\mathbf{T}_1\mathbf{A}\boldsymbol{x} = \mathbf{T}_{n-1}\mathbf{T}_{n-2}\ldots\mathbf{T}_2\mathbf{T}_1\boldsymbol{b}$

équivalent au système $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, mais dont la matrice est *triangulaire supérieure* !

La méthode de Gauss revient donc

à poser $\mathbf{A} = \mathbf{A}^{(0)}$, $\boldsymbol{b} = \boldsymbol{b}^{(0)}$ puis à calculer successivement les quantités suivantes :

$$\left\{ \begin{array}{l} m_{ik} = a_{ik}^{(k-1)} / a_{kk}^{(k-1)} \ , \quad i = k+1, k+2,..., n \\ b_i^{(k)} = b_i^{(k-1)} - m_{ik} b_k^{(k-1)} \ , \ i = k+1, k+2,..., n \\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - m_{ik} a_{kj}^{(k-1)} \ , i, \ j = k+1, k+2,..., n \end{array} \right\}$$

pour k = 1,2,…,n-1

et à appliquer enfin la phase d'élimination ascendante au système triangulaire résultant.

Coût de la méthode - Nombre d'opérations à effectuer

A l'étape k il faut calculer les

(n-k) termes m_{ik} ; $\Rightarrow$ (n-k) divisions
(n-k) termes $b_i^{(k)}$; $\Rightarrow$ (n-k) additions
$\Rightarrow$ (n-k) multiplications
$(n-k)^2$ termes $a_{ij}^{(k)}$; $\Rightarrow$ $(n-k)^2$ additions
$\Rightarrow$ $(n-k)^2$multiplications

Pour l'ensemble des n-1 étapes, il faut faire

$\sum_{k=1}^{n-1}(2(n-k)(n-k+1)+n-k)=\sum_{i=1}^{n-1}(2i(i+1)+i)$ opérations soit $\frac{2}{3}n^3+\frac{1}{2}n^2-\frac{7}{6}n$ auxquelles

il faut encore ajouter n^2 opérations pour l'élimination ascendante.

La méthode de Gauss exige donc finalement le calcul de $\frac{2}{3}n^3+\frac{3}{2}n^2-\frac{7}{6}n$ opérations en tout.

Tout ceci n'est évidemment valable que si les pivots successifs des transformations $\mathbf{T}_i$ sont non nuls.

Si, au cours des calculs, on « rencontre » un pivot nul, il faut permuter la ligne de ce pivot avec une ligne d'indice supérieur, présentant un pivot non nul dans la même colonne. Ceci sera toujours possible si le système donné est compatible (dét$\mathbf{A} \neq 0$).
En pratique, on ne permute pas les lignes entre-elles, on change simplement le numéro des lignes à permuter en stockant dans un vecteur de travail les indices des pivots successifs. Pour la « back-substitution » on tient compte de l'ordre utilisé lors de la triangularisation.

DÉTERMINANT DE LA MATRICE A

Théorème

Soit A une matrice n×n. Son déterminant est égal, au signe près, au produit des pivots issus de la méthode Gauss. On a :

dét A = $(-1)^p a_{11}a_{22}^{(1)}a_{33}^{(2)}\cdots a_{kk}^{(k-1)}\cdots a_{nn}^{n-1}$

où p est le nombre de permutations de lignes effectuées.

MATRICE INVERSE DE A

Soit **I** la matrice identité (n×n) et $\boldsymbol{e}^k$ sa k^{ième} colonne, c'est à dire un vecteur de taille n, dont toutes les composantes sont nulles sauf la k^{ième} qui vaut 1.

Le système $\mathbf{A}\boldsymbol{x}^k = \boldsymbol{e}^k$ admet la solution $\boldsymbol{x}^k = \mathbf{A}^{-1}\boldsymbol{e}^k$

Comme $\boldsymbol{x}^k$ est la k^{ième} colonne de $\mathbf{A}^{-1}$ on peut obtenir $\mathbf{A}^{-1}$ en résolvant les n systèmes $\mathbf{A}\boldsymbol{x}^k = \boldsymbol{e}^k$.

3.7. FACTORISATION **LU** DE LA MATRICE **A**

La méthode de Gauss fournit un système à matrice triangulaire supérieure :

$\mathbf{T}_{n-1}\mathbf{T}_{n-2}\ldots\mathbf{T}_2\mathbf{T}_1\mathbf{A}\boldsymbol{x} = \mathbf{T}_{n-1}\mathbf{T}_{n-2}\ldots\mathbf{T}_2\mathbf{T}_1\boldsymbol{b} = \boldsymbol{b}_1$

Posons $\mathbf{T}_{n-1}\mathbf{T}_{n-2}\ldots\mathbf{T}_2\mathbf{T}_1\mathbf{A} = \mathbf{U}$

Comme les $\mathbf{T}_i$ sont triangulaires inférieures à diagonale unitaire le produit $\mathbf{T}_{n-1}\mathbf{T}_{n-2}\ldots\mathbf{T}_2\mathbf{T}_1$ est aussi triangulaire inférieure à diagonale unitaire et l'on a :

$\mathbf{A} = (\mathbf{T}_{n-1}\mathbf{T}_{n-2}\ldots\mathbf{T}_2\mathbf{T}_1)^{-1}\mathbf{U} = \mathbf{LU}$

où $\mathbf{L} = (\mathbf{T}_{n-1}\mathbf{T}_{n-2}\ldots\mathbf{T}_2\mathbf{T}_1)^{-1}$ est une matrice

triangulaire inférieure à diagonale unitaire $\Rightarrow$

Si on connaît une factorisation (on dit aussi décomposition) **LU** de la matrice **A** du système $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, sa solution s'obtient de la façon suivante :

1°) on calcule la solution du système $\mathbf{L}\boldsymbol{y} = \boldsymbol{b}$
2°) on résout ensuite $\mathbf{U}\boldsymbol{x} = \boldsymbol{y}$

En effet : $\mathbf{A}\boldsymbol{x} = \mathbf{LU}\boldsymbol{x} = \mathbf{L}\boldsymbol{y} = \boldsymbol{b}$

La connaissance de la décomposition **LU** permet de résoudre le système de départ au prix de $2n^2$ opérations arithmétiques (*supplémentaires*) puisqu'il faut résoudre deux systèmes triangulaires à n inconnues.

FACTORISATION **LU** (*sans utiliser Gauss*)

On peut procéder à la décomposition en facteurs **L** et **U** par identification des éléments des deux matrices inconnues **L** et **U**, les a_{ij} étant des données.

Soit $\mathbf{A} = \mathbf{LU}$ avec $\mathbf{L} = (l_{ij})$ triangulaire inférieure à diagonale unitaire, c'est-à-dire :

$l_{ii} = 1, \quad i = 1, 2, \ldots, n \quad$ et $\quad l_{ij} = 0 \ \forall \ j > i$

et $\mathbf{U} = (u_{ij})$ triangulaire supérieure, soit : $u_{ij} = 0 \quad \forall \quad j < i.$

Donc, pour tout i, j on doit avoir :

$$a_{ij} = (\mathbf{LU})_{ij} = \sum_{k=1}^{n} l_{ik}u_{kj} = \sum_{k=1}^{i} l_{ik}u_{kj} = \sum_{k=1}^{\min(i,j)} l_{ik}u_{kj}$$

Ces formules permettent de calculer les lignes de **U** et les colonnes de **L**, en séquence, comme suit :

pour s = 1, 2, …, n

1°) calculer la s^{ième} ligne de **U** :

pour j = s, s+1,…, n calculer

$$u_{sj} = a_{sj} - \sum_{k=1}^{s-1} l_{sk} u_{kj}$$

2°) calculer la s^{ième} colonne de **L** :

poser $l_{ss} = 1$ et

pour i = s+1, s+2,…, n, calculer

$$l_{is} = \frac{1}{u_{ss}} \left\{ a_{is} - \sum_{k=1}^{s-1} l_{ik} u_{ks} \right\}$$

REMARQUES

Le coût de la factorisation **LU** est du même ordre que celui de la triangularisation de Gauss soit $2n^3/3$.

La méthode ne fonctionne qu'avec les matrice carrées régulières (dét **A** ≠ 0).

La méthode fournit la matrice inverse de **A** grâce à la relation $\mathbf{A}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$.

La matrice **U** est identique à la matrice triangulaire fournie par la méthode de Gauss

La matrice $\mathbf{L} = (\mathbf{T}_{n-1}\mathbf{T}_{n-2}\ldots\mathbf{T}_2\mathbf{T}_1)^{-1}$ s'écrit :

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{21} & 1 & 0 & \cdots & 0 \\ m_{31} & m_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & m_{n3} & \cdots & 1 \end{pmatrix}$$

où les m_{ik} sont les facteurs multiplicatifs rencontrés lors de la triangularisation de Gauss.

3.8. FACTORISATION DE CHOLESKI

Si la matrice carrée **A** est *symétrique* on peut tenter de la décomposer sous la forme d'un produit : $\mathbf{U}^t\mathbf{D}\mathbf{U}$ dans lequel **U** est une matrice triangulaire supérieure et **D** une matrice diagonale dont les éléments ont une valeur absolue unitaire.
Soit $\mathbf{U} = (u_{ij})$ triangulaire supérieure et $\mathbf{D} = (d_{ij})$ une matrice diagonale

On a :

$$(\mathbf{U}^t\mathbf{D}\mathbf{U})_{ij} = \sum_{k=1}^{n}\sum_{l=1}^{n} u_{ki} d_{kl} u_{lj} = \sum_{k=1}^{n} u_{ki} d_{kk} u_{kj} = \sum_{k=1}^{\min(i,j)} u_{ki} d_{kk} u_{kj}$$

car **D** est diagonale et **U** triangulaire supérieure.

Comme **A** est symétrique $a_{ij} = a_{ji}$ et il suffit de calculer a_{ij} pour $i \le j$.

De $(\mathbf{U}^t\mathbf{D}\mathbf{U})_{ij} = a_{ij}$ on tire, pour i = 1, 2,…,n :

$$a_{ij} = \sum_{k=1}^{i} u_{ki} d_{kk} u_{kj} = \sum_{k=1}^{i-1} u_{ki} d_{kk} u_{kj} + u_{ii} d_{ii} u_{ij}$$

pour i = j la formule devient : $a_{ii} = \sum_{k=1}^{i-1} u_{ki}^2 d_{kk} + u_{ii}^2 d_{ii}$ et $u_{ii}^2 d_{ii} = a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2 d_{kk}$

On choisit $d_{ii} = \text{signe}\left(a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2 d_{kk} \right)$

et $u_{ii} = \sqrt{\left| a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2 d_{kk} \right|}$

pour j > i, on a : $u_{ij} = \dfrac{a_{ij} - \sum_{k=1}^{i-1} u_{ki} d_{kk} u_{kj}}{u_{ii} d_{ii}}$

L'algorithme de Choleski s'écrit donc :

Pour i = 1, 2, ..., n

calculer la $i^{ième}$ ligne des matrices **D** et **U**

$d_{ii} = \text{signe}\left(a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2 d_{kk} \right)$ et $u_{ii} = \sqrt{\left| a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2 d_{kk} \right|}$

Pour j = i+1, i+2, ..., n

calculer $u_{ij} = \dfrac{a_{ij} - \sum_{k=1}^{i-1} u_{ki} d_{kk} u_{kj}}{u_{ii} d_{ii}}$

Exemple

Décomposons la matrice A par la méthode de Cholesky.

$$A = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 2 & 2 & 1 & 1 \\ 2 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Pour i = 1 on a : $d_{11} = \text{signe}(a_{11}) = +1$; $u_{11} = \sqrt{|a_{11}|} = 1$; $u_{1j} = \dfrac{a_{1j}}{u_{11} d_{11}} = a_{1j}$

$\Rightarrow$ $u_{12} = 2$; $u_{13} = 2$; $u_{14} = 1$

Pour i = 2 on trouve :

$$d_{22} = signe(a_{22} - u_{12}^2 d_{11}) = signe(-2) = -1$$
$$u_{22} = \sqrt{|a_{22} - u_{12}^2 d_{11}|} = \sqrt{2}$$

$$u_{2j} = \frac{a_{2j} - u_{12} d_{11} u_{1j}}{u_{22} d_{22}}$$

$\Rightarrow$ $u_{23} = \dfrac{3}{\sqrt{2}}$, $u_{24} = \dfrac{1}{\sqrt{2}}$

Pour i = 3 on trouve :

$$d_{33} = signe(a_{33} - u_{13}^2 d_{11} - u_{23}^2 d_{22}) = signe(\frac{1}{2})$$
$$u_{33} = \sqrt{|a_{33} - u_{13}^2 d_{11} - u_{23}^2 d_{22}|} = \sqrt{\frac{1}{2}} = \frac{1}{\sqrt{2}}$$

$$u_{34} = \frac{a_{34} - u_{13} d_{11} u_{14} - u_{23} d_{22} u_{24}}{u_{33} d_{33}} = \frac{1}{\sqrt{2}}$$

Enfin, pour i = 4 :

$$d_{44} = signe(a_{44} - u_{14}^2 d_{11} - u_{24}^2 d_{22} - u_{34}^2 d_{33}) = +1$$
car $signe(0) = +1$

La décomposition cherchée donne :

$$\mathbf{D} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{et} \quad \mathbf{U} = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 0 & \sqrt{2} & \frac{3}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

La matrice **A** n'est donc pas définie positive puisque **D** n'est pas l'identité.

REMARQUES

Le coût du calcul est de l'ordre de $n^3/2$ opérations arithmétiques et n extractions de racine carrée, c'est plus économique que la factorisation **LU** (*pour une matrice symétrique*).

Cette méthode fournit un test efficace pour déterminer si une matrice symétrique est définie positive ou non :

D est l'identité si la matrice est définie positive, et l'on a $\mathbf{A} = \mathbf{U}^t\mathbf{U}$.

Dans de très nombreux problèmes concrets, les systèmes à résoudre ont des matrices symétriques définies positives en raison même des phénomènes qu'ils modélisent. La méthode de factorisation de Choleski revêt donc une importance particulière.

Toutefois, comme toutes les méthodes directes, son coût de calcul et les erreurs d'arrondis deviennent un handicap quasiment insurmontable pour des systèmes comptant de l'ordre de 200 équations. En pratique on est surtout amené à résoudre des systèmes de plusieurs dizaines de milliers d'équations dont les matrices, fort heureusement, sont généralement creuses. La résolution de tels systèmes nécessite l'emploi de méthodes itératives.

4. RÉSOLUTION DE SYSTÈMES ALGÉBRIQUES LINÉAIRES MÉTHODES INDIRECTES (MÉTHODES ITÉRATIVES)

4.1. INTRODUCTION

La caractéristique essentielle des méthodes itératives est qu'elles procèdent par approximations successives et qu'on ne connaît pas *a priori* le nombre d'itérations (donc le nombre d'opérations) à effectuer pour atteindre une précision donnée.

Une deuxième caractéristique particulièrement importante est que lorsqu'une méthode itérative converge elle est *auto-correctrice* : si on commet une erreur de calcul au cours du processus, tout se passe comme si on recommençait le calcul à partir d'un nouveau vecteur initial $\boldsymbol{x}^{(0)}$.

De manière paradoxale, le fait de commettre une erreur peut donc éventuellement améliorer la convergence de la méthode utilisée. En effet, l'erreur commise peut avoir comme conséquence qu'on redémarre le calcul avec un vecteur plus proche de la solution. Ceci peut avoir un effet accélérateur de convergence. Cette caractéristique est d'ailleurs exploitée en pratique (cf. plus loin, relaxation).

Dans les méthodes itératives, on remplace le système à résoudre

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$$

par un système équivalent écrit sous la forme

$$\boldsymbol{x} = \mathbf{F}\boldsymbol{x} + \boldsymbol{c}$$

que l'on essaye de résoudre à l'aide d'un algorithme de point fixe, selon le schéma :

Choisir $\boldsymbol{x}^{(0)}$

Calculer $\boldsymbol{x}^{(k+1)} = \mathbf{F}\boldsymbol{x}^{(k)} + \boldsymbol{c}$

… *tant qu'il faut !*

Tout « l'art » consiste à bien choisir **F** et $\boldsymbol{c}$ et à déterminer de bons critères pour arrêter les calculs.

4.2. MÉTHODE DE JACOBI

On décompose A en une somme de trois matrices : $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$

D est la diagonale de **A**
U est la triangulaire supérieure de **A**
L est la triangulaire inférieure de **A**

On a alors $\mathbf{F} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$ et $\boldsymbol{c} = \mathbf{D}^{-1}\boldsymbol{b}$

Ce qui donne $\mathbf{F = I - D^{-1}(L + D + U) = -D^{-1}(L + U)}$ (**F** est la matrice de Jacobi)

Et la relation d'itération fournit *L'algorithme de Jacobi* :

$$x_i^{(k+1)} = -\sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)} + \frac{b_i}{a_{ii}} \quad , \quad i = 1, 2, ..., n.$$

Exemple :

Résoudre le système

$9x_1 - 2x_2 - 2x_3 = 11$
$2x_1 + 2x_2 + 8x_3 = 13$
$2x_1 - 9x_2 + 2x_3 = 15$

Voici un programme Matlab qui permet de résoudre ce système par la méthode de Jacobi :

```
% Méthode de Jacobi
% On part de [0 0 0]
% On constate que la méthode diverge (à faire) !
%
x10=0; x20=0; x30=0;
for k=1:30
   x(1,k)= ( 2*(x20 +   x30) + 11)/9;
   x(2,k)= (-2*(x10 + 4*x30) + 13)/2;
   x(3,k)= (-2* x10 + 9*x30) + 15)/2;
   x10 = x(1,k);
   x20 = x(2,k);
   x30 = x(3,k);
end
x'
```

Le programme suivant, qui applique la même méthode, converge

```
% Méthode de Jacobi
% On part de [0 0 0]
% On a interverti les équations 2 et 3 pour assurer
% dominance diagonale.
% On constate que la méthode converge (à vérifier) !
%
x10=0; x20=0; x30=0;
for k=1:30
   x(1,k)= ( 2*(x20 + x30) + 11)/9;
   x(2,k)= ( 2*(x10 + x30) - 15)/9;
   x(3,k)= (-2*(x10 + x20) + 13)/8;
   x10 = x(1,k);
   x20 = x(2,k);
   x30 = x(3,k);
end
x'
```

4.3. MÉTHODE DE GAUSS SEIDEL

Si on examine l'algorithme de Jacobi, on se rend compte que le calcul de la composante x_i à l'itération k+1 fait intervenir les composantes x_1, x_2 ... x_{i-1} à l'itération k.
On peut également constater que lorsque le calcul de x_i à l'itération k+1 est effectué, les composantes x_1, x_2 ... x_{i-1} sont déjà connues à cette même itération.

Si on tient compte de ce qui précède, on peut remplacer l'algorithme de Jacobi par le suivant :

$$x_i^{(k+1)} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k+1)} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)} + \frac{b_i}{a_{ii}} \quad , \quad i = 1, 2, ..., n.$$

connu sous le nom de «méthode de Gauss Seidel ».

La résolution du système de l'exercice ci-dessus par la méthode de Gauss Seidel peut être obtenue à l'aide du programme suivant :

```
% Méthode de Gauss Seidel
% On part de [0 0 0]
% On a interverti les équations 2 et 3 pour assurer
% dominance diagonale.
% On constate que la méthode converge (à vérifier) !
%
x10=0; x20=0; x30=0;
for k=1:30
   x(1,k)= ( 2*(x20    + x30)    + 11)/9;
   x(2,k)= ( 2*(x(1,k) + x30)    - 15)/9;
   x(3,k)= (-2*(x(1,k) + x(2,k) + 13)/8;
   x20 = x(2,k);
   x30 = x(3,k);
end
x'
```

Très souvent, l'algorithme de Gauss Seidel converge plus vite que l'algorithme de Jacobi (pourquoi ?).

4.4. MÉTHODES DE RELAXATION

On peut décomposer la matrice **A** d'une infinité de manières.

Soit $\mathbf{A} = \frac{1}{\omega}\mathbf{D} + \mathbf{L} + (\frac{\omega - 1}{\omega}\mathbf{D} + \mathbf{U})$ (le réel non nul ω est appelé *paramètre de relaxation*)

La méthode itérative devient alors : $(\frac{1}{\omega}\mathbf{D} + \mathbf{L})\,\boldsymbol{x}^{(k+1)} = -(\frac{\omega - 1}{\omega}\mathbf{D} + \mathbf{U})\,\boldsymbol{x}^{(k)} + \mathbf{b}$
C'est-à-dire, si l'inversion est possible :

$$\boldsymbol{x}^{(k+1)} = -(\frac{1}{\omega}\mathbf{D} + \mathbf{L})^{-1}(\frac{\omega - 1}{\omega}\mathbf{D} + \mathbf{U})\,\boldsymbol{x}^{(k)} + (\frac{1}{\omega}\mathbf{D} + \mathbf{L})^{-1}\,\mathbf{b}$$

Cette méthode est appelée méthode de relaxation. Elle est identique à la méthode de Gauss Seidel si $\omega = 1$ et nécessite également, à chaque itération, la résolution d'un système triangulaire.

Lorsque $\omega < 1$ on parle de sous-relaxation et si $\omega > 1$ de sur-relaxation.

Il est possible, dans certains cas, de démontrer qu'il existe une valeur optimale du paramètre de relaxation (pour plus de détails cf. « Introduction à l'analyse numérique » de Jacques Rappaz et Marco Picasso, Presses polytechniques et universitaires romandes, 1998).

4.5. CONDITION SUFFISANTE DE CONVERGENCE

On peut démontrer que les méthodes de Jacobi et Gauss Seidel sont convergentes si la matrice du système est à diagonale prépondérante.

4.6. CRITÈRES D'ARRÊT

Lorsqu'on programme un algorithme itératif il est vivement conseillé de prévoir au moins deux critères d'arrêt.

Le premier critère consistera à arrêter les calculs si deux approximations successives de la solution sont « *suffisamment proche*s » l'une de l'autre. Ce que signifie « *suffisamment proche*s » est laissé à l'appréciation du programmeur et nécessite la définition d'une mesure de la proximité de deux solutions approchées (distance, norme). On pourra par exemple décider que la *« convergence »* est atteinte si la norme euclidienne de la différence entre deux approximations successives n'excède pas une constante positive ε choisie a priori.

Le deuxième critère doit imposer que le programme soit arrêté après un nombre limité d'itérations. En effet, si la méthode produit une suite de vecteurs qui diverge, le programme s'arrêtera en général suite à un *« overflow »*. Cependant, le programme peut *« cycler »* : il n'y a ni convergence numérique ni *« overflow »*. La suite des approximations reproduit périodiquement une même succession de vecteurs ou donne naissance à une suite chaotique de vecteurs bornés. Dans pareil cas, si l'on ne limite pas le nombre d'itérations, le programme fonctionnera jusqu'à ce que l'ordinateur tombe en panne… sauf bien entendu si l'opérateur intervient pour provoquer l'arrêt de la machine !

5. PROGRAMMATION LINEAIRE

Les problèmes de programmation linéaire se rencontrent dans le domaine de la recherche opérationnelle et de l'optimisation. Ils permettent par exemple d'optimiser la composition d'une ration alimentaire ou la gestion de cultures agricoles.

5.1. ECHELON D'ECHANGE

On considère m fonctions linéaires y_i des n variables x_k définies par :

$$y_i = \sum_{k=1}^{n} a_{ik} x_k + b_i \quad (i = 1,2,...,m)$$

On met ces relations sous forme de tableau, par exemple, pour m=3 et n=4, on a :

	x_1	x_2	x_3	x_4	1
$y_1 =$	a_{11}	a_{12}	a_{13}	a_{14}	b_1
$y_2 =$	a_{21}	a_{22}	a_{23}	a_{24}	b_2
$y_3 =$	a_{31}	a_{32}	a_{33}	a_{34}	b_3

On appelle $x_1, x_2, \ldots, x_n$ les variables indépendantes (ou *de base*) et $y_1, y_2, \ldots, y_m$ les variables dépendantes (ou *auxiliaires*).

Soit la p^{ème} fonction y_i et soit $a_{pq} \neq 0$, on a :

$$x_q = \frac{1}{a_{pq}} y_p - \sum_{\substack{k=1 \\ k \neq q}}^{n} \frac{a_{pk}}{a_{pq}} x_k - \frac{b_p}{a_{pq}}$$

et pour $i \neq p$:

$$y_i = \frac{a_{iq}}{a_{pq}} y_p + \sum_{\substack{k=1 \\ k \neq q}}^{n} (a_{ik} - \frac{a_{iq} a_{pk}}{a_{pq}}) x_k + (b_i - \frac{a_{iq} b_p}{a_{pq}})$$

Ce qui donne m nouvelles fonctions linéaires où y_p joue à présent le rôle d'une variable indépendante et x_q celui d'une variable dépendante.

Il y a eu échange des rôles suite à ce que nous appellerons un *« échelon d'échange »* .

Si nous reprenons le cas m=3, n=4 et que nous supposons par exemple que p=2 et q=3, le tableau précédent devient :

	x_1	x_2	y_2	x_4	1
$y_1 =$	$^{*}a_{11}$	$^{*}a_{12}$	$^{*}a_{13}$	$^{*}a_{14}$	$^{*}b_1$
$x_3 =$	$^{*}a_{21}$	$^{*}a_{22}$	$^{*}a_{23}$	$^{*}a_{24}$	$^{*}b_2$
$y_3 =$	$^{*}a_{31}$	$^{*}a_{32}$	$^{*}a_{33}$	$^{*}a_{34}$	$^{*}b_3$

Où les éléments du nouveau tableau sont obtenus par les formules d'échange :

$$^{*}a_{pq} = \frac{1}{a_{pq}}$$

$$^{*}a_{pk} = -\frac{a_{pk}}{a_{pq}} \quad (k \neq q) \qquad ^{*}b_{p} = -\frac{b_{p}}{a_{pq}}$$

$$^{*}a_{iq} = \frac{a_{iq}}{a_{pq}} \quad (i \neq p)$$

$$^{*}a_{ik} = a_{ik} - \frac{a_{iq}a_{pk}}{a_{pq}} = a_{ik} + a_{iq}\,{}^{*}a_{pk} \quad (i \neq p, k \neq q)$$

$$^{*}b_{i} = b_{i} - \frac{a_{iq}b_{q}}{a_{pq}} = b_{i} + a_{iq}\,{}^{*}b_{p} \quad (i \neq p)$$

L'élément a_{pq} est appelé *pivot*, tandis que la ligne et la colonne qui le contiennent sont appelées *ligne pivot* et *colonne pivot*.

Exemple

	x_1	x_2	x_3	1
$y_1 =$	3	7	4	−13
$y_2 =$	−5	4	5	2
$y_3 =$	−1	2	3	−6

	x_1	y_3	x_3	1
$y_1 =$	6.5	3.5	−6.5	8
$y_2 =$	−3	2	−1	14
$x_2 =$	0.5	0.5	−1.5	3

0.5 −1.5 3 : *«ligne de base»*

5.2. INVERSION DE MATRICE

Après n échanges

$\boldsymbol{y} = A\boldsymbol{x}$ est transformé en $\boldsymbol{x} = A^{-1}\boldsymbol{y}$!

5.3. RÈGLES D'ÉCHANGE

♦ Le pivot est remplacé par son inverse

♦ Les éléments de la colonne pivot sont les anciens éléments divisées par le pivot

♦ Les éléments de la ligne pivot sont les anciens éléments changés de signe et divisés par le pivot ⇒ *ligne de base*

♦Les éléments restants sont les anciens éléments auxquels on a ajouté le produit de l'ancien élément de la même ligne qui se trouve dans la colonne contenant le pivot par le nouvel élément qui se trouve dans la même colonne et dans la ligne pivot (*règle du rectangle* - cf. ligne de base).

5.4. PREMIER PROBLEME

Un atelier de fabrication de chaussures dispose des ressources mensuelles suivantes :

Main d'œuvre (en heures) : 8000
Temps machine (en heures) : 2000
Matière première (cuir en dm^2) : 4500

La production consiste uniquement en chaussures pour hommes et chaussures pour dames. La confection d'une paire pour homme nécessite 10 heures de main d'œuvre, 5 heures de temps machine et 15 dm² de cuir tandis que pour fabriquer une paire pour dame il faut 20 heures de main d'œuvre 4 heures de temps machine et 6 dm² de cuir.

Chaque paire pour homme rapporte un bénéfice double de celui d'une paire pour dame, lequel est de 16 euro.

En supposant que toutes les chaussures produites sont vendues, combien de paires pour homme et combien de paires pour dame doit-on produire pour avoir un bénéfice maximum ?

Résumons les données dans un tableau

	Dame	Homme	Ressources
Main d'œuvre (h)	20	10	8000
Temps machine (h)	4	5	2000
Cuir (dm²)	6	15	4500
Bénéfice (euro)	16	32	

Soit x_1 le nombre de paires de chaussures pour dame et x_2 le nombre de paires pour homme à produire mensuellement.

On a les inégalités suivantes :

$$
\begin{aligned}
20x_1 + 10x_2 &\leq 8000 \\
4x_1 + 5x_2 &\leq 2000 \\
6x_1 + 15x_2 &\leq 4500 \\
x_1 &\geq 0 \\
x_2 &\geq 0 \\
16x_1 + 32x_2 &= \text{Max !}
\end{aligned}
$$

Les trois premières inégalités traduisent les contraintes dues aux ressources, les deux inégalités suivantes expriment le fait que l'on ne peut produire un nombre négatif de paires de chaussures. Enfin la dernière égalité signifie que l'on cherche à obtenir un bénéfice maximum.

L'ensemble des conditions reprises ci-dessus constitue un *programme linéaire*.

Ce programme est un problème typique d'*optimisation* où l'on essaye de maximiser une *fonction objectif* en tenant compte de contraintes imposées.

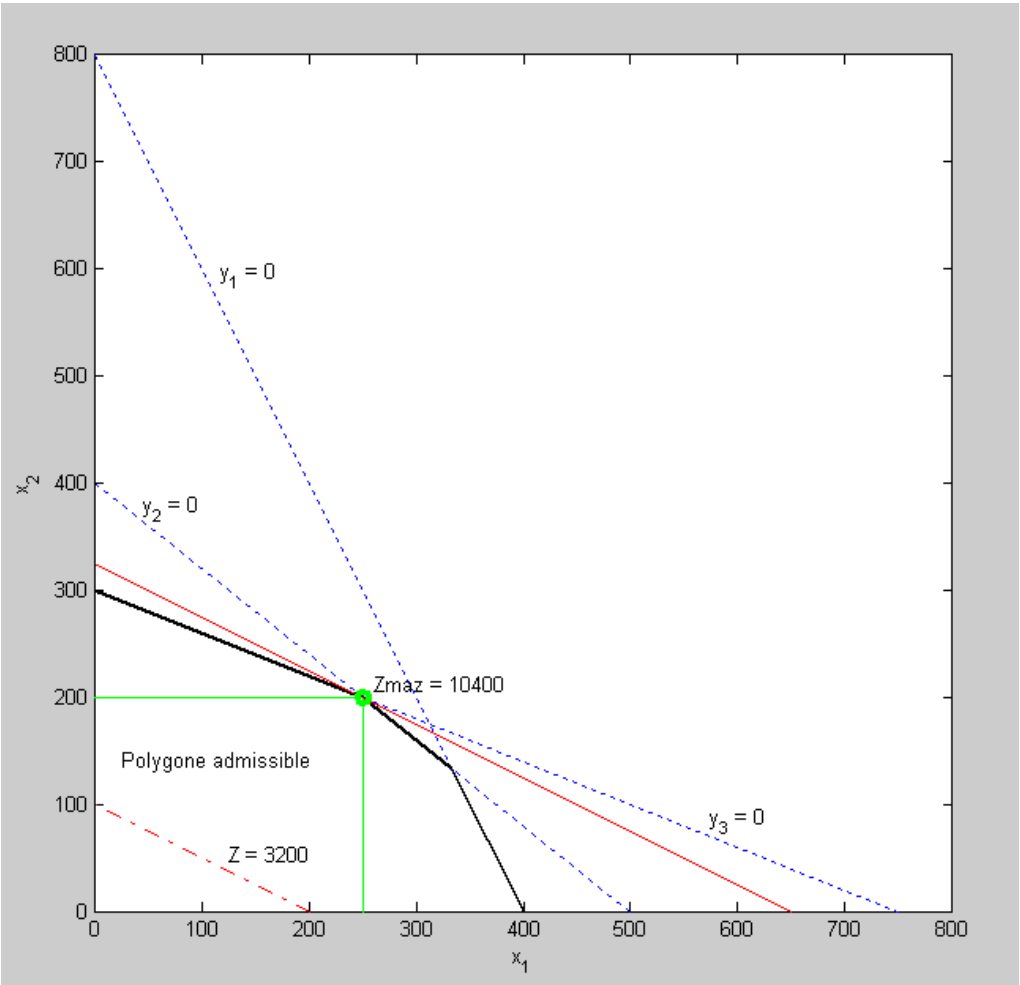
On peut systématiser le traitement d'un tel programme en introduisant une formulation uniforme des contraintes sous forme de fonctions linéaires qui doivent être non négatives. Ceci se fait en définissant des variables dépendantes y_j .

Si l'on note z, la fonction objectif à maximiser, on obtient le programme linéaire équivalent :

$$
\begin{aligned}
y_1 &= -20x_1 - 10x_2 + 8000 \geq 0 \\
y_2 &= -4x_1 - 5x_2 + 2000 \geq 0 \\
y_3 &= -6x_1 - 15x_2 + 4500 \geq 0 \\
& x_1 \geq 0 \\
& x_2 \geq 0 \\
z &= 16x_1 + 32x_2 = \text{Max !}
\end{aligned}
$$

$\Rightarrow$ non seulement les variables indépendantes x_i doivent être non négatives mais les variables dépendantes y_j également, … avec z = Max !

Le programme linéaire ci-dessus ne contient que deux inconnues : x_1 et x_2. On peut donc le résoudre graphiquement dans le plan (x_1, x_2).



L'ensemble des points P de coordonnées (x_1, x_2) satisfaisant à l'inéquation linéaire

$y = ax_1 + bx_2 + c \geq 0$

Consiste en un demi plan du plan (x_1, x_2), la frontière d'équation $ax_1 + bx_2 + c = 0$, étant comprise.

Les cinq inéquations du programme linéaire déterminent cinq demi plans dont les points communs sont contenus dans un polygone constitué de l'ensemble des points qui vérifient les inéquations. Ces points sont dits ***admissibles*** et le polygone est appelé ***domaine admissible***.

Dans le cas d'un problème à deux inconnues, le domaine admissible est un polygone convexe.

La fonction objectif est linéaire en x_1 et x_2 et ses lignes de niveau z = constante forment une famille de droites parallèles (cf. figure page précédente).

Parmi tous les points admissibles, celui qui permet de maximiser z est le point D. Ce point est un sommet du domaine admissible, il fournit la solution du problème :

$x_1 = 250$, $x_2 = 200$ et $z_{Max} = 10400$.

La résolution graphique nous enseigne de plus qu'au point solution on a $y_2 = 0$ et $y_3 = 0$ tandis que $y_1 > 0$ ce qui signifie que les ressources en temps machine et en cuir ont été épuisées alors qu'il reste une certaine capacité en heures de travail puisque $y_1>0$ (N.B. $y_1 = 1000$).

Les deux conditions $y_2 = 0$ et $y_3 = 0$ sont appelées ***restrictions essentielles***.

5.5. DEUXIÈME PROBLÈME

Une entreprise de transport dispose de deux dépôts A et B où sont garés, respectivement, 18 et 12 poids lourds. Elle doit dispatcher 11, 10 et 9 camions vers les sous-traitants R, S et T. Les distances (en km) séparant les dépôts des stations sous-traitantes sont données dans le tableau ci-dessous. Comment faut-il répartir les camions entre les sous-traitants si l'on veut minimiser la distance totale que devront parcourir les poids lourds

	R	S	T
A	5	4	9
B	7	8	10

On constate que le nombre de camions dont dispose l'entreprise est égal au nombre de camions à acheminer chez les sous-traitants. Ce problème de transport peut donc être décrit à l'aide de deux inconnues. Parmi différents choix possibles, nous définissons les inconnues comme suit :

x_1 = nombre de camions allant de A vers R
x_2 = nombre de camions allant de A vers S

Le nombre de poids lourds à diriger selon les autres trajets peut s'exprimer en fonction de ces deux variables et comme le nombre de camions allant d'un point à un autre ne peut être négatif, on obtient les inégalités suivantes, en introduisant les variables indépendantes y_i :

A → T : $y_1 = -x_1 - x_2 + 18 \geq 0$

B → R : $y_2 = -x_1 + 11 \geq 0$

B → S : $y_3 = - x_2 + 10 \geq 0$

B → T : $y_4 = x_1 + x_2 - 9 \geq 0$

A → R : $x_1 \geq 0$

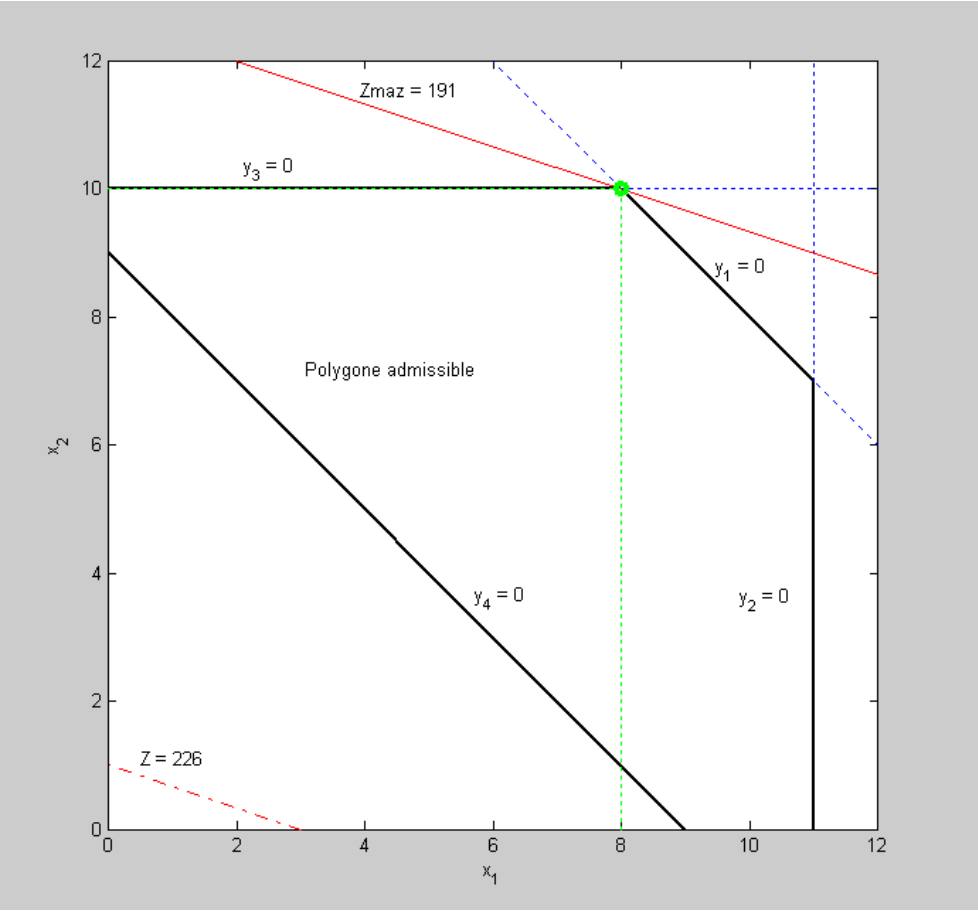
A → S : $x_2 \geq 0$

Le nombre total de kilomètres à parcourir vaut :

$z = 5x_1 + 4x_2 + 9(18 - x_1 - x_2) + 7(11 - x_1) + 8(10 - x_2) + 10(x_1 + x_2 - 9)$

et $z = -x_1 - 3x_2 + 229$ est la fonction objectif à minimiser.

On peut à nouveau résoudre le problème graphiquement dans le plan (x_1, x_2). Les six inégalités définissent six demi plans qui délimitent le *domaine admissible*. Les lignes de niveau de la *fonction objectif* z, correspondant aux valeurs z = 226 et zmax = 191 sont représentées sur la figure. Le point solution indiqué en vert.



Il faut donc envoyer $x_1 = 8$ camions de A vers R, $x_2 = 10$ camions de A vers S, $y_1 = 0$ camions de A vers T, $y_2 = 3$ camions de B vers R, $y_3 = 0$ camions de B vers S et $y_4 = 9$ camions de B vers T. Ce plan de transport correspond au nombre minimum de kilomètres à parcourir, soit $z_{Min} = 191$ km.

Les deux exemples ont en commun le fait que la solution du problème est unique et correspond à un sommet du polygone admissible. Une telle situation peut se généraliser à n dimensions. Ainsi, par exemple, lorsqu'on a trois variables indépendantes, le domaine admissible est un polyèdre convexe dont les faces ont pour équations $x_i = 0$ et $y_j = 0$. Les surfaces de niveau z = constante étant cette fois des plans parallèles.

Dans certains problèmes on obtient une ***infinité de solutions***. C'est le cas, dans le plan, si les lignes de niveau sont parallèles à un côté du domaine admissible qui passe par un sommet solution. Il en va de même dans l'espace à trois dimensions si une face du polyèdre admissible qui passe par un sommet solution est parallèle à une surface de niveau z = constante. Dans de telles situations, on dit qu'il y a ***dégénérescence***.
Les considérations géométriques précédentes montrent que dans un programme linéaire la(les) solution(s) du problème correspond(ent) à des points situés sur la ***frontière*** du domaine admissible. Nous allons donc développer un algorithme qui permet de passer de sommet en sommet pour atteindre la(les) solution(s).

Le démarrage de la procédure nécessitera, évidemment, la connaissance d'un sommet de départ appartenant au domaine admissible.

Dans le premier exemple, l'origine dont les coordonnées sont $x_1 = 0$ et $x_2 = 0$ est un sommet admissible qui peut servir de point de départ de l'algorithme.

Dans le deuxième exemple, l'origine n'est pas un point admissible et nous ne disposons pas d'un sommet de départ « évident ». Ce type de situation nécessitera un traitement particulier.

5.6. ALGORITHME DU SIMPLEXE

Sur la figure de la page 4, nous constatons que chaque coin du domaine admissible est caractérisé par le fait que deux variables s'y annulent, tandis que les autres prennent une valeur positive. Dans le but d'obtenir une suite de sommets qui conduise au sommet solution, nous partirons d'un sommet initial pour aller vers le suivant en longeant un côté du polygone.

Examinons par exemple, la transition du sommet A vers le sommet B dans le cas du premier exemple.

La situation de ces deux points est décrite comme suit :

Sommet A :

$$x_1 = 0,\ x_2 = 0,\ y_1 > 0,\ y_2 > 0,\ y_3 > 0,\ z = 0$$

Sommet B :

$$x_1 > 0,\ x_2 = 0,\ y_1 = 0,\ y_2 > 0,\ y_3 > 0,\ z = 6400$$

Lorsqu'on passe de A à B, le long de AB, x_2 garde sa valeur nulle tandis que x_1 passe de zéro à une valeur positive. Simultanément, y_1 passe d'une valeur positive à zéro. Toutes les autres variables restent positives et z augmente.

Les variables x_1 et y_1 ont échangé leurs rôles et on constate donc que pour passer d'un sommet du domaine admissible à un sommet adjacent il suffit d'appliquer un échelon d'échange à deux fonctions linéaires du programme linéaire.

En procédant par échanges successifs nous atteindrons, après un nombre fini d'étapes, le sommet solution.

Comme z doit augmenter à chaque étape, la procédure devra éviter un retour vers un sommet déjà rencontré.

Nous devons à présent définir les règles permettant le choix des pivots successifs conduisant à la maximisation de la fonction objectif pour un programme linéaire défini comme suit :

$$y_i = \sum_{k=1}^{n} a_{ik} x_k + c_i \geq 0 \quad (i = 1,2,...,m)$$
$$x_k \geq 0 \quad (k = 1,2,...,n)$$
$$z = \sum_{k=1}^{n} b_k x_k + d = Max!$$

Les inconnues non négatives, x_k, doivent être déterminées de façon à ce que les m inéquations linéaires soient vérifiées et que la fonction objectif soit maximisée.

L'origine est un point admissible si et seulement si :

$c_i \geq 0 \quad (i=1, 2, \ldots, m)$,

ce que nous supposerons pour la suite, de façon à disposer d'un sommet de départ (N.B. cette hypothèse sera levée plus loin).

Mettons les m inéquations à vérifier ainsi que l'expression de la <u>*fonction objet*</u> (on dit indifféremment objet ou objectif) sous la forme d'un tableau :

$$\begin{array}{cccccc}
 & x_1 \cdots & x_q \cdots & x_n & 1 \\
y_1 = & a_{11} \cdots & a_{1q} \cdots & a_{1n} & c_1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
y_i = & a_{i1} \cdots & a_{iq} \cdots & a_{in} & c_i \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
y_p = & a_{p1} \cdots & a_{pq} \cdots & a_{pn} & c_p \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
y_m = & a_{m1} \cdots & a_{mq} \cdots & a_{mn} & c_m \\
z = & b_1 \cdots & b_q \cdots & b_n & d
\end{array}$$

Les variables indépendantes x_k, qui caractérisent un sommet admissible, si on les annule, sont appelées *variables de base*. Les variables dépendantes y_i, qui prennent une valeur non

négative en un sommet admissible, sont appelées *variables auxiliaires*. La constante d dans le coin inférieur gauche du tableau est la valeur que prend la fonction z au point admissible « en cours ».

Echangeons à présent x_p et y_q en prenant comme pivot l'élément a_{pq} et en appliquant les règles des l'échelons d'échange.

Le problème à résoudre est de trouver les conditions pour qu'après l'échange le nouveau tableau :

$$\begin{array}{cccccc}
 & x_1 \cdots & y_p \cdots & x_n & 1 \\
y_1 = & a'_{11} \cdots & a'_{1q} \cdots & a'_{1n} & c'_1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
y_i = & a'_{i1} \cdots & a'_{iq} \cdots & a'_{in} & c'_i \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
x_q = & a'_{p1} \cdots & a'_{pq} \cdots & a'_{pn} & c'_p \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
y_m = & a'_{m1} \cdots & a'_{mq} \cdots & a'_{mn} & c'_m \\
z = & b'_1 \cdots & b'_q \cdots & b'_n & d'
\end{array}$$

fournisse un sommet admissible si nous égalons à zéro les nouvelles variables de base $x_1,\ldots,$ $y_p,\ldots, x_n$.

Pour avoir un sommet admissible les nouvelles variables auxiliaires $y_1,\ldots, x_q, \ldots y_m$ doivent être non négatives, c'est-à-dire que les conditions $c'_i \geq 0 \quad (i = 1,2,...,m)$ doivent être vérifiées.

De plus, si nous voulons que z augmente il faudra avoir $d' \geq d$.

Sur la base des règles des échelons d'échange, nous obtenons :

$$c'_p = -\frac{c_p}{a_{pq}}$$

$$c'_i = c_i - \frac{a_{iq} c_p}{a_{pq}} \; (i = 1,2,...,m; i \neq p)$$

$$d' = d - \frac{b_q c_p}{a_{pq}}$$

Ce qui impose de choisir un pivot a_{pq} négatif.

Pour choisir le « meilleur pivot », nous formons à partir du tableau initial les *quotients caractéristiques* définis par $q_i = c_i / a_{iq}$ et ce, pour tout $i \neq p$ et $a_{iq} < 0$.

Le plus grand quotient caractéristique ainsi formé désigne la colonne du pivot, ce qui assure que : $c'_i \geq c_i \geq 0$. Si cette condition est remplie, l'annulation des nouvelles variables de base correspond bien à un nouveau sommet admissible.

Enfin, puisque $c_p \geq 0$ et $a_{pq} < 0$ on doit avoir $b_q \geq 0$, condition qui doit être satisfaite pour la colonne du pivot. Cette condition garantit la croissance de la fonction objectif.

De ce qui précède, on définit les règles qui fixent le choix de l'élément pivot.

5.7. REGLES QUI FIXENT LE CHOIX DU PIVOT :

1° la colonne du pivot doit être choisie de façon à ce que son élément b_q soit positif (ou *nul*) ;

2° dans la colonne du pivot, il faut choisir un pivot négatif, on prend parmi les pivots possibles celui auquel correspond le plus grand quotient caractéristique $q_i = c_i/a_{iq}$ (le plus petit en valeur absolue !).

Ces règles constituent les bases de *l'algorithme du simplexe*. Elles permettent la détermination d'une suite de sommets du domaine convexe admissible appelé *simplexe.*

La valeur de la fonction objectif croît à chaque échange, sauf éventuellement, dans des cas particuliers tels que les cas de dégénérescence (b_q nul).

L'algorithme du simplexe s'arrête lorsque tous les éléments b_i sont négatifs ou nuls.

5.8. RETOUR AU PREMIER PROBLEME

Reprenons l'exemple de la manufacture de chaussures traité graphiquement précédemment.

Le schéma simplexe s'écrit :

	x_1	x_2	1	q_i
$y_1 =$	-20	-10	8000	-400
$y_2 =$	-4	-5	2000	-500
$y_3 =$	-6	-15	4500	-750
$z =$	16	32	0	
		$-\frac{1}{2}$	400	

Avec comme variables de base $x_1 = 0$ et $x_2 = 0$ on obtient le point A (cf. figure page 4).

On échange x1 et y1 en s'aidant des éléments de la *ligne de base.*

Après ce premier échange, les nouvelles variables de base sont y_1 et x_2 et le schéma simplexe devient :

	y_1	x_2	1	q_i
$x_1 =$	$-\frac{1}{20}$	$-\frac{1}{2}$	400	-800
$y_2 =$	$\frac{1}{5}$	-3	400	$-133\frac{1}{3}$
$y_3 =$	$\frac{3}{10}$	-12	2100	-175
$z =$	$-\frac{4}{5}$	24	6400	
	$\frac{1}{15}$		$\frac{400}{3}$	

Lors du premier échange, on est parti du point A pour aboutir au point B, en suivant le côté x_2=0 du polygone. Les nouvelles variables de base correspondent au sommet B du domaine admissible. La fonction objectif est passée de la valeur zéro à la valeur 6400. Les nouvelles variables auxiliaires sont toujours strictement positives. La deuxième étape de la résolution se fait par l'échange des variables x_2 et y_2 ce qui conduit à un nouveau tableau dont les variables de base y_1 et y_2 correspondent au sommet C.
En ce point la fonction objectif vaut 9600.

	y_1	y_2	1	q_i
$x_1 =$	$-\frac{1}{12}$	$\frac{1}{6}$	$\frac{1000}{3}$	-4000
$x_2 =$	$\frac{1}{15}$	$-\frac{1}{3}$	$\frac{400}{3}$	–
$y_3 =$	$-\frac{1}{2}$	4	500	-1000
$z =$	$\frac{4}{5}$	-8	9600	
		8	1000	

Bien que les deux inconnues x_1 et x_2 ont été échangées pour devenir des variables auxiliaires, le processus n'est pas terminé puisque dans le dernier tableau les éléments de la ligne des b_i ne sont pas tous négatifs. On doit donc continuer à procéder à des échelons d'échange. C'est nécessairement la première colonne qui contient le pivot suivant ($b_2 < 0$) et comme le plus grand quotient caractéristique est q_3, la ligne du pivot est aussi fixée. Le troisième échange fournit le tableau suivant :

	y_3	y_2	1
$x_1 =$	$\frac{1}{6}$	$-\frac{1}{2}$	250
$x_2 =$	$-\frac{2}{15}$	$\frac{1}{5}$	200
$y_1 =$	-2	8	1000
$z =$	$-\frac{8}{5}$	$-\frac{8}{5}$	10400

Avec ce dernier tableau l'algorithme s'arrête puisque tous les b_i sont négatifs, il correspond au sommet D, point solution.

En D, les dernières variables de base y_3 et y_2 s'annulent et on obtient comme solution :

$x_1 = 250$ et $x_2 = 200$;

avec la variable auxiliaire :

$y_1 = 1000$

et le maximum de la fonction objectif $z_{Max} = 10400$.

5.9. REMARQUES

1° Nombre d'opérations

L'algorithme du simplexe nécessite, *en général*, un nombre fini d'échanges et se termine donc après un nombre fini de calculs. Pour des problèmes où $m > 50$ et $m+n < 300$ le nombre d'opérations est de l'ordre de 3m/2.

2° Dégénérescence

Si au cours des calculs on se trouve dans une situation où la plus grande valeur des quotients caractéristiques q_i est atteinte par deux quotients simultanément, après échange, au moins un des c_j s'annule et une des variables auxiliaires également.

On est alors dans un cas de dégénérescence.

3° Programmation

On peut évidemment programmer l'algorithme du simplexe et c'est indispensable si l'on veut résoudre un problème présentant plusieurs centaines de contraintes. Nous fournissons ci-dessous un programme qui appelle la fonction **`simplex`** pour résoudre le premier exemple traité dans ce chapitre (et tout problème analogue…). Les résultats sont également donnés à titre d'information.

Programme :

```
clc; clear
a=[-20 -10; -4 -5; -6 -15];
b=[16; 32];
c=[8000; 2000; 4500];
simplex(a,c,b)
```

Fonction :

```
function simplex(a,c,b)
format short g
newa = 0*a;
newb = 0*b;
newc = 0*c;
z = 0;
n_var  = length(b);
n_cond = length(c);

if size(a) ~= [n_cond n_var]
    display('Dimensions ?')
    return
end

%  BOUCLE GENERALE
for boucle = 1:10
M = [a c ; b' z]
if max(c) < 0
    display('c < 0')
    break
end
[B,K] = sort(b);
if B(n_var) < 0
    display('b < 0')
    break
end
q = K(n_var);
[A,L] = sort(a(:,q));
if A(1) > 0
    display('a > 0')
    break
end
quot     = c./a(:,q);
[QUOT,Q] = sort(quot);
for i = n_cond : -1 : 1
    if QUOT(i) < 0
        p = i;
    end
    if QUOT(i) < 0
        display('q < 0')
        break
    end
end
p = Q(p);
q;
pivot = a(p,q);
PIVOT = [p q a(p,q)]


% CALCUL DES NOUVEAUX ELEMENTS

newa(p,:) = -a(p,:)/pivot;
newc(p)   =   -c(p)/pivot;
newa(:,q) =  a(:,q)/pivot;
newb(q)   =    b(q)/pivot;
newa(p,q) =       1/pivot;
z = z - b(q)*c(p)/pivot;

for i=1 : n_cond
    if i ~= p
```

```
        newc(i) = c(i) - a(i,q)*c(p)/pivot;
        for k = 1 : n_var
            if k ~= q
                newa(i,k) = a(i,k) - a(i,q)*a(p,k)/pivot;
            end
        end
    end
end
for k = 1 : n_var
    if k ~= q
        newb(k) = b(k) - b(q)*a(p,k)/pivot;
    end
end
a = newa;
b = newb;
c = newc;
end
```

Résultats :

```
M =

         -20         -10        8000
          -4          -5        2000
          -6         -15        4500
          16          32           0

q < 0

PIVOT =

     3     2   -15


M =

         -16      0.66667        5000
          -2      0.33333         500
        -0.4    -0.066667         300
         3.2      -2.1333        9600

q < 0

PIVOT =

     2     1    -2


M =

           8          -2        1000
        -0.5     0.16667         250
         0.2    -0.13333         200
        -1.6        -1.6       10400

b < 0
```

5.10. VARIABLES LIBRES

Il y a des problèmes de programmation linéaire où certaines variables ne doivent pas être non négatives, on les appelle *variables libres*.

De telles variables, sur lesquelles il n'y a pas de contrainte de signe, sont *éliminées* du programme linéaire.

On peut montrer (cf. Numerical analysis, H.R. Schwarz, Wiley ed. 1989) que la règle suivante préside à l'élimination des variables libres :

Si l'élément b_q, de la colonne pivot fixée par la variable libre x_q à éliminer, est positif (négatif ou nul), il faut calculer les quotients caractéristiques $q_i = c_i/a_{iq}$ correspondant aux éléments a_{iq} négatifs (positifs). La ligne pivot est alors déterminée par le plus *petit* de ces quotients en *valeur absolue*.

5.11. ELIMINATION

Une variable libre n'est soumise à aucune contrainte, lors de son échange, on supprime, dans le nouveau tableau, la ligne qui la contient.

5.12. TRANSLATION DE COORDONNEES

Lorsque, comme dans le deuxième exemple étudié graphiquement, l'origine $x_1=x_2=\ldots=x_n=0$, n'est pas un point admissible, c'est que les conditions $c_i \geq 0$ (i=1,2,…,m) ne sont pas toutes respectées. On ne dispose donc pas d'un point admissible de départ pour l'algorithme du simplexe.

Pour remédier à cette situation on effectue une translation de manière à placer l'origine des axes en un point du domaine admissible, tout en introduisant des variables libres ξ_k.

Soit $\underline{x}_k$ (k=1,2…,n) un point admissible.

Si l'on pose : $x_k = \underline{x}_k + \xi_k$ (k=1,2,…,n),

le programme linéaire devient :

$$y_i = \sum_{k=1}^{n} a_{ik}\xi_k + \underline{c}_i \geq 0 \quad (i = 1,2,...,m)$$

$$x_k = \xi_k + \underline{x}_k \geq 0 \quad (k = 1,2,...,n)$$

$$z = \sum_{k=1}^{n} b_k\xi_k + \underline{d} = Max\,!$$

Avec :

$$\underline{c}_i = \sum a_{ik}\underline{x}_k + c_i \geq 0, \quad (i = 1,2,...,m)$$

$$\underline{x}_k \geq 0, \quad (k = 1,2,...,n).$$

5.13. EXEMPLE

Le problème de la société de transport (cf. fig. p.5) présente un domaine admissible dont l'origine ne fait pas partie. Le tableau traduisant les contraintes, écrit avec comme variables de base x_1 et x_2, ne peut servir de tableau de départ pour appliquer l'algorithme du simplexe.

Chapitre 6

Systèmes mal conditionnés

6.1 Introduction

Soit le système

$$4.218613x_1 + 6.327917x_2 = 10.546530 \tag{6.1}$$
$$3.141592x_1 + 4.712390x_2 = 7.853982 \tag{6.2}$$

dont la solution est $x_1 = x_2 = 1$

et son « voisin » obtenu par « perturbation »

$$4.218611x_1 + 6.327917x_2 = 10.546530 \tag{6.3}$$
$$3.141594x_1 + 4.712390x_2 = 7.853980 \tag{6.4}$$

dont la solution est $x_1 = -5$ et $x_2 = 5$.

Ces deux systèmes sont, en apparence, presque identiques mais ils possèdent des solutions très différentes.

On dit qu'ils sont mal conditionnés. Une légère perturbation du système initial produit une solution très éloignée de la solution du système non perturbé. Une telle situation mérite une attention particulière... !

6.1.1 Interprétation géométrique

Les équations du système 6.1 représentent, dans le plan Ox_1x_2, deux droites quasiment parallèles. Leurs pentes respectives valent -1.499999407388163 et -1.500000636619905. Résoudre ce système revient à déterminer l'intersection de ces deux droites on trouve facilement $x_1 = x_2 = 1$.

Les droites représentées par les équations du système 6.3 sont également quasi parallèles et leurs pentes respectives diffèrent très peu de celles des droites du système 6.1 puisqu'elles valent -1.500000118522424 et -1.499999681690250. La solution du deuxième système est $x_1 = -5, x_2 = 5$. La légère modification des pentes des droites, provoque un fort déplacement de la position de leur point de rencontre qui, cette fois, se retouve dans la partie du plan où x_1 est négatif. [1]

Résoudre un système mal conditionné nécessite de travailler avec un grand nombre de chiffres significatifs exacts. Dans le cas présent, six chiffres exacts ne fournissent pas une solution « acceptable ».

Grâce à l'utilisation des normes vectorielles et matricielles, il y a moyen, en principe, de déterminer a priori si un système est mal conditionné.

6.2 Norme matricielle

Soit une matrice carrée réelle A de dimensions $N \times N$ et un vecteur colonne réel y de dimension N et de composantes $y_i, (1 \leq i \leq N)$. On définit la norme euclidienne du vecteur y (cf. cours d'algèbre) comme suit :

$$\|y\|_2 = \sqrt{\sum_{i=1}^{N} y_i^2} \tag{6.5}$$

La norme matricielle induite par la norme vectorielle euclidienne est définie par :

$$\|A\|_2 = \sup_{y \neq 0} \frac{\|Ay\|_2}{\|y\|_2} = \sup_{y \neq 0} \sqrt{\frac{y^T A^T A y}{y^T y}} \tag{6.6}$$

Remarques

1. $\frac{y^T A^T A y}{y^T y}$ est appelé quotient de Rayleigh.
2. si x est un vecteur quelconque on a, par définition $\|Ax\|_2 \leq \|A\|_2 . \|x\|_2$
3. On définit d'autres normes plus faciles à utiliser en pratique :

   $$\|A\|_\infty = \max_{i=1,\ldots,N} \sum_{j=1}^{N} |a_{ij}| \text{ et } \|A\|_1 = \max_{j=1,\ldots,N} \sum_{i=1}^{N} |a_{ij}|$$

   induites respectivement par

   $$\|y\|_\infty = \max_i |y_i| \text{ et } \|y\|_1 = \sum_{i=1}^{N} |y_i|$$

1. Si le second membre de la deuxième équation n'avait pas été perturbé, la solution aurait été proche de $x_1 = -2.813967507916267, x_2 = 3.542645120430333$.

Théorème

On peut démontrer le théorème suivant [3]

Soit A une matrice $N \times N$ et λ la plus grande valeur propre de $A^T A$ on a

$\|A\|_2 = \sqrt{\lambda}$

Définition - Nombre de condition spectral de A.

Soit A une matrice $N \times N$ régulière. On appelle nombre de condition spectral de A le nombre positif $\chi(A) = \|A\|_2 . \|A^{-1}\|_2$ où A^{-1} est la matrice inverse de A. On appelle aussi ce nombre le conditionnement spectral de A

6.2.1 Système perturbé - Majoration de l'erreur relative.

Soit le système de N équations à N inconnues

$Ax = b$

et le système perturbé

$Ay = b + \delta b$

On a, par linéarité

$y = x + \delta x$ avec $A\delta x = \delta b$

On peut majorer l'erreur relative $\frac{\|\delta x\|_2}{\|x\|_2}$ en fonction de l'erreur relative $\frac{\|\delta b\|_2}{\|b\|_2}$ commise sur le second membre grâce au...

Théorème

Soit A une matrice $N \times N$ régulière, b un vecteur non nul et δb un vecteur qui perturbe b. Alors, si x et δx sont tels que $Ax = b$ et $A\delta x = \delta b$ on a

$\frac{\|\delta x\|_2}{\|x\|_2} \leq \chi(A) \frac{\|\delta b\|_2}{\|b\|_2}$

où $\chi(A)$ est le nombre de condition spectral de A.

En effet $\|Ax\|_2 \leq \|A\|_2 . \|x\|_2$ appliqué à A^{-1}

en association avec les relations $y = x + \delta x$ et $A\delta x = \delta b$ donne

$\frac{\|\delta x\|_2}{\|x\|_2} = \frac{\|A^{-1}\delta b\|_2}{\|x\|_2} \leq \|A^{-1}\|_2 \frac{\|\delta b\|_2}{\|x\|_2}$

Comme on a aussi

$\|b\|_2 = \|Ax\|_2 \leq \|A\|_2 . \|x\|_2$

On trouve finalement

$\frac{1}{\|x\|_2} \leq \frac{\|A\|_2}{\|b\|_2}$

qui conduit au résultat escompté !

Commentaires

Lorsque la matrice A est symétrique, $\chi(A)$ est égal au rapport des valeurs absolues de la plus grande et de la plus petite valeur propre de A.

L'estimation $\frac{\|\delta x\|_2}{\|x\|_2} \leq \chi(A)\frac{\|\delta b\|_2}{\|b\|_2}$ est « optimale » mais elle s'avère souvent « pessimiste », voire « très pessimiste » en pratique.

Si $\frac{\|\delta b\|_2}{\|b\|_2}$ est de l'ordre de la précision relative $\eta = 10^{-p}$ (c'est-à-dire est exacte jusqu'au $p$ième chiffre) alors, on a, au pire : $\frac{\|\delta x\|_2}{\|x\|_2} = \chi(A).\eta = 10^{log_{10}\chi(A)-p}$.

Règle pratique

Si on calcule la solution du système $Ax = b$ avec un ordinateur qui utilise p chiffres significatifs (en base 10), on ne pourra pas, a priori, garantir plus de $[p - log_{10}\chi(A)]$ chiffres significatifs sur la solution (où [.] désigne la partie entière).

Dans le cas du système 6.1, $\chi(A)$ est de l'ordre de 10^7. On peut donc perdre jusqu'à 7 chiffres significatifs lors de la résolution du système et il faut un calculateur qui utilise 10 chiffres significatifs pour être sûr d'obtenir les trois premiers chiffres de la solution.

6.2.2 Qualité d'une solution et résidu.

Exemple

Soit le système

$$0.0003x_1 + 1.3910x_2 = 9.7460 \tag{6.7}$$
$$0.3961x_1 - 0.5210x_2 = 8.2360 \tag{6.8}$$

Si on le résout en arithmétique flottante à quatre chiffres avec arrondi au plus près en éliminant la première variable de la deuxième équation on obtient une première solution approximative $x^{(1)} = (53.33, 6.995)^T$.

Si on permute les équations avant de résoudre le système on trouve une deuxième solution approximative $x^{(2)} = (29.99, 7.002)^T$

Si au lieu d'éliminer la première variable de la deuxième équation, on élimine la seconde, on obtient deux nouvelles solutions approchées $x^{(3)} = (30.01, 7.000)^T$ et $x^{(4)} = (30.00, 6.994)^T$.

Résidu

Soit $\tilde{x}$ une solution approchée du système $Ax = b$.

On associe à $\tilde{x}$ le vecteur $r(\tilde{x}) = b - A\tilde{x}$ que l'on appelle résidu.

On a évidemment le résultat trivial suivant

x est solution exacte du système $Ax = b$ si et seulement si $r(x) = 0$.

Si l'on part de l'idée que la meilleure solution est celle qui minimise la norme du résidu, les solutions approximatives du système 6.7 peuvent être classées dans l'ordre décroissant de leur qualité : $x^{(3)}$, $x^{(2)}$, $x^{(4)}$ et $x^{(1)}$.

Cependant, l'hypothèse d'un lien rigoureux entre la grandeur du résidu et la précision de la solution est fausse. On le montre aisément à l'aide d'un contre-exemple.

Contre-exemple

Le système

$$0.783x_1 + 0.561x_2 = 0.222 \tag{6.9}$$
$$0.915x_1 - 0.655x_2 = 8.260 \tag{6.10}$$

Admet comme solution approchée $\tilde{x} = (0.284, 0.000)^T$ dont le résidu calculé en virgule flottante de base dix à trois chiffres est $r(\tilde{x}) = 0$.

Cependant, sa solution est $(1, -1)^T$ et non $\tilde{x}$!

6.3 Application

Proposons-nous de déterminer la fonction de contrainte dans une barre de section pentagonale régulière lorsqu'elle est soumise à une faible torsion. Après avoir procédé à une mise à l'échelle des variables d'espace x et y et de la fonction de torsion inconnue u (adimensionnalisation) on est conduit à la résolution d'un problème de Dirichlet pour l'équation de Poisson.

$$\Delta u(x) = -1 \ ; \qquad x \in \Omega \tag{6.11}$$
$$u(x) = 0 \ ; \qquad x \in \partial\Omega \tag{6.12}$$

où Ω représente le domaine pentagonal constitué par la section (réduite) de la barre et $\partial\Omega$ sa frontière. On suppose Ω inscrit dans un cercle de rayon unité.

La symétrie du problème, tant du point de vue géométrique qu'analytique nous permet de remplacer le problème de Dirichlet de départ par un problème aux conditions aux limtes mixtes sur le triangle OAC, moitié d'un des cinq quadrilatères schématisés à la figure 6.1. Le nouveau problème à résoudre s'écrit

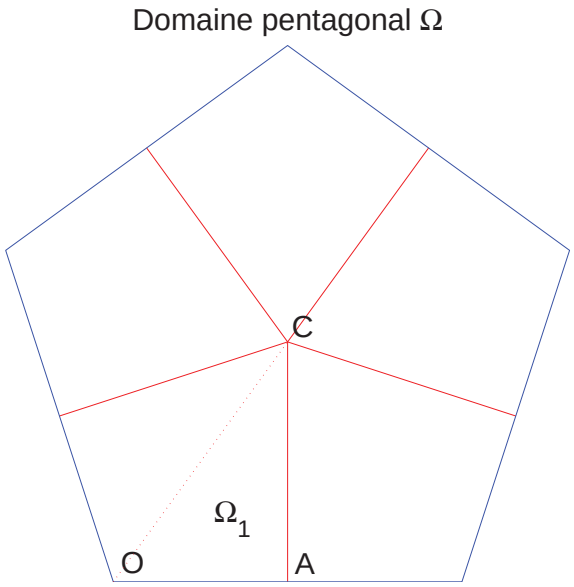


FIGURE 6.1 – Domaine en forme de pentagone régulier.

$$\Delta u(x) = -1 \; ; \qquad x \in \Omega_1 \tag{6.13}$$

$$u(x) = 0 \; ; \qquad x \in OA \tag{6.14}$$

$$\frac{\partial u}{\partial n} = 0 \; ; \qquad x \in AC \tag{6.15}$$

$$\frac{\partial u}{\partial n} = 0 \; ; \qquad x \in OC \tag{6.16}$$

Où Ω_1 est le triangle rectangle OAC de la figure 6.1.

Ou, dans un système de coordonnées polaires (r, θ)centrées en O, avec l'axe Ox orienté selon OA :

$$\Delta u(r,\theta) = \frac{\partial^2 u(r,\theta)}{\partial r^2} + \frac{1}{r}\frac{\partial u(r,\theta)}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u(r,\theta)}{\partial \theta^2} = -1 \; ; \qquad x \in \Omega_1 \tag{6.17}$$

$$u(r,0) = 0 \; ; \qquad x \in OA \tag{6.18}$$

$$\frac{\partial u}{\partial n} = 0 \; ; \qquad x \in OC \tag{6.19}$$

$$\frac{\partial u}{\partial n} = \frac{\partial u}{\partial x} = 0 \; ; \qquad x \in AC \tag{6.20}$$

Où n représente la normale extérieure à Ω_1.

On peut montrer, par simple substitution dans les équations 6.17 à 6.19 qu'elles sont vérifiées par les fonctions

$$u(r,\theta) = \frac{r^2}{4}\left[\frac{\cos(2(\beta-\theta))}{\cos 2\beta} - 1\right] + \sum_{n=1}^{\infty} a_n r^{(n-\frac{1}{2})\frac{\pi}{\beta}} \sin(n-\frac{1}{2})\frac{\pi}{\beta}\theta \qquad (6.21)$$

les coefficients a_n étant des constantes que l'on peut choisir arbitrairement et β l'angle AOC.

En pratique, on limite la somme dans 6.22 à N termes et on choisit les constantes a_n de manière à vérifier approximativement la condition 6.20. Une façon de procéder est d'exiger que cette condition de symétrie soit satisfaite en N points du segment AC ce qui conduit à la résolution d'un système de N équations à N inconnues. Une telle procédure porte le nom de *méthode de collocation*.

Si l'on pose

$$u(r,\theta) \simeq u_N(r,\theta) = \frac{r^2}{4}\left[\frac{\cos(2(\beta-\theta))}{\cos 2\beta} - 1\right] + \sum_{n=1}^{N} a_n r^{(n-\frac{1}{2})\frac{\pi}{\beta}} \sin(n-\frac{1}{2})\frac{\pi}{\beta}\theta \qquad (6.22)$$

cela revient à choisir N points $(r_i,\theta_i)\,(i=1,2,\ldots,N)$ sur AC et à résoudre le système $N\times N$ suivant :

$$\sum_{n=1}^{N}(n-\frac{1}{2})\frac{\pi}{\beta}a_n r_i^{(n-\frac{1}{2})\frac{\pi}{\beta}-1}\sin((n-\frac{1}{2})\frac{\pi}{\beta}-1)\theta_i = -\frac{1}{2}r_i\sin\theta_i\tan 2\beta \qquad (6.23)$$

car

$$\frac{\partial u_N}{\partial x} = \sum_{n=1}^{\infty}(n-\frac{1}{2})\frac{\pi}{\beta}a_n r_i^{(n-\frac{1}{2})\frac{\pi}{\beta}-1}\sin((n-\frac{1}{2})\frac{\pi}{\beta}-1)\theta_i) + \frac{1}{2}r_i\sin\theta_i\tan 2\beta \qquad (6.24)$$

Comme, sur AB, $r\cos\theta = \cos\beta$, le système 6.23 est équivalent à :

$$\sum_{n=1}^{N}\nu_n\frac{\pi}{\beta}a_n(\frac{\cos\beta}{\cos\theta})^{\nu_n\frac{\pi}{\beta}-1}\sin(\nu_n\frac{\pi}{\beta}-1)\theta_i = -\frac{1}{2}\tan\theta_i\cos\beta\tan 2\beta \qquad (0<\theta_i<\frac{\pi}{4}) \quad (6.25)$$

en posant $\nu_n = (n-\frac{1}{2})$.

Remarque

Il y a de nombreuses façons de choisir les points de collocation, c'est-à-dire les points (r_i,θ_i) en lesquels on impose à la condition 6.20 d'être vérifiée. Si l'on prend des points uniformément répartis sur AC on dira que l'on procède à une *collocation équidistante*. S'ils sont disposés de façon à ce que la distance angulaire entre deux points successifs $\theta_{j+1}-\theta_j$ soit constante on parlera de *collocation équi-angulaire*. Un choix fréquent consiste à placer les points en des positions qui soient homologues aux zéros de polynômes orthogonaux.

On parlera de *collocation orthogonale de Chebychev ou de Gauss* selon que les polynômes considérés sont ceux de Chebychev ou de Legendre [2].

On peut également procéder de manière plus subtile en essayant d'optimiser le choix des points. Une procédure pourrait être la suivante :
1°) Faire $N = 1$ et choisir comme point de collocation le point milieu du segment AC
2°) Calculer le résidu $\rho_N = \frac{\partial u}{\partial x}$ le long de AC
3°) Incrémenter N d'une unité et choisir le point de collocation $N + 1$ à l'endroit où le résidu ρ_N est le plus grand en valeur absolue
4°) Retourner à l'étape 2°) ou arrêter les calculs si le résidu est *suffisamment petit*.

Nous appellerons ce procédé *collocation minimisante*.

Le programme ci-dessous résout le problème par la méthode de *collocation équi-angulaire*, pour N allant de 1 à 30.

```
% Résolution de l'équation de Poisson dans un pentagone
% Méthode de collocation équi-angulaire
clc;clear
format long
%   Définition de quelques constantes utiles
beta=3*pi/10;
nu=pi/beta;
C=cos(beta);
%   Boucle générale - Fixe le nombre de paramètres à calculer
for N=1:30
%   Calcul du second membre et de la matrice du système algébrique
    for i=1:N
        dtheta=beta/(N+1);
        theta(i)=i*dtheta;
        b(i)=-C*tan(theta(i))*tan(2*beta)/2;
    for n=1:N
    fac=(n-0.5)*nu;
    M(i,n)=fac*(C/cos(theta(i)))^(fac-1)*sin((fac-1)*theta(i));
    end
    end
%   Résolution du système et calcul du conditionnement de la matrice
    sol=M\b';
    condM(N)=cond(M);
    solutions(N,1:N)=sol;
%   Calcul de la dérivée normale approchée en 1001 points de AC
    for i=1:1001
    t=(i-1)*beta/1000;
    residu(i)=C*tan(t)*tan(2*beta)/2;
```

2. Les zéros des polynômes de Legendre sont dans l'intervalle $(-1, 1)$. On les appelle points de Gauss cf.chapitre 8 sur l'intégration numérique.

```
    for k=1:N
        fac=(k-0.5)*nu;
        residu(i)=residu(i)+sol(k)*fac*(C/cos(t))^(fac-1)*sin((fac-1)*t);
    end
    end
%   Evaluation de la valeur maximum du résidu
%   NB : un résidu nul partout sur AC correspond à la solution exacte
    ERR(N)=max(abs(residu));  % Evaluation de la valeur maximum du résidu
end
%   Représentation des résultats
figure(1)
hold on
plot(log10(ERR),'-o')
figure(2)
hold on
plot(log10(condM),'-o')
format long e
%   Affichage des résultats
[ERR' condM']
solutions'
```

On l'adapte facilement à l'application de la *collocation minimisante*. Pour obtenir le programme plus élaboré suivant :

```
clc;clear
format long
beta=3*pi/10;
nu=pi/beta;
C=cos(beta);
j=0;
theta(1)=beta/2;
for N=1:30
    j=j+1;
    for i=1:N
    b(i)=-C*tan(theta(i))*tan(2*beta)/2;
    for n=1:N
    fac=(n-0.5)*nu;
    M(i,n)=fac*(C/cos(theta(i)))^(fac-1)*sin((fac-1)*theta(i));
    end
    end
    sol=M\b';
    condM(N)=cond(M);
    solutions(N,1:N)=sol;
    for i=1:2001
```

```
    t=(i-1)*beta/2000;
    residu(i)=C*tan(t)*tan(2*beta)/2;
    for k=1:N
        fac=(k-0.5)*nu;
        residu(i)=residu(i)+sol(k)*fac*(C/cos(t))^(fac-1)*sin((fac-1)*t);
    end
    end
    [ERR(N) i]=max(abs(residu));
    theta(j+1)=(i-1)*beta/2000;
end
figure(1)
hold on
plot(log10(ERR),'-om')
figure(2)
hold on
plot(log10(condM),'-om')
format long e
[ERR' condM']
solutions'
```

Les figures 6.2 et 6.3 présentent les résultats obtenus avec diverses méthodes de collocation. On note (cf. figure 6.2) la très grande supériorité de la méthode de *collocation minimisante* qui permet d'obtenir un résidu inférieur à 10^{-15} en valeur absolue alors que les autres méthodes donnent, au mieux, un résidu légèrement inférieur à 10^{-8}(cf. figure 6.2)

On voit également (cf. figure 6.3) que la règle pratique de la page 62 est très pessimiste. Ainsi, par exemple, la *collocation orthogonale de Gauss* conduit, dans le cas où l'on choisit de garder 15 termes dans la somme, à une matrice dont le nombre de condition vaut 1.16×10^{13} alors que le résidu correspondant est inférieur en module à 10^{-8} !

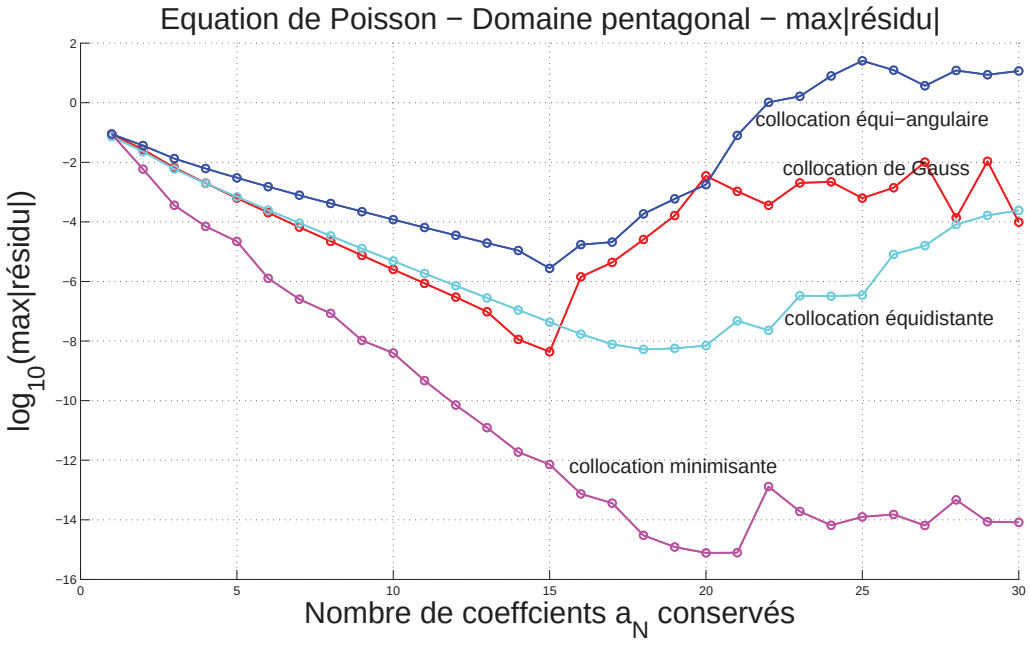


FIGURE 6.2 – Logarithme en base 10 du maximum de la valeur absolue du résidu.

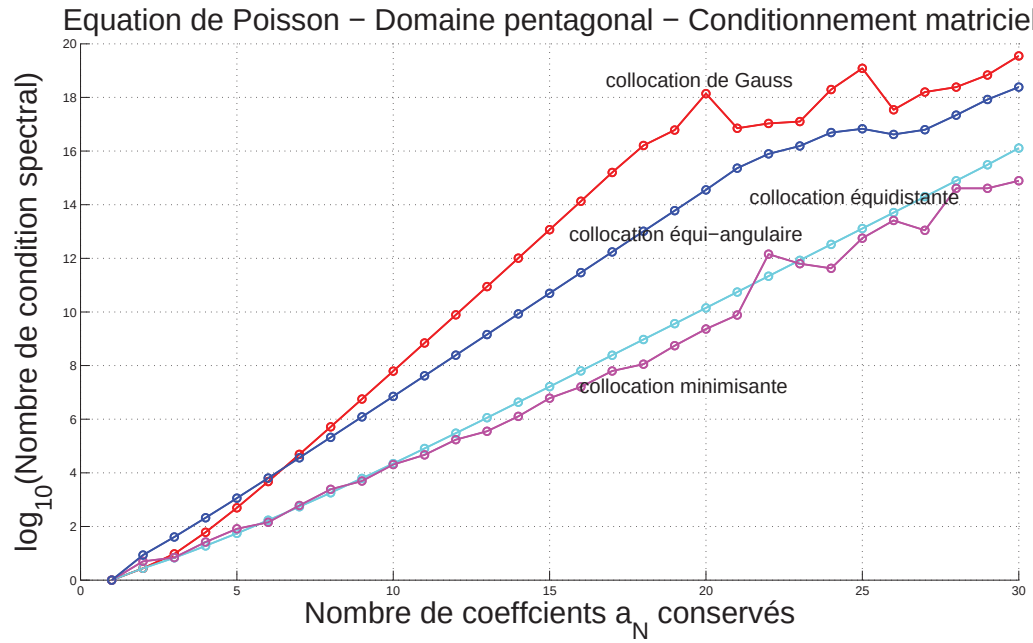


FIGURE 6.3 – Logarithme en base 10 du nombre de condition spectral.

Chapitre 7

Systèmes incompatibles
Méthode des moindres carrés

7.1 Systèmes incompatibles

7.1.1 Introduction

Dans de très nombreuses disciplines (économie, physique, chimie, biologie, etc.), on est confronté au problème de la détermination de grandeurs inconnues. Elles interviennent, par exemple, dans une loi naturelle ou dans un modèle mathématique : conductibilité thermique, vitesse de réaction, viscosité d'un fluide, vitesse de prolifération de micro-organismes, coefficient de diffusion etc.

La détermination de ces grandeurs à partir de mesures expérimentales conduit à ce qu'on nomme l'*estimation de paramètres*.

Généralement, les mesures expérimentales dont on dispose sont plus nombreuses que les paramètres à déterminer. On est alors conduit à résoudre des systèmes d'équations qui contiennent plus d'équations que d'inconnues. En raison des erreurs de mesures ces systèmes sont incompatibles et on cherche à les résoudre « au mieux ».

7.2 Méthode des moindres carrés.

7.2.1 Exemple

La chute d'un corps, assimilé à un point matériel, lancé verticalement vers le haut avec une vitesse initiale v_0, dans le vide (sans frottement de l'air), se fait à une vitesse (comptée positivement si le point monte) :

$v = v_0 - g(t - t_0) \quad [m/s]$

où :

t représente le temps (s), t_0 est l'instant où le corps a été lancé (s) et g est l'accélération due à la pesanteur (m/s^2)

Si l'on suppose v_0 et g inconnues et $t_0 = 0s$, on peut, à partir d'une série de mesures de v en fonction de t, calculer une approximation de ces inconnues.

En principe deux couples de valeurs v et t suffisent pour déterminer les deux inconnues.

Cependant, lorsqu'on effectue des mesures expérimentales, celles-ci sont toujours entachées d'erreurs (dont on connaît souvent un majorant).

Ainsi, si l'on dispose de dix mesures, il y a moyen de former 45 couples de deux équations à deux inconnues conduisant à 45 couples de valeurs de v_0 et g !

Une méthode simple pour déterminer v_0 et g consiste à représenter, dans un système d'axes orthogonaux, les points de mesures (couples v, t) et à « faire passer » une droite entre ces points puisque la relation qui lie v à t est linéaire. La pente de la droite donne g et son abscisse à l'origine fournit v_0.

Une autre méthode, particulièrement bien adaptée lorsqu'on dispose d'un grand nombre de mesures entachées d'erreurs dont la distribution est *normale* (cf. cours de statistiques) est la méthode des moindres carrés.

7.2.2 Equations d'erreur - Equations normales

On considère le système surdéterminé de m équations à n inconnues $x_1, x_2, \ldots x_n$:

$\sum_{j=1}^{n} c_{ij}x_j + d_i = r_i$ avec $i = 1, 2, \ldots, m$

où r_i sont les composantes du vecteur résidu.

Ces équations, appelées équations d'erreur, peuvent s'écrire sous forme matricielle :

$Cx + d = r$

où C, matrice $m \times n$ supposée de rang n et d sont des données, tandis que x est le vecteur des inconnues et r le vecteur résidu.

Si toutes les composantes du résidu étaient nulles, le système serait compatible, bien que comptant plus d'équations que d'inconnues. En pratique, on ne rencontre jamais cette situation et la question se pose de calculer un vecteur x qui résolve *au mieux* le système.

Une façon de procéder est de calculer un vecteur x qui minimise le résidu r selon une mesure que l'on se donne.

Un critère de minimisation classique consiste à rendre minimum la somme des carrés des résidus r_i, c'est-à-dire, à minimiser la norme euclidienne du résidu, ou, ce qui est équivalent, à minimiser la somme des carrés des écarts entre les composantes de Cx et celle de $-d$.

Ce critère est appelé critère des *moindres carrés*.

La recherche du minimum de la norme euclidienne du résidu : $r^T r = Min!$ revient à déterminer les valeurs x_i qui rendent minimum une certaine forme quadratique.

On a :

$r^T r = (Cx+d)^T(Cx+d) = x^T C^T Cx + x^T C^T d + d^T Cx + d^T d = x^T C^T Cx + 2(C^T d)^T x + d^T d$

et la norme euclidienne de r est donc une fonction quadratique des n inconnues x_i.

Pour alléger l'écriture, on pose :

$C^T C = A$ et $b = C^T d$

Comme C, par hypothèse, est de rang n, la matrice symétrique A est définie positive.

En effet, la forme quadratique $Q(x)$, associée à A, vérifie :

$Q(x) = x^T Ax = x^T C^T Cx = (x^T C^T)(Cx) \quad \leq 0 \quad \forall \quad x$

$Q(x) = 0 \Leftrightarrow (Cx) = 0 \Leftrightarrow x = 0$

On doit donc, finalement, minimiser la fonction quadratique :

$F(x) = r^T r = x^T Ax + 2b^T x + d^T d$

Une condition nécessaire pour que $F(x)$ soit minimum au point x est que son gradient $\nabla F(x)$ soit nul en ce point.

Ce qui équivaut à :

$\sum_{j=1}^{n} a_{ij} x_j = -b_i$ avec $i = 1, 2, \ldots, n$

Ce système de n équations à n inconnues constitue les *équations normales de Gauss* et s'écrit, matriciellement :

$Ax + b = 0$

La solution de ce système existe et est unique, on peut l'obtenir, par exemple, au moyen de la méthode de Choleski.

Elle correspond bien à un minimum de r car la matrice hessienne de $F(x)$, constituée de ses dérivées partielles secondes, est la matrice définie positive A.

La méthode classique pour résoudre le système des équations d'erreur consiste en 4 étapes :

1. Calculer $A = C^T C$ et $b = C^T d$ (équations normales)
2. Décomposer $A = LU = U^T U$ (Choleski)
3. Résoudre $Ly + b = 0$ puis $Ux + y = 0$ (systèmes triangulaires)
4. Eventuellement calculer $r = Cx + d$ (évaluer le résidu)

Une autre façon de procéder est de résoudre directement le système $C^T Cx = -C^T d$.

Remarques

1°) Si on considère les vecteurs colonnes c_i de la matrice C, on obtient facilement les éléments de la matrice A et du vecteur b :

$a_{ik} = c_i^T c_k$ et $b_i = c_i^T d_k$ avec $i, k = 1, 2, \ldots, n$

qui apparaissent alors comme des produits scalaires.
2°) La matrice A est symétrique on ne calcule donc que ses éléments diagonaux et sa triangulaire inférieure.
3°) Le calcul complet (résidu inclus) nécessite $Z = m \times n \times (n+5)/2 + n^3/6 + 3n^2/2 + n/3$ multiplications et n extractions de racines carrées.

7.2.3 Application - Equation de Laplace Problème aux conditions aux limites mixtes

Soit un barreau infini de section carrée. Deux faces adjacentes sont isolées thermiquement. Les deux autres sont maintenues à deux températures constantes différentes. En l'absence de source, une approximation du champ de température peut être obtenue en résolvant l'équation de Laplace sur la section carrée du barreau avec les conditions aux limites pertinentes.

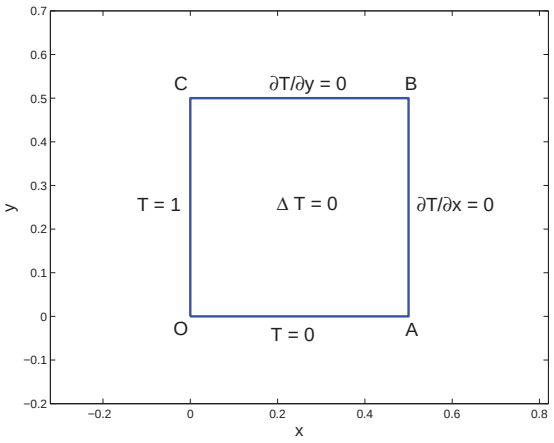


FIGURE 7.1 – Domaine carré de côté $\frac{1}{2}$. Equation et conditions aux limites en variables réduites.

On doit donc résoudre le problème suivant :

$$\Delta T(x,y) = \frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2} = 0 \tag{7.1}$$

$$T(x,y) = 0 \quad \text{sur OA} \tag{7.2}$$

$$T(x,y) = 1/2 \quad \text{sur OB} \tag{7.3}$$

$$\frac{\partial T(x,y)}{\partial x} = 0 \quad \text{sur AB} \tag{7.4}$$

Ou, dans un système de coordonnées polaires centrées en O, le problème équivalent :

$$\Delta T(r,\theta) = \frac{\partial^2 T(r,\theta)}{\partial r^2} + \frac{1}{r}\frac{\partial T(r,\theta)}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T(r,\theta)}{\partial \theta^2} = 0 \tag{7.5}$$

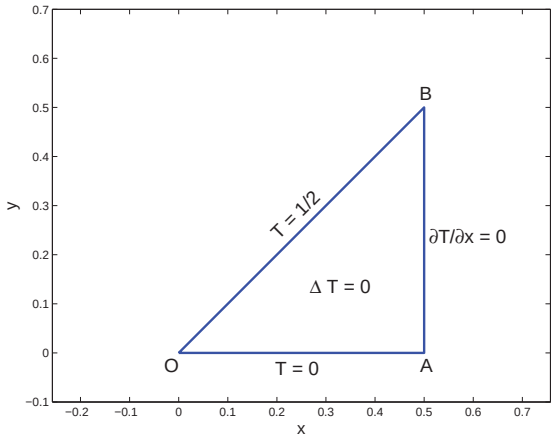


FIGURE 7.2 – La symétrie permet de remplacer le probème de départ par un problème équivalent défini sur une moitié du carré initial.

$$T(r,0) = 0 \tag{7.6}$$

$$T(r,\pi/4) = 1/2 \tag{7.7}$$

$$\frac{\partial T}{\partial x} = 0 \quad \text{sur AB} \tag{7.8}$$

Il est facile de montrer que les fonctions

$$T(r,\theta) = \frac{2\theta}{\pi} + \sum_{n=1}^{\infty} a_n r^{4n} \sin(4n\theta) \tag{7.9}$$

vérifient les équations (7.5,7.6,7.7), les coefficients a_n étant des constantes que l'on peut choisir arbitrairement. En pratique, on limite la somme dans (7.9) à N termes et on choisit les constantes $a_n (n = 1, 2, \ldots, N)$ de manière à vérifier approximativement la condition (7.8). Une façon de procéder est d'exiger que la condition de symétrie (7.8) soit satisfaite en N points du segment AB ce qui conduit à la résolution d'un système de N équations à N inconnues (Méthode de collocation, cf.chapitre 6). Si l'on pose

$$T(r,\theta) \simeq T_N(r,\theta) = \frac{2\theta}{\pi} + \sum_{n=1}^{N} a_n r^{4n} \sin(4n\theta) \tag{7.10}$$

cela revient à choisir N points $(r_i, \theta_i)\,(i = 1, 2, \ldots, N)$ sur AB et à résoudre le système $N \times N$ suivant :

$$\sum_{n=1}^{N} 4n a_n r_i^{4n-1} \sin((4n-1)\theta_i) = \frac{2\sin(\theta_i)}{\pi r_i} \tag{7.11}$$

car

$$\frac{\partial T(r,\theta)}{\partial x} = \sum_{n=1}^{\infty} 4n a_n r^{4n-1} \sin((4n-1)\theta) - \frac{2\sin(\theta)}{\pi r} \tag{7.12}$$

Comme, sur AB, $r\cos(\theta) = \frac{1}{2}$, le système (7.11) est équivalent à :

$$\sum_{n=1}^{N} 4na_n \frac{\sin((4n-1)\theta_i)}{(2\cos(\theta_i))^{4n-1}} = \frac{2\sin(2\theta_i)}{\pi} \qquad (0 < \theta_i < \frac{\pi}{4}) \tag{7.13}$$

On demande :

1°) de calculer des constantes a_n pour différentes valeurs de N ($N = 1, \ldots, 15$ par exemple)
2°) d'estimer la qualité de la solution approchée obtenue pour chaque N en évaluant l'approximation de la dérivée (7.12) sur AB
3°) d'évaluer, pour chaque N, le conditionnement spectral de la matrice du système d'équations à résoudre
4°) de construire un système de M équations du type (7.13) à N inconnues avec $M = 3\times N$ et de le résoudre au sens des moindres carrés
5°) de comparer, à même N, la solution approchée fournie par la méthode des moindres carrés à celle obtenue précédemment.

Solution

1°) et 2°) Le programme suivant permet d'obtenir les figures 7.3, 7.4 et 7.5. On constate que la convergence est exponentielle (figure 7.5). En effet le logarithme en base 10 de la valeur absolue de l'écart maximum entre $\frac{\partial T_N(r,\theta)}{\partial x}$ et zéro sur AB décroît linéairement. Pour $N = 15$ l'erreur locale sur AB n'excède pas 10^{-12} !
3°) Le conditionnement de la matrice est évalué au moyen de la fonction `cond` de Matlab. Le programme prévoit la représentation du logarithme en base 10 de ce conditionnement, sur une figure qui n'est pas reproduite ici.
4°) Pour résoudre le problème au sens des moindres carrés il suffit de remplacer la ligne 15 par `y=(1:3*N)/(6*N)`. Le logiciel fait le reste car l'opérateur `\` utilisé à la ligne 26 résout le système incompatible ($3N \times N$) au sens des moindres carrés. On aurait évidemment aussi pu modifier le programme et passer par les équations normales de Gauss.
5°) La figure 7.6 permet de constater que la méthode des moindres carrés fournit un meilleur résultat. Les figures 7.7 et 7.8 montrent le comportement de l'erreur locale sur AB pour $N = 10$.

```
clc;clear
figure(1)
hold on
grid on
figure(2)
hold on
grid on
figure(3)
hold on
grid on
```

```
ypts=(1:1000)/2000; % Ordonnées de 1000 points équirépartis sur AB
tpts=atan(2*ypts);  % Angles theta associés aux ordonnées (sur AB, x=0.5)
format long
for N=1:15         % Boucle sur N, nombre de coefficients a_n
y=(1:N)/(2*N);      % Choix des ordonnées des points (r_i,theta_i)
theta=atan(2*y);    % Sur AB x=0.5 et theta_i=Arctg(2*y_i)
for i=1:length(y)           % Boucles sur i et j pour construire la matrice
    t=theta(i);     % "mat" du système et son second membre "sec"
    sec(i)=2*sin(2*t)/pi;
    for j=1:N
        f1=4*j;
        f2=f1-1;
        mat(i,j)=f1*sin(f2*t)/((2*cos(t))^f2);
    end
end
a=mat\sec';         % Résolution du système pour trouver les a_n
a
condi(N)=cond(mat); % Calcul du conditionnement de la matrice "mat"
err=0;
for i=1:N           % Calcul de l'erreur en 1000 points de AB
    f1=4*i;
    f2=f1-1;
    err=err+a(i)*f1*sin(f2*tpts)./((2*cos(tpts)).^(f2));
end
err=err-2*sin(2*tpts)/pi;
erreur(N)=max(abs(err));   % Maximum du module de l'erreur absolue
figure(1)
plot(ypts,err/erreur(N),'r')
xlabel('y = r.sin(\theta)')
ylabel('Ecart entre  \partialT/\partialx  et zéro sur AB')
title('Erreur locale sur AB rapportée à son maximum pour N = 1,...,15')
pause                % Pause entre les dessins des courbes
figure(2)
plot(ypts,err,'r')
xlabel('y = r.sin(\theta)')
ylabel('Ecart entre  \partialT/\partialx  et zéro sur AB')
title('Erreur sur AB pour N = 1,...,15')
end
figure(3)
plot(log10(erreur),'r')
xlabel('Nombre de coefficients a_n')
ylabel('log_1_0(Ecart maximum entre  \partialT/\partialx  et zéro sur AB)')
title('log_1_0(Valeur absolue de l''erreur maximale sur AB) en fonction de N')
figure(4)
plot(log10(condi),'r')
```

```
xlabel('Nombre de coefficients a_n')
ylabel('log_1_0(Conditionnement de la matrice NxN)')
title('log_1_0(Conditionnement de la matrice NxN) en fonction de N')
grid on
```

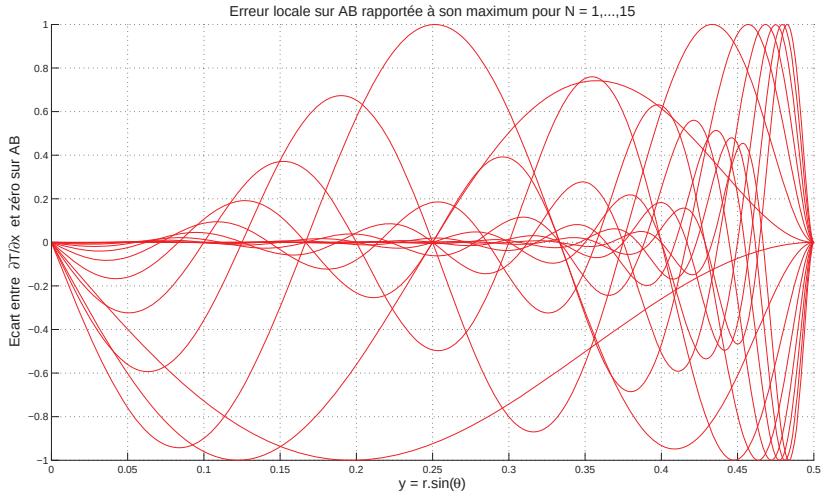


FIGURE 7.3 – Erreur locale sur AB rapportée à son maximum pour N = 1,...,15.

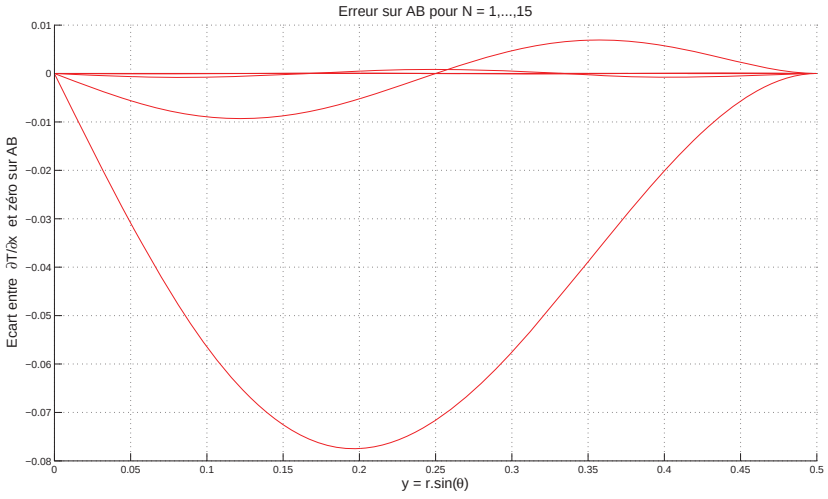


FIGURE 7.4 – Erreur locale sur AB pour différentes valeurs de N.

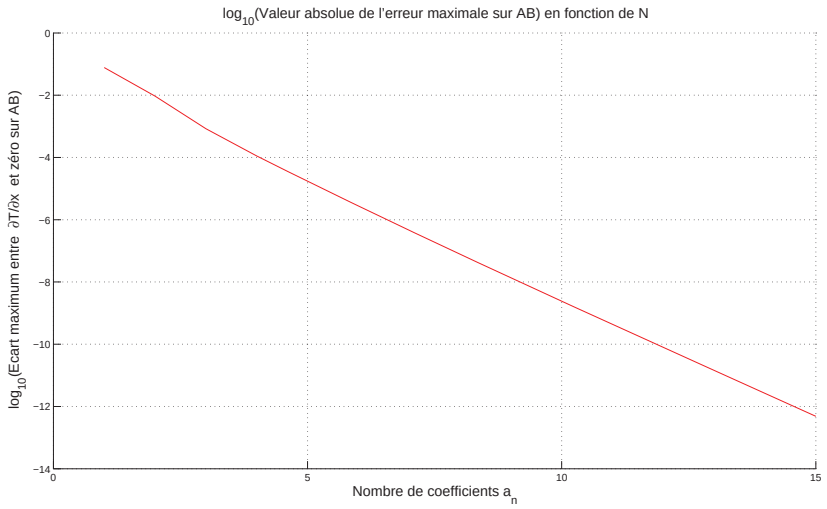


FIGURE 7.5 – Logarithme en base 10 de l'erreur absolue maximale le long de AB.

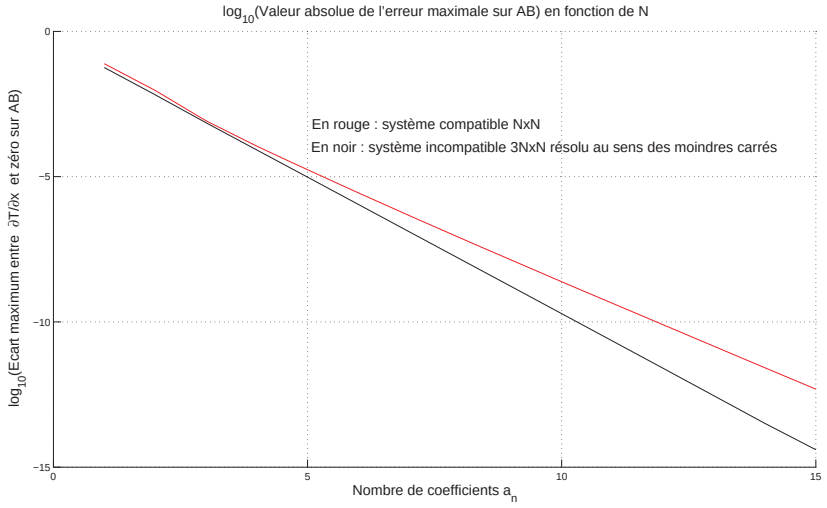


FIGURE 7.6 – Comparaison des erreurs commises par la méthode de collocation "normale" et la méthode de collocation par moindres carrés, en fonction de N.

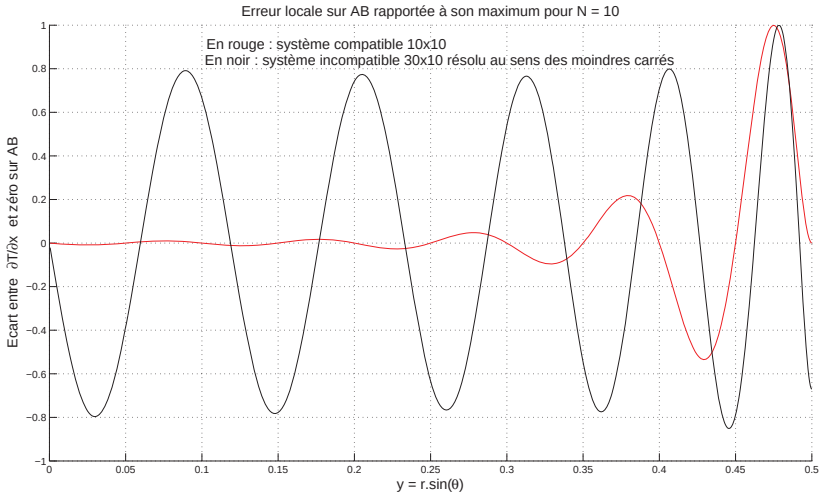


FIGURE 7.7 – Comparaison des erreurs locales rapportées à leur maximum obtenues, pour N = 10, par les deux méthodes (collocation ordinaire et collocation au sens des moindres carrés).

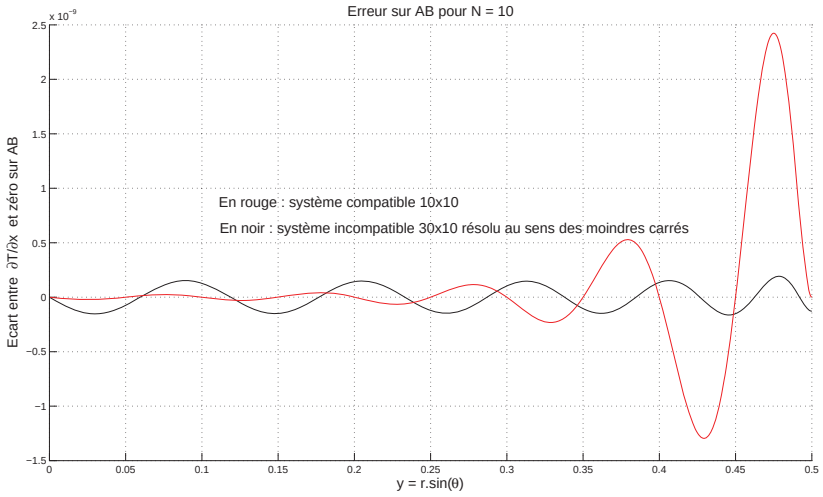


FIGURE 7.8 – Comparaison des erreurs locales obtenues, pour N = 10, par les deux méthodes (collocation ordinaire et collocation au sens des moindres carrés). Remarquez l'échelle verticale : 10^{-9}.

8. INTERPOLATION - LISSAGE

8.1. INTERPOLATION POLYNOMIALE DE VANDERMONDE

Position du problème

On veut trouver un polynôme p de degré $n \geq 0$, qui, pour des valeurs $t_0, t_1, \ldots, t_n$ données, distinctes, prenne les valeurs $p_0, p_1, \ldots, p_n$, respectivement, c'est-à-dire

$p(t_j) = p_j$ pour $0 \leq j \leq n$. (n+1 points)

Une manière directe et (apparemment) simple de résoudre ce problème est d'écrire

$p(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n$

Où $a_0, a_1, a_2, \ldots, a_n$ sont des coefficients à déterminer.

Si ces coefficients sont connus, le polynôme p l'est aussi.

On a :

$p_j = a_0 + a_1 t_j + a_2 t^2_j + \ldots + a_n t^n_j \qquad 0 \leq j \leq n.$

Puisque les valeurs t_j et p_j sont connues, les dernières relations constituent un système de n+1 équations à n+1 inconnues : les $a_0, a_1, \ldots, a_n$.

Ce système peut être écrit sous la forme matricielle suivante :

$$\mathbf{Va} = \mathbf{p}$$

Où la matrice de Vandermonde **V** est la matrice $(n+1) \times (n+1)$ associée aux points t_j :

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & \cdots & t_0^n \\ 1 & t_1 & t_1^2 & t_1^3 & \cdots & t_1^n \\ 1 & t_2 & t_2^2 & t_2^3 & \cdots & t_2^n \\ 1 & t_3 & t_3^2 & t_3^3 & \cdots & t_3^n \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & t_n & t_n^2 & t_n^3 & \cdots & t_n^n \end{bmatrix}$$

a et **p** étant les vecteurs colonne formés respectivement des coefficients a_i du polynôme et des valeurs p_j prises par celui aux abscisses t_j.

Bien que la méthode ci-dessus soit particulièrement simple et évidente dans son principe, et qu'elle conduise à la résolution d'un système linéaire de n+1 équations à n+1 inconnues, toujours compatible, elle n'est pas à conseiller en raison des problèmes numériques qu'elle peut poser.

8.2. INTERPOLATION POLYNOMIALE DE LAGRANGE

Base de Lagrange

Soit les fonctions φ_k de t définies par :

$$\varphi_k(t) = \frac{(t-t_0)(t-t_1)...(t-t_{k-1})(t-t_{k+1})...(t-t_n)}{(t_k-t_0)(t_k-t_1)...(t_k-t_{k-1})(t_k-t_{k+1})...(t_k-t_n)}$$

elles satisfont aux propriétés suivantes :

φ_k est un polynôme de degré n

$\varphi_k(t_j) = 0$ si $j \neq k$, $0 \leq j \leq n$

$\varphi_k(t_k) = 1$

A chaque point t_k, nous avons donc associé un polynôme φ_k de degré n valant un en t_k et zéro aux autres points t_j, $j \neq k$.

Ces polynômes, linéairement indépendants, forment, comme la base canonique 1, t, t^2, …, t^n, une base pour l'espace vectoriel P_n, de dimension n+1, des polynômes de degré inférieur ou égal à n.

Définition

Les φ_k constituent la ***base de Lagrange*** de P_n associée aux points t_0, t_1, t_2, …, t_n.

Polynôme d'interpolation de Lagrange

Le polynôme p cherché est défini par :

$$p(t) = \sum_{j=0}^{n} p_j \varphi_j(t)$$

Exemple

Trouver un polynôme de degré deux qui prenne les valeurs $p_0 = 8$, $p_1 = 3$ et $p_2 = 6$, respectivement en $t_0 = -1$, $t_1 = 0$ et $t_2 = 1$.

La base de Lagrange P_2 associée aux points –1, 0, et 1 est formée des trois paraboles

$\varphi_0 = t.(t-1)/2$; $\varphi_1 = 1 - t^2$; $\varphi_2 = t.(t+1)/2$

et le polynôme demandé est :

$p(t) = 8\varphi_0 + 3\varphi_1 + 6\varphi_2 = 4t^2 - t + 3$

8.3. LISSAGE POLYNOMIAL D'UNE COURBE DONNÉE

Procédure polyfit de Matlab

Syntaxe

p = polyfit(x,y,n)

Description

p = polyfit(x,y,n) calcule les coefficients d'un polynôme $P(x)$ de degré n qui approche les données y(i) par $P(x(i))$ au sens des moindres carrés.

Le résultat, p, est un vecteur ligne à n+1 composantes qui contient les coefficients de $P(x)$ dans l'ordre décroissant des puissances de x.

$P(x) = p_1x^n + p_2x^{n-1} + \ldots + p_nx + p_{n+1}$

Exemple

On se propose d'approcher la fonction d'erreur, erf(x), à l'aide d'un polynôme en x.

On commence par générer un vecteur contenant des abscisses uniformément réparties sur l'intervalle [0,2.5] :

```
x =  (0 : 0.1 : 2.5)' ;  % Transposée !
```

Puis on calcule les valeurs (approchées de erf(x))

```
y = erf(x) ;
```

On calcule et on affiche les coefficients du polynôme de degré six qui approche la fonction au sens des moindres carrés :

```
p = polyfit(x,y,6)
```

Ceci fournit le polynôme d'approximation :

$0.0084x^6 - 0.0983x^5 + 0.4217x^4 - 0.7435x^3 + 0.1471x^2 + 1.1064x + 0.0004$

Pour avoir une idée de la qualité de l'approximation, on évalue le polynôme obtenu aux points d'abscisses données :

```
f = polyval(p,x) ;
```

On calcule et affiche une table contenant les données, les valeurs calculées et l'erreur absolue :

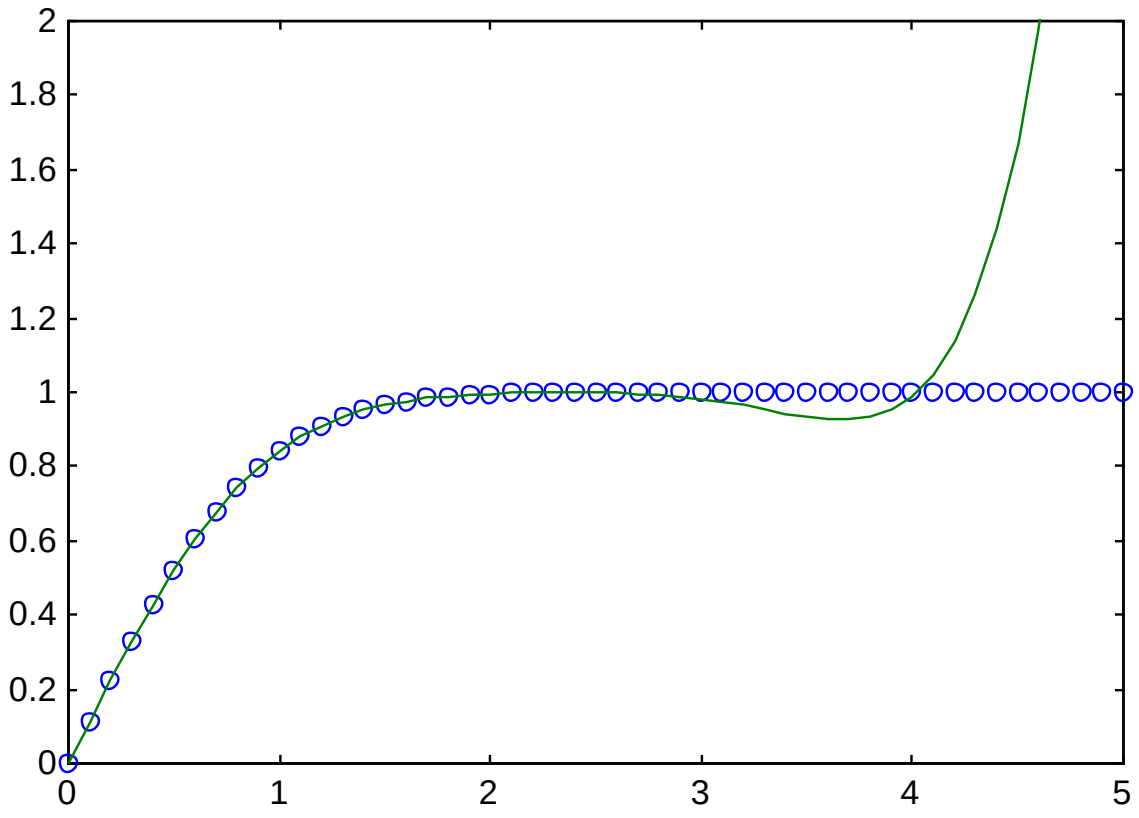
```
table = [x y f y-f]
```

table =

0	0	0.0004	-0.0004
0.1000	0.1125	0.1119	0.0006
0.2000	0.2227	0.2223	0.0004
0.3000	0.3286	0.3287	-0.0001
0.4000	0.4284	0.4288	-0.0004
…			
2.0000	0.9953	0.9951	0.0002
2.1000	0.9970	0.9969	0.0001
2.2000	0.9981	0.9982	-0.0001
2.3000	0.9989	0.9991	-0.0003
2.4000	0.9993	0.9995	-0.0002
2.5000	0.9996	0.9994	0.0002

Les résultats sont bons dans l'intervalle considéré. Toutefois, si on utilise le même polynôme pour approcher erf(x) sur l'intervalle [0,5] les résultats sont catastrophiques.

```
x = (0: 0.1 :5)';  %  Transposée !
y = erf(x);
f = polyval(p,x);
plot(x,y,'o',x,f,'-')
axis([0 5 0 2])
```



Algorithme

Le programme forme la matrice de Vandermonde, dont les éléments sont les puissances de x_i :

$$v_{ij} = x_i^{n-j}$$

Ensuite, l'opérateur « backslash » \ est utilisé pour résoudre le système des équations d'erreur par la méthode des moindres carrés (NB : la matrice de Vandermonde est rectangulaire).

Remarque

On peut modifier le « script » pour utiliser d'autres fonctions de base que x^n.

Deuxième exemple

On considère cette fois un système qui comporte le même nombre d'équations que d'inconnues. On a donc un problème d'interpolation polynomiale « normal », on ne procède pas à un lissage qui minimise un résidu au moyen d'une *certaine norme*.
Le polynôme d'interpolation passe par chacun des points donnés.

Le « script » suivant donne un exemple d'un tel problème :

```
x = (0:1:5)';
y = [-12 2 0 11 -7 1]';
format short
tablxy = [x y]
copol = polyfit(x,y,5)
f = polyval(copol,x);
table = [x y f (y-f)*10^12]
```

Il produit les résultats suivants :

```
tablxy =

   0  -12
   1    2
   2    0
   3   11
   4   -7
   5    1


copol =

    1.4000  -16.9583   71.5833  -125.0417   83.0167  -12.0000
```

```
table =

    0.0000   -12.0000   -12.0000    -2.4496
    1.0000     2.0000     2.0000     0.5063
    2.0000     0.0000    -0.0000     0.0515
    3.0000    11.0000    11.0000     0.2647
    4.0000    -7.0000    -7.0000    -0.8580
```

Il faut remarquer que la dernière colonne de ce tableau présente l'erreur $\times 10^{12}$!

8.4. INTERPOLATION POLYNOMIALE PAR MORCEAUX

L'interpolation par des polynômes de degré trop « élevé » conduit à des instabilités numériques et est d'une utilisation délicate.

En pratique, on lui préfère *l'interpolation polynomiale par morceaux* :

on divise l'intervalle sur lequel il faut procéder à l'interpolation en sous-intervalles présentant un petit nombre de points d'interpolation par lesquels on fait passer un polynôme de faible degré, 1, 2 ou 3, par exemple.

Interpolation linéaire

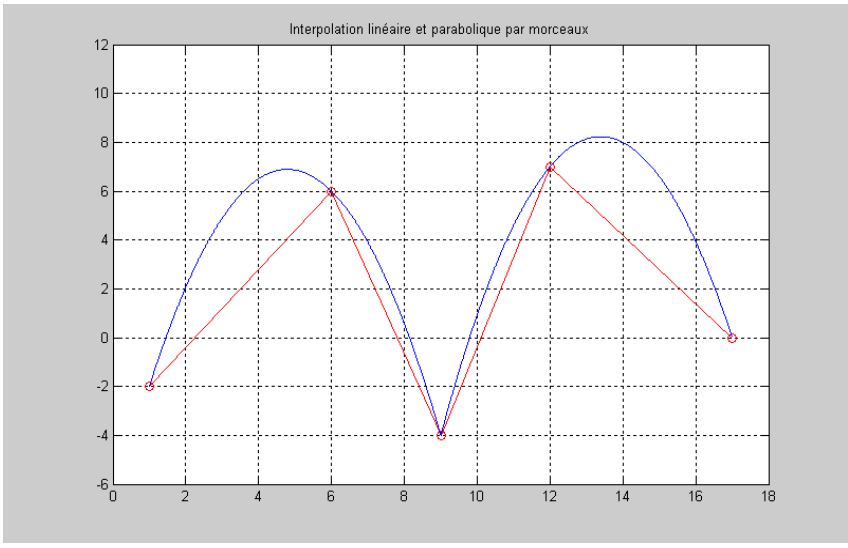
Deux points d'interpolation successifs sont reliés par un segment de droite.

Interpolation quadratique ou parabolique

Trois points d'interpolation successifs sont reliés par un segment de parabole.

Ce type d'interpolation est très utilisé en pratique, il est la base de nombreuses méthodes de résolution d'équations différentielles, ordinaires ou aux dérivées partielles : différences finies, éléments finis, volumes finis, par exemple.

Malheureusement, ce mode d'interpolation présente l'inconvénient de fournir des *fonctions interpolantes* de classe C^0, comme on peut le voir sur le graphe ci-dessous :



8.5. INTERPOLATION D'HERMITE

Pour assurer la continuité C^p des fonctions interpolantes on impose des ***conditions de raccordement*** à leurs représentations sur les sous-intervalles.

On exige, par exemple, la continuité de la dérivée première ou des dérivées jusqu'à l'ordre p.

Une méthode très efficace en pratique et très utilisée est la méthode d'interpolation par ***splines cubiques***.

8.6. INTERPOLATION PAR SPLINES CUBIQUES

Supposons que l'on ***veuille interpoler par morceaux*** une fonction $y(x)$ sur un intervalle $[x_0,x_n]$ que l'on a partitionné en sous-intervalles $[x_{i-1},x_i]$, i=1,2,…,n.

L'interpolation par splines cubiques consiste à remplacer, sur chaque sous-intervalle, la fonction *y* par un polynôme du troisième degré, de sorte que la fonction interpolante soit continue ainsi que ses dérivées première et seconde sur tout l'intervalle $[x_0,x_n]$.

Les courbes splines cubiques simulent l'attitude prise par une fine latte de bois dont la forme est imposée à l'aide de piquets. La déformation de la latte entre deux piquets successifs est représentée par une fonction dont la dérivée quatrième s'annule, c'est-à-dire par un polynôme du troisième degré ou cubique.

Spline est en anglais, le nom d'une latte flexible utilisée en dessin industriel.

Les cubiques d'interpolation s'écrivent :

$$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i\,,\ \ x_{i-1} \le x \le x_i\,,\ \ i = 1,2,...,n$$

Posons $f_i''(x_i) = f_i''$, $i = 1,2,...,n$

La continuité de la dérivée seconde s'écrit :

$$f_i''(x_i) = f_i'' \quad et \quad f_i''(x_{i-1}) = f_{i-1}'' \ , \quad i = 1,2,...,n$$

ou, comme la dérivée seconde est linéaire :

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

soit, en posant $h_i = x_i - x_{i-1}$:

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

En intégrant deux fois il vient :

$$f_i(x) = f''_{i-1}\frac{(x_i - x)^3}{6h_i} + f''_i\frac{(x - x_{i-1})^3}{6h_i} + r_i x + s_i$$

On détermine les constantes d'intégration r_i et s_i à l'aide des valeurs connues de $f_i(x)$ en x_i et x_{i-1} :

$f_i(x_{i-1}) = y_{i-1}$, $f_i(x_i) = y_i$

Après quelques manipulations algébriques, il vient :

$$f_i(x) = f''_{i-1}\frac{(x_i - x)^3}{6h_i} + f''_i\frac{(x - x_{i-1})^3}{6h_i} + \left[\frac{y_{i-1}}{h_i} - f''_{i-1}\frac{h_i}{6}\right](x_i - x) + \left[\frac{y_i}{h_i} - f''_i\frac{h_i}{6}\right](x - x_{i-1})$$

Les fonctions $f_i(x)$ seront entièrement connues quand nous aurons calculé les valeurs de f''_i.

Pour obtenir ces valeurs, nous utilisons les conditions de continuité des dérivées premières aux points intérieurs.

Après dérivation de $f_i(x)$ nous imposons :

$$f'_i(x_i) = f'_{i+1}(x_i) \quad , \quad i = 1,2,...,n-1$$

pour obtenir :

$$h_i f''_{i-1} + 2(h_i + h_{i+1}) f''_i + h_{i+1} f''_{i+1} = \frac{6}{h_{i+1}}(y_{i+1} - y_i) + \frac{6}{h_i}(y_{i-1} - y_i) \quad ; \quad i = 1,2,...,n-1.$$

soit un système de n-1 équations à n+1 inconnues, les f''_i.

Il nous reste donc la possibilité d'imposer deux conditions supplémentaires, obtenues par exemple, par les <u>*conditions aux limites*</u> de l'intervalle $[x_0, x_n]$.

Si nous imposons les deux conditions suivantes :

$$f''_1(x_0) = 0 \quad et \quad f''_n(x_n) = 0$$

nous obtenons les <u>*splines cubiques naturelles*</u>.

Les n-1 inconnues f''_1, f''_2 ,..., f''_{n-1} sont alors solution du système linéaire $\mathbf{A}\boldsymbol{f} = \boldsymbol{b}$, écrit ci-dessus.

Ce système tridiagonal est symétrique.

Les éléments diagonaux de sa matrice **A** sont :

$a_{i,i} = 2(h_i + h_{i+1})$, $i = 1, 2, \ldots, n-1$.

Tandis que les éléments des diagonales secondaires de **A** s'écrivent :

$a_{i+1,i} = a_{i,i+1} = h_{i+1}$, $i = 1, 2, \ldots, n-1$.

Le second membre ***b*** a comme éléments :

$$b_i = \frac{6}{h_{i+1}}(y_{i+1} - y_i) + \frac{6}{h_i}(y_{i-1} - y_i) \ , i = 1,2,\ldots,n-1.$$

Ce système possède une solution unique car sa matrice tridiagonale est à diagonale dominante et est donc inversible.

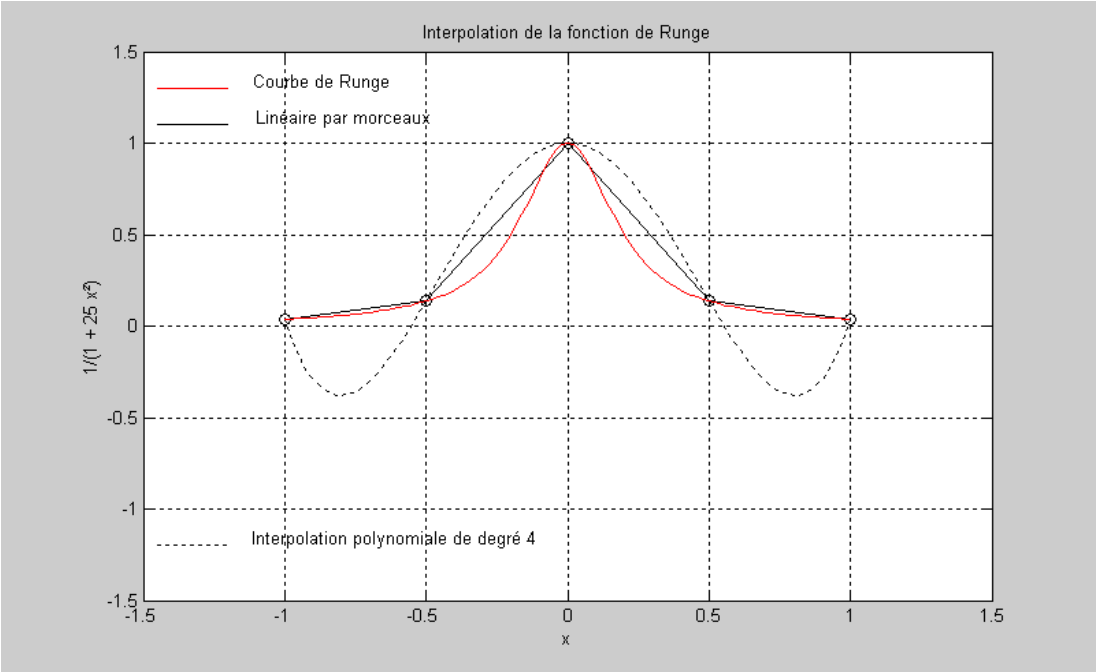
Si les points d'interpolation sont répartis uniformément sur $[x_0,x_n]$, tous les h_i sont égaux et le système devient, pour i = 1, 2, …, n-1 :

$$f''_{i-1} + 4 f''_i + f''_{i+1} = \frac{6}{h^2}[y_{i+1} - 2y_i + y_{i-1}]$$

Remarques

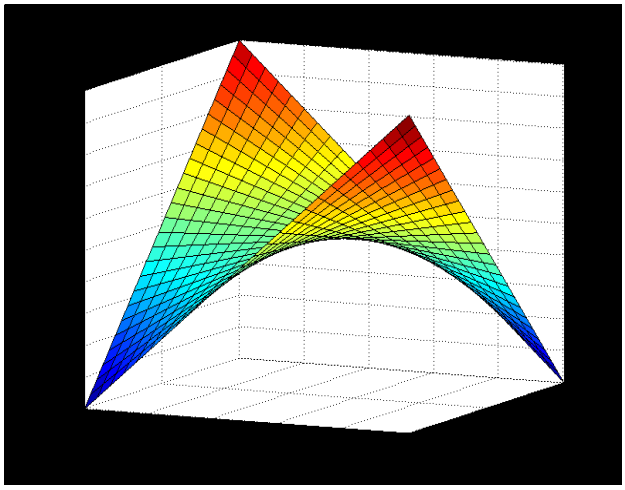
Les splines naturelles sont obtenues en imposant une dérivée seconde nulle aux extrémités x_0 et x_n de l'intervalle. On peut évidemment fixer d'autres conditions en ces points, telles que, par exemple : $f'_1(x_0) = d_0$ *et* $f'_n(x_n) = d_n$.

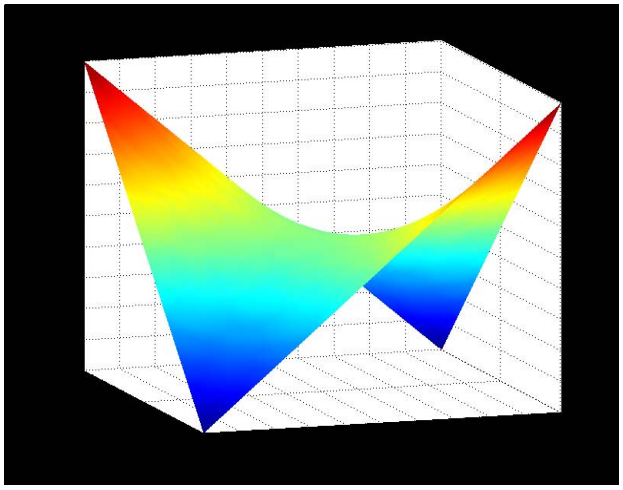
- L'utilisation d'abscisses d'interpolation uniformément réparties est en général déconseillée (*phénomène de Runge* cf. figure ci-dessous). On lui préfère le choix de points d'interpolation distribués comme les *zéros de polynômes orthogonaux* sur l'intervalle $[x_0,x_n]$ (Legendre, Chebyshev, cf. cours d'analyse) .



- Nous nous sommes limités ici, à l'approximation et l'interpolation d'une fonction par des polynômes. Il existe de nombreuses autres possibilités telles que l'utilisation de fonctions trigonométriques, d'exponentielles, de fonctions logarithmiques, de fractions rationnelles, etc.

- La notion d'interpolation ou d'approximation s'étend au cas des fonctions de plusieurs variables. Voici par exemple un « script » matlab qui permet de représenter un paraboloïde hyperbolique dont la figure est obtenue par interpolation linéaire dans les directions x et y.





```
% Représentation d'un paraboloïde hyperbolique
% On peut le voir sous toutes ses coutures !
%
for j=1:26
   x(j)=(j-1)/25;
   for i=1:26
      y(i)=(i-1)/25;
      z(i,j)=(1-x(j))*(1-y(i))+x(j)*y(i);
   end
end
surf(x,y,z) % Donne l'image de gauche
rotate3d on
title('Interpolation bidirectionnelle - Paraboloïde
hyperbolique')
pause
shading interp % On interpole les couleurs (image de droite)
```

8.7. Les fonctions splines et Matlab.

En frappant `help spline` dans la fenêtre de commande de Matlab on obtient ceci :

SPLINE Cubic spline data interpolation.

YY = SPLINE(X,Y,XX) uses cubic spline interpolation to find YY, the values of the underlying function Y at the points in the vector XX. The vector X specifies the points at which the data Y is given. If Y is a matrix, then the data is taken to be vector-valued and interpolation is performed for each column of Y and YY will be length(XX)-by-size(Y,2).

PP = SPLINE(X,Y) returns the piecewise polynomial form of the cubic spline interpolant for later use with PPVAL and the spline utility UNMKPP.

Ordinarily, the not-a-knot end conditions are used. However, if Y contains two more values than X has entries, then the first and last value in Y are used as the endslopes for the cubic spline. Namely:
f(X) = Y(:,2:end-1), df(min(X)) = Y(:,1), df(max(X)) = Y(:,end)

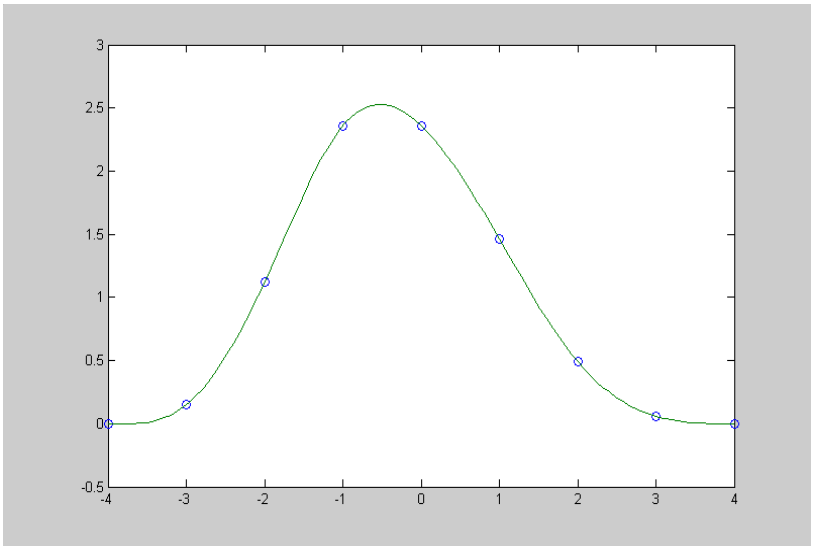
Example: This generates a sine curve, then samples the spline over a finer mesh:

```
x = 0:10;  y = sin(x);
xx = 0:.25:10;
yy = spline(x,y,xx);
plot(x,y,'o',xx,yy)
```

Example: This illustrates the use of clamped or complete spline interpolation where end slopes are prescribed. Zero slopes at the ends of an interpolant to the values of a certain distribution are enforced:

```
x = -4:4; y = [0 .15 1.12 2.36 2.36 1.46 .49 .06 0];
cs = spline(x,[0 y 0]);
xx = linspace(-4,4,101);
plot(x,y,'o',xx,ppval(cs,xx),'-');
```

Ce dernier programme produit la figure ci-dessous :



9. INTÉGRATION NUMÉRIQUE

9.1. INTRODUCTION

Nous développons ci-après quelques méthodes qui permettent de calculer, sur un intervalle fini [*a*,*b*], l'*intégrale définie* $\int_a^b f(x)dx$ d'une fonction f continue donnée.

Ces méthodes sont particulièrement utiles dans le cas où les primitives de f ne sont pas des fonctions élémentaires ou sont trop difficiles à calculer.

C'est le cas par exemple pour les intégrales $\int_0^b e^{-x^2}dx$ *et* $\int_0^b \frac{\sin x}{x}dx$.

Nous distinguerons deux optiques :

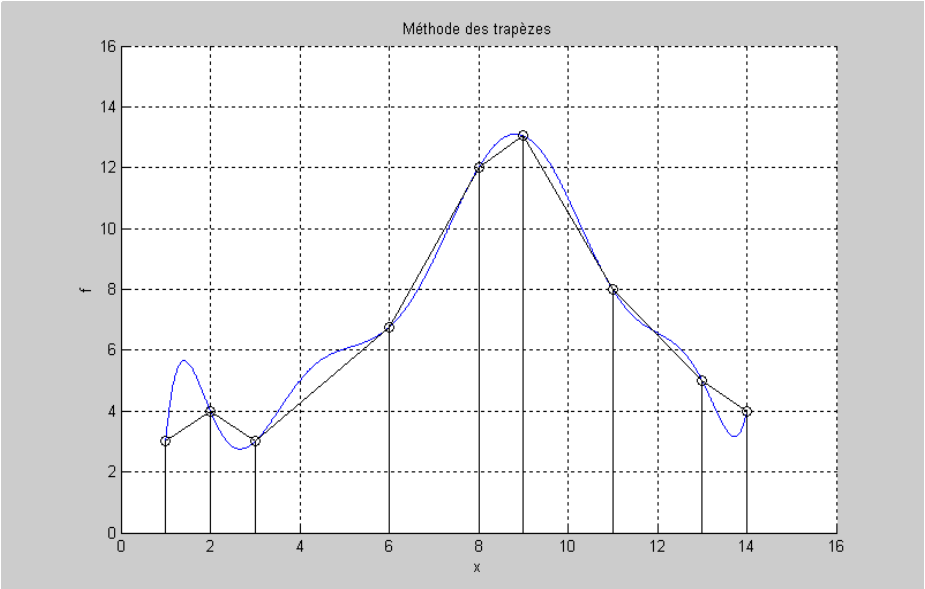
- la fonction à intégrer est remplacée par une fonction interpolante ou par une fonction d'approximation ;

- l'intégrale est approchée par une somme pondérée de valeurs prises par la fonction en des points *situés dans un voisinage* de [*a*,*b*].

9.2. FORMULE DES TRAPÈZES

On subdivise l'intervalle [*a*,*b*] en sous- intervalles $\{[x_{i-1},x_i]$, i = 1,2,…, n; $x_0 = a$; $x_n = b\}$ sur lesquels la fonction f est remplacée par le segment de droite qui joint les points $(x_{i-1}, f(x_{i-1}))$ et $(x_i, f(x_i))$.

Cette procédure revient à remplacer, sur [*a*,*b*], f par une fonction d'interpolation linéaire par morceaux. D'un point de vue géométrique, on assimile l'aire comprise entre le graphe de f et l'axe des x à la somme des aires de n trapèzes.



Considérons que la division en sous-intervalle est uniforme et posons :

$x_i = a + ih$ où $h = \frac{b-a}{n}$ et $f(x_i) = f_i$; i = 0, 1, 2, …, *n*.

Sur l'intervalle $[x_i , x_{i+1}]$ l'aire $\int_{x_i}^{x_{i+1}} f(x)dx$ est remplacée par $h(f_i + f_{i+1})/2$, aire du trapèze correspondant.

En additionnant les aires des n trapèzes, on obtient la *formule des trapèzes* :

$$\int_a^b f(x)dx \approx \frac{h}{2}(f_0 + 2 f_1 + 2 f_2 + \cdots + 2 f_{n-1} + f_n)$$

On peut montrer que l'erreur commise est proportionnelle à h^2 (si la fonction f est deux fois continûment dérivable).

On dit que la méthode des trapèzes est *d'ordre 2*.

La formule est « exacte » pour les fonctions *f* de degré ≤ 1.

9.3. FORMULE DES TRAPÈZES AMÉLIORÉE

La formule des trapèzes peut être améliorée en procédant comme suit, on développe $f(x)$ en série de Taylor au voisinage de x_0 :

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{1}{2!}(x - x_0)^2 f''(x_0) + ...$$

en intégrant de x_0 à x_1, on obtient :

$$\int_{x_0}^{x_1} f(x)dx = hf(x_0) + \frac{h^2}{2!} f'(x_0) + \frac{h^3}{3!} f''(x_0) + ...$$

comme : $f_1 = f(x_1) = f_0 + hf'(x_0) + \frac{h^2}{2!} f''(x_0) + ...$

en multipliant par $h/2$, on a :

$$\frac{h^2}{2!} f'(x_0) = \frac{h}{2}(f_1 - f_0) - \frac{h^3}{4} f''(x_0) + ...$$

et l'intégrale devient :

$$\int_{x_0}^{x_1} f(x)dx = hf(x_0) + [\frac{h}{2}(f_1 - f_0) - \frac{h^3}{4} f''(x_0)] + [\frac{h^3}{3!} f''(x_0)] + ... = \frac{h}{2}(f_0 + f_1) - \frac{h^3}{12} f''(x_0) + ...$$

l'intégrale totale vaut :

$$\int_{x_0}^{x_n} f(x)dx \approx \frac{h}{2}[f_0 + 2f_1 + 2f_2 + ... + 2f_{n-1} + f_n] - \frac{h^2}{12}\sum_{k=0}^{n-1} hf''(x_k)$$

Comme le dernier terme est une approximation de $\int_{x_0=a}^{x_n=b} f''(x)dx$ en le remplaçant par $f'(b) - f'(a)$, on obtient la formule des trapèzes améliorée :

$$\int_a^b f(x)dx \approx \frac{h}{2}[f_0 + 2f_1 + 2f_2 + ... + 2f_{n-1} + f_n] - \frac{h^2}{12}[f'(b) - f'(a)]$$

L'erreur commise est proportionnelle à h^4, et cette méthode est donc d'ordre 4.

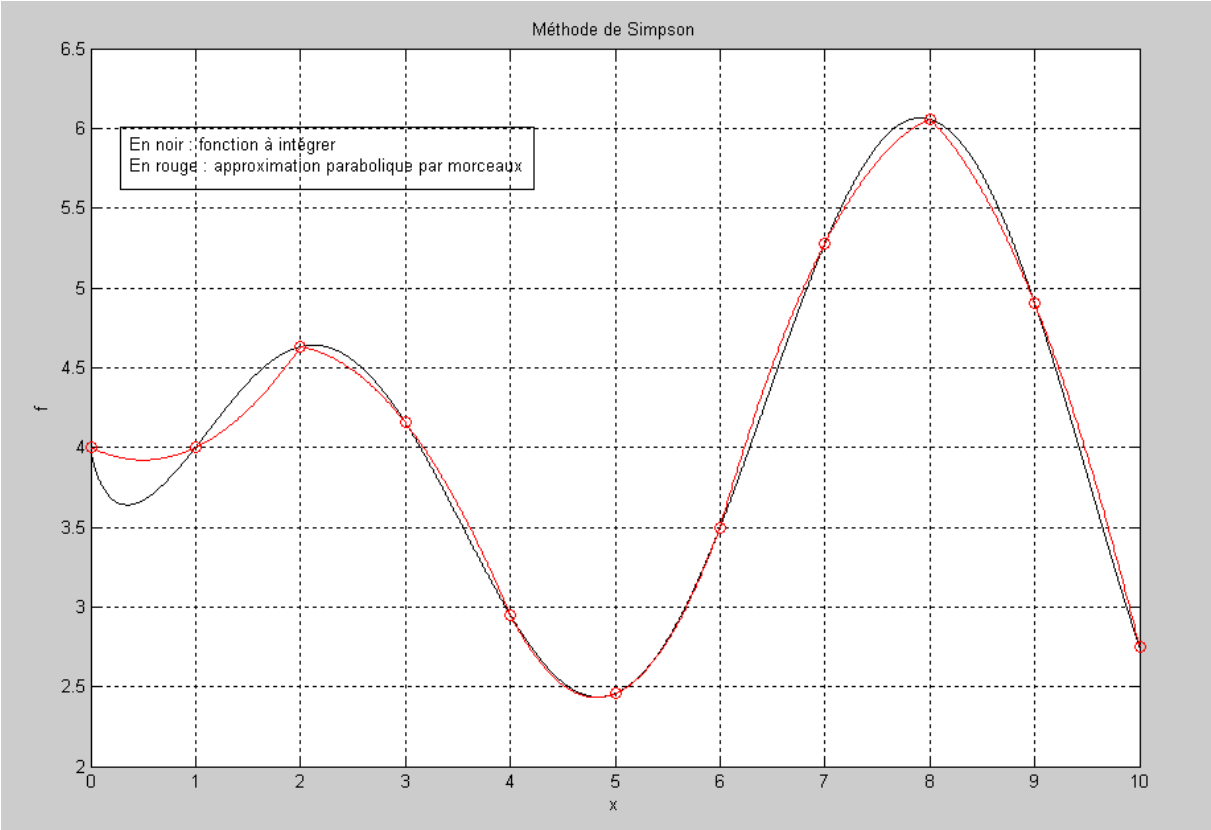
Cette formule montre également que la formule des trapèzes est bien d'ordre 2 et que son erreur est donnée par :

$$E(h) = -\frac{h^2}{12}(b-a)f''(\xi) \quad , \quad \xi \in [a,b].$$

9.4. FORMULE DE SIMPSON

Développons à présent une méthode d'ordre 4 qui équivaut à remplacer la fonction à intégrer par des segments de paraboles définis sur des sous-intervalles comprenant trois abscisses d'intégration successives (cf. figure).

Approximation parabolique par morceaux



On suppose que l'intervalle $[a,b]$ est partagé en n sous-intervalles égaux :

$[x_{i-1}, x_i]$,tels que $x_i = a + ih$, avec $h = (b-a)/n$.

On groupe les points par trois ; n doit donc être pair !

$a = x_0, x_1, x_2 \mid x_2, x_3, x_4 \mid \ldots \mid x_{n-2}, x_{n-1}\, x_n = b.$

Et on remplace, sur chaque intervalle $[x_{i-1}, x_{i+1}]$, la fonction f par une parabole.

Pour l'intervalle $[x_0, x_2]$, la courbe représentée par $f(x)$ est approchée par la parabole d'équation (interpolation par polynôme de Lagrange) :

$$p(x) = f_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f_1 \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

$$= f_0 \frac{(x-x_1)(x-x_2)}{2h^2} + f_1 \frac{(x-x_0)(x-x_2)}{-h^2} + f_2 \frac{(x-x_0)(x-x_1)}{2h^2}$$

et l'intégrale est alors approchée par : $\int_{x_0}^{x_2} f(x)dx \approx \int_{x_0}^{x_2} p(x)dx = \frac{h}{3}[f_0 + 4f_1 + f_2]$

En répétant ce procédé pour les *n/2* intervalles $[x_i, x_{i+2}]$, on a finalement la

formule de Simpson : $\int_a^b f(x)dx \approx \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + ... + 2f_{n-2} + 4f_{n-1} + f_n]$

Cette formule est exacte pour les polynômes $f(x)$ de degré ≤ 3.

9.5. FORMULE DE SIMPSON AMELIOREE

Tout comme pour la méthode des trapèzes, on peut obtenir une *formule de Simpson améliorée* :

$$\int_a^b f(x)dx \approx \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + ... + 2f_{n-2} + 4f_{n-1} + f_n] - \frac{h^4}{180}[f'''(b) - f'''(a)]$$

L'erreur de cette formule est de l'ordre de h^6.

La méthode de Simpson est donc d'ordre 4 et son erreur est donnée par :

$$E(h) = -\frac{h^4}{180}(b-a)f^{(4)}(\xi) \quad , \quad \xi \in [a,b].$$

9.6. MÉTHODE DE ROMBERG (1955)

Désignons la somme des aires des *n* trapèzes de la méthode des trapèzes par $S_0(n)$. Lorsque $n \to \infty$, la suite $S_0(n)$ converge en général assez lentement vers la valeur exacte de l'intégrale. La méthode de Romberg permet d'accélérer considérablement la convergence.

L'idée est la suivante :

Selon la formule des trapèzes améliorée, $S_0(n)$ est liée à la valeur exacte de l'intégrale par une relation de la forme :

$S_0(n) = \int_a^b f(x)dx + Ch^2 + O(h^4) \quad , \quad h = \frac{b-a}{n}$ où la constante *C* ne dépend pas de *h*.

Si on double le nombre d'intervalles on a :

$$S_0(2n) = \int_a^b f(x)dx + C\frac{h^2}{4} + O(h^4) \quad , \quad h = \frac{b-a}{n}$$

Par élimination de l'inconnue *C* on obtient :

$$\int_a^b f(x)dx = \frac{4S_0(2n) - S_0(n)}{3} + O(h^4)$$

On est donc passé d'une précision en h^2 (formule des trapèzes) à une précision en h^4.

Introduisons une nouvelle suite :

$$S_1(2n) = \frac{4S_0(2n) - S_0(n)}{3} \quad , \quad n = 1,2,4,8,...$$

Cette suite converge plus vite vers la valeur de l'intégrale que la suite $S_0(n)$ (si $C \neq 0$).

On voit aisément que $S_1(2n)$ est égale à l'approximation fournie par la méthode de Simpson pour une subdivision de $[a,b]$ en $2n$ intervalles égaux.

On a, en particulier :

$$S_1(2) = \frac{b-a}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$$

Il en résulte donc que :

$$S_1(n) = \int_a^b f(x)dx + C^* h^4 + O(h^6) \quad , \quad h = \frac{b-a}{n}$$

Où C^* est une constante indépendante de h.

En procédant comme ci-dessus, c'est-à-dire en doublant le nombre de sous-intervalles et en éliminant l'inconnue C^* on trouve :

$$\int_a^b f(x)dx = \frac{16S_1(2n) - S_1(n)}{15} + O(h^6)$$

On est ainsi passé d'une précision en h^4 à une précision en h^6, la suite

$$S_2(2n) = \frac{16S_1(2n) - S_1(n)}{15} \quad , \quad n = 2,4,8,...$$

converge donc en général plus vite vers la valeur exacte de l'intégrale.

Cette démarche se généralise (pour une fonction suffisamment dérivable) et donne

l'algorithme de Romberg :

Pour $n = 1, 2, 4, 8, \ldots$ calculer :

$$S_0(n) = \frac{h}{2}[f_0 + 2f_1 + 2f_2 + \ldots + 2f_{n-1} + f_1],\ h = \frac{b-a}{n}$$

Pour $n = 2^{m-1}, 2^m, 2^{m+1}, \ldots$; $m = 1, 2, 3, \ldots$

calculer :

$$S_m(2n) = \frac{4^m S_{m-1}(2n) - S_{m-1}(n)}{4^m - 1}$$

9.7. MÉTHODE DES COEFFICIENTS INDÉTERMINÉS

Soit à calculer l'intégrale $\int_{-1}^{+1} f(x)dx$ à l'aide d'une formule d'intégration qui soit exacte pour tout polynôme de degré inférieur ou égal à n (on généralise aisément si l'intervalle d'intégration est $[a,b]$).

On peut procéder de la manière suivante :

1°) on subdivise l'intervalle d'intégration en n sous-intervalles égaux
$x_i = -1 + i.h,\ h = 2/n,\ i = 0, 1, 2, \ldots, n.$

2°) on pose $\int_{-1}^{+1} f(x)dx \approx w_0 f(x_0) + w_1 f(x_1) + \ldots + w_n f(x_n)$

3°) on impose que la dernière expression soit exacte pour $1, x, x^2, x^3, \ldots, x^n$.

4°) on résout le système de $n+1$ équations qui fournit les *poids d'intégration* w_n.

Cette procédure donne les formules de *Newton-Cotes* dont les formules des trapèzes et de Simpson sont des cas particuliers. Elles sont exactes pour les polynômes de degré n si n est impair et de degré $n+1$, si n est pair. On constate donc qu'il peut être intéressant de diviser l'intervalle d'intégration en un nombre pair de sous-intervalles.

NB : il n'est pas indispensable que les sous-intervalles soient égaux.

9.8. INTÉGRATION DE GAUSS-LEGENDRE

Si on applique la méthode des coefficients indéterminés sans imposer, a priori, la position des *points d'intégration* (les x_i), on dispose de $2n + 2$ grandeurs ajustables pour construire une formule d'intégration.

Ces grandeurs sont les $n+1$ abscisses x_i et les $n+1$ poids d'intégration w_i.

On peut donc, en principe, en n'utilisant que n+1 points d'intégration, obtenir des formules d'intégration numériques exactes pour les polynômes de degré $2n + 1$.

9.8.1. FORMULE À DEUX POINTS

Etablissons la *formule de Gauss-Legendre à deux points* d'intégration :

$$\int_{-1}^{+1} f(x)dx \approx G_2(f) = w_0 f(x_0) + w_1 f(x_1)$$

exacte pour les polynômes de degré ≤ 3.

La formule d'intégration sera exacte pour toute cubique $f(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ si elle est exacte pour les quatre fonctions $1, x, x^2, x^3$ (linéarité + additivité).

Les conditions sur x_0, x_1, w_0 et w_1 sont donc :

$$\int_{-1}^{+1} 1dx = 2 = w_0 + w_1$$
$$\int_{-1}^{+1} xdx = 0 = w_0x_0 + w_1x_1$$
$$\int_{-1}^{+1} x^2dx = \frac{2}{3} = w_0x_0^2 + w_1x_1^2$$
$$\int_{-1}^{+1} x^3dx = 0 = w_0x_0^3 + w_1x_1^3$$

La résolution de ce système d'équations non linéaire fournit, sans trop de difficultés :

$w_0 = w_1 = 1$ et $-x_0 = x_1 = 1/\sqrt{3}$

Et la règle de Gauss-Legendre à deux points s'énonce comme suit :

si f(x) est continue sur [-1, +1], alors $\int_{-1}^{+1} f(x)dx \approx G_2(f) = f(-1/\sqrt{3}) + f(1/\sqrt{3})$

Cette formule est exacte pour tout polynôme de degré 3 et, si $f \in C^4[-1,+1]$, on a :

$$\int_{-1}^{+1} f(x)dx \approx f(-1/\sqrt{3}) + f(1/\sqrt{3}) + \frac{f^{(4)}(\xi)}{135} \quad \text{avec } \xi \in [-1,+1].$$

9.8.2. FORMULE À N POINTS

La règle d'intégration de Gauss-Legendre à N points est exacte pour tout polynôme de degré inférieur ou égal à $2N - 1$ et s'écrit :

$$G_N(f) = w_{N,1} f(x_{N,1}) + w_{N,2} f(x_{N,2}) + \ldots + w_{N,N} f(x_{N,N})$$

Les abscisses et les poids d'intégration ont été tabulés et sont disponibles dans la littérature (cf. Handbook of Mathematical Functions, Abramowitz and Stegun (1968), pp.916 et seq. : *A.S.*) et dans divers logiciels de calcul numérique.

9.9. INTÉGRATION MULTIDIMENSIONNELLE

Les formules d'intégration de fonctions d'une variable peuvent se généraliser au cas des fonctions de plusieurs variables.

A une formule du type $\int_a^b f(x)dx = \sum_{i=1}^{n} w_i f(x_i)$,

on peut faire correspondre une formule

$$\int_c^d \int_a^b f(x,y)dxdy = \sum_{i=1}^{n}\sum_{j=1}^{m} w_i w_j f(x_i, y_j)$$

qui permet d'intégrer de manière approchée, une fonction continue de deux variables. sur le rectangle $[a,b]\times[c,d]$.

Les poids et abscisses sont les mêmes que pour l'intégration de fonctions d'une variable.

Ils sont définis sur les intervalles $[a,b]$ et $[c,d]$.

La généralisation à un nombre plus élevé de dimensions est immédiate mais elle ne fournit pas la méthode la plus efficace et la moins coûteuse en temps de calcul (d'ailleurs, à deux dimensions non plus !).

9.10. REMARQUES

Il existe de très nombreuses méthodes dédiées à l'intégration de fonctions présentant des caractéristiques particulières :

- fonctions oscillant très fortement sur l'intervalle d'intégration (méthodes adaptatives);
- fonctions dont certaines dérivées sont connues en des points de l'intervalle d'intégration;
- fonctions à intégrer sur un intervalle non borné (Hermite-Legendre);
- fonctions définies dans un polygone ou un polyèdre;
- etc.

- dans Matlab « `help quad` » donne ce qui suit :

QUAD Numerically evaluate integral, adaptive Simpson quadrature.

Q = QUAD(FUN,A,B) tries to approximate the integral of function FUN from A to B to within an error of 1.e-6 using recursive adaptive Simpson quadrature.
The function Y = FUN(X) should accept a vector argument X and return a vector result Y, the integrand evaluated at each element of X.

Q = QUAD(FUN,A,B,TOL) uses an absolute error tolerance of TOL
instead of the default, which is 1.e-6. Larger values of TOL result in fewer function evaluations and faster computation, but less accurate results. The QUAD function in MATLAB 5.3 used a less reliable algorithm and a default tolerance of 1.e-3.

[Q,FCNT] = QUAD(...) returns the number of function evaluations.

QUAD(FUN,A,B,TOL,TRACE) with non-zero TRACE shows the values of [fcnt a b-a Q] during the recursion.

QUAD(FUN,A,B,TOL,TRACE,P1,P2,...) provides for additional arguments P1, P2, ... to be passed directly to function FUN, FUN(X,P1,P2,...). Pass empty matrices for TOL or TRACE to use the default values. Use array operators .*, ./ and .^ in the definition of FUN so that it can be evaluated with a vector argument.

Function QUADL may be more efficient with high accuracies and smooth integrands.

Example:

FUN can be specified three different ways.

A string expression involving a single variable:

```
Q = quad('1./(x.^3-2*x-5)',0,2);
```

An inline object:

```
F = inline('1./(x.^3-2*x-5)');
Q = quad(F,0,2);
```

A function handle:

```
Q = quad(@myfun,0,2);
```

where myfun.m is an M-file:

```
function y = myfun(x)
y = 1./(x.^3-2*x-5);
```

See also QUADL, DBLQUAD, INLINE, @.

9.11. EXEMPLES

9.11.1. Intégrale simple

Calculons une valeur approchée de $\int_1^5 \frac{dt}{t}$ par la formule de Gauss-Legendre à trois points.

La formule de Gauss-Legendre à trois points, pour l'intervalle standard [-1,+1], est (cf. tables de poids et abscisses, p.e. : *A.S.*) :

$$\int_{-1}^{+1} f(x)dx \approx G_3(f)$$

$$avec \quad G_3(f) = \frac{5f(-\sqrt{3/5}) + 8f(0) + 5f(\sqrt{3/5})}{9}$$

Si $f \in C^6[-1,+1]$ l'erreur $E_3(f)$ est : $E_3(f) = \frac{f^{(6)}(\xi)}{15.750}$ $avec$ $\xi \in [-1,+1]$

Comme l'intervalle d'intégration n'est pas l'intervalle [-1,+1], on effectue un changement de variable pour obtenir la formule d'intégration :

$$\int_a^b f(t)dt = \frac{b-a}{2} \sum_{k=1}^{N} w_{N,k} \; f\left(\frac{a+b}{2} + \frac{b-a}{2} x_{N,k}\right)$$

Le calcul approché de $\int_1^5 \frac{dt}{t}$ donne, avec $a = 1$ et $b = 5$:

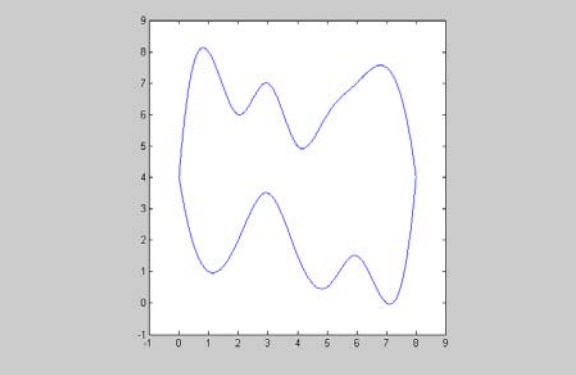
$$G_3(f) = 2\frac{5f(3-2(0.6)^{1/2}) + 8f(3+0) + 5f(3+2(0.6)^{1/2})}{9}$$

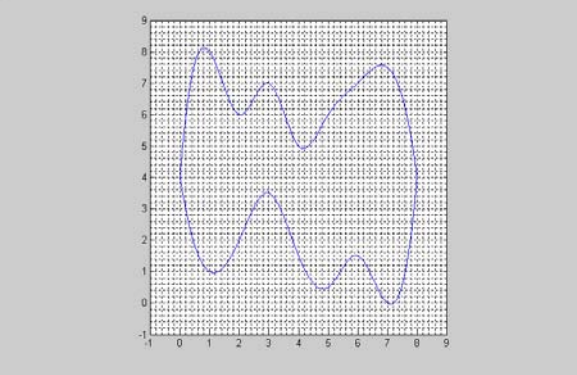
$$G_3(f) = 2\frac{3.446359 + 2.666667 + 1.099096}{9} = 1.602694$$

valeur approchée à comparer à $n(5) - ln(1) \approx 1.609438$

9.11.2. Intégrale double

On veut calculer $\iint_\Omega f(x,y)dxdy$ où Ω représente le domaine ci-dessous





On place une grille à mailles carrées sur le domaine Ω (figure de droite) et sur chaque maille on utilise une formule d'intégration adaptée au carré (cf. *A.S.*, p.892).

On somme ensuite les contributions de chaque carré à l'intégrale.

L'erreur commise dépend évidemment de la taille des mailles.

Ce type de méthode est fréquemment utilisé lorsqu'on résout des équations aux dérivées partielles et se généralise aisément à trois dimensions (*méthode des éléments finis*, *méthode des volumes finis*).

On emploie très souvent un maillage triangulaire dans le plan et tétraédrique dans l'espace.

Les points où l'on évalue la fonction à intégrer sont parfois appelés *points de Gauss*.

9.12. PROGRAMME À DÉCORTIQUER ET EXPÉRIMENTER

```
% Calcul approché d'une intégrale - Illustration graphique
% Règles des rectangles - Règle des trapèzes
function Integration_Numerique(NB_Intervalles)
clc;clf
Q = quad(@F,-1,1); Integrale = num2str(Q);
t = -1:0.01:1; f=F(t);
plot(t,f,'r-','LineWidth',2)
hold on; grid on; axis([-1 1 0 5]); pause
deltaX=2/NB_Intervalles;
TRAP=(F(-1)+F(1))/2;
for i=1:NB_Intervalles-1
    abscisse=-1+i*deltaX;
    TRAP=TRAP+F(abscisse);
end
TRAP=TRAP*deltaX;
TRAP
portions(-1,NB_Intervalles,'g')
gtext(Integrale,'Fontsize',18,'Color','b')
pause
clf
plot(t,f,'r-','LineWidth',2)
hold on; grid on; axis([-1 1 0 5]);
pause
portions(-1,NB_Intervalles,'d')
gtext(Integrale,'Fontsize',18,'Color','b')
pause
clf
plot(t,f,'r-','LineWidth',2)
hold on; grid on; axis([-1 1 0 5]);
pause
portions(-1,NB_Intervalles,'c')
gtext(Integrale,'Fontsize',18,'Color','b')
pause
clf
plot(t,f,'r-','LineWidth',2)
hold on; grid on; axis([-1 1 0 5]);
pause
portions(-1,NB_Intervalles,'t')
gtext(Integrale,'Fontsize',18,'Color','b')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function portions(XRG,NB,type)
if type=='g'
    fac=0; col='k';
end
if type=='d'
    fac=1; col='c';
end
if type=='c'
    fac=0.5; col='g';
end
if type=='t'
    fac=0; col='k';
end
rot=0;
if NB > 9
    rot=-90;
end
deltaX=2/NB;
for n=1:NB
 Y1=F(XRG+fac*deltaX);
 Y2=Y1;
 if type=='t'
     Y2=F(XRG+deltaX);
 end
 X=[XRG XRG+deltaX XRG+deltaX XRG];
 Y=[Y1 Y2        0          0  ];
 fill(X,Y,col)
 plot(X,Y,'r-','LineWidth',2)
 XRG=XRG+deltaX;
 Aire(n)=deltaX*Y1;
 if type=='t'
     Aire(n)=(Y1+Y2)*deltaX/2;
 end
end
t=-1:0.01:1; f=F(t);
plot(t,f,'r-','LineWidth',2)
for n=1:NB
 aire=num2str(Aire(n));
 gtext(aire,'Fontsize',12,'Color','r','Rotation',rot)
end
Airetotale=sum(Aire);
airetotale=num2str(Airetotale);
 if type=='g'
     airetotale=['Rectangles de gauche ' airetotale];
 end
 if type=='d'
     airetotale=['Rectangles de droite ' airetotale];
 end
 if type=='c'
     airetotale=['Rectangles centrés    ' airetotale];
 end
 if type=='t'
     airetotale=['Trapèzes             ' airetotale];
 end
gtext(airetotale,'Fontsize',18,'Color','r')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=F(x)
y=3+exp(-x).*sin(-2*x+5*x.^4);
```

Chapitre 10

Equations différentielles ordinaires

10.1 Introduction

On se propose de résoudre l'équation différentielle suivante

$$\frac{dy}{dt}(t) = y'(t) = f(t, y(t)) \quad t \in [a, b]$$

avec comme condition initiale : $y(a) = y_0$.

En ce qui concerne les théorèmes d'existence et d'unicité de ce problème de Cauchy nous renvoyons le lecteur au cours d'analyse mathématique.

Sauf dans de rares cas particuliers, il est impossible de trouver une solution analytique d'un tel problème. On est donc amené à essayer de calculer des valeurs approchées de la fonction inconnue y en certains points de l'intervalle $[a, b]$.

En pratique nous allons générer une suite de points (t_k, y_k) tels que y_k soient des valeurs approchées de la fonction inconnue : $y(t_k) \simeq y_k$.

Divisons l'intervalle $[a, b]$ en N sous-intervalles que nous choisissons égaux par facilité. Le $k$ième intervalle est limité par t_{k-1} et t_k, et l'on pose : $t_k = a + k.h$ pour $k = 0, 1, 2, \ldots, N$ avec $h = \dfrac{b-a}{N}$.

On appelle h le pas d'intégration et l'on a évidemment $a = t_0$ et $b = t_N$.

Les points t_k forment ce qu'on appelle indifféremment grille, réseau ou maillage.

10.2 Méthodes d'Euler

Supposons que les fonctions $y(t), y'(t)$ et $y''(t)$ sont continues sur $[a, b]$.

Le théorème de Taylor nous permet d'écrire :

$y(t_1) = y(t_0 + h) = y(t_0) + h.y'(t_0) + \frac{h^2}{2}y''(\xi) \quad$ où $\quad \xi \in [a, b]$.

Comme $y'(t_0) = f(t_0, y_0)$, si h est suffisamment petit, on peut négliger le terme en h^2 et obtenir la formule d'Euler progressive, ou formule d'Euler explicite :

$y_1 = y_0 + h.f(t_0, y_0)$.

Remarque

Cette formule peut également être obtenue par la méthode d'intégration du rectangle.

Supposons h suffisamment petit pour que l'on puisse approcher $f(t, y(t))$ sur $[t_0, t_1]$ par $f(t_0, y_0)$, on obtient alors :

$\int_{t_0}^{t_1} y'(t)dt = y(t_1) - y(t_0) = \int_{t_0}^{t_1} f(t, y(t))dt \simeq \int_{t_0}^{t_1} f(t_0, y_0)dt = h.f(t_0, y_0)$

et l'on retrouve bien la formule d'Euler explicite : $y_1 = y_0 + h.f(t_0, y_0)$.

Partant du point (t_0, y_0) on a donc construit le point (t_1, y_1). De ce point on peut déduire le point suivant (t_2, y_2) et, de proche en proche, l'ensemble des points $(t_k, y_k), k = 1, 2, \ldots, N$.

La formule d'Euler explicite permet donc de trouver des valeurs approchées du problème de Cauchy selon l'algorithme suivant :

Choisir N

$h = \dfrac{b - a}{N}$

$t_0 = a$

pour $k = 0, 1, 2, \ldots, N - 1$

faire

$y_{k+1} = y_k + h.f(t_k, y_k)$
$t_{k+1} = t_k + h.$

10.2.1 Exemple 1

```
% Résolution du problème de Cauchy
%
% y'(t) = (t-y)/2 ;  y(0) = 1
%
% par la méthode d'Euler explicite (ou progressive).
%
% Comparaison avec la solution exacte sur [t0 tf]
%
% Appelle euler qui utilise @fex1
%
figure(1)
t0=0; tf=6;
format short
hold on; grid on
axis([0 6 -1 5]);
title('Problème de Cauchy     y''(t) = ( t - y ) / 2  ;  y(0) = 1','FontSize',14)
xlabel('t','FontSize',14); ylabel('y(t)','FontSize',14)
for N=2:2:12
    E=euler(@fex1,t0,tf,1,N);
    plot(E(:,1),E(:,2))
    pause
end
%
N=100;
h=(tf-t0)/N; t=t0:h:tf; y=3*exp(-t/2) - 2 + t; plot(t,y,'k'); pause
E=euler(@fex1,t0,tf,1,N); plot(E(:,1),E(:,2),'r'); pause
%
figure(2)
title('Erreur relative','FontSize',14)
xlabel('t','FontSize',14); ylabel('erreur relative','FontSize',14);
grid on
hold on
for N=[150 300 600]
h=(tf-t0)/N; t=t0:h:tf; y=3*exp(-t/2) - 2 + t; E=euler(@fex1,t0,tf,1,N);
Y=E(:,2);
errel=(y-Y')./y;
plot(t,errel)
legend=strcat('h=',num2str(h));
gtext(legend,'FontSize',14);
pause
pas=h
errmax=max(abs(errel))
end
```

```
gtext('Méthode d''Euler explicite O(h)','FontSize',14)

% FONCTION EULER

function E=euler(f,t0,tf,y0,N)
%
% Résout le problème de Cauchy
% y' = f(t,y),  y(t0) = y0
% par la méthode d'Euler explicite à pas constant
%
% ENTREES
%
% f est le second membre de l'équation à intégrer - (string)
% t0 et tf sont les bornes de l'intervalle d'intégration
% y0 est la condition initiale : y(t0) = y0
% N est le nombre de pas d'intégration
%
% SORTIES
%
% E = [T' Y'] où T est le vecteur des abscisses de calcul
% et Y celui des valeurs approchées de la fonction cherchée

h=(tf-t0)/N;
T=zeros(1,N+1); Y=zeros(1,N+1);
T=t0:h:tf;
Y(1)=y0;
for k=1:N
    Y(k+1) = Y(k) + h*feval(f,T(k),Y(k));
end
E = [T' Y'];

% FONCTION fex1

function f=fex1(t,y)
f=(t-y)/2;
```

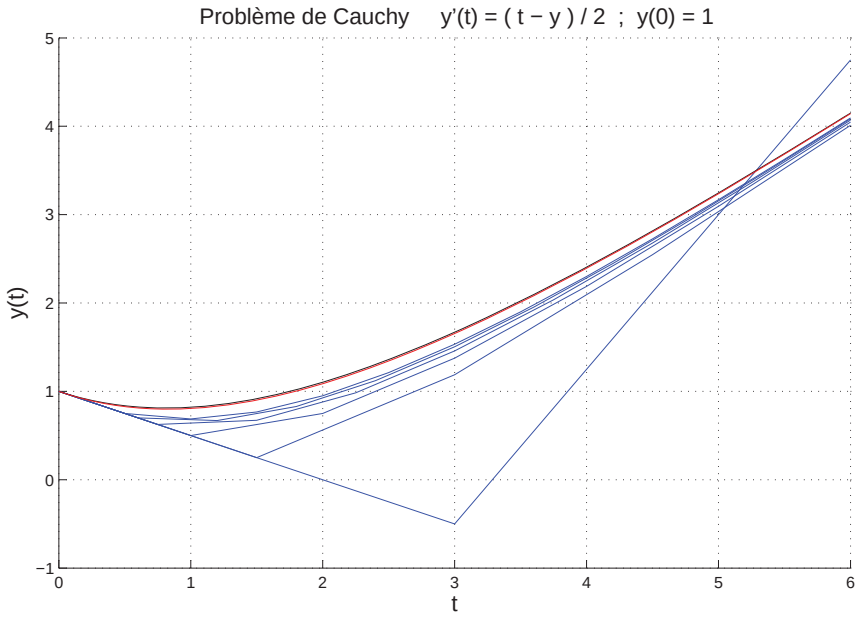


FIGURE 10.1 – Résolution approchée du problème de Cauchy pour divers pas d'intégration. La solution analytique $y(t) = 3e^{-t/2} - 2 + t$ est représentée en noir. La solution approchée obtenue avec N=100 est en rouge.

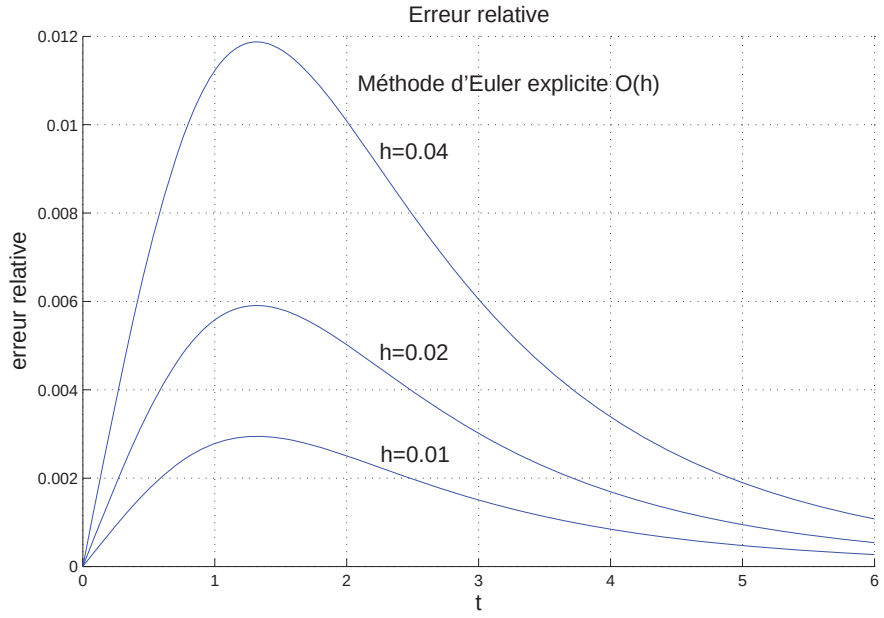


FIGURE 10.2 – Résolution approchée du problème de Cauchy, erreur relative pour trois pas d'intégration différents. On constate que l'erreur maximum décroît linéairement avec le pas d'intégration.

Au lieu d'exprimer $y(t_1)$ à partir de $y(t_0)$ grâce au théorème de Taylor, on peut procéder autrement et exprimer $y(t_0)$ à partir de $y(t_1)$ selon la formule :

$y(t_0) = y(t_1 - h) = y(t_1) - h.y'(t_1) + \frac{h^2}{2}y''(\xi) \quad \text{où} \quad \xi \in [a, b].$

En négligeant, comme précédemment, le terme en h^2 on obtient la formule d'Euler régressive, ou formule d'Euler implicite :

$y_1 = y_0 + h.f(t_1, y_1).$

On dit que ce schéma est implicite dans la mesure où la quantité que l'on désire calculer, y_1, n'est pas connue explicitement.

Dans ce cas, le calcul de y_1 nécessitera, en général [1], le recours à une méthode de résolution d'une équation à une inconnue.

Ainsi, si nous définissons la fonction $g(x) = x - h.f(t_1, x) - y_0$ nous pouvons obtenir y_1 au moyen de l'algorithme de Newton :

$$x_0 = y_0, \quad x_{m+1} = x_m - \frac{g(x_m)}{g'(x_m)}, \quad m = 0, 1, 2, \ldots$$

La plupart du temps on pourra se contenter d'effectuer un très petit nombre d'itérations de la méthode de Newton dans la mesure où, pour h petit, y_1 est proche de y_0.

Remarque

Comme la formule explicite, la formule d'Euler implicite peut aussi être obtenue par la méthode d'intégration du rectangle [2].

Supposons h suffisamment petit pour que l'on puisse approcher $f(t, y(t))$ sur $[t_0, t_1]$ par $f(t_1, y_1)$, on obtient alors :

$\int_{t_0}^{t_1} y'(t)dt = y(t_1) - y(t_0) = \int_{t_0}^{t_1} f(t, y(t))dt \simeq \int_{t_0}^{t_1} f(t_1, y_1)dt = h.f(t_1, y_1)$

et l'on retrouve bien la formule d'Euler implicite : $y_1 = y_0 + h.f(t_1, y_1)$.

Les figures 10.3 et 10.4 montrent les comportements des méthodes d'Euler explicite et implicite dans le cas du problème de Cauchy de l'exemple précédent.

1. Il est parfois possible d'expliciter y_1 à partir de la formule d'Euler implicite.
2. La hauteur du rectangle vaudra cette fois $f(t_1, y_1)$, a priori inconnue, puisqu'elle dépend de y_1 !

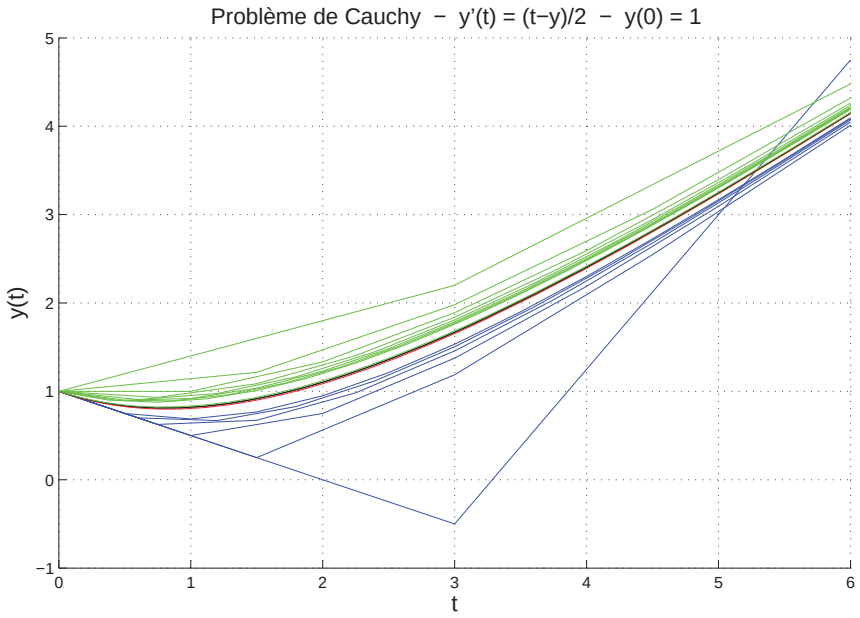


FIGURE 10.3 – Les solutions approchées par la méthode explicite sont en bleu et celles obtenues par la méthode implicite sont en vert. On remarque que l'une des méthodes approche la solution par le bas et l'autre par le haut.

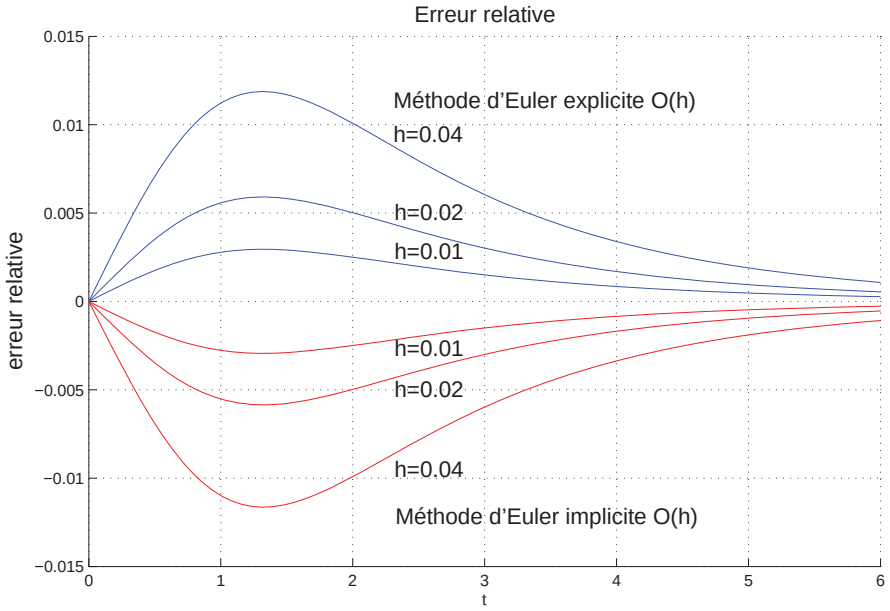


FIGURE 10.4 – Résolution approchée du problème de Cauchy, erreur relative pour trois pas d'intégration différents. Les courbes bleues correspondent à la méthode explicite et les rouges à la méthode implicite. On constate la quasi symétrie des courbes d'erreur et que le maximum de l'erreur décroît linéairement avec le pas d'intégration h.

10.3 Stabilité

A première vue, la formule d'Euler explicite semble préférable au schéma implicite. Ce n'est cependant généralement pas le cas dans la mesure où elle présente des problèmes de stabilité que le schéma implicite n'occasionne pas.

Considérons le problème de Cauchy particulier suivant :

$y'(t) = -\beta y(t) \quad t > 0$

avec comme condition initiale : $\quad y(0) = y_0$.

Si β est positif, ce que nous supposerons, la solution de ce problème est : $y(t) = y_0 e^{-\beta t}$ et $y(t)$ décroît exponentiellement quand t vers l'infini.

Soit un nombre réel petit $h > 0$.

Posons $t_n = n \times h$ avec $n = 0, 1, 2, \ldots$

le schéma d'Euler explicite nous donne comme valeurs approchées de la solution du problème :

$y_{n+1} = (1 - \beta \times h) \times y_n$ ou encore : $y_n = (1 - \beta \times h)^n \times y_0$ pour $n = 0, 1, 2, \ldots$

Cette dernière expression nous montre donc que si $y_0 \neq 0$ et $1 - \beta \times h < -1$, la suite y_n, qui est une solution approchée du problème, tend vers l'infini en alternant les signes, si n tend vers l'infini, au lieu de tendre vers zéro comme le fait la solution exacte !

Pour éviter ce phénomène, il faut choisir h de manière à avoir : $-1 \leq 1 - \beta \times h$ c'est-à-dire imposer $h \leq \frac{2}{\beta}$.

Cette condition qui restreint le pas h lorsqu'on utilise le schéma d'Euler explicite est appelée condition de stabilité.

Le schéma d'Euler implicite ne présente pas cet inconvénient.

En effet, dans le cas du problème type étudié ici, le schéma implicite fournit :

$y_n = (1 + \beta \times h) \times y_{n+1}$, ou encore : $y_n = (\frac{1}{1+\beta \times h})^n \times y_0$ pour $n = 0, 1, 2, \ldots$

Nous voyons donc que $\forall h \lim_{n \to \infty} y_n = 0$,

et le schéma d'Euler implicite est donc stable quel que soit h, qui ne doit pas être limité a priori.

NB : on peut montrer que les schémas d'Euler explicite et implicite sont tous deux d'ordre 1 en h. Ceci signifie que lorsqu'on divise le pas d'intégration par deux l'erreur obtenue pour une valeur de t fixée est divisée par deux également [3].

Plus précisément, pour $h = \frac{t}{N}$ on a : $|y(t) - y_N| \leq \frac{C}{N}$ où C est une constante qui ne dépend pas de N mais qui peut dépendre de la valeur de t. Lorsque N tend vers l'infini, c'est-à-dire lorsque h tend vers zéro (pour t fixé) on a :

$\lim_{h \to 0} |y(t) - y_N| = 0$ et $|y(t) - y_N| = O(h)$.

Pour plus de détails, voir par exemple : J.Rappaz et M.Picasso, Introduction à l'analyse numérique, Presses polytechniques et universitaires romandes 1998, p.144.

10.3.1 Exemple 2

On peut facilement illuster le phénomène d'instabilité (voir figures 10.5 et 10.6). Il suffit par exemple de chercher des approximations du problème de Cauchy

$\frac{dy}{dt}(t) = y'(t) = -5.y(t) \quad \text{avec} \quad y(0) = 2 \quad \text{pour} \quad 0 < t < 3$

```
function testeuler_2
% Instabilité de la méthode d'Euler explicite
% Résolution du problème de Cauchy%
% y'(t) = -5y + t ;  y(0) = y0
%
H=[0.45 0.3 0.15 0.075 0.0015]; t0=0; tf=3; y0=2;
N=round((tf-t0)./H)+1
figure(2); format short; hold on; grid on; xlabel('t'); ylabel('y(t)')
titre=strcat('Problème de Cauchy  -  y''(t) = -5y + t;  y(0) = ',num2str(y0));
title(titre'FontSize',14); col=['k' 'r' 'b' 'm' 'g'];
for i=1:length(H)
E=euler(@fex2,t0,tf,y0,N(i))
plot(E(:,1),E(:,2),col(i),'LineWidth',2)
pause
end
% Solution analytique
u=0:0.001:3; v=(51*exp(-5*u)+u*5-1)/25;
plot(u,v,'c','LineWidth',2)
gtext('Méthode d''Euler explicite','FontSize',14);
%
function f=fex2(t,y)
f=-5*y+t;
```

3. Comme sur les figures 10.2 et 10.4

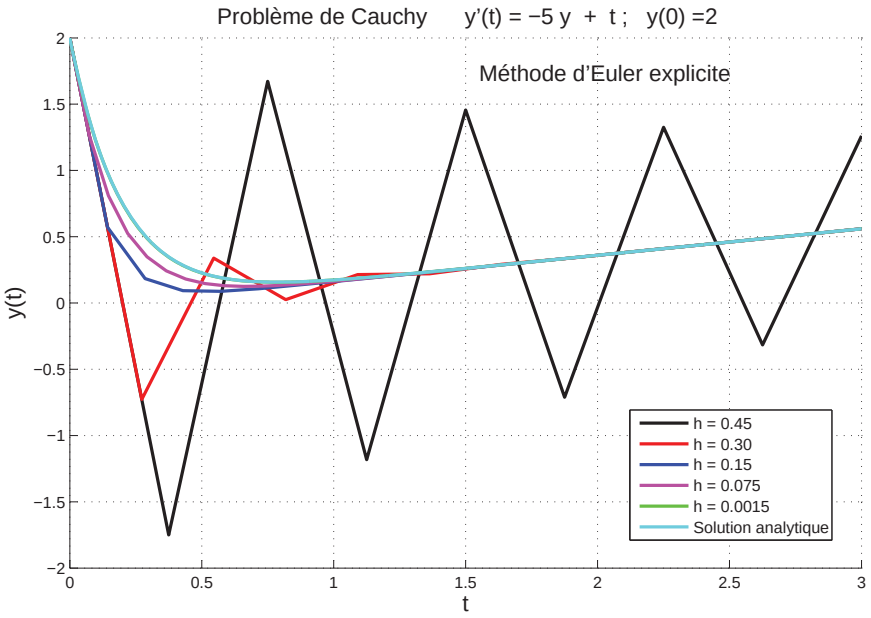


FIGURE 10.5 – Instabilité de la méthode d'Euler explicite. Lorsque le pas h est suffisamment petit, on ne distingue pas le graphe de la solution approchée (en vert) de celui de la solution analytique (en cyan).

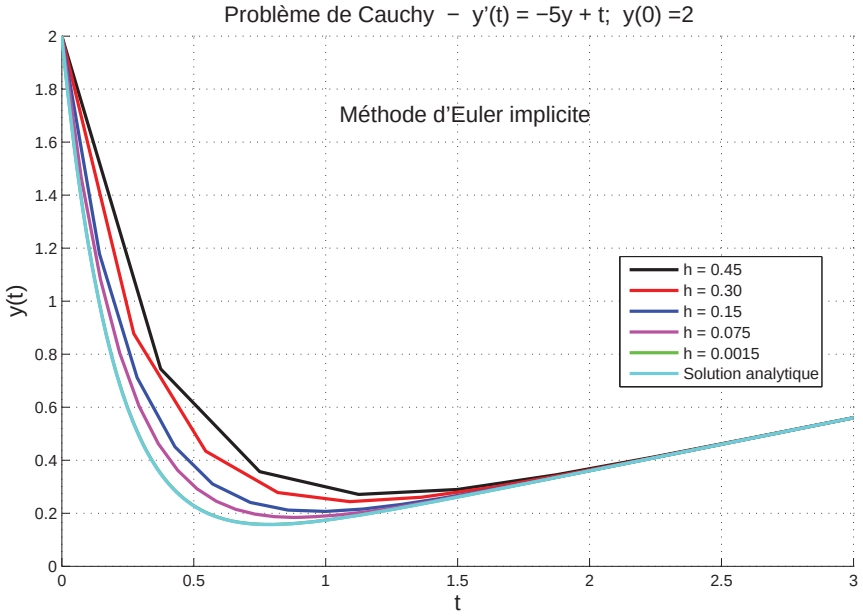


FIGURE 10.6 – Stabilité de la méthode d'Euler implicite. Les différents pas utilisés sont les mêmes que sur la figure ptécédente.

10.4 Schémas d'intégration à un pas

Revenons au problème de Cauchy du point 10.1.

$\frac{dy}{dt}(t) = y'(t) = f(t, y(t)) \quad t \in [a, b]$

avec comme condition initiale : $\quad y(a) = y_0$.

Par intégration entre t_n et t_{n+1} nous obtenons :

$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt$

Comme la fonction $y(.)$ est inconnue, nous ne pouvons calculer l'intégrale ci-dessus que de manière approchée. Selon la façon d'approcher cette intégrale, nous obtenons différents schémas de résolution du problème de Cauchy.

Supposons que $h = t_{n+1} - t_n$ est suffisamment petit pour pouvoir considérer que l'intégrand est constant sur l'intervalle d'intégration. Selon le choix de cette constante on obtient :

Schéma d'Euler explicite (Règle du rectangle)

On remplace l'intégrand par $f(t_n, y_n)$.

Schéma d'Euler implicite (Règle du rectangle)

On remplace l'intégrand par $f(t_{n+1}, y_{n+1}) = f(t_n + h, y_{n+1})$

Schéma de Crank-Nicholson (Règle du trapèze)

On remplace l'intégrand par $\frac{1}{2}[f(t_n, y_n) + f(t_{n+1}, y_{n+1})] = \frac{1}{2}[f(t_n, y_n) + f(t_n + h, y_{n+1})]$

Ce schéma est évidemment implicite et on peut montrer qu'il est d'ordre 2 en h. Il correspond également à la moyenne obtenue à partir des schémas d'Euler explicite et implicite. Il s'écrit :

$y_{n+1} = y_n + \frac{h}{2}[f(t_n, y_n) + f(t_{n+1}, y_{n+1})]$ pour $n = 0, 1, 2, \ldots$

Schéma de Heun ou de Runge-Kutta à deux paramètres

Pour éviter le calcul implicite de y_{n+1} dans le schéma de Crank-Nicholson, on peut remplacer, dans $f(t_{n+1}, y_{n+1})$, y_{n+1} par une valeur approchée obtenue par une prédiction $\widetilde{y_{n+1}}$ fournie par le schéma d'Euler explicite :

$y_{n+1} = y_n + \frac{h}{2}[f(t_n, y_n) + f(t_{n+1}, y_n + h.f(t_n, y_n))] = y_n + \frac{h}{2}(k_1 + k_2)$

où $k_1 = f(t_n, y_n)$, $\widetilde{y_{n+1}} = y_n + h.f(t_n, y_n)) = y_n + h.k_1$ et $k_2 = f(t_{n+1}, \widetilde{y_{n+1}})$

On obtient ainsi la méthode de Heun qui fait partie des méthodes de Runge-Kutta explicites d'ordre deux, à deux paramètres, que l'on nomme souvent RK2. Les méthodes présentées dans ce paragraphe sont dites à un pas car le calcul de y_{n+1} ne nécessite que la connaissance de l'approximation au pas précédent : y_n.

10.4.1 Retour à l'exemple 1

Reprenons le problème de Cauchy de l'exemple 1 et cherchons-en des solutions approchées par les méthodes de Crank-Nicholson et de Heun de manière à comparer les erreurs commises avec celles produites par les méthodes d'Euler.

Les figures 10.7 et 10.9 présentent les solutions approchées obtenues en partitionnant l'intervalle d'intégration $[0, 6]$ en 2,4,8,10 et 12 sous-intervalles égaux. Elles sont à rapprocher des figures 10.1 et 10.3. On voit clairement que la convergence est plus rapide pour les méthodes $O(h^2)$. Les figures 10.8 et 10.10 présentent les erreurs relatives et sont à rapprocher des figures 10.2 et 10.4.

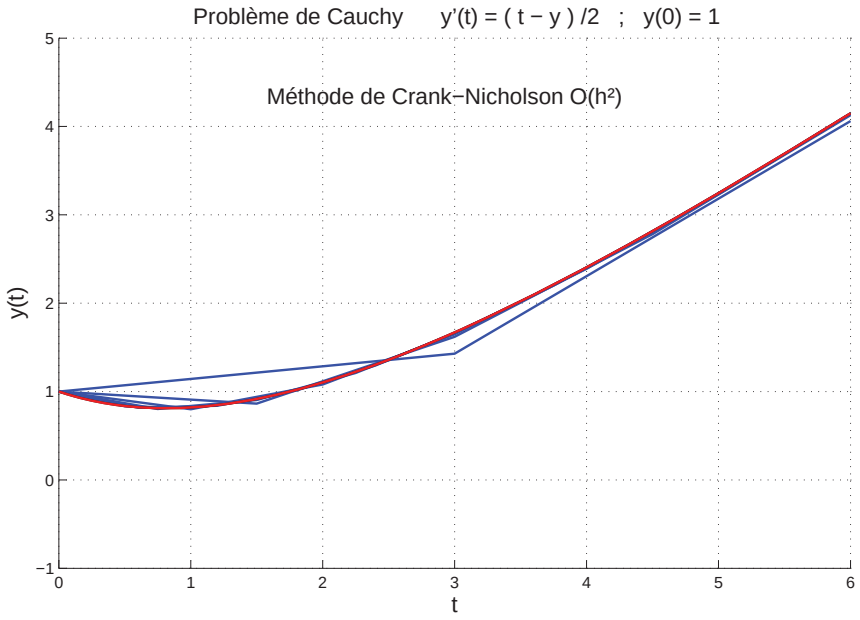


FIGURE 10.7 – Solutions approchées par la méthode de Crank-Nicholson, à comparer avec les figures 10.1, 10.3 et 10.9.

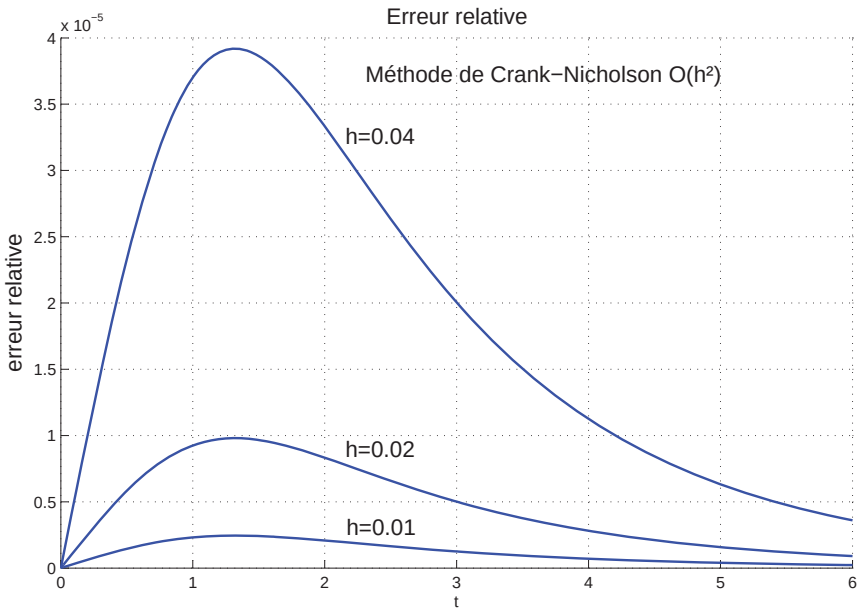


FIGURE 10.8 – Résolution approchée du problème de Cauchy par la méthode de Crank-Nicholson. Le maximum de l'erreur relative décroît quadratiquement avec le pas d'intégration h. A comparer avec les figures 10.2, 10.4 et 10.10.

Le tableau ci-dessous résume les observations que l'on peut faire à partir de ces figures.

Méthode	h	Erreur relative maximale en valeur absolue	Ordre
Euler explicite	0.04	1.187510^{-2}	h
Euler explicite	0.02	5.907310^{-3}	h
Euler explicite	0.01	2.946210^{-3}	h
Euler implicite	0.04	1.163910^{-2}	h
Euler implicite	0.02	5.848410^{-3}	h
Euler implicite	0.01	2.931510^{-3}	h
Crank-Nicholson	0.04	3.918610^{-5}	h^2
Crank-Nicholson	0.02	9.796210^{-6}	h^2
Crank-Nicholson	0.01	2.449010^{-6}	h^2
Heun ou RK2	0.04	7.955610^{-5}	h^2
Heun ou RK2	0.02	1.974010^{-5}	h^2
Heun ou RK2	0.01	4.916510^{-6}	h^2

On constate que les erreurs maximales sont du même ordre de grandeur pour les méthodes d'Euler explicite et implicite[4] et qu'il en va de même en ce qui concerne les méthodes de Crank-Nicholson et de Heun. Toutefois, pour ces deux méthodes qui sont $O(h^2)$ les erreurs sont plus petites de plusieurs ordres de grandeurs. On remarque aussi que la méthode de Heun est moins précise que celle de Crank-Nicholson. Cela provient du fait que la méthode de Heun utilise une valeur de démarrage fournie par la méthode d'Euler explicite.

4. Leurs signes sont toutefois opposés.

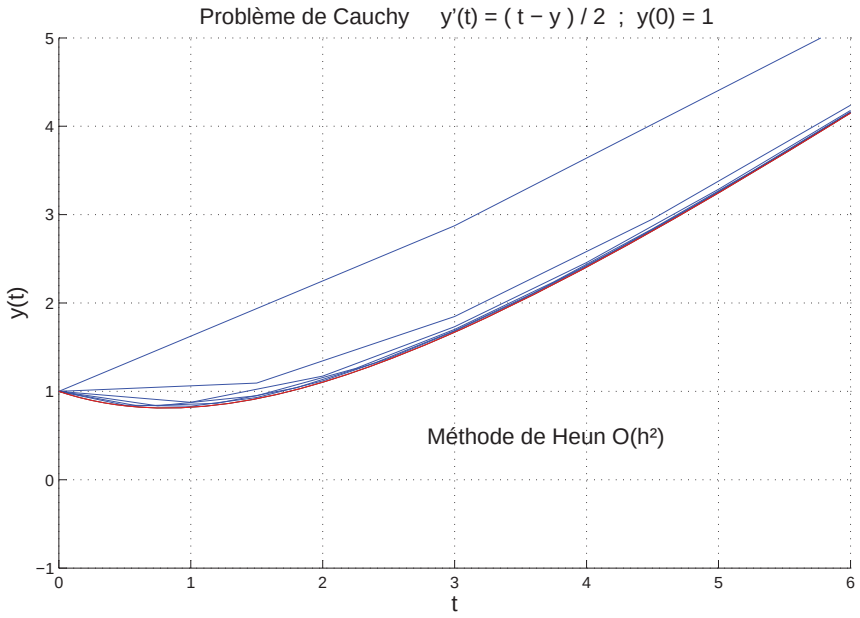


FIGURE 10.9 – Solutions approchées par la méthode de Heun ou RK2, à comparer avec les figures 10.1, 10.3 et 10.7.

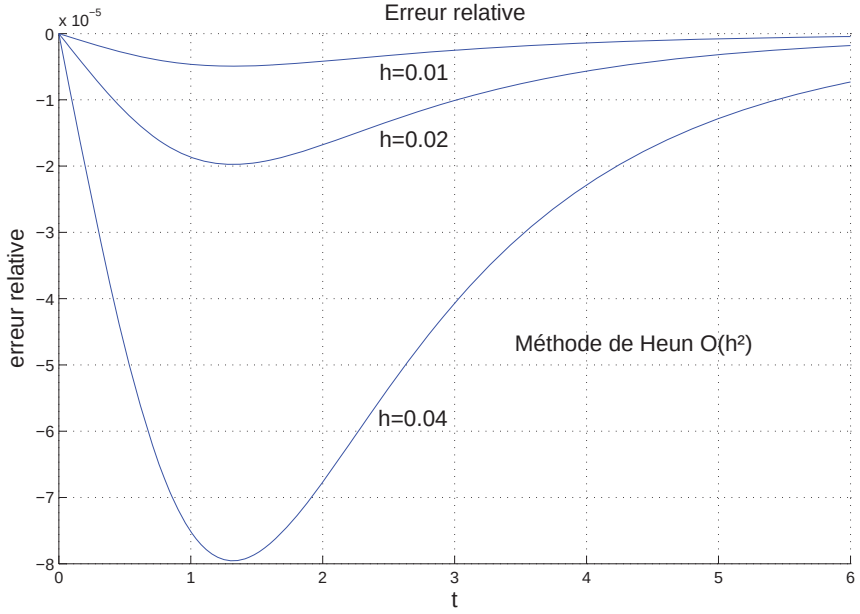


FIGURE 10.10 – Résolution approchée du problème de Cauchy par la méthode de Heun ou RK2, erreur relative pour trois pas d'intégration différents. Le maximum de l'erreur décroît quadratiquement avec le pas d'intégration h. A comparer avec les figures 10.2, 10.4 et 10.8.

10.5 Schémas d'intégration à pas multiples

Supposons que pour approcher l'intégrale $\int_{t_n}^{t_{n+2}} f(t, y(t))dt$ on utilise la formule de Simpson :

$\int_{t_n}^{t_{n+2}} f(t, y(t))dt \simeq \frac{h}{3}[f_n + 4f_{n+1} + f_{n+2}]$

on obtient alors, en posant $f_n = f(t_n, y_n)$, l'algorithme suivant :

$y_0 = y(t_0); \quad y_{n+1} = y_{n-1} + \frac{h}{3}[f_{n-1} + 4f_n + f_{n+1}]; \quad n = 1, 2, \ldots \quad \text{avec} \quad h = t_{n+1} - t_n$

où le pas d'intégration h est supposé constant.

Une telle méthode est dite à deux pas car le calcul de y_{n+1} utilise non seulement y_n mais aussi y_{n-1}. On constate que pour que le processus fonctionne, il faut disposer d'une approximation y_1 de $y(t_1)$. Cette approximation est obtenue à l'aide d'une formule que l'on appelle formule de démarrage qui peut, par exemple, être la formule de Heun. Enfin, la méthode est implicite dans la mesure où le calcul de f_{n+1} nécessite la connaissance de y_{n+1} c'est-à-dire de la quantité que l'on veut calculer !

On généralise facilement aux méthodes à plus de deux pas. Elles découlent de la manière dont on approche l'intégrale $\int_{t_n}^{t_{n+p}} f(t, y(t))dt$ où p est le nombre de pas de la méthode [5].

10.6 Systèmes d'équations différentielles ordinaires du premier ordre

Problème de Cauchy

Le problème de Cauchy s'écrit comme dans le cas d'une seule équation (cas scalaire) :

$\frac{dy}{dt}(t) = y'(t) = f(t, y(t)) \quad t \in [a, b]$

avec la condition initiale : $\quad y(a) = y_0$.

Toutefois, la signification des différents termes n'est plus la même :

$y(.)$ et $f(.,.)$ sont à présent des fonctions vectorielles et y_0 est le vecteur des conditions initiales.

On peut appliquer les différentes démarches développées ci-dessus au problème de Cauchy vectoriel.

5. Rappel : le pas $h = t_{k+1} - t_k$ est constant.

Par exemple, le schéma d'Euler explicite s'écrit, comme dans le cas scalaire :

Choisir N

$h = \frac{b-a}{N}$

$t_0 = a$

pour $\quad k = 0, 1, 2, \ldots, N-1$

faire $\quad y_{k+1} = y_k + h.f(t_k, y_k)$

$t_{k+1} = t_k + h$

10.6.1 Exemple 3 - Simulation d'une épidémie de grippe.

Appliquons la méthode d'Euler explicite pour résoudre un problème qui simule l'évolution d'une épidémie de grippe. L'épidémie est modélisée par un système de trois équations différentielles ordinaires non-linéaires. Ce modèle dû à Kermack et McKendrick, qui date de 1927, est connu sous le nom de modèle *SIR*.

Ses équations s'écrivent

$\frac{dS}{dt} = S' = -\beta \times S.I \quad$ (Nombre d'individus par jour)

$\frac{dI}{dt} = I' = \beta \times S.I - \gamma \times I \quad$ (Nombre d'individus par jour)

$\frac{dR}{dt} = R' = \gamma \times I \quad$ (Nombre d'individus par jour)

avec comme conditions initiales

$S(0) = S_0 \quad I(0) = I_0 \quad$ et $\quad R(0) = R_0.$

Dans ces équations, la population concernée par l'épidémie est divisée en trois classes, les individus sains ou susceptibles de contracter la maladie (S), les individus malades ou infectés qui sont contagieux (I) et les individus qui ont recouvré la santé (R). [6]

On suppose que l'infection se propage [7] par la rencontre entre susceptibles et malades (terme $\beta \times S.I$) et que la guérison se fait au bout de $1/\gamma$ jours (terme $\gamma \times I$). Enfin, on fait l'hypothèse simplificatrice que pendant la durée de l'épidémie la population totale $S + I + R$ reste constante.

6. On suppose que la maladie est bénigne et que personne n'en meurt.
7. La durée d'incubation est supposée négligeable.

Le programme ci-dessous permet de simuler la propagation d'une épidmie de grippe dans une population de 10.000 personnes dont une seule est infectée à l'instant initial. Par modification des paramètres, d'autres types d'épidémies peuvent également être simulées. Les résultats de la simulation sont présentés à la figure 10.11.

```
% Simulation du modèle SIR de Kermack et McKendrick (1927).
% cf.  Mathematical biology - Ed. Springer Verlag (1990) par J.D.Murray (page 613).
%
% Intégration par la méthode d'Euler explicite.
%
% Le taux de propagation de l'épidémie est appelé beta.
% Le taux de guérison est noté gamma et 1/gamma est la durée de l'infection.
%
% On peut discuter l'influence de beta et gamma.
%
% S : nombre d'individus sains (susceptibles);
% I : nombre de malades (infectives);
% R : nombre de guéris ou décédés (recovered).
%
% NB : gamma peut aussi inclure le taux de décès, il s'agit en fait du taux
% de passage de la catégorie I à la catégorie R (mort et/ou guéri)
%
clf;
figure(1)
hold on
axis([0 100 0 10000]); title('Modèle de Kermack et McKendrick')
xlabel('Temps (jours)'); ylabel('S I R  (nombre d''individus)')
dt = 0.05;
Tmax=100;
beta=1/20000;
gamma=1/3;
I=1;                    % Un seul malade est introduit dans la population
R=0;                    % Au départ, personne n'est immunisé
S=10000-I;              % La population totale initiale est de 10000 individus
for t=0:dt:Tmax
% On définit cinq variables auxiliaires
T1 = beta*S*I;
T2 = gamma*I;
DS = -T1;
DI = T1 - T2;
DR = T2;
% Euler explicite
S = S + dt*DS;
I = I + dt*DI;
R = R + dt*DR;
```

```
%
% Petite astuce, on utilise des complexes pour représenter les résultats
%
pointS = t + i*S;
pointR = t + i*R;
pointI = t + i*I;
plot(pointS,'g','LineWidth',1.5);  % Individus sains courbe verte
plot(pointI,'r','LineWidth',1.5);  % Individus malades (infectives) courbe rouge
plot(pointR,'b','LineWidth',1.5);  % Individus guéris (Removed) courbe bleue
end
grid on
gtext('Sains','FontSize',14)
gtext('Infectés (malades contagieux)','FontSize',14)
gtext('Guéris','FontSize',14)
```

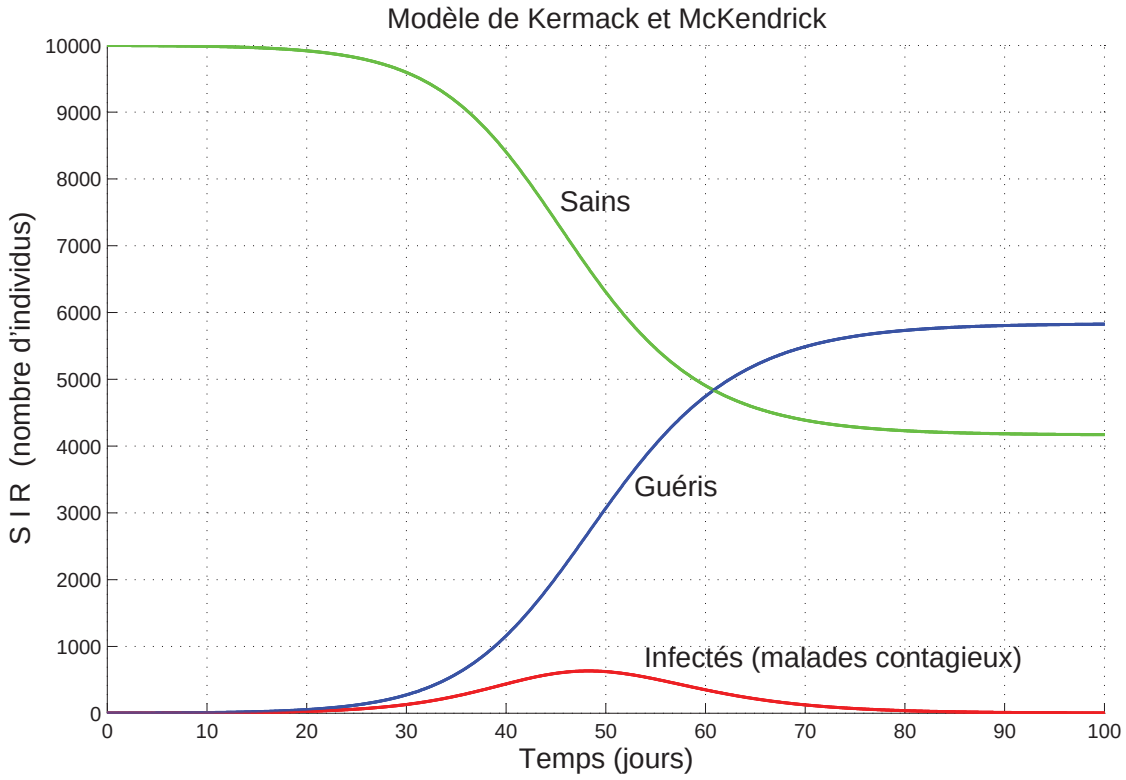


FIGURE 10.11 – Modèle de Kermack et McKendrick pour simuler l'évolution d'une épidémie de grippe.

Remarque

Une équation différentielle d'ordre n peut être remplacée par un systèmes de n équations du premier ordre.

10.6.2 Exemple 4

Soit l'équation du 3ème ordre

$y'''(x) = \mu[y''(x) + (1 - y^2(x))].y'(x) + \sin x, \quad$ où μ est une constante

et les conditions initiales associées

$y(0) = y_0, \quad y'(0) = y'_0 \quad$ et $\quad y''(0) = y''_0$

Posons $\quad y(x) = u_1(x), \quad y'(x) = u_2(x) \quad$ et $\quad y''(x) = u_3(x)$

Il vient

$u'_1(x) = u_2(x)$
$u'_2(x) = u_3(x)$
$u'_3(x) = \mu[u_3(x) + (1 - u_1^2(x))].u_2(x) + \sin x$

et

$u_1(0) = y_0$
$u_2(0) = y'_0$
$u_3(0) = y''_0$

La résolution numérique de ce problème de Cauchy peut-être obtenue aisément à l'aide des procédures d'intégration de Matlab. Voici un exemple de programme qui utilise la procédure ode45 qui fait appel aux méthodes de Runge-Kutta explicites à 4 et 5 paramètres à pas adaptatif (dans les zones où la fonction inconnue ou une de ses dérivées varie fortement, le pas d'intégration est automatiquement réduit) :

```
function ode3
global MU
MU= 0.001;  xspan = [0; 100];  u0 = [1; 5; 0];  ode45(@uprime,xspan,u0);
title(['Résolution d''un système de trois équations, \mu = ' num2str(MU)]);
xlabel('Position x');  ylabel('solution u(x)');

% --------------------------------------------------------------------
```

```
function dydx = uprime(x,u)
global MU
dydx = [            u(2)
                    u(3)
          MU*(u(3)+(1-u(1)^2))*u(2) + sin(x) ];
```

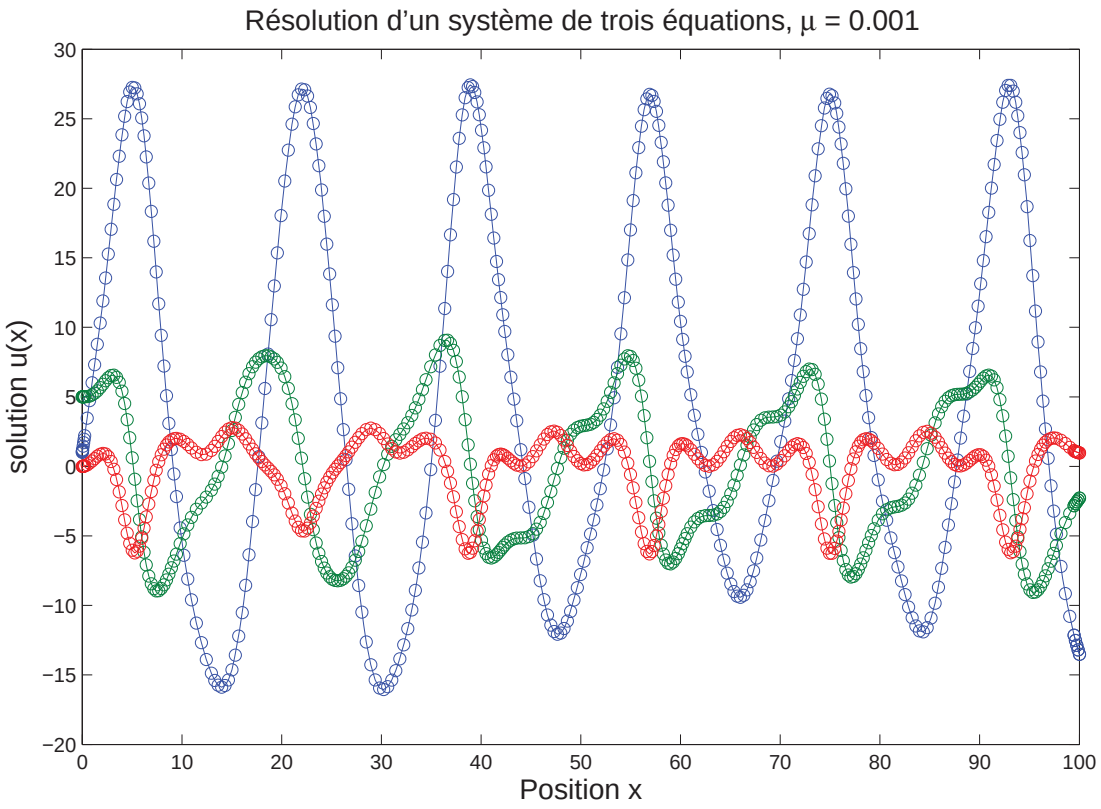


FIGURE 10.12 – Graphe fourni par ode3.

On trouve plusieurs exemples de systèmes d'équations différentielles ordinaires ainsi que de systèmes différentiels-algébriques dans les "demos" de Matlab. (Cf. par exemple vdpode dans Matlab701\toolbox\matlab\demos ainsi que odedemo et les divers programmes dont le nom se termine par ode).

10.7 Prudence et petits pas

La résolution de problèmes numériques à l'aide de procédures "toutes faites" que l'on nomme souvent "boîtes noires" ne peut se faire sans précautions.

10.7.1 Exemple 5

L'équation différentielle linéaire du second ordre $y''(x) + 2y'(x) + y(x) = 0$, associée aux conditions initiales $y(0) = 0, y'(0) = 1$ peut être remplacée par le problème de Cauchy suivant (on pose $y(x) = u_1(x)$ et $y'(x) = u_2(x)$)

$u_1'(x) = u_2(x)$
$u_2'(x) = -u_1(x) - 2u_2(x)$

$u_1(0) = 0$
$u_2(0) = 1$

elle admet comme solution $y(x) = u_1(x) = xe^{-x}$.

La résolution du problème de Cauchy à l'aide des procédures standard de Matlab ode15s, ode23s et ode45 permet d'élaborer les figures 10.13 et 10.14. La figure 10.13 montre l'écart qui il y a entre les solutions approchées et la solution analytique. A première vue, les erreurs diminuent sensiblement avec x. Toutefois, la figure 10.14 nous apprend qu'en réalité les erreurs augmentent très fort pour $x \in [5, 15]$.

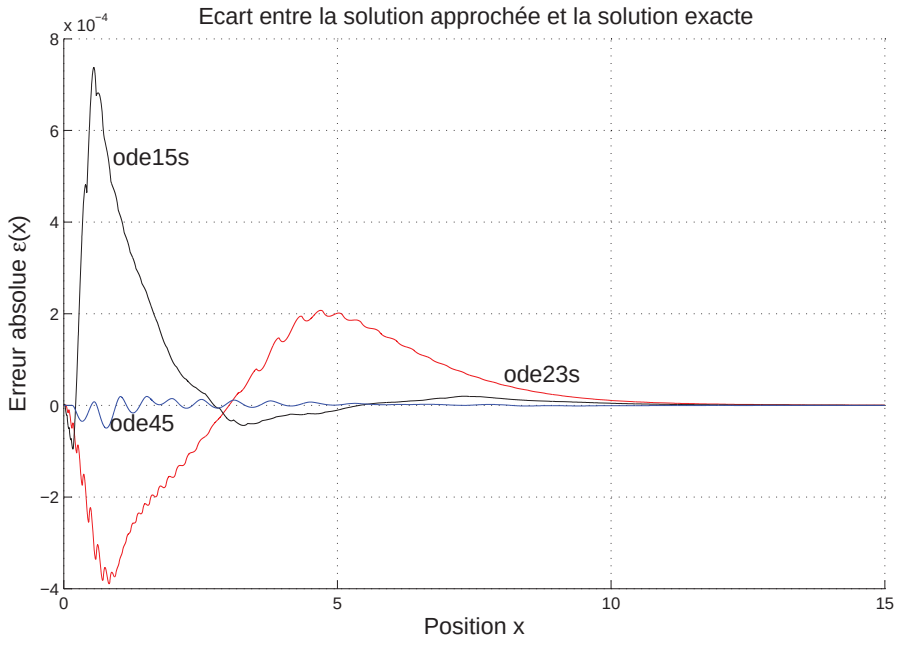


FIGURE 10.13 – Erreurs absolues commises lors de la résolution du problème de l'exemple 5 par les procédures standard ode15s, ode23s et ode45 de Matlab.

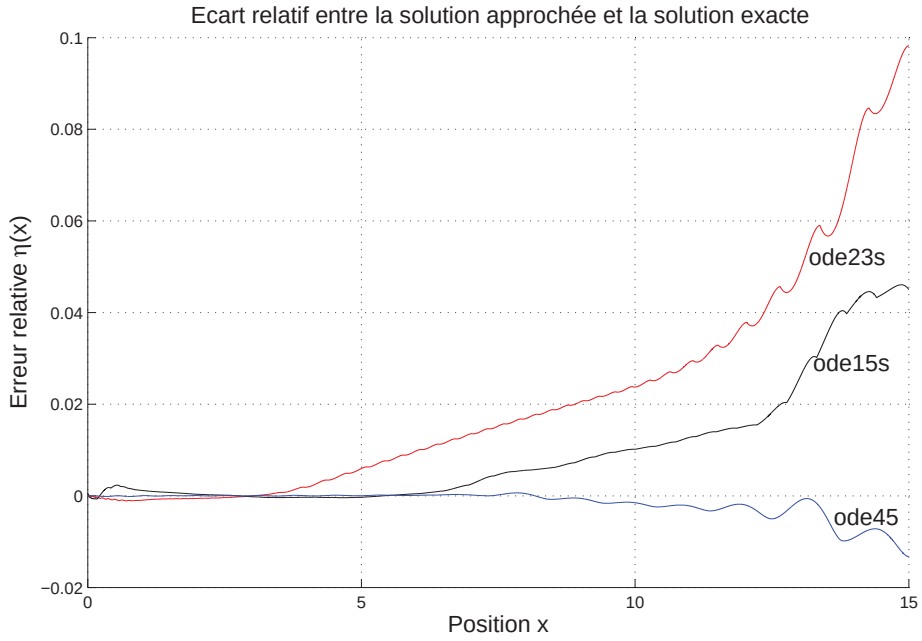


FIGURE 10.14 – Erreurs relatives commises lors de la résolution du problème de l'exemple 5 par les procédures standard ode15s, ode23s et ode45 de Matlab.

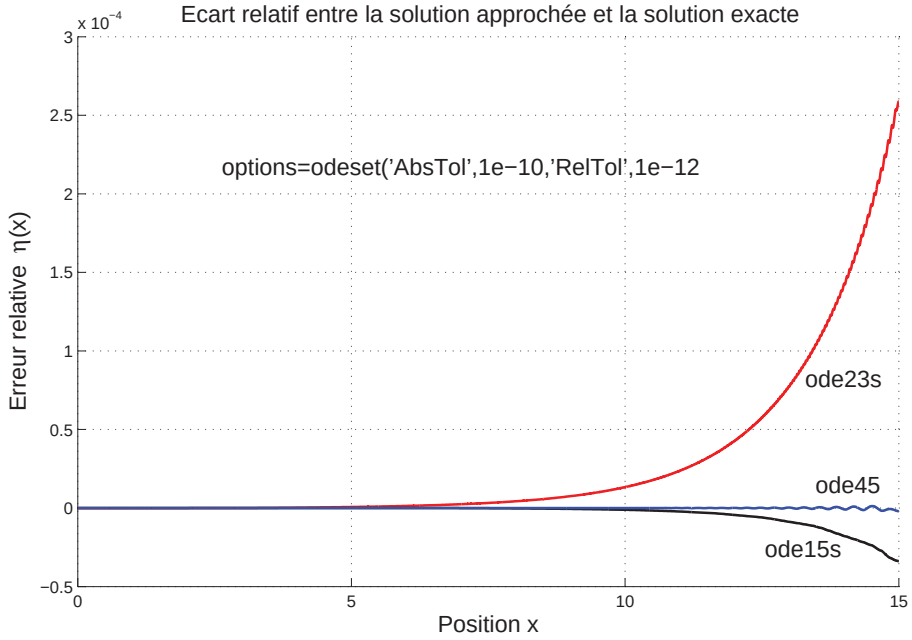


FIGURE 10.15 – Erreurs relatives commises lors de la résolution du problème de l'exemple 5 par les procédures ode15s, ode23s et ode45 de Matlab en modifiant les "tolérances".

10.8 Problèmes aux conditions aux limites

10.8.1 Introduction

Soient $c(x)$ et $f(x)$ deux fonctions continues sur l'intervalle $[0,1]$. Déterminer la fonction $u(x)$ deux fois continûment dérivable sur $[0,1]$ qui satisfait aux conditions suivantes

$$-u''(x) + c(x).u(x) = f(x) \quad \text{pour} \quad x \in [0,1] \quad \text{avec} \quad u(0) = u_0; \quad u(1) = u_f$$

consiste à résoudre un problème aux conditions aux limites du second ordre.

Ce problème est linéaire, non homogène, aux conditions aux limites non-homogènes.

Pour ce qui concerne l'existence et l'unicité de la solution de ce problème, nous renvoyons au cours d'analyse mathématique.

Remarques

1) Lorsque f est identiquement nulle sur $[0,1]$ on dit que le problème est homogène. Si $u_0 = u_f = 0$ on dit que les conditions aux limites sont homogènes.

2) Un problème homogène aux conditions aux limites homogènes admet toujours la solution triviale et, éventuellement une infinité de solutions appelées fonctions propres du problème.

3) Il existe évidemment des problèmes aux conditions aux limites d'ordre supérieur à 2. Ainsi, les solutions du système suivant correspondent aux modes propres de vibration d'une poutre encastrée à ses deux extrémités. L est la longueur de la poutre, $z(x)$ sa déflexion au point d'abscisse x et λ une constante liée à la fréquence de vibration de la poutre et à ses caractéristiques physiques

$$z''''(x) + \lambda^4 z(x) = 0$$
$$z(0) = z(L) = 0$$
$$z'(0) = z'(L) = 0$$

La résolution analytique de ce type de problèmes est souvent impossible, même dans le cas du problème du deuxième ordre présenté à la page précédente.

On doit, très généralement, se contenter de solutions numériques approchées. Nous nous limiterons ici à l'exposé très bref de trois méthodes de résolution :

1) La méthode des différences finies.
2) La méthode de tir au but.
3) La méthode de collocation.

10.8.2 Méthode des différences finies

Dérivée numérique d'une fonction en un point

La définition de la dérivée d'une fonction en un point

$$f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h} = \lim_{h\to 0} \frac{f(x-h)-f(x)}{h} = \lim_{h\to 0} \frac{f(x+h/2)-f(x-h/2)}{h}$$

nous permet de remplacer $u'(x_n)$ par une valeur approchée $O(h)$:

$$u'(x_n) \simeq \frac{u(x_n+h)-u(x_n)}{h} \simeq \frac{u(x_n)-u(x_n-h)}{h}$$

ou, mieux, $O(h^2)$:

$$u'(x_n) \simeq \frac{u(x_n+h)-u(x_n-h)}{2h} \simeq \frac{u(x_n+h/2)-u(x_n-h/2)}{h}$$

pour autant que h soit suffisamment petit.

Le calcul des dérivées d'ordre supérieur s'obtient aisément. Par exemple, pour la dérivée seconde, on obtient, par application de la dernière approximation de la dérivée première :

$$u''(x_n) \simeq \frac{u'(x_n+h/2)-u'(x_n-h/2)}{h} \simeq \frac{u(x_n+h)-2u(x_n)+u(x_n-h)}{h^2} + O(h^2).$$

Bien entendu, les valeurs $u(x_n)$ sont inconnues et en pratique, on les remplacera par des approximations notées : u_n , telles que $u_n \simeq u(x_n)$.

Equations aux différences finies

Nous supposons que l'intervalle sur lequel nous recherchons la fonction $u(x)$ est partitionné en $N+1$ sous-intervalles égaux, de longueur $h = 1/(N+1)$, au moyen d'abscisses de discrétisation $x_0 = 0, x_1 = h, \ldots, x_j = j.h, \ldots, x_{N+1} = 1$.

Le problème continu de départ est remplacé par le problème discret approché qui consiste à calculer les inconnues u_j solutions du système d'équations aux différences suivant :

$$\begin{aligned}
&\frac{-u_{j-1}+2u_j-u_{j+1}}{h^2} + c(x_j)u_j = f(x_j) \quad 1 \le j \le N,\\
&u(x_0) = u_0\\
&u(x_{N+1}) = u_f
\end{aligned}$$

Il s'agit d'un système tridiagonal, qui dans le cas particulier étudié ici, est linéaire.

Remarque

Les problèmes d'ordre 2 et 4 présentés ci-dessus ne contiennent que la fonction et sa dérivée d'ordre maximum (2 ou 4). L'équation du second ordre pourrait évidemment contenir un terme en $y'(x)$ tout comme l'équation d'ordre 4 pourrait présenter des termes en $y'(x), y''(x)$ et $y'''(x)$.

10.8.3 Exemple 6

Calculons une solution approchée du problème aux conditions aux limites :

$y''(x) + \sin(x).y(x) = cos(x) \quad 0 \leq x \leq \pi.$
$y(0) = y(\pi) = 1.$

A cette fin, pour $N = 4, 8, 16, 32, 64$:

a) on pose $x_n = n.h_N$, avec $h_N = \dfrac{\pi}{N+1}$ et $n = 0, 1, 2, \ldots, N+1$;
b) on discrétise l'équation selon $y''_n + \sin(x_n).y_n = \cos(x_n)$ avec $y(x_n) \simeq y_n$
et $y''_n \simeq \dfrac{y_{n-1} - 2y_n + y_{n+1}}{h_N^2}$;
c) on résout le système tridiagonal $N \times N$ obtenu.

Voici le programme qui permet de résoudre ce problème et la figure qu'il produit.

```
clc; clear; figure(1)
hold on; grid on;
col=['g' 'b' 'm' 'k' 'c'];
xlabel('x','FontSize',14);
ylabel('y','FontSize',14);
title('Problème aux conditions aux limites','FontSize',14);
for i=2:6
    N=2^i; h=pi/(N+1);
    for n=1:N
        x(n)=h*n;
    end
    d=h*h*sin(x)-2;
    Mat=diag(d) + diag(ones(N-1,1),1) + diag(ones(N-1,1),-1);
    Sec=h*h*cos(x);
    Sec(1)=Sec(1)-1;
    Sec(N)=Sec(N)-1;
    sol=Mat\Sec'; % "Division matricielle de Matlab"
    t=[0 x pi]; y=[1 sol' 1];
    plot(t,y,col(i-1),'LineWidth',1.5)
end
```

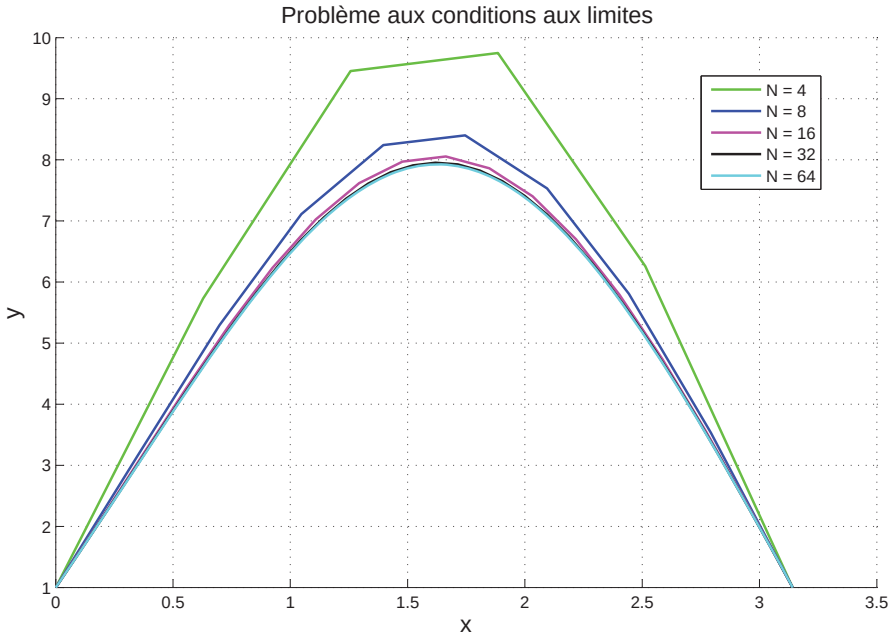


FIGURE 10.16 – Approximations de la solution du problème de l'exemple 6. Les courbes correspondant à $N = 32$ et $N = 64$ sont quasi confondues.

10.8.4 Méthode de tir au but

Soit le problème aux conditions aux limites du second ordre

$$y''(x) + a(x, y(x), y'(x)).y'(x) + b(x, y(x), y'(x)).y(x) + c(x, y(x), y'(x)) = 0 \qquad x \in [a, b]$$
$$y(a) = y_0$$
$$y(b) = y_f$$

dont nous supposons qu'il admet une solution unique.

Il s'agit d'un problème très général qui est linéaire si on remplace les fonction a, b et c par des fonctions qui ne dépendent que de x. Nous pourrions chercher une solution approchée de ce problème par discrétisation de $y''(x)$ et $y'(x)$ comme nous l'avons fait au paragraphe précédent, ce qui nous conduirait à la résolution d'un système algébrique non linéaire en les inconnues y_i. Une autre possibilité consiste à résoudre une suite de problèmes de Cauchy.

Remplaçons, d'une part, l'équation du second ordre par un système de deux équations du premier ordre en posant $y(x) = u_1(x)$ et $y'(x) = u_2(x)$; et, d'autre part, la deuxième condition aux limites par une condition initiale $y'(a) = C$, où C est une constante à déterminer de façon à avoir $y(b) = y_f$.

Pour calculer C, nous allons procéder par approximations successives, ce qui nous amènera résoudre une suite ($k = 1, 2, \dots$) de problèmes de Cauchy du type

$u_1'(x) = u_2(x)$
$u_2'(x) = -a(x, u_1(x), u_2(x)).u_2(x) - b(x, u_1(x), u_2(x)).u_1(x) - c(x, u_1(x), u_2(x))$
$u_1(a) = y_0$
$u_2(a) = C_k$

Exemple 7 - Problème linéaire

Soit le probème aux conditions aux limites

$y''(x) + y'(x) + \sin x.y(x) - \cos x = 0 \quad x \in [0, \pi]$
$y(0) = 1$
$y(\pi) = -1$

Ce problème est linéaire et nous le remplaçons par les problèmes de Cauchy

$u_1'(x) = u_2(x)$
$u_2'(x) = -u_2(x) - \sin x.u_1(x) + \cos(x)$
$u_1(0) = 1$
$u_2(0) = C_k \quad k = 1, 2, \ldots$

La stratégie que nous utilisons pour déterminer une suite de C_k convergeant vers la valeur permettant de réaliser la condition $u_1(\pi) = y(\pi) = -1$ est fort simple, nous procédons par interpolation :

1) on choisit une valeur C_1 et on calcule $u_{1,1}(\pi)$
2) si $u_{1,1}(\pi) = -1$ le problème est résolu
3) si $u_{1,1}(\pi) \neq -1$ on se donne une valeur C_2 et on calcule $u_{1,2}(\pi)$
4) si $u_{1,2}(\pi) = -1$ le problème est résolu
5) si $u_{1,2}(\pi) \neq -1$ on choisit $C_3 = C_1 + (u_1(\pi) - u_{1,1})\dfrac{C_2 - C_1}{u_{1,2} - u_{1,1}}$ et on calcule $u_{1,3}(\pi)$

Sauf cas pathologique, l'approximation obtenue avec C_3 fournit la solution "exacte" du problème, c'est à dire des valeurs y_n quasiment égales à $y(x_n)$ avec $x \in [0, pi]$, $y(0) = 1$ et $y(\pi) = -1$.

En effet, comme le problème est linéaire, la deuxième condition aux limites dépend linéairement du choix de C et par interpolation linéaire à partir des couples de valeurs $(C_1, u_{1,1})$ et $(C_2, u_{1,2})$ on obtient la valeur C_3 de C qui permet de satisfaire à la deuxième condition aux limites.

Dans le cas d'un problème non linéaire la procédure d'interpolation devra être répétée jusqu'à ce que, pour une certaine valeur de C_k, la deuxième condition aux limites soit vérifiée dans des limites acceptables, si possible...

Le programme ci-dessous et les fonctions `odebut` et `uprime` qui l'accompagnent permettent de résoudre le problème linéaire de l'exemple 7. Les solutions approchées sont représentées à la figure 10.17.

```
% PROGRAMME
C1=-12;  Y1=odebut(C1,'C_1');
C2= 8;   Y2=odebut(C2,'C_2');
YFINAL=-1; C3=C1+(YFINAL-Y1)*(C2-C1)/(Y2-Y1);
odebut(C3,'C_3'); grid on
%-----------------------------------------------------
% FONCTIONS
%
function Y_final_appx=odebut(C,texte)
% Résolution d'un problème aux limites par tir au but
figure(1); hold on; grid on; axis([0 pi -10 10])
x0=0; xfin=pi; y0=1; yfin=-1; uprimedebut=C;
xspan = [x0 xfin];  u0 = [y0; uprimedebut];
[t,y] = ode45(@uprime,xspan,u0);
plot(t,y(:,1),'LineWidth',2)
% Après chaque tir on fournit la valeur d'arrivée en x=pi
% Le but est d'atteindre la bonne cible.
% Ici c'est le point xfin=pi, yfin=-1 (point rouge)
gtext(num2str(y(end,1)),'FontSize',14)
gtext(texte,'FontSize',14)
titre='Résolution d''un problème aux limites par tir au but';
title(titre,'FontSize',14);
xlabel('Position x','FontSize',14);
ylabel('Approximations successives de la solution y(x)','FontSize',14);
plot(xfin,yfin,'or','LineWidth',3)
Y_final_appx=y(end,1);
% ---------------------------------------------------------------------
function dydx = uprime(x,u)
a=inline('1'); b=inline('sin(x)'); c=inline('cos(x)');
dydx = [ u(2)
         -a(x)*u(2)-b(x)*u(1)+c(x) ];
```

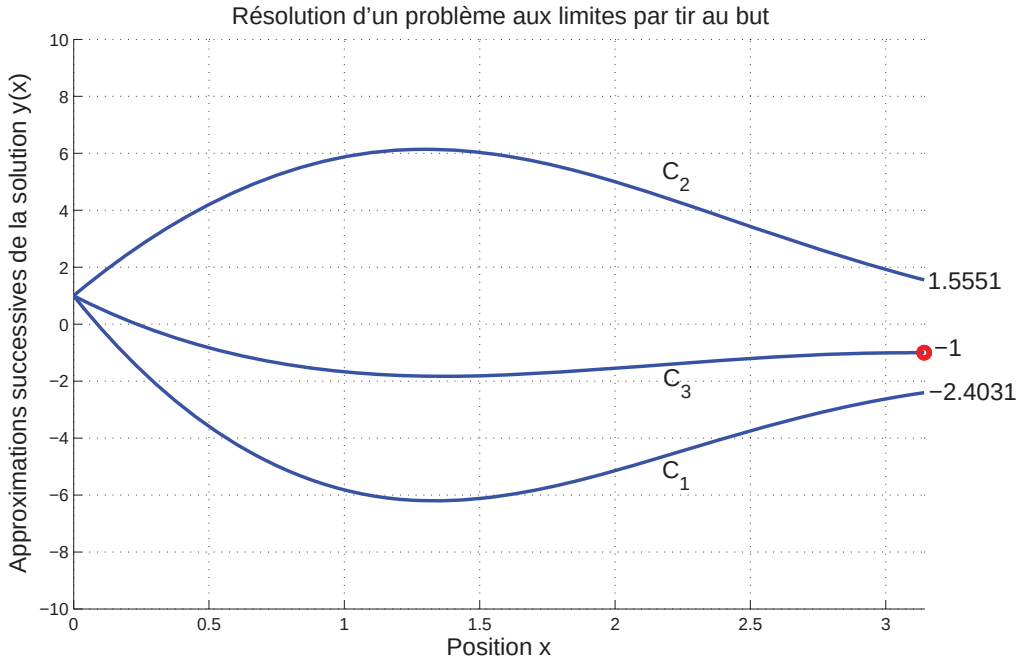


FIGURE 10.17 – Approximations de la solution du problème aux limites linéaire de l'exemple 7. Une seule interpolation calculée à partir de deux valeurs de C suffit pour obtenir la solution "exacte".

Exemple 8 - Problème non linéaire

La procédure d'interpolation utilisée pour résoudre le problème linaire de l'exemple 7 peut également être appliquée dans le cas de problèmes non linéaires. Nous l'illustrons ici pour le problème suivant

$$y''(x) - \frac{y'(x)}{1+y^2(x)} + \cos x = 0 \text{ avec } x \in [0, 3\pi]$$
$$y(0) = 1 \text{ et } y(3\pi) = 2$$

Les $k = 1, 2, \ldots$ problèmes de Cauchy associés s'écrivent

$$u_1'(x) = u_2(x)$$
$$u_2'(x) = \frac{u_2(x)}{1+u_1^2(x)} - \cos x$$
$$u_1(0) = y(0) = 1$$
$$u_2(0) = C_k$$

Le programme `interpolodebut` permet de résoudre ce problème et on constate qu'il faut faire une quinzaine d'itérations pour atteindre une précision de l'ordre 10^{-14}. La figure 10.18 a été élaborée avec ce programme et montre les 18 approximations de la solution qu'il calcule.

```
function interpolodebut
% Résolution d'un problème aux limites par tir et interpolation linéaire
% YFINAL est la condition à l'extrémité droite du domaine soit en x=pi.
clc; clear; format long
xspan = [0 3*pi]; YFINAL=2;
C1=0.25; u0 = [1; C1]; [t,y] = ode45(@uprime,xspan,u0);
Y1=y(end,1); [C1 Y1]
plot(t,y(:,1),'LineWidth',1.5); hold on
gtext('C_1','FontSize',14);
pause
C2=-0.25; u0 = [1; C2]; [t,y] = ode45(@uprime,xspan,u0);
Y2=y(end,1); [C2 Y2]
plot(t,y(:,1),'LineWidth',1.5)
gtext('C_2','FontSize',14);
pause
col='b'; LW=1.5;
Nmax=18;
for k=1:Nmax
    if k==Nmax
        col='r'; LW=2;
    end
C3=C1-(Y1-YFINAL)*(C1-C2)/(Y1-Y2); [t,y] = ode45(@uprime,xspan,[1;C3]);
Y1=Y2; Y2=y(end,1); C1=C2; C2=C3;
[C2 Y2]
plot(t,y(:,1),col,'LineWidth',LW)
texte=['k=' num2str(k)];
if k<=6 | k==8 | k==18
gtext(texte,'FontSize',14)
end
pause
end
titre='Problème aux limites non linéaire - Méthode de tir au but';
title(titre,'FontSize',14)
xlabel('x','FontSize',14)
ylabel('Approximations successives de y(x) x  \in [0,3\pi]','FontSize',13)
grid on

function dydx = uprime(x,u)
dydx = [ u(2)
         u(2)./(1+u(1).^2)-cos(x) ];
```

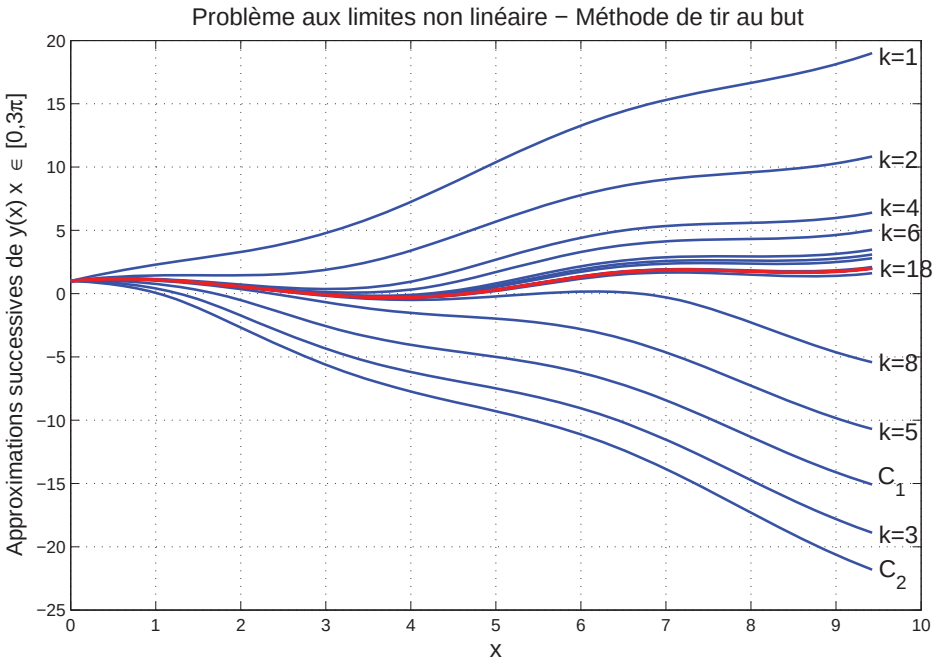


FIGURE 10.18 – Approximations de la solution du problème non linéaire de l'exemple 7. Il faut faire 18 interpolations pour obtenir $y(3\pi) = 2.00000000000002$. A partir de la quinzième itération les courbes sont quasi confondues.

10.8.5 Méthode de collocation

Le principe de la méthode de collocation a été exposé dans les chapitres 6 et 7 et à plusieurs reprises au cours oral. Il a été exploité dans la note sur les cordes et poutres et dans le cadre de la douzième séance d'exercices. Tous les problèmes traités étaient linéaires. La méthode reste valable dans le cas des problèmes non linéaires mais elle conduit à la résolution de systèmes algébriques non linéaires, sujet que nous n'avons abordé que de manière très superficielle (cf. séance d'exercices portant sur la méthode de Newton-Raphson).

Table des matières

Table des figures

Bibliographie

[1] Jean-Guy Dion and Rolland Gaudet. *Méthodes d'Analyse Numérique : de la théorie à l'application*. MODULO, 1996.

[2] John H. Mathews and Kurtis.D. Fink. *Numerical Methods Using Matlab*. PRENTICE HALL, third edition, 1999.

[3] J. Rappaz and M Picasso. *Introduction à l'analyse numérique*. Presses Polytechniques Romandes, 1998.