Notions d'analyse numérique
Licence de Physique - Paris 7

Jacques Le Bourlot

Avril 2006

Contents

1 Introduction

Ces notes ne constituent pas un cours. Elles sont extraites d'un cours d'analyse numérique du Master Pro "Outils et systèmes de l'Astrophysique et de l'Espace". De nombreux ouvrages sur l'analyse numérique existent. Un choix restreint est proposé en fin de texte.

Comme souvent, la pratique est essentielle, et il est indispensable de mettre en oeuvre soi même les méthodes présentées. Les premiers chapitres présentent des méthodes de base, suffisamment simples pour être programmées en quelques (dizaine de) minutes. Les derniers exigent le recours à des bibliothèques. numériques Le but est d'aider l'utilisateur à savoir ouvrir une boite noire en cas de besoin, mais à ne pas réinventer la roue quand c'est inutile. Savoir évaluer dans quel cas on se trouve relève de la pratique seule. L'analyse numérique reste un art, et non une science.

Trois aspects, dont chacun est suffisant, caractérisent le fait que ces notes ne sont pas un cours:

1. Des méthodes et techniques essentielles ne sont même pas citées.

2. Aucun résultat n'est démontré au sens mathématique du terme (les conditions d'application n'étant souvent même pas citées).

3. Aucune étude sérieuse de convergence n'est faite.

Il est donc indispensable d'utiliser en parallèle un véritable manuel d'analyse numérique. La bibliographie présente une courte sélection d'ouvrages:

Acton (1990) est à lire absolument pour comprendre vraiment ce que l'on fait (et pas seulement appliquer des recettes).

Kernighan et al. (1999) indispensable avant d'entreprendre un projet un peu ambitieux, et le reste du temps aussi. Pas un manuel de programmation, mais de réflexion sur les "bonnes" pratiques.

Mathews (1992) est un bon ouvrage d'introduction. Simple pour le débutant, et en même temps rigoureux. Il ne faut pas utiliser les éditions récentes (basées sur Matlab), mais préférer la deuxième.

Press et al. (1992) est très (trop?) connu. C'est un très bon ouvrage pour aborder un problème quand on part de rien. On s'aperçoit à l'usage qu'il est vite limité lorsque l'on veut vraiment approfondir, ou que le problème est délicat.

2 Résolution de systèmes linéaires

2.1 Présentation

Soit à résoudre un système de n équations linéaires écrit sous la forme:

$$A\,X = B$$

Avec:

$$A = (a_{ij})\,, \qquad X = (x_j)\,, \qquad B = (b_i)$$

Dès que n dépasse quelques unités, un tel système se résout numériquement. Pour choisir l'algorithme le plus adapté, il convient d'examiner trois aspects du problème:

1. Doit-on résoudre le système pour un seul vecteur B, ou pour un grand nombre de B avec la même matrice A?
2. Quelle est la structure de A?
3. Quelle est la taille de A?

La dernière question a l'air triviale. Elle ne l'est pas car la réponse "intéressante" (c'est à dire "taille raisonnable" ou "grande taille") dépend de la machine utilisée, et de la structure de A.

En fonction des réponses à ces questions, on sera amené à choisir parmi un grand nombre de méthodes disponibles, en général en utilisant une bibliothèque de fonctions[1]. Ces méthodes se divisent en deux grandes catégories:

- Les méthodes itératives.
- Les méthodes finies.

La première catégorie est plutôt utilisée pour les matrices "grandes", et dont la diagonale est dominante. L'archétype est la méthode de Jacobi, ou sa variante de Gauss-Seidel ((Mathews, 1992, Chap 3.7)). Ces méthodes convergent (quand

[1] La plus utile, est la bibliothèque LAPACK, disponible sur le web, et généralement fournie sous forme optimisée par la plupart des compilateurs. Sur les système LINUX, le mode d'emploi de chaque routine existe sous la forme de page du manuel (commande `man`), ce qui rend l'utilisation très simple.

tout va bien) vers la solution recherchée en un nombre infini d'itérations. Elles doivent donc être accompagnées d'un critère de "qualité" permettant de décider (automatiquement) quand la précision atteinte est "suffisante". Nous n'en parlerons pas plus ici.

La deuxième catégorie contient les méthodes qui donnent un résultat définitif en un nombre fini d'étapes. Dès que la matrice A possède des particularités (matrice bande par exemple), celles-ci **doivent** être utilisées, car elles permettent en général un gain de temps important. La méthode de base de cette catégorie est la méthode d'élimination de Gauss, qui ne fait que reproduire automatiquement la procédure manuelle consistant à utiliser une équation pour déterminer une première variable, la reporter dans les autres équations, et recommencer sur le système de taille $n-1$.

Cette procédure est inefficace. Toujours dès que n est "assez grand", mais aussi très souvent quand la matrice A est "mal conditionnée". En pratique, de nombreux développements successifs ont permis d'élaborer des méthodes plus robustes, plus stables et plus rapides partant de cette idée de base. Nous en étudierons une seule, qui est la méthode de choix en l'absence de raison particulière de ne pas l'utiliser (!), la méthode **LU**.

2.2 Technique

2.2.1 Principe

Cette méthode reste proche de la méthode d'élimination traditionnelle. Elle est basée sur l'idée de décomposer la matrice A sous la forme d'un produit de deux matrices triangulaires:

$$A = LU$$

où L est une matrice triangulaire inférieure ("lower"), et U une matrice triangulaire supérieure ("upper"). Une fois cette décomposition accomplie, le système linéaire de départ $AX = B$ s'écrit:

$$LU\,X = B$$

On pose alors $Y = UX$, et on résout le système auxiliaire $LY = B$. Ceci est trivial, grâce à la structure triangulaire de L! En effet, la première ligne se réduit

à:

$$l_{11}.y_1 = b_1$$

d'où l'on tire $y_1 = \frac{b_1}{l_{11}}$. On peut alors reporter cette solution dans la deuxième ligne, qui est:

$$l_{21}.y_1 + l_{22}.y_2 = b_2$$

d'où l'on tire:

$$y_2 = \frac{1}{l_{22}} \ (b_2 - l_{21}.y_1)$$

De proche en proche, on peut ainsi calculer les y_i successifs, jusqu'à y_n. On reprend alors le système $UX = Y$, où maintenant Y est **connu**! On part ici de la dernière ligne qui est:

$$u_{nn}.x_n = y_n$$

d'où l'on tire $x_n = \frac{y_n}{u_{nn}}$. On peut alors reporter cette valeur dans la ligne $n-1$ et en tirer x_{n-1}. De proche en proche, on remonte ainsi jusqu'à x_1, et le système est résolu.

Cette méthode est particulièrement élégante lorsque l'on a plusieurs jeux de seconds membres B pour une même matrice A. En effet, on peut montrer que le temps de calcul nécessaire pour décomposer A varie comme n^3, alors que les substitutions successives ont un coût variant en n^2. Or la **même** décomposition est utilisée pour tous les seconds membres, ce qui est particulièrement économique. La résolution se déroule donc en deux grandes étapes:

1. Décomposition de A en un produit LU.

2. Substitutions successives pour tous les seconds membres B.

La principale faiblesse de la méthode (pour l'instant), se lit sur les équations précédentes: à chaque étape de substitution, il faut diviser par un élément diagonal de L ou de U. Donc, si l'un d'entre eux est proche de 0 des difficultés sont à craindre.

La solution (partielle) vient de la liberté laissée à l'utilisateur de choisir l'ordre des équations. En permutant deux d'entre elles, on ne change pas le résultat final. L'étape de décomposition se fait donc en examinant à chaque pas s'il est utile d'inverser deux équations pour maximiser les éléments diagonaux, ...et en

gardant en mémoire ces inversions pour "démélanger" le résultat final (il faudra aussi permuter les b_i en calculant la solution).

2.2.2 Décomposition

On va d'abord supposer qu'aucune permutation n'est nécessaire. Les "pivots" seront pris en compte dans un deuxième temps. Soit $A = (a_{ij})$ une matrice carré d'ordre n. On suppose qu'elle n'est pas singulière (donc $\det(A) \neq 0$). On cherche deux matrices L et U telles que $l_{ij} = 0$ si $j > i$, $u_{ij} = 0$ si $j < i$, et $A = LU$. On peut donc écrire:

$$a_{ij} = \sum_{k=1}^{n} l_{ik}.u_{kj} \tag{1}$$

En pratique, cette équation ne détermine pas complètement L et U. En effet, on peut toujours les remplacer par $\frac{1}{\alpha}L$ et αU, si $\alpha \neq 0$. Ici, on imposera $u_{ii} = 1$.

La décomposition se fait en exploitant les 0 de L et U et l'ordre dans lequel les équations (1) sont écrites. On commence par la première colonne de A:

$$a_{i1} = \sum_{k=1}^{n} l_{ik}.u_{k1} = l_{i1}$$

parce que $u_{k1} = 0$ pour tout $k > 1$ et $u_{11} = 1$ par convention. On en déduit $l_{i1} = a_{i1}$ pour $i = 1$ à n.

Cela nous donne la première colonne de L. on écrit maintenant l'expression des coefficients de la première ligne:

$$a_{1j} = \sum_{k=1}^{n} l_{1k}.u_{kj} = l_{11}.u_{1j}$$

parce que $l_{1k} = 0$ pour tout k >1. On en tire $u_{1j} = a_{1j}/l_{11}$ pour $j = 2$ à n, ce qui fournit la première ligne de U.

On peut maintenant passer à la deuxième colonne de A, en commençant par la ligne 2 (la ligne 1 est terminée). On obtient:

$$a_{i2} = l_{i1}\, u_{12} + l_{i2}$$

D'où:

$$l_{i2} = a_{i2} - l_{i1}\, u_{12} \tag{2}$$

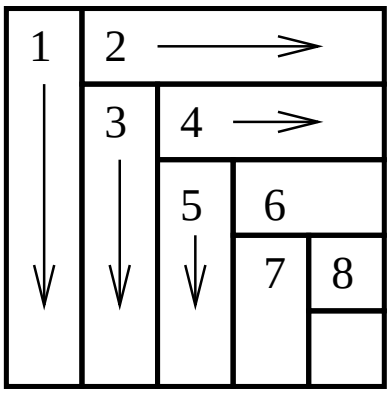
La deuxième ligne nous donne elle:

$$a_{2j} = l_{21}.u_{1j} + l_{22}.u_{2j}$$

D'où:

$$u_{2j} = (a_{2j} - l_{21}.u_{1j})/l_{22}$$

Le calcul se poursuit ainsi en alternant colonnes et lignes. A chaque j, les lignes 1 à $j-1$ permettent de calculer u_{ij}, et ceux-ci sont ensuite utilisés pour calculer l_{ij} pour i allant de j à n.



Pour une colonne quelconque j, on a pour $i \geq j$:

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}.u_{kj}$$

puis pour la ligne correspondante pour tout $j > i$:

$$u_{ij} = \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik}.u_{kj} \right) / l_{ii}$$

Une particularité particulièrement intéressante est que les a_{ij} ne sont utilisés qu'une seule fois, au moment de calculer le l_{ij} ou le u_{ij} correspondant. On peut donc effectuer la décomposition "sur place", en remplaçant au fur et à mesure les coefficients de A par ceux de L et de U de mêmes indices. Le choix de $u_{ii} = 1$ fait qu'il n'y a pas non plus d'ambiguïté pour la diagonale.

2.2.3 Pivot

Reste à s'assurer que l'on ne créera pas de division par 0. On peut pour cela utiliser la généralisation de l'équation 2 (j est fixé):

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}.u_{kj} \tag{3}$$

Tous les l_{ij} peuvent être calculés pour i allant de j à n. Il suffit donc de déterminer l'indice i_0 tel que $|l_{ij}|$ soit maximum, et d'intervertir les équations j et i_0, c'est à dire les lignes j et i_0 de A (y compris les coefficients de L déjà calculés!). Cela ne modifie pas les coefficients de U déjà calculés qui restent en place, ni ceux de L dont seul l'indice change.

Pour pouvoir par la suite associer le "bon" second membre à la "bonne" équation, il suffit de créer avant le début de l'algorithme un vecteur P tel que $p_i = i$, et de lui appliquer les mêmes permutations. Le vecteur final contiendra l'ordre d'utilisation des valeurs de B.

Même si l'équation (3) donne une valeur de l_{ii} différente de 0, on a intérêt à pivoter systématiquement pour minimiser les erreurs d'arrondi.

3 Résolution de systèmes non-linéaires

3.1 Présentation

Soit à résoudre un système de n équations non linéaires écrites sous la forme:

$$F(X) = 0$$

où X est un vecteur de n inconnues x_i, et F un vecteur de n fonctions $f_i(X)$. Dès que n est différent de 1 ce problème est intrinsèquement difficile. En effet, chaque équation individuelle $f_i(X) = 0$ défini dans l'espace de dimension n un sous-espace (une variété) de dimension $n-1$. La solution recherchée est à l'intersection (si elle existe) de ces différents sous-espace. or:

- Il peut ne pas y avoir de solution
- Il peut y en avoir plusieurs...
- Même si une équation particulière a un sens précis, il n'y a aucune raison pour l'intersection de deux variétés se fasse en des points particulièrement remarquables.
- On ne dispose pas de la possibilité "d'encadrer" la solution comme à une dimension.

Il est donc indispensable de bien analyser d'abord le problème physique associé et de ne pas chercher "au hasard". Si possible, il faut partir d'une solution approchée (voire même un ordre de grandeur raisonnable).

Toutes les méthodes utilisées en pratique sont itératives, et nécessitent donc de définir un critère de convergence permettant d'arrêter le calcul lorsque le résultat est "satisfaisant". Deux quantités sont souvent utilisées pour cela:

$$\epsilon_x = \sum_i \left| \frac{\overline{x_i} - x_i}{x_i} \right|$$

$$\epsilon_f = \sum_i \left| f_i\left(\overline{X}\right) \right|$$

où $\overline{X}$ est la solution approchée. ϵ_x est une mesure de l'écart relatif entre la solution recherchée et la solution approchée. En pratique, comme on ne connaît

pas X, on utilise souvent:

$$\epsilon_x = \sum_i \left| \frac{\delta_i}{x_i} \right|$$

où δ_i est la variation de la variable x_i lors de la dernière itération. Deux conditions sont nécessaires pour que cette définition soit utilisable:

- Tous les x_i doivent être du même ordre de grandeur. Le problème doit donc être dédimensionné.
- Aucun x_i ne doit être nul.

ϵ_f est une mesure de la qualité de la solution. On utilise une erreur absolue car, par construction, $f_i = 0$ pour tout i pour la "vrai" solution.

Il est théoriquement nécessaire de rendre à la fois ϵ_x **et** ϵ_f plus petits qu'un seuil fixé pour se garder des cas "pathologiques" ou le gradient de l'un des f_i est proche de 0 ou de ∞ au point solution.

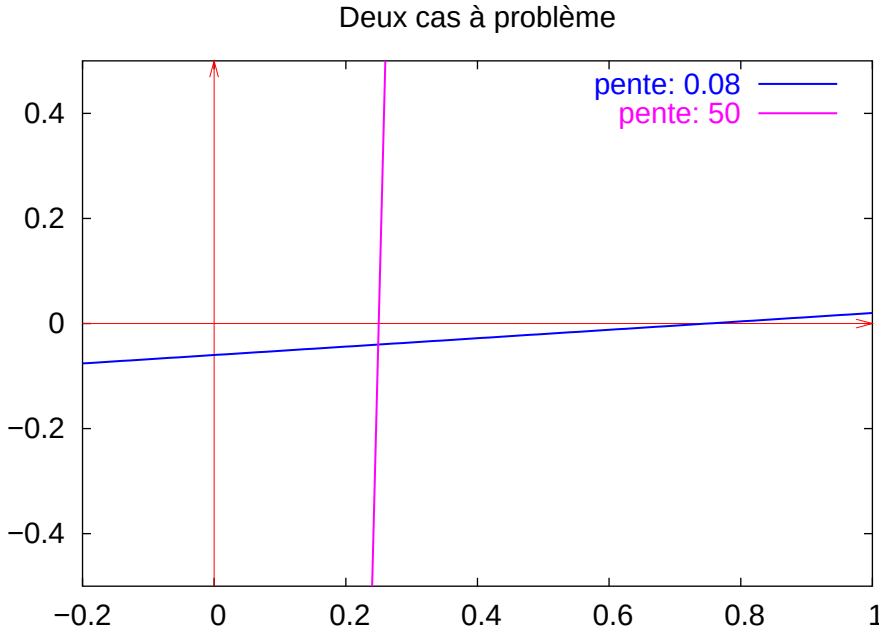


Figure 1: Recherche de zéro

Sur la figure (1), on voit en effet que pour la courbe bleu f reste proche de 0 sur une large gamme de x au voisinage de la solution (0.75). On peut donc avoir ϵ_f petit alors que l'on est loin de la solution. Sur la courbe mauve en revanche,

on peut avoir x proche de la solution (0.25), et donc ϵ_x petit alors que f reste significativement différent de 0. Savoir si cela est gênant ou non dépend bien sur du problème physique considéré...

La méthode générale à utiliser en l'absence de raison précise de faire un autre choix est la méthode de Newton-Raphson. Elle présente un avantage principal: sa convergence est quadratique (quand elle converge). C'est à dire que le nombre de chiffres significatifs double à chaque itération. En revanche, elle est redoutablement efficace pour partir dans les décors si l'on prend un point de départ au hasard. De plus elle exige de connaître le Jacobien de la fonction dont on cherche un zéro, ce qui n'est pas toujours facilement accessible.

3.2 Technique

On part d'un développement limité au premier ordre de la fonction F au voisinage de la solution X_0:

$$F\left(X_0 + \delta X\right) = F\left(X_0\right) + J\left(X_0\right).\delta X$$

où δX est une (petite?) perturbation au voisinage de X_0, et $J\left(X_0\right)$ est la matrice Jacobienne (ou Jacobien) de F calculée au point X_0, c'est à dire telle que:

$$j_{ij} = \frac{\partial f_i}{\partial x_j}\left(X_0\right)$$

Or, par construction, $F\left(X_0\right) = 0$. Partant d'un vecteur X astucieusement choisi, on voit donc que l'écart δX entre X (connu) et X_0 (recherché) est solution d'un système d'équations **linéaire**:

$$J\left(X_0\right).\delta X = F(X)$$

Il reste une difficulté: comme on ne connaît pas X_0, on ne peut pas calculer J en ce point. Comme d'habitude, on va donc le calculer au point X en espérant que l'approximation introduite n'est pas trop catastrophique.

L'algorithme consiste donc à partir d'un point $\overline{X_1}$, à résoudre le système:

$$J\left(\overline{X_1}\right).\delta X_1 = F(\overline{X_1}) \tag{4}$$

à poser $\overline{X_2} = \overline{X_1} + \delta X_1$, et à recommencer tant que les critères de convergence ne sont pas satisfaits.

On voit sur l'équation (4) pourquoi un mauvais choix de conditions initiales amène de sérieux problèmes. Si $J\left(\overline{X_i}\right)$ est mal conditionnée à un moment quelconque de l'itération (ou pire, singulière), alors l'une des composante de δX_i a toutes les chances de prendre une valeur élevée (quasi infinie). le point suivant $\overline{X}_{i+1}$ n'a alors plus de signification et l'algorithme ne converge pas.

Pour limiter cette sensibilité, on accepte souvent de dégrader les performances de l'algorithme, en utilisant un facteur de "relaxation" α inférieur à 1. On pose alors: $\overline{X}_{i+1} = \overline{X_i} + \alpha.\delta X_i$. La convergence est moins rapide, mais on oblige chaque itération à rester dans un voisinage de l'itération précédente, ce qui limite le risque de partir dans le décors.

3.3 Approximation numérique du Jacobien

Lorsque les fonctions constitutives du vecteur F ont une expression analytique suffisamment simple, on peut calculer analytiquement le jacobien J. Si cela est possible, c'est **toujours** préférable, quitte à utiliser des outils de calcul symbolique pour les expressions trop lourdes.

Dans certains cas, on ne peut malheureusement pas mener à bien la dérivation analytique, et il faut alors se résoudre à calculer une dérivée numériquement. C'est toujours un pis-aller, car il s'agit d'une des opérations les plus instables de l'analyse numérique. On l'illustrera ici seulement sur le cas d'une fonction à une variable, mais la généralisation est immédiate.

Soit une fonction $f(x)$ dont on cherche à calculer la dérivée en un point x_0. Le point de départ est la définition mathématique de la dérivée:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Une première erreur (grossière) est d'utiliser cette formule sans la symétriser. Cela revient à ne "prendre" de l'information sur la dérivée recherchée que d'un seul coté du point x_0. On utilisera donc l'expression approchée suivante:

$$f'(x_0) \simeq \frac{f(x_0 + h) - f(x_0 - h)}{2\,h}$$

Avec une valeur de h "bien" choisie. La deuxième erreur (plus subtile, mais grave) consiste à choisir une valeur de h trop petite (en particulier, proche de la précision machine). Dans ce cas, en effet, les valeurs de $f(x_0 + h)$ et de $f(x_0 - h)$ sont extrêmement proches (puisque f est continue). Leurs représentations en mémoire sont donc identiques à l'exception des quelques derniers bits (de poids le plus faible). Lorsque l'on calcule leur différence, on a donc très peu de bits significatifs, et le nombre en mémoire est complété au hasard (en fonction du bon vouloir des concepteurs du compilateur), souvent par des 0, mais pas obligatoirement.

L'opération suivante consiste à diviser ce nombre très mal connu par $2h$ qui est **très** petit (exprès). On obtient donc un effet d'amplification majeur de l'erreur par une (presque) division par 0.

Le résultat est en général dépourvu du moindre intérêt.

Une "bonne" valeur de h dépend, comme d'habitude, du problème considéré, mais on peut dire que pour un problème bien dédimensionné (donc pour lequel x et $f(x)$ sont proches de 1), et en travaillant en double précision, une valeur de $h \sim 10^{-6}$ est raisonnable.

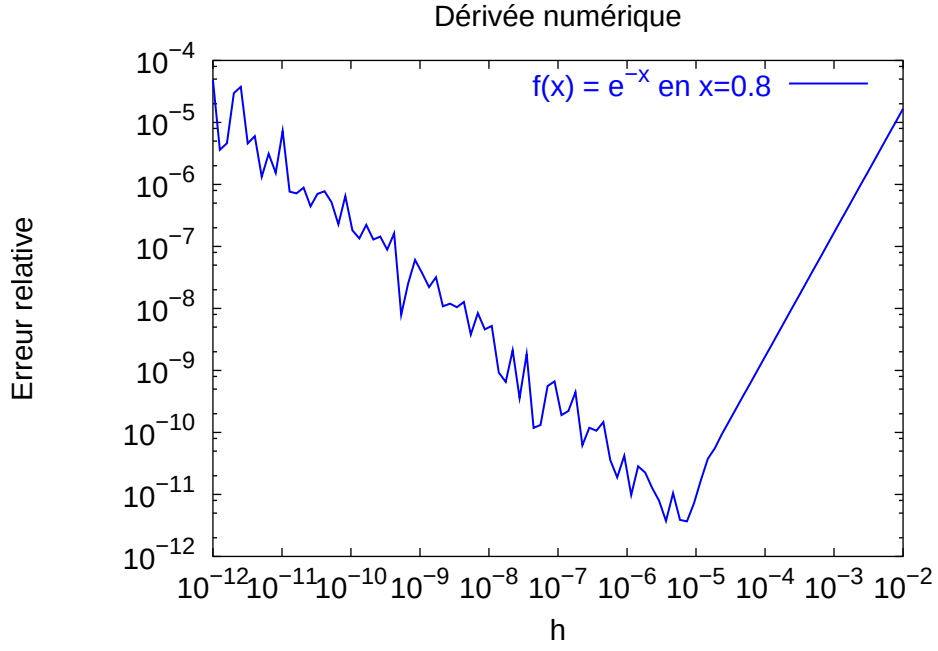


Figure 2: Choix du "bon" pas pour dériver

La figure (2) illustre ce phénomène sur une fonction élémentaire ($f(x) =$

$exp(-x)$). L'optimum est un peu au dessus de $h = 10^{-6}$, et l'erreur croit (en loi de puissance) quand on s'en éloigne.

4 Calcul d'intégrales numériquement

4.1 Présentation

Le calcul numérique d'une intégrale définie $\int_a^b f(x)\,dx$ est très différent suivant que l'on est libre de calculer $f(x)$ en tout point x de son choix, ou bien si elle n'est n'est connue qu'en un nombre de points x_i donnés à l'avance.

Dans le premier cas, le choix de méthodes est très grand et nous ne le décrirons que très brièvement. Le second est beaucoup plus pauvre, bien que très fréquent.

Dans tous les cas, on remplace une intégrale (continue) par une somme discrète:

$$\int_a^b f(x)\,dx \simeq \sum_i \omega_i\, f(x_i)$$

Les méthodes se différencient par la façon de choisir les x_i (si on le peut) et les ω_i pour que le résultat soit aussi précis que possible. Généralement, on essaye de faire en sorte que l'approximation soit exacte pour toute une classe de fonctions, souvent des polynômes.

4.1.1 Méthodes de Gauss

Ces méthodes, très spécialisées, sont suffisamment performantes quand on a la chance de pouvoir les utiliser pour mériter une mention ici, même si on se contente de principes généraux. Elles concernent les intégrales de la forme:

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

où $g(x)$ est une fonction analytique simple comme par exemple $exp(-x^2)$, $\frac{1}{\sqrt{1-x^2}}$, ou ...1 (!). Ces fonctions permettent de construire des familles de polynômes orthogonaux, dont la théorie sort complètement du cadre de ce cours. Dans ce cas, on peut montrer qu'en choisissant pour les x_i les racines du polynôme de degré $n+1$, on peut calculer les ω_i de telle sorte que:

$$\int_a^b g(x)\,f(x)\,dx = \sum_i^n \omega_i\, f(x_i)$$

La formule étant exacte pour tout polynôme de degré inférieur à n. Il faut

noter que la fonction $g(x)$ n'intervient **plus** dans la sommation! La fonction $f(x)$ est souvent à variations "assez" lentes; l'essentiel de la difficulté d'intégration étant concentrée dans g. On peut alors souvent obtenir un résultat remarquablement précis avec très peu de points, ce qui rend ces méthodes extrêmement performantes.

Les méthodes associées portent le nom de Gauss-Legendre ($g(x) = 1$), Gauss-Hermite ($g(x) = e^{-x^2}$), Gauss-Chebyshev ($g(x) = (1 - x^2)^{-1/2}$), Gauss-Laguerre ($g(x) = x^\alpha e^{-x}$), etc... en fonction de la fonction $g(x)$. Les valeurs des x_i et ω_i associés sont tabulées pour toutes (petites) valeurs de n. On peut également construire une méthode spécialisée à partir de (presque) n'importe quelle fonction $g(x)$ en se fatigant un peu (voir (Press et al., 1992, Par. 4.5)).

4.1.2 Intégrations "polynômiales"

Si l'intégrale ne se prête pas à une intégration de Gauss, on utilise l'une ou l'autre des méthodes "polynômiales" classiques. Elles sont construites sur l'idée que l'on ne sait "bien" calculer que des intégrales analytiques. Donc, pour obtenir une bonne approximation de notre intégrale, il faut d'abord trouver une bonne approximation analytique de la fonction $f(x)$, puis calculer analytiquement l'intégrale de celle-ci.

Comme souvent, les approximations en questions sont des polynômes de divers degrés:

0: fonction en escalier. Pédagogique, mais inutilisable en pratique.

1: interpolation linéaire. Méthode des "trapèzes". L'outil de base pour tout le reste.

2: interpolation parabolique. Méthode de Simpson.

Au delà, on ne les programme jamais directement. En revanche, on peut montrer que ces approximations successives peuvent se déduire l'une de l'autre à (relativement) peu de frais. C'est la méthodes de Romberg, qui est la méthode à préférer si on est libre de choisir ses abscisses. C'est celle qui est présentée ci-dessous.

Si la fonction f n'est connue qu'en un nombre fini et imposé de points, on se limite souvent à la méthode des trapèzes, que l'on peut éventuellement étendre à une méthode de Simpson à pas non constants. C'est assez laborieux...

4.2 Technique - Méthode de Romberg

Soit à calculer $I = \int_a^b f(x)\,dx$. On suppose que $f(x)$ peut être calculée à (relativement) peu de frais en tout point x de l'intervalle $[a, b]$. L'idée est de calculer une approximation de I utilisant un découpage régulier de l'intervalle $[a, b]$, puis de raffiner le découpage en coupant en deux chaque intervalle $[x_i, x_{i+1}]$ pour obtenir une approximation basée sur une interpolation polynômiale d'un ordre plus élevée. La beauté de la chose est que l'on n'a pas besoin de recalculer f aux points déjà utilisés! De plus, on peut montrer que la méthode est inconditionnellement stable.

4.2.1 Règle du trapèze récursive

On part de l'approximation d'ordre 0:

$$I_0^T = \frac{1}{2}(b-a)\ (f(a)+f(b))$$

Cette relation est exacte pour une fonction affine. En introduisant un point $x_1 = \frac{a+b}{2}$, on obtient une deuxième approximation du trapèze par:

$$I_1^T = \frac{1}{2}(f(a)+f(x_1))\ (x_1-a) + \frac{1}{2}(f(x_1)+f(b))\ (b-x_1)$$

ou bien:

$$I_1^T = \frac{b-a}{2}\,\frac{1}{2}\ (f(a)+2f(x_1)+f(b))$$

On constate que $I_1^T = \frac{1}{2}I_0 + \frac{1}{2}(b-a)\,f(x_1)$

On peut généraliser cette procédure par:

$$I_J^T = \frac{1}{2}\,I_{J-1}^T + h\sum_{k=1}^{2^{J-1}} f(x_{2k-1})$$

Avec $h = (b-a)/2^J$ et $x_i = a + ih$. Dans cette expression, on n'utilise qu'un x_i sur deux (d'où l'indice $2k-1$), les autres étant déjà pris en compte dans I_{J-1}^T.

Cette procédure permet déjà d'améliorer récursivement l'approximation de I en réutilisant les points déjà calculés.

4.2.2 Règle de Simpson récursive

On peut également construire une approximation "de Simpson" par:

$$I_1^S = \frac{b-a}{2}\,\frac{1}{3}\,\left(f(a) + 4f(x_1) + f(b)\right)$$

Et cette fois-ci $I_1^S = \frac{1}{3} I_0^T + \frac{2}{3}\,(b-a)\,f(x_1)$, que l'on peut réécrire:

$$I_1^S = \frac{4I_1^T - I_0^T}{3}$$

Cette formule se généralise également à:

$$I_J^S = \frac{4I_J^T - I_{J-1}^T}{3}$$

On peut donc obtenir une approximation de I basée sur des paraboles à partir d'approximations basées sur des droites à l'aide uniquement de deux multiplications et d'une soustraction!

On peut généraliser cette idée, et passer d'une approximation à l'aide de polynômes de degré $n-1$ à des polynômes de degré n à un coup numérique négligeable. C'est le principe de l'intégration de Romberg

4.2.3 Règle de Romberg

Soit $R(J,K)$ une approximation de l'intégrale I telle que:

- Pour $K = 0$ on a une séquence de règles du trapèze.
- Pour $K = 1$ on a une séquence de règles de Simpson.

Alors on peut passer d'interpolations par des polynômes de degré K à des polynômes de degré $K+1$ par:

$$R(J,K) = \frac{4^K\,R(J,K-1) - R(J-1,K-1)}{4^K - 1}$$

En pratique, on compare les valeurs de $R(J-1,J-1)$ et $R(J,J)$. Si la différence (relative) est supérieure à un seuil, on applique l'algorithme suivant:

1. Créer une nouvelle série de points avec un pas $h_{J+1} = (b-a)/2^{J+1}$

2. Calculer la nouvelle approximation trapézoïdale $R(J+1,0)$ comme indiqué en (4.2.1).

3. En déduire les approximations successives jusqu'à $R(J+1,J+1)$.

Dans la plupart des cas "normaux", cet algorithme est le plus efficace possible.

5 Résolution de systèmes d'équations différentielles ordinaires

5.1 Présentation

Un système d'équations différentielles ordinaires (EDO) s'écrit sous forme canonique:

$$\dot{X} = F(X, t)$$

où t est une variable réelle continue (souvent le temps, mais pas forcément), $X(t)$ est un vecteur de fonctions inconnues (dimension n), et F un vecteur de fonctions données. Pour que le problème soit "bien posé", il est nécessaire de lui adjoindre des "conditions initiales" sous la forme $X(t = 0) = X_0$. Ce système est du premier ordre. On peut toujours ramener un système d'ordre supérieur au premier ordre en augmentant le nombre de variables.

Si t intervient explicitement dans F, le système est dit "non-autonome", ce qui est une difficulté numériquement. On peut là aussi éliminer t en ajoutant une variable supplémentaire $x_{n+1} = t$, ce qui donne une équation supplémentaire:

$$\dot{x}_{n+1} = 1$$

Dans la suite, nous considérerons donc un système de dimension n, autonome, sous la forme:

$$\dot{X} = F(X)$$

Bien que X soit continue, on ne peut numériquement le connaître qu'en des points discrets. La résolution du système d'équations différentielles consiste donc à déterminer les meilleures approximations possibles d'un nombre fini de valeurs (t_i, X_i) où $X_i = X(t_i)$.

Il y a deux grandes catégories de méthodes, suivant que pour calculer X_{i+1} on utilise uniquement X_i (méthodes à pas libre), ou également les points X_{i-1}, X_{i-2}, etc... (méthodes à pas liés). Ces dernières ont été extrêmement populaires (classe des méthodes d'Adams: Adams-Bashford, Adams-Moulton, etc...). Elles sont moins à la mode actuellement car difficiles à implémenter, lourdes à initialiser (il faut connaître X en plusieurs valeurs de t), et sujettes à des oscillations difficiles à contrôler dans un certain nombre de cas importants. Nous n'en parleront pas

plus ici.

Nous nous limiterons donc aux méthodes permettant de calculer X_{i+1} à partir de la seule connaissance de X_i et de $F(X_i)$. La plupart se regroupe sous le vaste chapeau de "Runge et Kutta", dont la classique "RK4" n'est qu'un cas particulier. Nous détaillerons une méthode du deuxième ordre, et n'utiliserons la méthode d'ordre 4 qu'à partir d'une bibliothèque.

5.2 Technique

Soit à calculer $X_{i+1} = X(t_{i+1})$, où $t_{i+1} = t_i + h$, à partir de X_i et $\dot{X}_i = F(X_i)$ à t_i. Le pas h peut varier ou non d'un point au suivant. Nous examinerons ce point plus loin.

Le point de départ de la plupart des méthodes est de faire un développement limité de X_{i+1}:

$$X(t_{i+1}) = X(t_i) + h\,\dot{X}(t_i) + o(h^2)$$

Soit:

$$X_{i+1} \simeq X_i + h\,F(X_i) \tag{5}$$

On peut être tenté d'utiliser l'équation (5) directement. C'est presque toujours une mauvaise idée[2]. Les solutions calculées de cette façon ont en effet fortement tendance à être instables.

On voit sur la figure (3) que pour aller directement du point $M_i : (t_i, X_i)$ à $M_{i+1} : (t_{i+1}, X_{i+1})$ la "bonne" formule à appliquer est:

$$X_{i+1} = X_i + \alpha\,h$$

où α est la pente de la **corde** entre les deux points. ...Évidemment, cette pente est inconnue, et tout le jeu consiste à essayer d'en trouver la meilleure approximation possible.

Une deuxième idée peut être (par exemple) de prendre la pente de la courbe au point d'arrivée M_{i+1}, ce qui donne:

$$X_{i+1} = X_i + h\,F(X_{i+1})$$

[2]Sauf comme outil de base dans certaines méthodes de résolution d'équations aux dérivées partielles, qui sortent du cadre de ce cours.

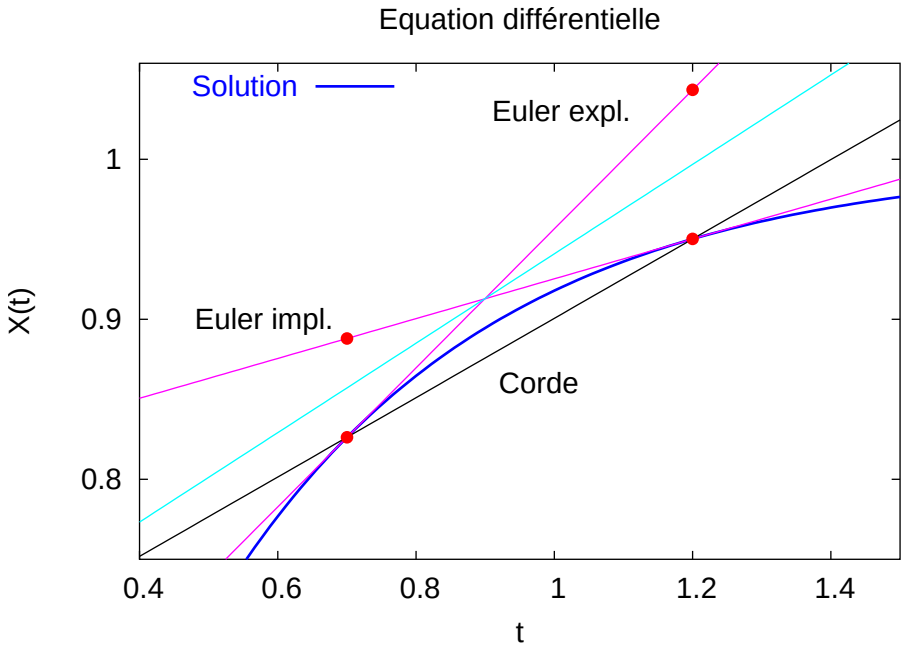


Figure 3: Un pas d'intégration

Comme on ne connaît pas (et pour cause) X_{i+1} pour calculer $F(X_{i+1})$, on utilise une approximation, qui est ...la formule (5)! Ce qui donne finalement:

$$X_{i+1} = X_i + h\,F\left(X_i + h\,F(X_i)\right) \tag{6}$$

Il s'agit du premier (et du plus simple) exemple des méthodes dites "prédicteur - correcteur". On calcule une première approximation (médiocre) du résultat recherché, puis on utilise ce résultat lui-même pour calculer une deuxième approximation (en principe) meilleure.

Ici le résultat n'est pas vraiment meilleur, à un détail (important) près: la méthode est stable! Elle est donc souvent utilisée pour rechercher un résultat "quick and dirty". On peut cependant faire beaucoup mieux à très peu de frais. On voit clairement sur la figure (3) que, si la courbure est constante entre les deux points M_i et M_{i+1}, la pente en l'un surestime α alors qu'en l'autre elle le sous-estime. Une meilleure approximation est donc de prendre la moyenne:

$$X_{i+1} = X_i + \frac{1}{2}\,h\,\left(F(X_i) + F\left(X_i + h\,F(X_i)\right)\right) \tag{7}$$

On peut démontrer que cette formule est équivalente à un développement de X au deuxième ordre:

$$X_{i+1} = X_i + h\,F(X_i) + \frac{h^2}{2}\,\frac{d}{dt}F(X_i) + o(h^3)$$

Notons au passage, que $\frac{d}{dt}F(X) = F'(X)\,F(X)$ (pourquoi?), ce qui permet potentiellement de calculer un développement limité à n'importe quel ordre. En pratique, ce type de formule est indispensable pour établir les résultats théoriques sur l'ordre des différentes méthodes, mais pas très pratique à programmer. On préfère donc utiliser des formules utilisant le même principe que (7). Celle-ci constitue la première méthode vraiment utilisable en pratique, et est une des formes de la méthode de Runge-Kutta d'ordre 2. Il est particulièrement remarquable de noter qu'elle utilise le **même** nombre d'appel à la fonction F que Euler implicite, et ne coûte donc pas plus cher à calculer alors qu'elle est d'un ordre supérieur.

Les "vrais" méthodes de Runge-Kutta (d'ordre 4 ou 4-5), s'inspirent du même principe: trouver à l'aide d'approximations successives une bonne valeur de α, pente de la corde. L'ordre de la méthode provient du fait qu'elle approxime le développement limité de X jusqu'à ce même ordre.

5.3 Contrôle de l'erreur

Pour que le résultat soit utile, il est nécessaire qu'à chaque pas l'erreur commise reste inférieure à un seuil choisi par l'utilisateur. Si le pas est constant, il est donc imposé par la position sur la trajectoire où l'erreur croit le plus vite, et doit être gardé à cette valeur même lorsque ce n'est pas nécessaire. On perd donc un temps de calcul important, et on accumule aussi des erreurs de sommation. l'idéal est qu'à chaque pas on puisse adapteer la valeur de h à la précision voulue. Pour cela on peut utiliser la technique suivante:

Connaissant X_i on détermine X_{i+1} deux fois de deux façons différentes. Tout d'abord, $X_{i+1}^{(1)}$ est calculé en un pas de longueur h, puis $X_{i+1}^{(2)}$ est calculé par deux pas de longueur $\frac{h}{2}$. Si l'algorithme utilisé est d'ordre n, on a alors:

$$X_{i+1}^{(1)} = X_i + h\,\Delta^{(1)} + h^{n+1}\,C^{(1)}$$

$$X_{i+1}^{(2)} = X_i + h \left(\frac{\Delta_A^{(2)} + \Delta_B^{(2)}}{2} \right) + \frac{h^{n+1}}{2^n} \left(\frac{C_A^{(2)} + C_B^{(2)}}{2} \right)$$

où Δ est l'approximation de la variation donnée par la méthode à chaque pas (la corde), et $C = F^{(n+1)}(\bar{t})$ avec $\bar{t} \in [t_i, t_{i+1}]$ est la valeur de la dérivée cinquième de F en un point intermédiaire (et inconnu) de l'intervalle. Les indices A et B désignent les deux demi-pas de la deuxième évaluation.

Une estimation de l'erreur est donnée par l'écart entre ces deux évaluations. Si l'on veut une précision **relative** de ϵ alors on peut se "permettre" un écart entre $X_{i+1}^{(1)}$ et $X_{i+1}^{(2)}$ de l'ordre de:

$$E_0 = \epsilon \left| X_{i+1}^{[2]} \right| \sim h_0^{n+1} C$$

alors que l'erreur réellement commise est de l'ordre de:

$$E = \left| X_{i+1}^{(1)} - X_{i+1}^{(2)} \right| \sim h^{n+1} C$$

On obtient donc la valeur h_0 de h qu'il aurait fallu utiliser par:

$$h_0 = h \left(\frac{E_0}{E} \right)^{1/(n+1)}$$

Il faut encore se prémunir contre un risque; on a négligé les termes d'ordres 6 ici. La valeur de h_0 est donc "limite". Il est donc prudent de taper "un peu en dessous" en la mutipliant par un facteur de sécurité de l'ordre de (par exemple) 0.95.

L'expression finale de h_0 est donc:

$$h_0 = 0.95\, h \left(\frac{\epsilon \left| X_{i+1}^{(2)} \right|}{\left| X_{i+1}^{(1)} - X_{i+1}^{(2)} \right|} \right)^{1/(n+1)}$$

L'algorithme est alors simple:

- Si $h > h_0$ la précision n'est pas suffisante; On rejette le résultat et on recommence avec le nouveau pas h_0.
- Si $h < h_0$ la précision est suffisante. On conserve le résultat et on remplace h par h_0 pour aller aussi "vite" que possible.

Bien entendu, s'il s'agit d'un système d'équations différentielles, le calcul de h_0 se fait sur la variable qui présente le plus grand écart E.

Le prix à payer est qu'il a fallu trois évaluations de F au lieu d'une, ce qui allonge le temps de calcul. Mais ce n'est pas complètement perdu. En effet, dans tous les cas "normaux", on peut considérer que la dérivée cinquième de F ne doit "pas trop" varier sur l'intervalle $[t_i, t_{i+1}]$. Donc $C^{(1)} \sim C_A^{(2]} \sim C_B^{(2)}$ (au moins au premier ordre). Et donc:

$$X_{i+1}^{(1)} = X_i + h\,\Delta^{(1)} + h^{n+1}\,C + o(h^{n+2})$$

$$X_{i+1}^{(2)} = X_i + h\,\Delta^{(2)} + \frac{h^{n+1}}{2^n}\,C + o(h^{n+2})$$

On peut donc améliorer X_{i+1} à l'aide de la formule:

$$X_{i+1} = \frac{2^n\,X_{i+1}^{(2)} - X_{i+1}^{(1)}}{2^n - 1} + o(h^{n+2})$$

qui est d'ordre 5! On a donc, en plus d'un algorithme permettant d'ajuster le pas, une méthode meilleur d'un ordre d'approximation sur la même méthode à pas constant. Dans tout ce qui précède, la façon de calculer Δ n'est pas précisé, et l'algorithme s'adapte donc à toute méthode à pas non liés.

6 Optimisation non-linéaire

6.1 Présentation

On dispose d'une série de données "expérimentales" (i.e. entachées d'une certaine erreur) (x_i, y_i, ω_i) dans laquelle les y_i présentent une incertitude ω_i. On veut les modéliser à l'aide d'une "loi" $f_a(x)$, dans laquelle α représente un certain nombre de paramètres réels. Le problème est de déterminer le "meilleur" jeu de paramètres a, compte tenu des incertitudes ω_i.

Lorsque la loi f_a est linéaire, ce problème consiste à trouver la classique "droite des moindre carrés", qui, pour être bien posé, n'en est pour autant toujours facile à bien utiliser.

Nous allons l'étudier ici dans le cas général où f_a n'est **pas** linéaire. Le principe reste le même, on construit une certaine quantité dont on espère qu'elle caractérise correctement l'écart entre le modèle et les données expérimentales, et on cherche à la rendre la plus petite possible en faisant varier les paramètres. Le jeu de paramètres résultant est réputé représenter au mieux les données.

Évidemment l'intérêt de la procédure est entièrement lié au choix astucieux du modèle f_a... Ce qui nécessite encore une fois une compréhension physique correcte du problème.

Traditionnellement, la quantité physique à minimiser est notée χ^2. Elle est définie par:

$$\chi^2(a) = \sum_{i=1}^{n} \left(\frac{y_i - f_a(x_i)}{\omega_i} \right)^2$$

C'est une fonction des paramètres a.

Il est très important d'utiliser de "bonnes" valeurs des incertitudes ω_i. Tout d'abord, cela permet d'avoir un χ^2 sans dimension. On peut ainsi comparer (ou modéliser) des quantités ayant des unités et/ou des ordres de grandeur très différents, sans que l'un des y_i domine les autres. D'autre part, si les ω_i sont effectivement représentatifs de l'écart typique de la valeur y_i mesurée par rapport à une "vrai" valeur, alors on **doit** s'attendre à ce que l'écart entre y_i et $f_a(x_i)$ soit de l'ordre de ω_i. A l'optimum (a_0), on doit donc avoir:

$$\chi^2(a_0) \simeq n$$

Si χ^2 est grand devant n, alors le modèle est mauvais, soit parceque le jeu de paramètres a est mal calculé, soit tout simplement parceque le choix du modèle f_a ne reflète pas la physique du problème. Inversement, si χ^2 est petit devant n, il faut se poser des questions. Le cas le plus fréquent est celui où les barres d'erreur ont été sur-évaluées. Il convient donc de ré-analyser les données. Il peut aussi arriver que l'on cherche à sur-interpréter l'expérience, par exemple en utilisant trop de paramètres libres par rapport aux contraintes dont on dispose. Le modèle "marche" parfaitement, mais il ne veut rien dire.

6.2 Technique

6.2.1 Minimisation du χ^2

Ce développement est tiré de (Press et al., 1992, Par 15.5). Le modèle f_a étant supposé dérivable par rapport aux paramètres a (M valeurs a_k), on peut cherche à minimiser χ^2 en cherchant les zéros de ses dérivées par rapport aux a_k:

$$\frac{\partial \chi^2}{\partial a_k} = 0 \tag{8}$$

Or, ce gradient s'écrit:

$$\frac{\partial \chi^2}{\partial a_k} = -2 \sum_{i=1}^{n} \frac{y_i - f_a(x_i)}{\omega_i^2} \frac{\partial f_a(x_i)}{\partial a_k}$$

Au voisinage de la solution, on peut de nouveau développer chacune des composantes du gradient par rapport aux composantes de a, ce qui donne:

$$\frac{\partial^2 \chi^2}{\partial a_k \partial a_l} = 2 \sum_{i=1}^{n} \frac{1}{\omega^2} \left[\frac{\partial f_a(x_i)}{\partial a_k} \frac{\partial f_a(x_i)}{\partial a_l} - (y_i - f_a(x_i)) \frac{\partial^2 f_a(x_i)}{\partial a_k \partial a_l} \right]$$

Pour simplifier la suite des formules, on pose:

$$\beta_k = -\frac{1}{2} \frac{\partial \chi^2}{\partial a_k}$$

$$\alpha_{kl} = \frac{1}{2} \frac{\partial^2 \chi^2}{\partial a_k \partial a_l}$$

On peut alors écrire le vecteur $a = a_0 + \delta a$, où a_0 est la solution recherchée,

et reporter cette expression dans l'équation (8). Le gradient du χ^2 s'écrit alors:

$$\bigtriangledown\chi^2(a) = D(a) \cdot \delta a$$

où le vecteur $\bigtriangledown\chi^2$ a pour composantes β_k, la matrice D pour composantes α_{kl} (à un facteur 2 près, utile pour la suite). On peut donc (en principe) passer d'une valeur approchée de a à la valeur recherchée a_0 en résolvant un système linéaire:

$$\sum_{l=1}^{M} \alpha_{kl}\, \delta_l = \delta k$$

6.2.2 Méthode de Levenberg-Marquardt

En pratique, les valeurs initiales des paramètres a sont rarement assez bons pour que l'on puisse directement utiliser le système précédent, de façon analogue à la méthode de Newton-Raphson. La méthode standard utilisée pour résoudre ce problème est celle de Levenberg-Marquardt, dont le principe est exposé dans Press et al. (1992). Il n'est pas souhaitable chercher à l'implémenter soi-même, et on trouve ici un exemple typique de problème pour lequel il **faut** utiliser une bibliothèque numérique.

References

Acton, F.S., 1990, "Numerical methods that work", Corrected edition, Washington, Mathematical Association of America.

Kernighan, B.W., Pike, R., 1999, "The practice of programming", Addison-Wesley.

Mathews, J.H., 1992, "Numerical methods for mathematics, science and engineering", 2nd edition, Prentice-Hall Inc.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 1992, "Numerical Recipes in C", 2nd edition, Cambridge university press.