complément

OPTIMISATION

Comme nous l'avons fait pour l'intégration numérique, l'*optimisation* étant un très vaste et complexe sujet, notre but n'est pas de le traiter de façon exhaustive, ni dans le détail (cela est très bien fait dans les ouvrages spécialisés), mais d'en dégager, aussi simplement que possible, les éléments utiles pour comprendre comment fonctionne un optimisateur et comment l'utiliser correctement.

Bibliographie

- *Numerical Recipes in C*. http://www.nr.com
(Chapitre 10 : Minimization or Maximization of Functions ; Chapitre 15 : Modeling of Data)

- *Function minimization.* F. James, CERN, Genève, Proceeding of the 1972 CERN Computing and Data Processing School, Pertisau, Austria

- *Programmation mathématique (tomes 1 et 2),* Michel Minoux, 1983, Editions Dunod, ISBN : 2-04-015487-6 et 2-04-015542-2

- *Optimization by Simulated Annealing.* S. Kirkpatrick, C. Gelatt and M. Vecchi, *Science*, 1983, *220*, 671-680

- N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, *Journal of Chemical Pysics*, 1953, *21*, 1087-1092

- *Optimisation par Recuit Simulé en Dynamique Chimique*, Nathalie Simon, *Rapport de stage de fin d'études en mathématiques appliquées, I.N.S.A. Toulouse*, 1996

1. Situation du problème

1.1 Minimisation d'une fonction

Soit une fonction $f(\boldsymbol{x})$, où $\boldsymbol{x}$ est un vecteur de n variables, que nous supposerons ici réelles. Elle correspond à une hyper-surface dans l'espace $\mathbf{R}^{n+1}$. Par analogie avec notre espace géométrique habituel, cette hyper-surface se présente comme un "paysage" (Fig. 1), avec des montagnes, des cuvettes, des vallées plus ou moins sinueuses, des gorges, des falaises, des surplombs, des tunnels, des plateaux, des

plaines... et même un peu plus, serait-on tenté de dire, dans un espace de dimension supérieure !

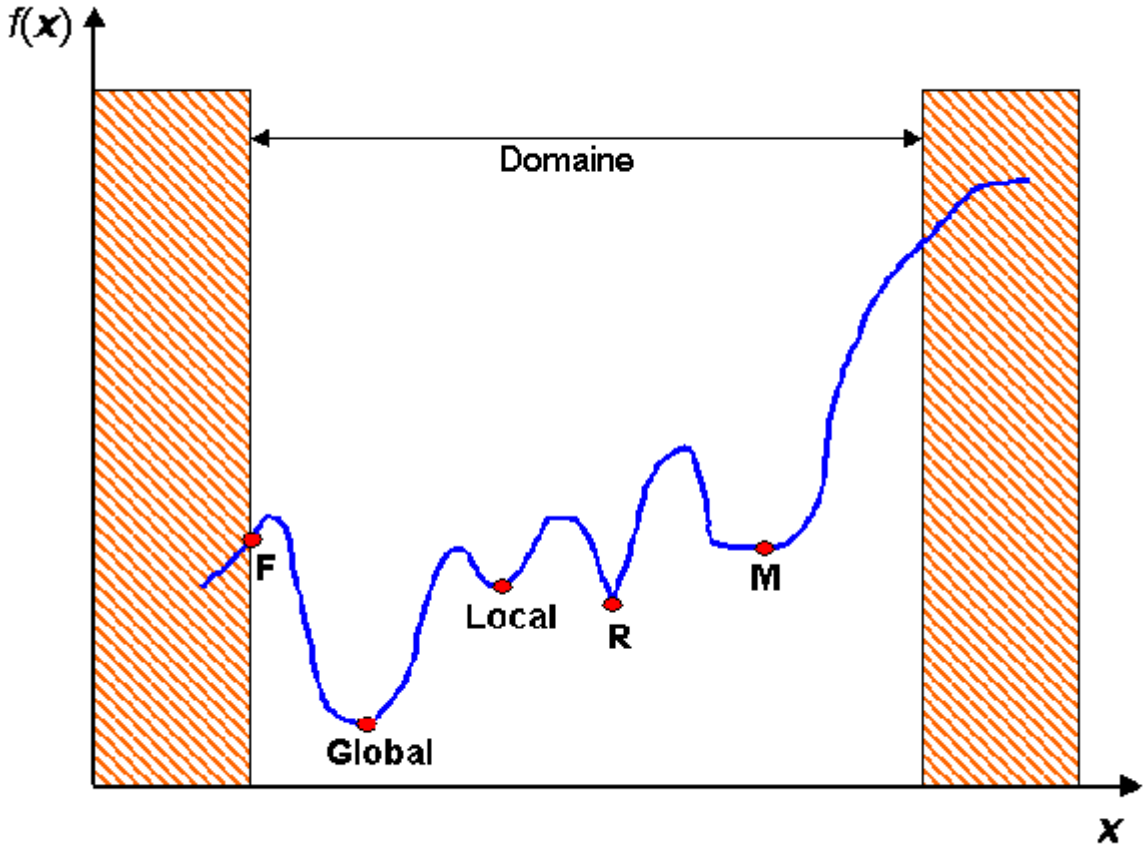


Fig. 1 **"Paysage" de $f(x)$**

x est supposé ici à une seule dimension (n = 1) : le "paysage" se réduit à la courbe bleue. Il devient une surface pour n = 2, et une hyper-surface pour n > 2

Parmi ces différents aspect du paysage, on est souvent intéressé par la recherche des points les plus bas, ou *minimisation*, ou les plus haut, *maximisation*. Il apparaît tout de suite que tout problème de maximisation se ramène à une minimisation en considérant la fonction opposée, $-f(x)$. C'est pourquoi on utilise le terme minimisation d'une façon générale.

La fonction peut présenter plusieurs minima, dits *locaux*, et, parmi eux, le *minimum global*, c'est-à-dire le plus bas. Ainsi le problème de recherche de minimum peut se présenter sous différentes formes :

- recherche d'un minimum local dans un certain domaine des variables x, où l'on sait qu'un seul minimum est présent ; c'est le problème le plus facile à résoudre, pour lequel il existe de nombreux algorithmes

- recherche du minimum global, dans un certain domaine présentant plusieurs minima, ou dans l'espace R^n entier ; c'est le cas en général dans les problèmes d'ajustement qui nous intéressent ici, et c'est un problème nettement plus difficile. En fait, on effectuera le plus souvent une recherche de minimum local, puis on essayera d'autres recherches à partir de points de départ différents pour tester s'il s'agit d'un minimum local ou global. Dans ce contexte, un minimum local est aussi appelé *faux minimum*

- recherche de tous les minima possibles ; c'est le cas, par exemple, dans les problèmes d'optimisation de structures moléculaires, où toutes les configurations possibles doivent être explorées.

Les dérivées partielles $\partial f/\partial x_i$ (i = 1 à n) s'annulent le plus souvent à un minimum, mais pas forcément : celui-ci peut être situé à la limite du domaine considéré, de pente non nulle (Fig. 1, **F**), ou être un point de rebroussement (Fig. 1, **R**).

Inversement, un point en "plaine" (Fig. 1, **M**) ou sur un "plateau", peut présenter des dérivées partielles nulles, du moins par rapport à certaines variables, sans être pour autant un minimum. Dans ce cas de figure, relativement fréquent, il existe une infinité de valeurs de ces variables donnant la même valeur de la fonction $f(\boldsymbol{x})$.

Si les variables, ou certaines d'entre elles, doivent être maintenues dans un certain domaine (valeurs positives, par exemple), l'optimisation est dite *contrainte*. Il peut y avoir également d'autres types de contraintes : variables entières ou ne pouvant prendre que certaines valeurs prédéfinies, etc.. Nous n'aborderons pas ici ce type d'optimisation.

La minimisation d'une fonction $f(\boldsymbol{x})$ linéaire (multi-linéaire si n > 1), est l'objet de ce qui est appelé la *programmation linéaire*. Dans le cas, plus général, d'une fonction quelconque, non-linéaire, on parle d'*optimisation non-linéaire*.

1) Dans l'expression *programmation linéaire*, le mot "programmation" est utilisé dans le sens de *programmation mathématique*, c'est-à-dire l'étude des *algorithmes* permettant d'atteindre un objectif donné, et non dans le sens de programmation informatique. Le but des programmes informatiques est évidemment de traduire le mieux possible ces algorithmes, avec les contraintes supplémentaires qu'impose la numérisation des données, avec une précision limitée.

2) L'augmentation de la dimensionnalité, c'est-à-dire du nombre de variables n, ne se traduit pas seulement par une simple augmentation du niveau, mais, d'une certaine façon, par un **changement de nature de complexité**.

Pour s'en convaincre, on peut évoquer par exemple le théorème de Pólya concernant les *marches au hasard isotropes* sur un réseau $\mathbf{Z}^n$ à n dimensions (les pas sont donc discrets et égaux, et aucune direction n'est privilégiée) :

- pour n = 1 et n = 2, la marche est *récurrente*, c'est-à-dire que le marcheur reviendra **toujours** à son point de départ

- pour n > 2, la marche n'est plus récurrente ; ainsi la probabilité de revenir au point de départ n'est plus que de 34% environ pour n = 3, et de 10% pour n = 6.

On voit donc que la *marche au hasard* dans un espace de dimension élevée devient de plus en plus hasardeuse, dans le sens anglais de "hazardous" ! Bien qu'il s'agisse d'un problème différent, on peut imaginer dès lors qu'il faudra se munir de méthodes éprouvées pour se promener "le moins au hasard possible" dans le paysage d'une fonction de n variables réelles afin d'en trouver le minimum.

1.2 Ajustement de données par un modèle : Fonction d'erreur

Le problème qui nous intéresse en cinétique, comme dans toute modélisation, est de déterminer, si possible, les paramètres (constantes de vitesse, coefficients d'extinction molaire, etc.) d'un modèle de telle sorte que les données calculées à l'aide de ce modèle et de ces paramètres soient le moins distantes possible de données expérimentales. Autrement dit, nous voulons *ajuster* le modèle sur ces données.

Dès lors, il faut disposer d'une ***fonction d'erreur***, mesurant cette *distance*, ou *erreur résiduelle*.

La fonction d'erreur est aussi appelée dans d'autres contextes : fonction *coût*, fonction de *mérite*, fonction *économique*, fonction *objectif*, etc..

L'ajustement consistera donc en la minimisation de la fonction d'erreur, dont les variables sont les paramètres du modèle :

$$E = f(\boldsymbol{p})$$

p désignant le vecteur des paramètres.

Les données expérimentales sont, d'une manière générale, les valeurs d'une certaine grandeur $y_e(x_i)$ obtenues pour un certain nombre de valeurs d'une variable indépendante x_i (i = 1 à m) (temps pour la cinétique, concentration d'une espèce, etc.). Le modèle permet, quant à lui, de calculer les valeurs correspondantes $y_c = h(x_i, \boldsymbol{p})$

Toute fonction $E\left(y_e(x_i),\ y_c(x_i, \boldsymbol{p})\right),\ > 0\ \forall \mathrm{i},$ et telle que $E = 0$ si $y_e(x_i) = y_c(x_i, \boldsymbol{p})\ \forall \mathrm{i}$
peut constituer une fonction d'erreur, susceptible d'être minimisée.

Il s'agit là d'une définition, mais en réalité une fonction d'erreur n'est jamais nulle, même pour un modèle parfait, d'une part à cause de l'erreur expérimentale sur y_e, et d'autre part à cause de la précision des calculs numériques de y_c.

Ainsi la fonction

$E = \sum_{i=1 \text{ à } m} | y_c(x_i, \boldsymbol{p}) - y_e(x_i) |$

pourrait constituer une fonction d'erreur.

Mais on utilise en général la ***somme des carrés des écarts*** :

$$E = \sum_{i=1 \text{ à } m} \left(y_c(x_i, p) - y_e(x_i) \right)^2 / m \quad (1)$$

la valeur des ***moindres carrés*** obtenue étant une estimation de probabilité maximum des paramètres ajustés, à la condition, toutefois, que les erreurs expérimentales soient indépendantes, normalement distribuées et avec une déviation standard, σ, constante.

Il est quelquefois préférable d'utiliser une somme *pondérée* des carrés des écarts :

$$E = \sum_{i=1 \text{ à } m} w_i \left(y_c(x_i, \boldsymbol{p}) - y_e(x_i) \right)^2 / m \quad (2)$$

ou w_i désigne le poids attribué au point i et dépend de la confiance que l'on peut accorder à la valeur expérimentale correspondante.

Idéalement, si la déviation standard σ_i de chaque point expérimental est connue, les poids à affecter devraient être les inverses des variances, $w_i = 1 / \sigma_i^2$, de sorte que la fonction d'erreur soit alors le ***chi-carré*** :

$$E = \chi^2 = \Sigma_{i=1 \text{ à } m} [(y_c(x_i, \boldsymbol{p}) - y_e(x_i)) / \sigma_i]^2 / (m - n - 1) \quad (3)$$

où n désigne le nombre de paramètres optimisés, et (m − n − 1) représente le nombre de degrés de liberté. La valeur de χ^2 devrait tendre alors vers l'unité pour un ajustement parfait et représente un estimateur absolu de la qualité de l'ajustement (on trouve couramment indiqué qu'une valeur de χ^2 comprise entre 0.9 et 1.2 signifie un ajustement correct).

Toutefois, cela n'est correct en toute rigueur que si l'erreur expérimentale suit une *distribution normale*, d'une part, et si le modèle est linéaire, d'autre part. Or, ni l'un, ni l'autre ne sont garantis en général. Par exemple, des données obtenues par comptage d'événements (comme le comptage de photons en fluorescence) suivent plutôt une distribution de Poisson. Quant à la linéarité, une simple réaction monomoléculaire conduit à une équation cinétique en e^{-kt}.

De plus, la variance réelle de chaque donnée expérimentale est en général inconnue. On doit alors en fournir une estimation. Mais si cette estimation n'est pas correcte, le *chi-carré*, qui reste bien sûr une fonction d'erreur utilisable, n'a plus de signification d'estimateur absolu.

Dès lors, il n'y a plus de raison d'utiliser le *chi-carré*, et on utilise en général, la simple *somme des carrés des écarts*, éventuellement pondérée à la demande (voir le Manuel de *Sa*, Chapitre XI, p. 53). Noter que tout cela n'a qu'un impact mineur sur l'optimisation proprement dite (on minimise toujours une fonction d'erreur). Simplement, la valeur de l'erreur résiduelle obtenue ne constitue pas, à elle seule, un estimateur de qualité de l'ajustement.

Pour calculer la fonction d'erreur, on doit calculer la fonction $y_c(x_i,\boldsymbol{p})$ pour chaque point. Cette dernière, *a priori* quelconque, peut demander un temps de calcul important. Il faudra donc rechercher les algorithmes de minimisation nécessitant le moins d'appel possible de la fonction d'erreur.

2. Recherche d'un minimum local

Oubliant pour l'instant toute la complexité possible du paysage de la fonction d'erreur, supposons que nous nous situons dans un domaine réduit à un ***bassin d'attraction*** vers un minimum unique, P*. Partant d'un point de départ P_0, quelconque (mais situé dans ce bassin), il s'agit donc de trouver un algorithme sûr et efficace pour atteindre le minimum P*. Même dans cette situation très simplifiée, le problème peut s'avérer plus difficile qu'il n'y paraît : il suffit d'imaginer par exemple le cas d'un bassin d'attraction en forme de vallée très sinueuse.

Tous les algorithmes que nous évoquerons sont des ***algorithmes itératifs***, c'est-à-dire que du point P_k, on va chercher un point P_{k+1} tel que la fonction d'erreur y soit plus faible, jusqu'à ce qu'un certain critère soit atteint (une valeur limite de la fonction d'erreur, ou un déplacement du point P inférieur à une valeur donnée, par exemple).

L'algorithme sera dit ***globalement convergent*** ou ***stable*** si la suite $\{P_0, P_1, ..., P_k\}$ converge vers P* $\forall$ P_0. La convergence globale indique donc la sûreté d'un algorithme, dans le cadre limité du bassin d'attraction où nous nous sommes situés. Elle ne signifie nullement la convergence vers un minimum global.

La ***convergence asymptotique*** mesure l'efficacité de l'algorithme, par le comportement de la suite $\{P_k\}$ au voisinage de P* :

- elle est ***linéaire*** si $\alpha = \lim_{(k\to\infty)} \| P_{k+1} - P^* \| / \| P_k - P^* \| < 1$
- et ***superlinéaire*** si $\alpha \to 0$

Pour résoudre le problème général, multi-dimensionnel, il est nécessaire de faire appel successivement à des *minimisation à une dimension*, suivant des directions de déplacement données. Cette opération devant être effectuée un grand nombre de fois, il est nécessaire d'utiliser un algorithme efficace.

2.1 Minimisation à une dimension

Une première possibilité serait de diviser le domaine en un certain nombre d'intervalles égaux, de calculer la fonction d'erreur pour chaque point ainsi déterminé et de choisir le point pour lequel cette fonction est la plus petite. La précision du

paramètre obtenu dépendrait alors de la finesse de la grille de balayage ainsi constituée. Cette méthode est de toute évidence stable (les résultats contiendront obligatoirement le minimum), et effectue un nombre déterminé à l'avance de calculs de la fonction d'erreur. Mais elle suppose un domaine strictement délimité et, surtout, elle peut être qualifiée de stupide, car elle ne tient aucun compte des calculs déjà effectués ! Les méthodes de balayage par *dichotomie* ou suivant la *section d'or* en constituent des variantes intelligentes.

Il faut distinguer les méthodes avec et sans calcul de dérivées. Dans notre cas, nous privilégions les méthodes sans calcul de dérivées, de façon à limiter autant que possible le nombre d'appels de la fonction d'erreur, qui peut être coûteux en temps de calcul. Nous ne décrirons, brièvement, que deux de ces méthodes, souvent utilisées en complément l'une de l'autre.

Soit p le paramètre, variable unique de la fonction d'erreur.

Méthode de succès-échecs, sans dérivée, de Rosenbrock

L'algorithme est très simple :

♦ partant de la valeur p_0, on effectue un déplacement arbitraire, d,
et on calcule $E(p_0)$ et $E(p_0+d)$

♦ si $E(p_0+d) < E(p_0)$, l'opération est un **succès** :
on remplace alors p_0 par $p_1 = p_0+d$ et d par $d = \alpha d$ $(\alpha > 1)$ (α = 2 par exemple)

♦ sinon, l'opération est un **échec** :
on remplace simplement d par $-\beta d$ $(\beta < 1)$ (on refait un pas, plus petit, dans la direction opposée, à partir du même point ; β = 0.4 par exemple)

♦ et ainsi de suite jusqu'à ce que les critères d'arrêt soient satisfaits.

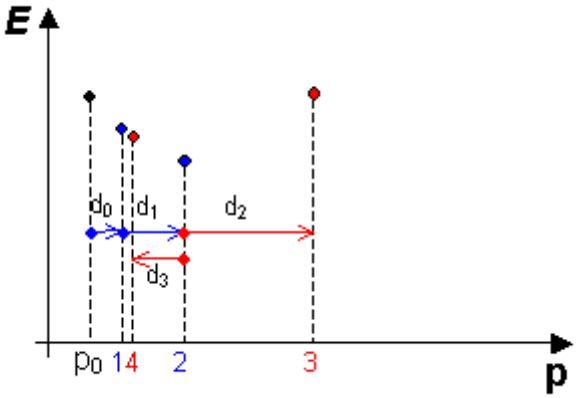


Fig. 2 Méthode de succès-échecs

Les points bleus (1 et 2) représentent des succès, et les points rouges (3 et 4) des échecs (α = 2; β = 0.4).

Les critères d'arrêt des itérations peuvent être : erreur résiduelle inférieure à une valeur donnée, variation de l'erreur résiduelle d'un point au suivant inférieure à une valeur donnée, variation du paramètre inférieure à une valeur donnée, etc. ou une combinaison de ces différents critères.

Cet algorithme est **stable sur un domaine infini**, grâce à son mécanisme d'expansion (en cas de succès) et de contraction (en cas d'échec). En fait, deux échecs cernent un minimum, réduisant ainsi automatiquement, et rapidement, le domaine à explorer. De plus, il n'est pas nécessaire que la fonction soit continue et différentiable, il suffit qu'elle soit *unimodale*, c'est-à-dire qu'elle possède un minimum local unique dans le domaine considéré.

Par contre il peut être peu efficace à proximité du minimum. On peut alors l'améliorer en le couplant avec l'interpolation quadratique lorsque cette proximité est atteinte.

Interpolation quadratique

Si la fonction d'erreur est continue et dérivable jusqu'à un ordre quelconque, ce que l'on peut supposer en général sur un petit domaine, elle peut être développée en série de Taylor :

$$E(p) = E(p_0) + \partial E/\partial p_{|p0}\,(p - p_0) + 0.5\,\partial^2 E/\partial p^2{}_{|p0}\,(p - p_0)^2 + \ldots \qquad (4)$$

Près d'un minimum, sauf cas particulier comme un point de rebroussement, on peut donc négliger les termes de degré supérieurs à 2 dans (4) et approximer $E(p)$ par une parabole.

Une telle parabole, Q, peut être déterminée par les 3 points P_1, P_2 et P_3 (les 3 derniers points obtenus par la méthode de succès-échecs, par exemple).

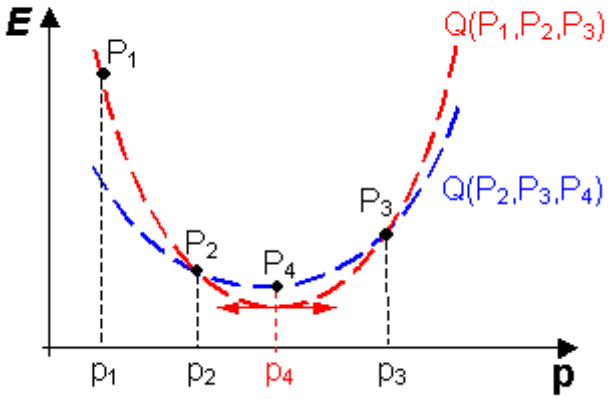


Fig. 3 Interpolation quadratique

On pourrait aussi utiliser 2 points et une dérivée, ou encore un seul point et les dérivées première et seconde.

La parabole $Q(P_1,P_2,P_3)$ étant déterminée, on calcule la position du minimum, p_4, puis la valeur réelle de l'erreur en ce point, $E(p_4)$. Ce nouveau point P_4 est substitué au point le plus éloigné parmi les trois précédents, soit P_1. Et l'opération est répétée avec les trois nouveaux points P_2, P_3 et P_4, et ainsi de suite jusqu'à la satisfaction des critères d'arrêt.

Utilisée près d'un minimum, comme indiqué ici, cette méthode est stable en général et converge très rapidement. Elle suppose que la fonction soit continue dans le domaine défini par les trois points considérés, ce qui est très généralement le cas, mais peut poser problème dans le cas inverse.

Toutefois, sa stabilité n'est pas assurée d'une façon générale dans l'ensemble du domaine, en particulier en présence d'un maximum, ou dans le cas de points alignés, par exemple. Elle peut donner dans certains cas des solutions qui oscillent indéfiniment entre deux valeurs.

2.2 Minimisation multi-dimensionnelle

Quelques rappels

A. Gradient

L'équivalent de la dérivée pour une fonction *E* de plusieurs variables (vecteur ***p***) est le *gradient*, c'est-à-dire le vecteur de ses dérivées partielles :

$$\boldsymbol{g} = \nabla E = \{\partial E/\partial p_i\} \quad (\mathrm{i} = 1 \text{ à } \mathrm{n})$$

qui indique la *plus grande pente* et sa *direction* en un point donné.

Pour calculer le gradient en un point $\boldsymbol{p_0}$, il n'y a en général pas d'autres possibilité qu'une méthode d'*accroissements finis*, successivement sur chaque variable p_i :

$$\partial E/\partial p_{i|\boldsymbol{p0}} \approx \left(E(\boldsymbol{p_0} + \mathrm{d_i}) - E(\boldsymbol{p_0})\right) / \mathrm{d_i} \quad (\mathrm{i} = 1 \text{ à } \mathrm{n}) \quad (5)$$

où $\mathrm{d_i}$ désigne un accroissement effectué dans la direction i, le plus petit possible.

La taille de $\mathrm{d_i}$ est en réalité limitée par la précision du calculateur. L'erreur commise est de l'ordre de $\delta \approx 0.5\ \mathrm{d_i}\ \partial^2 E/\partial p_i^{\,2}{}_{|\boldsymbol{p0}}$ et il faut n+1 appels de la fonction d'erreur pour obtenir le gradient en un point, ce qui peut devenir très vite un coût considérable en temps de calcul (en fait, le coût est n le plus souvent car $E(\boldsymbol{p_0})$ doit être calculé de toute façon). C'est la raison pour laquelle on choisira de préférence un algorithme sans calcul de gradient, ou avec le calcul de ce dernier sur un minimum de points. Toutefois, pour bien comprendre le fonctionnement et l'intérêt d'un tel algorithme, il sera nécessaire d'évoquer aussi les méthodes avec calcul de gradient.

B. Développement en série, matrice hessienne, matrice de covariance

Le développement en série de Taylor *(4)* se généralise pour $E(\boldsymbol{p})$:

$$E(\boldsymbol{p}) = E(\boldsymbol{p_0}) + \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{p} - \boldsymbol{p_0}) + 0.5\ (\boldsymbol{p} - \boldsymbol{p_0})^{\mathrm{T}}\ \boldsymbol{H}\ (\boldsymbol{p} - \boldsymbol{p_0}) + \ldots \quad (6)$$

où ***H*** est la matrice des dérivées partielles secondes ou *matrice hessienne* :

$$\boldsymbol{H} = \left[\partial^2 E/\partial p_i \partial p_j\right] \quad (\mathrm{i}, \mathrm{j} = 1 \text{ à } \mathrm{n})$$

g et ***H*** sont calculés en $\boldsymbol{p_0}$; $\boldsymbol{g}^{\mathrm{T}}$ et $(\boldsymbol{p} - \boldsymbol{p_0})^{\mathrm{T}}$ désignent les vecteurs transposés.

1) La matrice hessienne, $\boldsymbol{H}$, est symétrique. Elle représente la courbure multidirectionnelle de la fonction d'erreur. Elle est presque constante sur une très petite région (rigoureusement constante si la fonction d'erreur est quadratique).

2) On peut utiliser une formule plus précise pour le calcul du gradient, en effectuant des accroissements symétriques :

$$\partial E/\partial p_{i|\boldsymbol{p0}} \approx (E(\boldsymbol{p_0} + d_i) - E(\boldsymbol{p_0} - d_i)) / (2\, d_i) \quad (i = 1 \text{ à } n) \quad (7)$$

l'erreur commise est plus petite, mais cela demande cette fois 2n appels de la fonction d'erreur ! Toutefois, cette méthode peut être acceptable si l'on désire calculer également la matrice hessienne, car elle en donne presque gratuitement la diagonale :

$$\partial^2 E/\partial p^2_{i|\boldsymbol{p0}} \approx (E(\boldsymbol{p_0} + d_i) + E(\boldsymbol{p_0} - d_i) - 2E(\boldsymbol{p_0})) / d_i^2 \quad (i = 1 \text{ à } n) \quad (8)$$

L'équation *(6)* montre que l'on peut approximer la fonction, sur un petit domaine, par une fonction quadratique de n variables, analogue de la parabole uni-dimensionnelle (Fig. 3). La méthode de minimisation dite de Newton consiste en interpolations quadratiques multi-dimensionnelles. A partir d'un point P_0 défini par $\{\boldsymbol{p_0}, \boldsymbol{g_0}, \boldsymbol{H_0}\}$, le minimum quadratique (analogue de p_4 sur la Fig. 3) s'atteint directement :

$$\boldsymbol{p}^* = \boldsymbol{p_0} - \boldsymbol{H_0}^{-1}\boldsymbol{g_0} \quad (9)$$

L'équation *(9)* nécessite donc le calcul de la matrice hessienne et son inversion, soit des calculs relativement coûteux. La matrice $\boldsymbol{V} = \boldsymbol{H}^{-1}$, près du minimum, est appelée la *matrice de covariance* (des paramètres optimisés).

C. Directions de déplacement

Nous avons utilisé sans autre précision la notion de *directions* dans l'espace des paramètres à optimiser, $\boldsymbol{p}$. Ces derniers étant supposés être des variables indépendantes, leurs vecteurs directeurs sont autant de directions, également indépendantes. Il s'agit d'ailleurs en général des *directions initiales*, utilisées au démarrage de la minimisation. Cependant, toute combinaison linéaire de ces vecteurs constitue également une *direction*, suivant laquelle il est possible d'effectuer un *déplacement*, et nous verrons que l'essentiel du problème de minimisation multidimensionnelle consiste à trouver les **meilleures directions de déplacement**.

Méthode de relaxation, sans gradient

La méthode sans calcul de dérivées la plus simple consiste à opérer des minimisations uni-dimensionnelles successivement suivant chacune des directions initiales, qui sont donc orthogonales. L'opération est réitérée jusqu'à l'obtention de la précision désirée, sans jamais changer ces directions. Le pas de déplacement d'un point au suivant n'est pas prédéterminé : il dépend de la minimisation uni-dimensionnelle.

Cette méthode produit une trajectoire en zig-zag, peu efficace près du minimum, particulièrement dans une topologie du type vallée, comme suggéré ci-dessous pour n = 2.

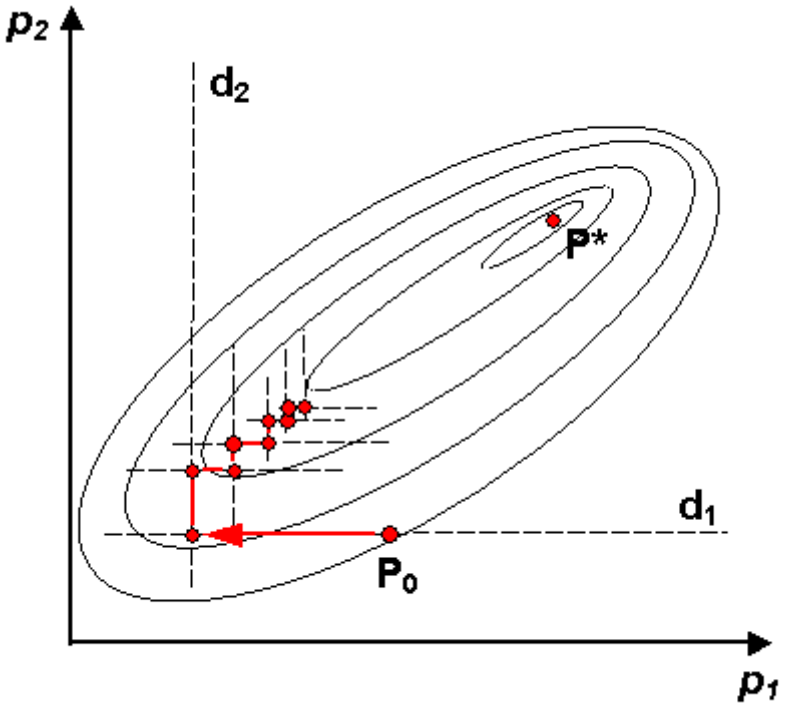


Fig. 4 Méthode de relaxation

(n = 2)
Partant de **P_0**, on minimise suivant la direction d_1 en maintenant p_2 constant , puis suivant la direction d_2 en maintenant p_1 constant, et ainsi de suite jusqu'à se rapprocher suffisamment du minimum P*.

Méthode de descente suivant le gradient

Cette méthode est dite aussi de Cauchy, ou de *descente de plus forte pente* (steepest descent). Elle consiste en effet à effectuer des minimisations uni-dimensionnelles successives suivant les directions de plus forte pente, données par le calcul du gradient, ***g***, en chacun des points P_0, P_1, P_2, etc. La descente est en général rapide au début, mais si la fonction d'erreur est quadratique, ce qui tend presque toujours à être le cas près du minimum, la procédure génère des directions orthogonales et devient donc équivalente à la méthode de relaxation.

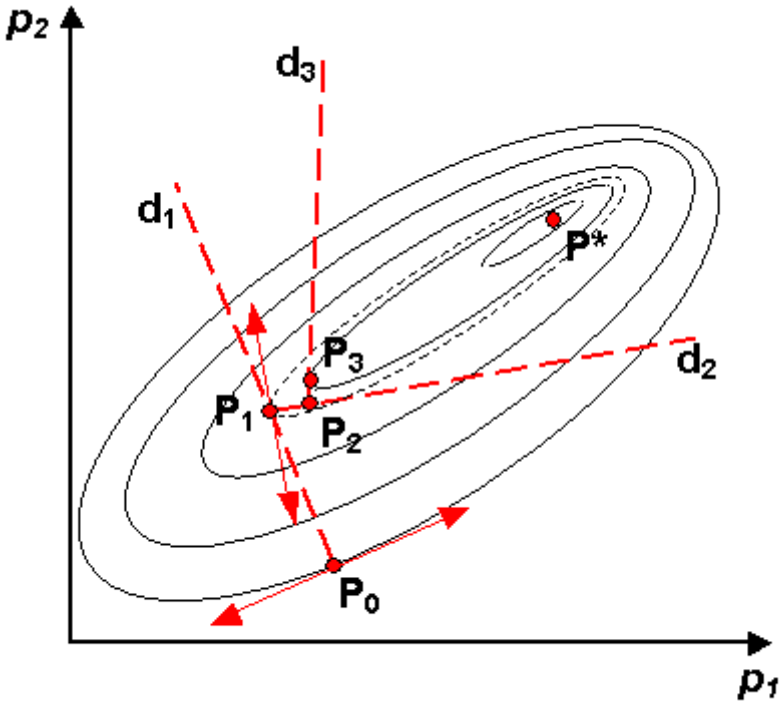


Fig.5 Descente suivant le gradient

(n = 2) On minimise suivant la direction d_1 donnée par le gradient en P_0, puis suivant la direction d_2 du gradient en P_1, et ainsi de suite. Les directions successives tendent à devenir orthogonales à mesure qu'on s'approche du minimum.

Méthode de gradient conjugué

Rappelons que deux vecteurs $\mathbf{d_i}$ et $\mathbf{d_j}$ sont *orthogonaux* si $\mathbf{d_i}^T\mathbf{d_j} = 0$, qui peut s'écrire encore :

$$\mathbf{d_i}^T\mathbf{I}\,\mathbf{d_j} = 0 \qquad (10)$$

$\mathbf{I}$ désignant la matrice unité.

De même, ils sont dits *conjugués* par rapport à la matrice carrée, symétrique définie positive, $\mathbf{A}$, si l'on a :

$$\mathbf{d_i}^T\mathbf{A}\,\mathbf{d_j} = 0 \qquad (11)$$

Sans rentrer dans le détail, les méthodes de *gradient conjugué* consistent à générer des directions successives de minimisation uni-dimensionnelle, $\mathbf{d_i}$, $\mathbf{d_{i+1}}$, mutuellement conjuguées par rapport à la matrice hessienne :

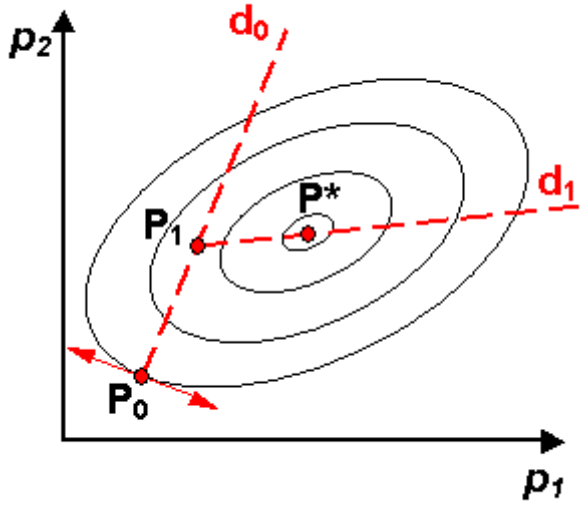


Fig. 6 Méthode de gradient conjugué

(n = 2, cas d'une fonction quadratique exacte)
d_0 est la direction du gradient en P_0, et P_1 le minimum sur cette direction. La direction d_1 est conjuguée de d_0. Le minimum sur d_1 est le minimum de la fonction, atteint donc en n = 2 coups.

$$\mathbf{d_{i+1}}^{T} \boldsymbol{H}\, \mathbf{d_i} = 0 \qquad (12)$$

La première direction est celle du gradient au point de départ P_0. Il est en général nécessaire de réinitialiser périodiquement suivant le gradient pour assurer la convergence globale.

Dans le cas d'une fonction quadratique exacte, on montre que le minimum est atteint en **n itérations**. Il faut, naturellement, davantage d'itérations pour une fonction quelconque, mais le nombre de points à calculer reste limité et cela demande moins de calculs que la méthode de Newton. Surtout, cette méthode, comparée aux méthodes qui utilisent ou génèrent des directions orthogonales, ne "traîne" pas au voisinage de minimum.

L'efficacité de cette méthode tient au fait que le conjugué, pourrait-on dire, corrige ce qui est faux (incomplet) dans la direction du gradient.

Toutefois, elle fait appel à la différentiation, par une méthode d'accroissements finis, puisque l'on doit calculer les gradients (le calcul de la matrice hessienne n'est en réalité pas nécessaire dans l'algorithme de Fletcher - Reeves - Powell). Cela suppose, comme on l'a vu, de nombreux appels de la fonction et pose évidemment un problème si celle-ci n'est pas partout différentiable.

Algorithme de Powell

Il s'agit d'un algorithme subtil, mis au point pour remédier aux défauts précédents :

(1) départ de $\mathbf{p_0}$

minimiser uni-dimensionnellement dans chacune des n directions indépendantes (initialement celles des axes de coordonnées), en repartant à chaque fois du dernier point obtenu ; on obtient donc une suite de vecteurs : $\mathbf{p_1}$, $\mathbf{p_2}$, …, $\mathbf{p_n}$ (les indices se réfèrent aux directions de minimisation et non aux numéros des paramètres)

ces calculs permettent de retenir au passage :

• $\mathbf{d_g} = \mathbf{p_n} - \mathbf{p_0}$: direction globale de déplacement du cycle

• $\mathbf{d_m}$: meilleure direction, suivant laquelle le gain, $\Delta_m = E(\mathbf{p_{m-1}}) - E(\mathbf{p_m})$, est le plus grand

• $E_0 = E(\mathbf{p_0}), E_1 = E(\mathbf{p_1}), \ldots, E_n = E(\mathbf{p_n})$ avec $E_0 \geq E_1 \geq \ldots \geq E_n$

calculer, de plus, l'erreur au symétrique $\mathbf{p_q}$ de $\mathbf{p_n}$ par rapport à $\mathbf{p_0}$:

$E_q = E(2\mathbf{p_n} - \mathbf{p_0})$

(2) substitution de $\mathbf{d_g}$ à $\mathbf{d_m}$ **?**

le plus souvent, en particulier lorsque la fonction d'erreur est quadratique, la direction $\mathbf{d_g}$ est une meilleure direction de descente que la meilleure du cycle précédent, $\mathbf{d_m}$, et peut donc avantageusement lui être substituée. Mais ce n'est pas obligatoirement le cas et les test suivants permettent d'en décider :

si $E_q < E_0$ **et** $(E_0 - 2E_n + E_q)(E_0 - E_n - \Delta_m)^2 < 0.5\,\Delta_m\,(E_0 - E_q)^2$

- nouveau $\mathbf{p_0}$ = le minimum dans la direction $\mathbf{d_g}$
- nouvel ensemble de directions : sauter $\mathbf{d_m}$ et ajouter $\mathbf{d_g}$ à la fin, soit
  $\mathbf{d_1}$, $\mathbf{d_2}$, …, $\mathbf{d_{m-1}}$, $\mathbf{d_{m+1}}$, …, $\mathbf{d_n}$, $\mathbf{d_g}$
- retour à **(1)**

sinon

- nouveau $\mathbf{p_0} = \mathbf{p_n}$
- retour à **(1)** (directions inchangées)

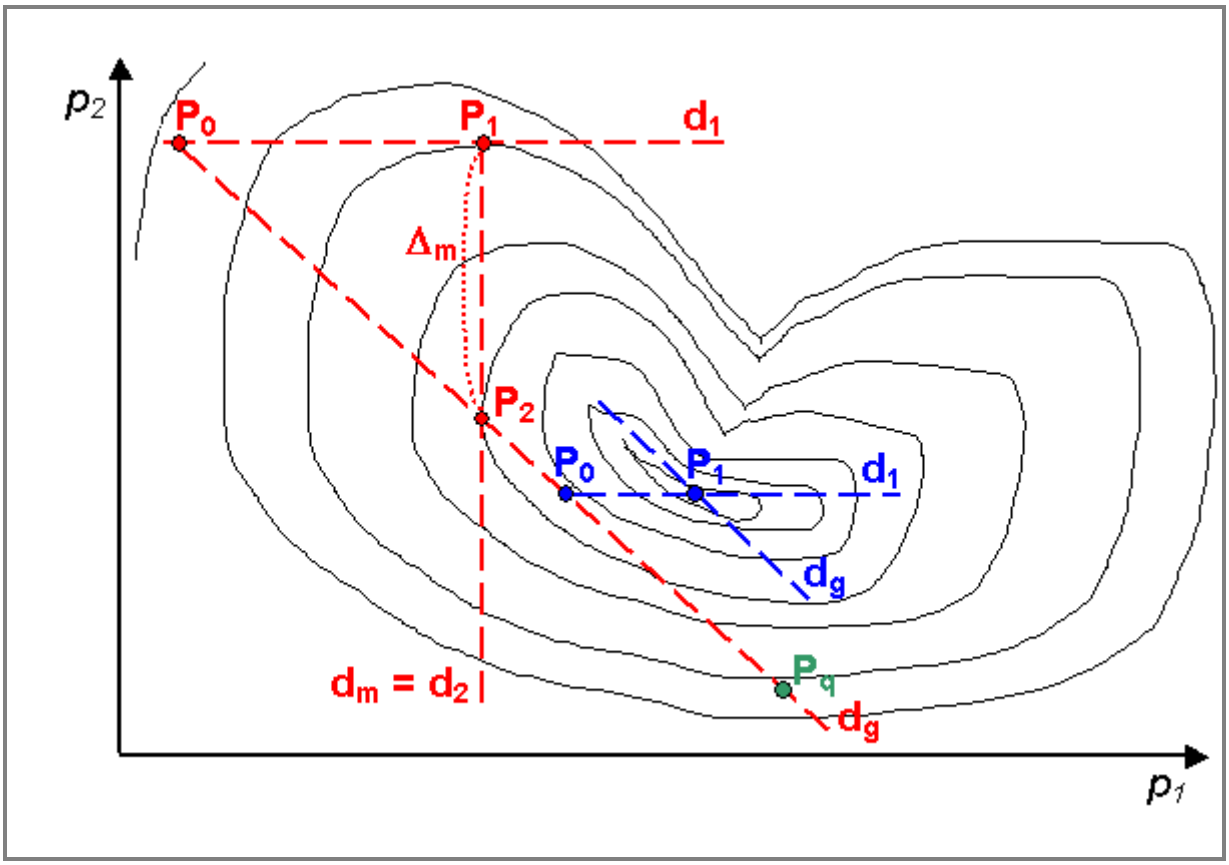


Fig. 7 Trajectoire engendrée par l'algorithme de Powell

(n = 2, fonction d'erreur quelconque)
Le premier cycle, en rouge, est effectué avec les directions initiales des axes de coordonnées, orthogonales. Le meilleur gain Δ_m se trouve sur la direction $\mathbf{d_2}$ (saut de 2 courbes de niveau, un peu moins sur $\mathbf{d_1}$). On suppose les conditions $E_q < E_0$ et $(E_0 - 2E_n + E_q)(E_0 - E_n - \Delta_m)^2 < 0.5\,\Delta_m\,(E_0 - E_q)^2$ satisfaites et on repart pour un deuxième cycle, en bleu, à partir du minimum sur la direction $\mathbf{d_g}$ qui remplacera $\mathbf{d_2}$. Seul le premier point de ce deuxième cycle est indiqué. Les directions $\mathbf{d_1}$ et $\mathbf{d_g}$ sont "quasi conjuguées" (elles le seraient si la fonction était quadratique) et donnent lieu à des déplacements très efficaces.

On peut montrer qu'appliqué à une fonction quadratique, cet algorithme génère progressivement des directions mutuellement conjuguées. Il est stable et sa convergence asymptotique est superlinéaire comme toutes les méthodes de directions conjuguées.

Naturellement, ces propriétés ne sont pas garanties dans le cas d'une fonction quelconque, mais il reste très efficace d'une manière général. De plus, il ne comporte aucune différentiation, donc relativement peu d'appels de la fonction d'erreur, et est utilisable même pour des fonctions non différentiables.

Par contre, les algorithmes de ce type sont très sensibles à l'ordre des directions initiales. C'est pourquoi il est très utile de pouvoir les modifier.

L'optimisateur ***RP*** (Rosenbrock - Powell) de *Sa* est une implémentation de l'algorithme de Powell ci-dessus, avec utilisation de la méthode de succès-échecs de Rosenbrock pour les minimisations uni-dimensionnelles. Il permet, de plus, et nécessite, de préciser des bornes inférieures et supérieures pour chaque paramètre optimisé. Cela peut être très utile en particulier, et c'est sa raison d'être, dans le cas où des valeurs de paramètres en dehors de ces limites provoquent des erreurs de calcul, car l'optimisateur, sans cela, est libre d'amener les paramètres où il veut. Il est recommandé dans ce cas de terminer l'optimisation avec *va04a*, une fois un point proche du minimum obtenu avec *RP*.

Quant à l'optimisateur ***va04a*** de *Sa*, utilisé par défaut et recommandé en général, il s'agit également d'un algorithme de type Powell, fortement amélioré par Patrick André (CICT, Toulouse).

Quelques aspects de la marche vers le minimum

Afin d'illustrer un peu les difficultés que peut rencontrer un optimisateur, nous allons examiner en détail un problème réel. Choisissons pour cela un problème extrêmement simple, mais très courant :

• Ajuster les données cinétiques d'une réaction de type $A \leftrightarrow B$, réversible ou non, suivie par une grandeur *y* proportionnelle aux concentrations :

Le modèle est donc la fonction exponentielle

$$y = y_\infty + (y_0 - y_\infty)\, e^{-kt} \qquad (13)$$

qui comprend
- un paramètre cinétique, la constante de vitesse k
- deux paramètres d'amplitude : les valeurs initiales et finales, y_0 et y_∞ respectivement.

Les données sont un ensemble de points (t_i, Y_i) (i = 1 à m) et la fonction d'erreur est

$$E\,(y_0, y_\infty, k) = \sum_{i=1 \text{ à } m} \left(Y_i - y_\infty - (y_0 - y_\infty)\, e^{-k\, t_i} \right)^2 / m \qquad (14)$$

Nous utiliserons en réalité des données simulées à l'aide de l'équation *(13)*, avec
k = 0.1 (s^{-1})
y_0 = 1
y_∞ = 0.25
et auxquelles on pourra éventuellement ajouter du bruit.

Voisinage proche du minimum

Fixons d'abord y_0 à sa valeur, 1 (c'est en fait la situation très courante où la valeur initiale est connue). L'erreur est alors une fonction de k et y_∞ seulement et a la forme de la Fig. 8.

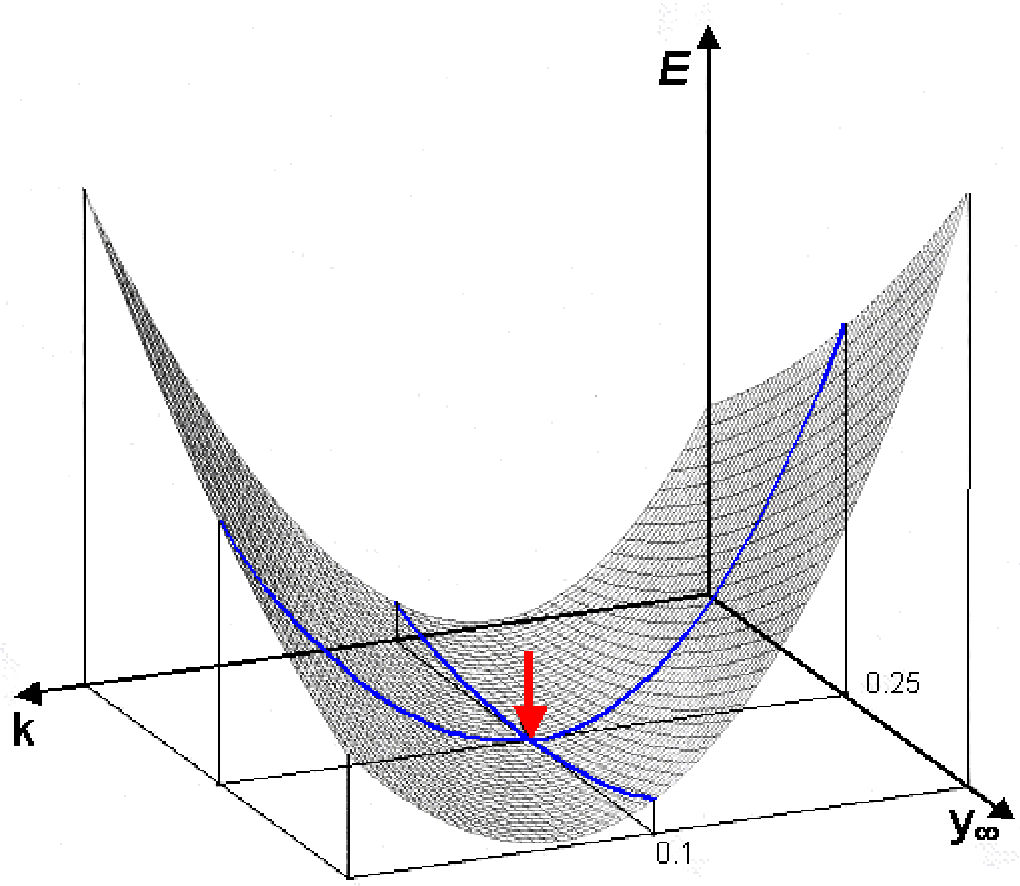


Fig. 8 Ajustement de k et y_∞ (y_0 = 1, fixé)

La flèche rouge indique la position du minimum. Les courbes en bleu représentent la variation de l'erreur en fonction d'un des deux paramètres, l'autre étant fixé à sa valeur du minimum. (données non-bruitées)

Nous constatons d'abord que, proche du minimum, la fonction d'erreur est bien une fonction quasi quadratique des deux paramètres.

Mais surtout, la *sensibilité de la fonction d'erreur* est très différente selon la direction : beaucoup plus sensible dans la direction k que dans la direction y_∞ (cette différence serait d'ailleurs encore plus frappante si les échelles étaient les mêmes sur ces deux axes). Cette forme de paysage en ***vallée*** correspond à la nature très différente de ces deux paramètres et signifie qu'il sera plus difficile d'obtenir la même précision relative sur y_∞ que sur k. Noter que les courbes en bleu correspondent exactement à ce qui est fait par l'outil "sensibilité globale" de *Sa*.

Fixons maintenant k à sa valeur, 0.1, et ajustons les deux paramètres d'amplitude y_0 et y_∞. L'erreur se présente maintenant comme sur la Fig. 9.

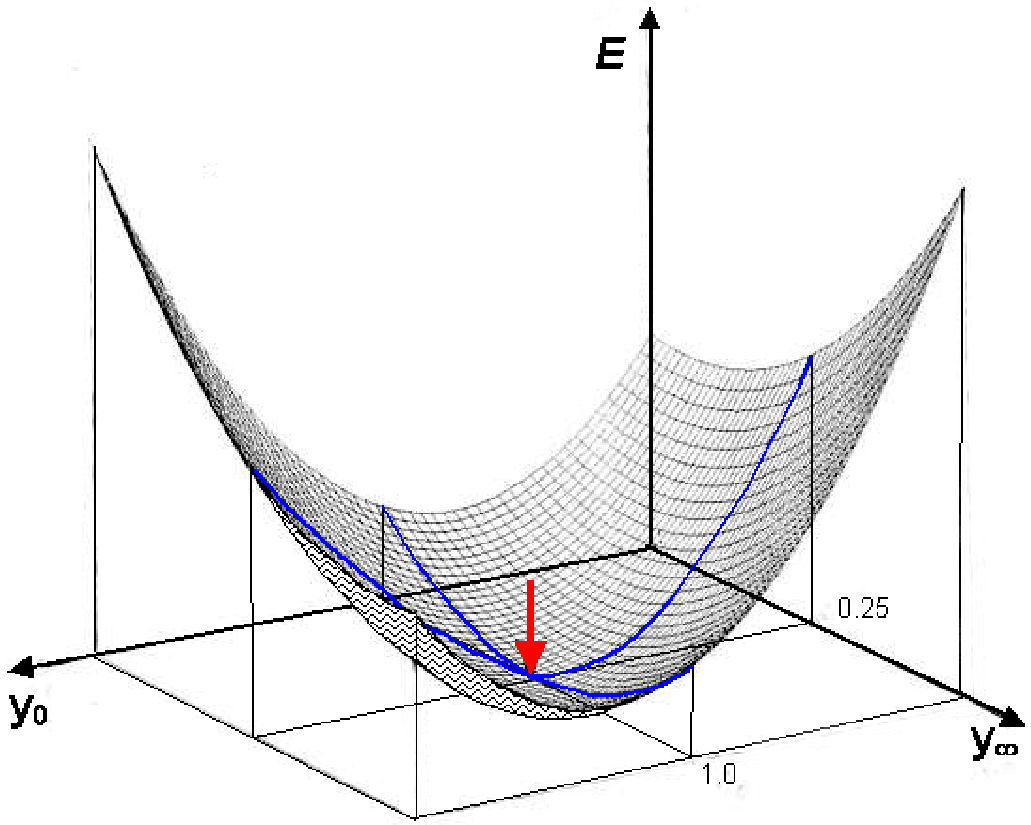


Fig. 9 Ajustement de y_0 et y_∞ (k = 0.1, fixé)

La flèche rouge indique la position du minimum. Les courbes en bleu représentent la variation de l'erreur en fonction d'un des deux paramètres, l'autre étant fixé à sa valeur du minimum.. (données non-bruitées)

Nous constatons cette fois que les sensibilités de l'erreur en fonction de ces deux paramètres d'amplitude sont voisines, et le paysage n'apparaît plus comme une vallée, mais comme une cuvette.

Un peu plus loin du minimum

Revenons maintenant à la situation, plus réaliste, où y_0 est fixé à sa valeur, comme sur la Fig. 8, mais éloignons-nous un peu plus du minimum. Le paysage devient alors celui de la Fig. 10.

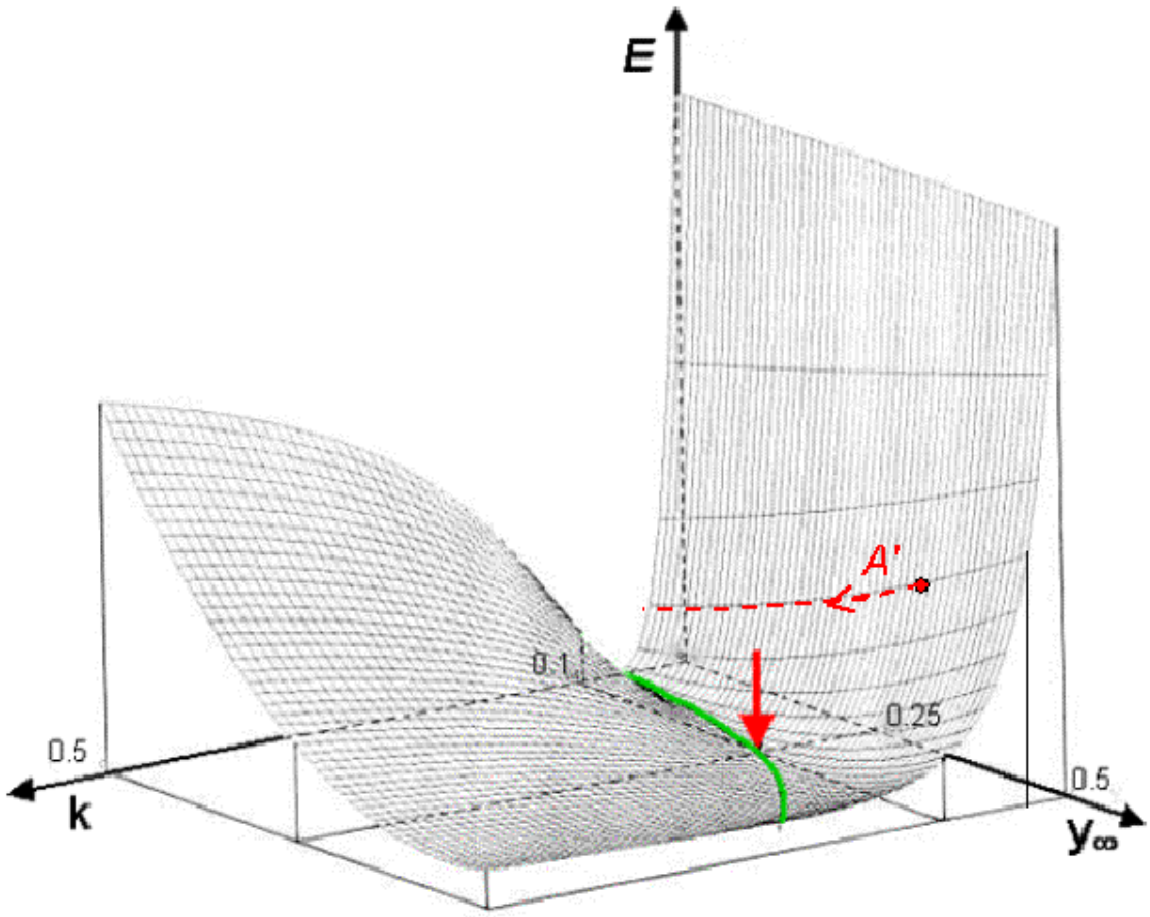


Fig. 10 Ajustement de k et y_∞ (y_0 = 1, fixé)

La flèche rouge indique la position du minimum. La courbe en vert suit le fond de la vallée.
La trajectoire *A'* correspond à une minimisation dans la direction y_∞ à partir du point k = 0.03, y_∞ = 0.3 (voir plus bas)
(données non-bruitées)

Nous voyons ici que l'ajustement d'un modèle aussi simple qu'une exponentielle, et de deux paramètres seulement, conduit à une forme de la fonction d'erreur plus compliquée qu'on ne pouvait l'imaginer : la vallée loin d'être droite, tourne et, de plus, s'élargit sur un de ses flans. Ainsi, on peut voir que si l'on s'éloigne encore du minimum, vers l'avant et la gauche de la figure, on va trouver des régions quasiment plates, d'où le cheminement vers le minimum sera de plus en plus difficile.

Trajectoires de minimisation

La Fig. 11 montre quelques trajectoires de minimisation réelles, dans différentes situations de départ.

Remarquons d'abord que les ***itérations*** (points numérotés) correspondent à **un cycle complet sur toutes les directions**, et non à chacune des minimisations uni-dimensionnelles, bien que chacune de celles-ci aboutisse à un point de plus faible

erreur. La raison en est que les tests permettant de déclarer l'optimisation terminée doivent porter naturellement sur tous les paramètres optimisés, et par conséquent n'être effectués qu'à la fin d'un cycle complet.

Dans quatre des situations envisagées, le nombre d'itérations est 3 (2 itérations intermédiaires). Il est de 4 pour le trajet **C** et 5, seulement , pour le trajet *A'*. Ces nombres pourraient être plus grands pour des modèles plus complexes, comportant plus de paramètres à ajuster. Mais on peut observer ici que l'algorithme utilisé (*va04a*) est efficace dès le départ, et jusqu'au bout, près du minimum, ce qui est une qualité d'un grand intérêt pratique.

La comparaison des trajets en lignes continues avec leurs correspondants en tirets montre l'influence importante de **l'ordre des directions initiales**, dans les algorithmes de type Powell. Le minimum est malgré tout atteint, ici, dans tous les cas, et en peu d'itérations. On peut facilement imaginer cependant, à la vue de ces trajectoires, que dans certaines situations un mauvais ordre des directions initiales peut conduire soit à une trajectoire très longue, soit à rester piégé dans un faux minimum, s'il en existe.

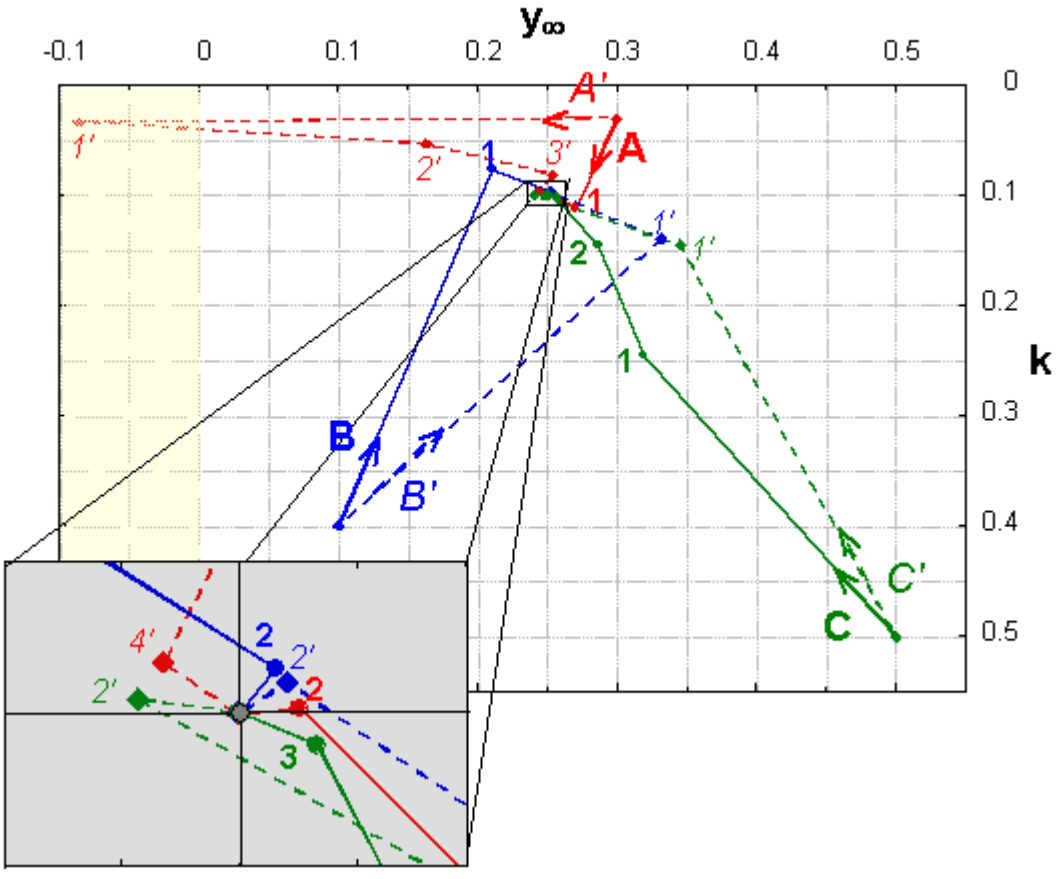


Fig. 11 Trajectoires de minimisation de k et y_∞ (y_0 = 1, fixé)

A partir des points (k = 0.03 ; y_∞ = 0.3), (k = 0.4 ; y_∞ = 0.1) et (k = 0.5 ; y_∞ = 0.5), respectivement, les trajets **A**, **B** et **C** sont effectués avec l'ordre des directions initiales {k, y_∞}, et les trajets *A'*, *B'* et *C'* l'ordre {y_∞, k}. Les points correspondant à chaque itération sont numérotés (sauf les derniers, au minimum).(optimisations effectuées par *va04a*, données non-bruitées)

La trajectoire *A'* mérite quelques commentaires particuliers : partant relativement près du minimum, mais avec y_∞ comme première direction initiale, le point correspondant à la première itération "s'éloigne" nettement du minimum (point *1'* rouge), dans l'espace {k, y_∞} (mais l'erreur y est tout de même plus faible qu'au départ), pour adopter ensuite une trajectoire d'allure plus intuitive. Cet éloignement initial vient de la première minimisation uni-dimensionnelle, effectuée dans la direction y_∞ à k constant, qui doit aller chercher très loin un point de moindre erreur, comme on peut le comprendre en situant ce point sur la Fig. 10 (trajectoire A').

Mais ce n'est pas tout : ce point "éloigné" se situe à une valeur négative de y_∞. Or une telle valeur n'aurait en général aucune signification physique. Il faut distinguer en réalité la démarche mathématique d'optimisation, dans laquelle il convient de laisser de la liberté aux paramètres, au risque, sinon, de gêner la procédure, et l'examen final des paramètres obtenus, qui doivent évidemment avoir un sens physiquement. En toute rigueur, on ne devrait imposer de contraintes (limites) aux paramètres ajustables, que lorsque le non respect de ces contraintes risque de provoquer des erreurs de calculs non gérables. En pratique, toutefois, il est le plus souvent sans inconvénient d'obliger certains paramètres à rester positifs, par exemple. Mais il ne faut pas perdre de vue que de telles contraintes peuvent parfois gêner l'optimisation.

En partant du point (0.5, 0.5), relativement éloigné du minimum , c'est par contre la trajectoire *C'* (y_∞ comme première direction initiale) qui s'avère plus directe que **C**.

Si l'on partait de plus loin encore, par exemple de k = 10 et y_∞ = 0.4, l'optimisation échouerait, avec le message "l'erreur ne varie plus". C'est qu'on serait alors sur une sorte de "plateau" de la fonction d'erreur, alors insensible aux variations des paramètres. En pratique, on évite le plus possible de telles situations en s'approchant manuellement du minimum, avec des valeurs physiquement réalistes des paramètres (voir exercice 16), mais il peut toujours y avoir dans le modèle des paramètres insensibles, qu'il convient alors soit de fixer, soit de ramener dans un domaine sensible.

Lors d'une optimisation avec *Sa*, le détail des itérations, trajectoire et valeurs de la fonction d'erreur, est automatiquement enregistré dans le fichier d'extension *.sar* (voir exercice 16).

Influence d'un paramètre fixé faux

Il est courant de fixer certains paramètres, connus ou supposés tels, d'un modèle tandis que les autres sont ajustés. C'est même parfaitement recommandable en général, mais cela impose en contrepartie qu'ils soient corrects. En effet, fixer un paramètre à une valeur fausse peut avoir un effet important sur la position du minimum.

Ainsi, par exemple, une erreur de 5% sur la valeur imposée de y_0 (y_0 = 0.95 ou 1.05) déplace le minimum de 2% sur y_∞ et de presque 10% sur k (Fig. 12). Ces chiffres dépendent de la courbure de la fonction d'erreur dans ces directions. Ici, une erreur sur y_0 est synonyme d'erreur sur une valeur initiale (concentration, mesure

d'absorbance...). C'est une situation très fréquente, l'opérateur se montrant très sûr de ses valeurs initiales ! Une autre situation est celle où un paramètre est connu par ailleurs (littérature, mesure indépendante...).

Dans l'exemple présent, et c'est souvent le cas, ce déplacement du minimum se traduit par une erreur résiduelle importante et, visuellement, un mauvais ajustement. Mais il peut se faire que l'ajustement reste relativement correct, les paramètres mobiles compensant partiellement les paramètres fixés faux.

D'un autre côté, laisser tous les paramètres libres dès le départ d'un ajustement, c'est prendre le risque de s'égarer dans de fausses solutions.

Pour éviter, ou du moins apprécier, ce genre d'erreur, nous préconisons la stratégie suivante, même dans le cas où le premier ajustement s'avère satisfaisant :

(1) ajuster en fixant les paramètres connus

(2) à partir de la solution obtenue, ré-ajuster en libérant ces paramètres (éventuellement un par un, suivant le problème). Noter les différences obtenues sur tous les paramètres entre ces deux ajustements. Apprécier si la variation des paramètres initialement fixés est dans leur marge d'erreur possible. Si oui, retenir cette solution.

(3) si ce n'est pas le cas, fixer ces paramètres à leur limite d'erreur admissible, dans le sens indiqué par leur ajustement en (2), et ré-ajuster les paramètres inconnus.
Apprécier la qualité du nouvel ajustement obtenu.

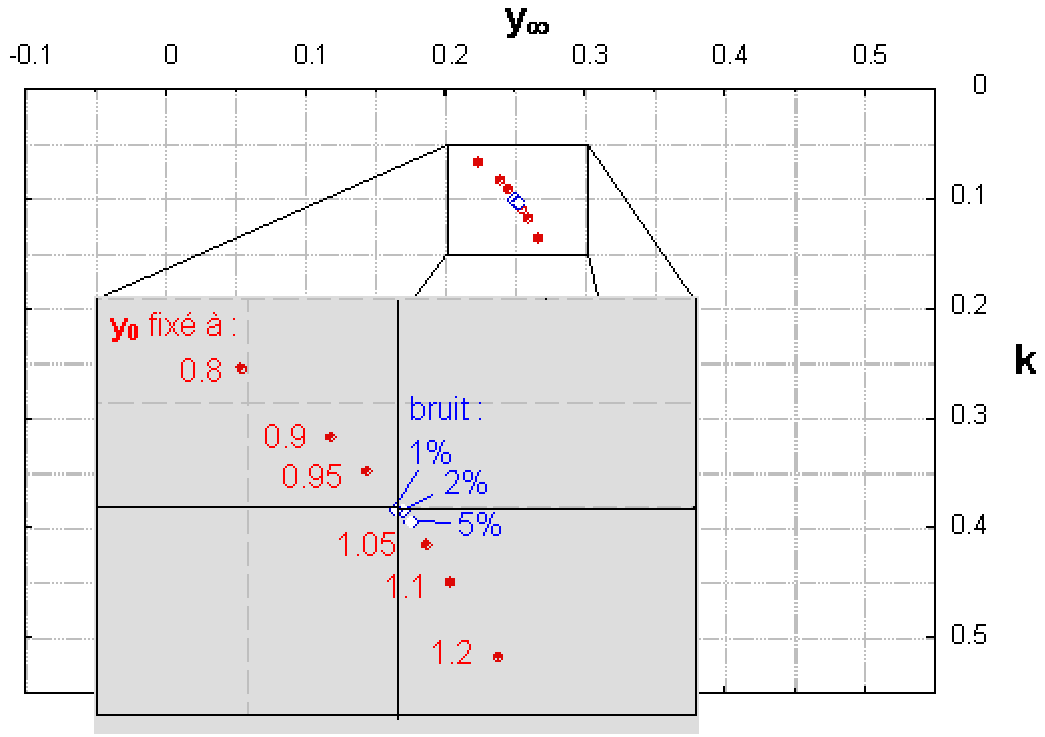


Fig. 12 Influence d'un paramètre fixé faux et du bruit expérimental

Points rouges : position du minimum atteint en fixant y_0 à une valeur fausse (données non-bruitées)
Diamants bleus : position du minimum atteint avec des données expérimentales bruitées ($y_0 = 1$; bruit gaussien sur 60 points)

Influence du bruit expérimental

Dans les minimisations ci-dessus, les données expérimentales étaient non-bruitées et les valeurs des paramètres aux minima obtenus étaient donc les valeurs attendues, à la précision numérique du calcul près. Il n'en va plus de même dans un ajustement sur des données réelles, car la présence de bruit n'a pas seulement pour effet d'augmenter l'erreur résiduelle, mais aussi de déplacer la position du minimum.

La Fig. 12 montre l'effet d'un bruit gaussien appliqué sur les 60 points servant à l'ajustement. Cet effet, bien que relativement limité, n'est pas négligeable. Il peut être, et est le plus souvent, beaucoup plus important, si le nombre de points est moins important et si le bruit n'est pas gaussien.

Cela ne veut surtout pas dire qu'il faudrait éliminer le bruit avant optimisation, par exemple par un lissage ! L'effet serait pire encore, car tout traitement, y compris un

lissage, introduit un *biais* dans les données. On peut d'ailleurs déplorer que beaucoup d'appareils de mesure moderne ne délivrent que des données pré-traitées... souvent simplement parce que c'est plus joli, et ne permettent pas d'accéder aux données brutes, toujours préférables pour un ajustement par un modèle physique.

Minimum vrai ou faux ?

Malgré toute les qualités des algorithmes utilisés, rien ne garantit que le minimum $\boldsymbol{p^*}$ trouvé soit le minimum global, ni même qu'il soit le minimum local le plus proche du point de départ $\boldsymbol{p_0}$. Il importe donc de savoir critiquer le minimum obtenu.

L'erreur résiduelle doit être de l'ordre attendu en fonction de l'erreur expérimentale. Si ce n'est pas le cas, la raison peut en être bien sûr que le modèle n'est pas entièrement correct. Mais elle peut être aussi que le minimum intéressant n'est pas atteint, autrement dit que le système est resté piégé dans un faux minimum, et cette hypothèse doit toujours être testée avant de rejeter le modèle.

Les valeurs des paramètres obtenues doivent être plausibles. Il est important également de regarder la sensibilité de la fonction d'erreur envers chaque paramètre.

Même si tout semble correct, on ne peut pas exclure qu'il existe d'autres minima, aussi valables. La section suivante donne quelques aspects de la recherche de tous les minima ou du minimum global. Mais plus simplement, on devrait toujours au moins tester qu'en partant de points de départ $\boldsymbol{p_0}$ différents, et/ou avec des ordres de directions initiales différents, on obtient, ou non, le même minimum.

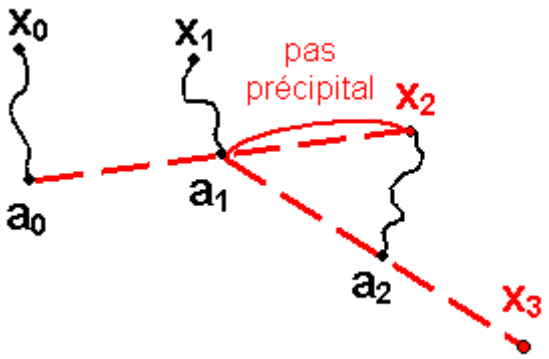
3. Recherche du minimum global

Nous ne donnerons ici qu'un aperçu très rapide de quelques méthodes qui peuvent être utilisées lorsqu'il existe plusieurs minima, soit pour les obtenir tous, soit pour rechercher le minimum global. Ces méthodes, en particulier le *recuit simulé*, peuvent être utilisées aussi pour sortir de, ou éviter, un "faux" minimum (qui ne convient pas physiquement), sans que le "vrai" soit obligatoirement le minimum global.

Une première méthode, dite de Monte-Carlo, consiste à tirer au hasard un certain nombre de points à partir desquels on effectue une minimisation locale. Si le nombre de points de départ est suffisamment grand, on peut espérer couvrir tous les bassins d'attraction de la fonction et atteindre ainsi tous les minima. Il est clair cependant que cela ne sera jamais totalement garanti, qu'il faudra un nombre rapidement prohibitif de points, et par conséquent de minimisations locales, au fur et à mesure que le nombre de paramètres augmente. La méthode, d'autre part, ne bénéficie d'aucune intelligence et conduit à rechercher plusieurs fois le même minimum si plusieurs points de départ sont dans un même bassin.

3.1 Course de précipice

L'idée en est l'opposé de voler de sommet en sommet. Il s'agit donc plutôt de creuser des tunnels de creux en creux.



L'algorithme de Gelfand est très grossièrement le suivant (illustré ci-contre) :

- on part de deux points différents $\mathbf{x_0}$ et $\mathbf{x_1}$ à partir desquels on suppose qu'on atteint (grossièrement) deux minima locaux distincts $\mathbf{a_0}$ et $\mathbf{a_1}$.
- dans la direction définie par $\mathbf{a_0}$ et $\mathbf{a_1}$, on effectue un ***pas précipital***, qui amène en $\mathbf{x_2}$
- depuis $\mathbf{x_2}$ on atteint un nouveau minimum local $\mathbf{a_2}$
- dans la direction définie par $\mathbf{a_1}$ et $\mathbf{a_2}$, on effectue un nouveau *pas précipital*, qui amène en $\mathbf{x_3}$... et ainsi de suite.

La méthode dépend principalement de la longueur du *pas précipital*, qui ne peut être choisi qu'au cas par cas par des essais. Elle est utilisée plutôt pour détecter la présence de minima que pour converger précisément vers leurs positions.

3.2 Méthode de Goldstein-Price

Cette méthode donne une idée d'une autre façon élégante de passer d'un minimum à un autre. Nous n'en donnons que la ligne générale. Il faut, pour pouvoir l'appliquer, que la fonction d'erreur soit dérivable et que les minima locaux ne soient pas singuliers, c'est-à-dire que le gradient y soit nul. Dans ces conditions, si $\mathbf{x_1}$ est un minimum, supposé déjà trouvé (trajectoire **(I)** sur la Fig. 13-A), on a $\boldsymbol{g}_{|\mathbf{x1}} = 0$ et le développement de la fonction (éq.6) devient

$$E(\boldsymbol{x}) = E(\mathbf{x_1}) + 0.5\,(\boldsymbol{x} - \mathbf{x_1})^{\mathrm{T}}\,\boldsymbol{H}\,(\boldsymbol{x} - \mathbf{x_1}) + t.o.s. \quad (15)$$

où le deuxième terme est positif et *t.o.s.* désigne l'ensemble des termes d'ordre 3 et supérieur, très importants puisque ce sont eux qui peuvent donner lieu à d'autres minima.

On construit une **nouvelle fonction** E_1 de $\mathbf{x_1}$ et $\boldsymbol{x}$:

$$E_1(\mathbf{x_1}, \boldsymbol{x}) \;=\; 2\left(E(\boldsymbol{x}) - E(\mathbf{x_1})\right) / \left((\boldsymbol{x} - \mathbf{x_1})^{\mathrm{T}}\, \boldsymbol{H}\, (\boldsymbol{x} - \mathbf{x_1})\right) \;=\; 1 + t.o.s. \qquad (16)$$

dont le signe est donné par la différence $E(\boldsymbol{x}) - E(\mathbf{x_1})$ et dans laquelle le minimum $\mathbf{x_1}$ a été comme "enlevé" : elle ne contient plus que les éventuels minima des *t.o.s.*.

On minimise alors E_1 à partir d'un point très voisin de $\mathbf{x_1}$ (trajectoire **(II)** sur la Fig. 13-A). (on ne peut pas partir précisément de $\mathbf{x_1}$, en effet, car la fonction E_1 n'est pas définie en $\mathbf{x_1}$)

Si E_1 devient < 0, c'est-à-dire $E(\boldsymbol{x}) < E(\mathbf{x_1})$, comme en $\mathbf{x'_1}$, cela indique que la barrière séparant deux minima est franchie, et on peut utiliser une position très voisine de $\mathbf{x'_1}$ pour relancer une minimisation de la fonction originelle $E(\boldsymbol{x})$ (trajectoire **(III)** sur la Fig. 13-A), qui en donnera un nouveau minimum, $\mathbf{x_2}$.

Si E_1 ne devient pas < 0, et qu'on trouve un minimum de E_1, $\mathbf{x''_1} > 0$, ce minimum peut correspondre à un nouveau bassin et servir de point de départ pour y rechercher un nouveau minimum local, plus haut cette fois, de $E(\boldsymbol{x})$. Cette situation est illustrée sur la Fig. 13-B où le premier minimum trouvé est supposé être $\mathbf{x_2}$, et la fonction E_1 est construite sur ce point.

L'opération peut bien sûr être réitérée à partir de ce nouveau minimum si nécessaire.

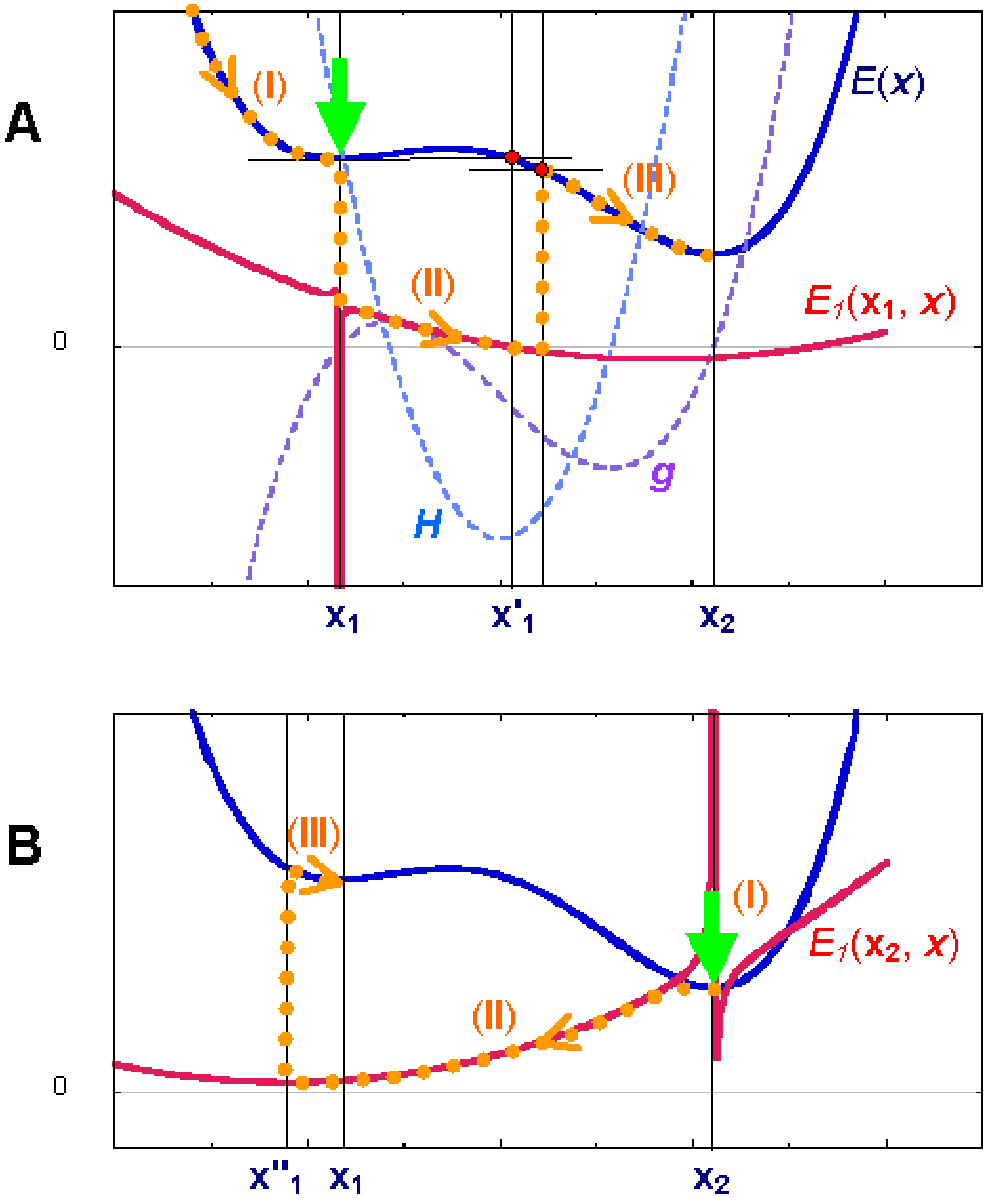


Fig. 13 Illustration uni-dimensionnelle de la méthode de Golstein-Price

Les flèches vertes indiquent le premier minimum connu, sur lequel on construit la fonction E_1.

A : situation où l'algorithme permet d'atteindre un meilleur minimum

B : situation où il permet d'atteindre un autre minimum, moins bon.

3.3 Recuit simulé

Concept

Les différents états possibles de la matière, prenons un métal par exemple, peuvent être représentés par leur énergie, en fonction des différents paramètres qui les caractérisent (distances entre les atomes, orientation, etc.). Chaque état correspond à un minimum local de l'énergie, et l'état le plus stable, mono-cristallin par exemple, correspond au minimum le plus bas.

A température élevée, les molécules ont toute liberté de mouvement. Elles la perdent au fur et à mesure d'un refroidissement. Mais, si le refroidissement a lieu **lentement**, elles peuvent atteindre le niveau d'énergie le plus bas, leur état le plus stable. Chauffer puis refroidir lentement un métal est ce qu'on appelle le *recuit* en métallurgie. A l'inverse, un refroidissement rapide, ou *trempe*, a pour effet de geler la matière dans l'état où elle se trouvait, sans relaxation possible.

L'analogie de la fonction énergie d'un système matériel avec une fonction d'erreur est évidente. Pour la rendre plus complète encore, il faut trouver une **variable *T* analogue à la température** qui caractérise la mobilité dans le paysage de la fonction d'erreur : il sera d'autant plus facile de sauter d'un minimum à un autre que la température sera élevée. L'idée du ***recuit simulé*** est de mimer la nature en opérant un abaissement progressif de cette variable *T*, un "refroidissement" lent, de façon à amener au minimum global de la fonction, ou du moins à un meilleur minimum.

A l'opposé de ce concept, tous les algorithmes de minimisation envisagés plus haut, visaient à descendre le plus vite possible et s'apparenteraient, de ce fait, plutôt à une *trempe*.

Algorithme de Metropolis

En amont du phénomène de recuit pour la matière, il y a la distribution de probabilité de Boltzmann :

$$\mathrm{P}(E) \sim \mathrm{e}^{-E/\mathrm{k}T} \qquad (17)$$

où k désigne la constante de Boltzmann, et qui exprime qu'un système en équilibre thermique à la température *T*, peut se trouver à tous les états d'énergie possible avec une certaine probabilité, d'autant plus faible que la température est basse et l'énergie élevée, mais jamais totalement nulle.

Un corollaire est qu'il existe toujours une certaine chance pour le système de "sortir" du minimum d'énergie local où il se trouve et d'atteindre un autre minimum. Autrement dit, des variations positives de l'énergie, des "remontées", sont également possibles, d'autant plus que la température est élevée.

On peut appliquer cette idée à une fonction quelconque $E(\boldsymbol{p})$ en attribuant une probabilité de déplacement du point $\mathbf{p_1}$ vers $\mathbf{p_2}$, fonction de $\Delta E = E(\mathbf{p_2}) - E(\mathbf{p_1})$ et d'une variable *ad hoc* T :

$$P = e^{-\Delta E/T} \qquad (18)$$

avec $P = 1$ si $\Delta E \leq 0$ (19) (le nouveau point $\mathbf{p_2}$ n'est pas plus haut que $\mathbf{p_1}$)

C'est-à-dire que l'**on autorise toujours une descente** et **quelquefois seulement une remontée**, moins souvent . D'une façon générale, cette démarche est l'algorithme de Metropolis (1953).

Minimisation par recuit simulé

Pour appliquer le concept du recuit simulé aux problèmes de minimisation qui nous intéressent, il faut adjoindre à l'expression *(18-19)* des mécanismes

- de déplacement dans l'espace des paramètres
- de contrôle et de variation de la "température" *T*
- de validation de l'acceptation ou du refus d'un déplacement .

Ces mécanismes doivent être adaptés aux types de problèmes traités. Nous n'évoquerons ici que ceux qui sont utilisés dans le *Recuit Simulé* de *Sa*, particulièrement adaptés au problèmes de cinétique.

Générateur de déplacements

Il s'agit de générer des déplacements aléatoires, des "pas", dans l'espace R^n des paramètres, de façon à évaluer le ΔE correspondant. En fait, il est indispensable de limiter cet espace à un certain pavé, déterminé par des bornes inférieures et supérieures pour chaque paramètre.

La première question concerne la direction de ce pas, parmi les n possibles, ou leur combinaison. La solution retenue consiste en un tirage au sort sur chaque paramètre pour déterminer si il doit être modifié ou non. **La direction de chaque pas est ainsi aléatoire**. Ce type de déplacement d'un pas est appelé *déplacement partiel*.. Le mode *déplacement total*, où tous les paramètres sont modifiés simultanément, est toutefois disponible dans la phase de préparation, dite de positionnement, avant le recuit proprement dit.

Dans les problèmes de cinétique qui nous intéressent ici, comme dans beaucoup d'autres d'ailleurs, on est le plus souvent en présence de paramètres dont les valeurs peuvent varier sur plusieurs décades : il y a une forte hétérogénéité entre les bornes du domaine acceptable, comme, d'ailleurs, entre les paramètres eux-mêmes. Afin de balayer correctement l'espace des paramètres, il faut donc utiliser un **pas**

multiplicatif, et non additif. Et un tirage au sort indique si on multiplie ou si on divise par ce pas la valeur d'un paramètre.

Lorsqu'un paramètre est amené près d'une de ses bornes, il faut éviter de le faire "rebondir", ce qui reviendrait à privilégier cette région de l'espace. Il faut donc considérer un **domaine cyclique**, c'est-à-dire qu'un franchissement d'une borne renvoie à la borne opposée. Dans ce cas, la détermination du sens de déplacement n'est plus aléatoire : si le déplacement se fait au delà de la borne supérieure, alors on multiplie la borne inférieure par le pas, et si il se fait en deçà de la borne inférieure, alors on divise la borne supérieure.

Pour cette raison, bien que les paramètres puissent être positifs ou négatifs (mais non nuls), **leurs bornes doivent être de même signe** (valeur min et max de même signe pour un paramètre donné). Si ce n'est pas le cas, il est toujours possible de s'y ramener par changement de variable.

Enfin, d'une part, la taille du pas doit être aléatoire sinon cela reviendrait à quadriller l'espace et donc à exclure totalement certaines positions. D'autre part, elle doit être adaptée à la température, car il importe d'avoir aux hautes températures un pas assez grand pour explorer tout l'espace, tandis qu'aux basses températures il doit être relativement petit pour ne pas sauter trop loin du bassin d'attraction et descendre vers le minimum global. Ainsi, une taille de *pas moyen* est déterminée, en fonction (linéaire) de la température. **La taille de chaque pas réel est alors tirée au sort** selon une distribution gaussienne centrée sur le *pas moyen* et d'*écart-type* réglable σ.

Variation de la température

Après une phase de positionnement initial dont le but est de se placer dans un bassin d'attraction susceptible de contenir le minimum global, le recuit simulé proprement dit est opéré en abaissant la température par paliers successifs. La progression peut être linéaire ou géométrique.

Il est possible de régler le nombre d'itérations (de pas) sur chaque palier, en fonction de la température, comme la taille du pas. Un nombre d'itérations élevé au départ et petit à la fin privilégie une grande mobilité aux températures élevées, permettant de franchir des obstacles, tandis que la descente sera pauvre. C'est le but habituellement poursuivi avec le recuit simulé, et il est toujours possible de parfaire ensuite la descente à l'aide d'un algorithme de descente efficace.

En toute rigueur, le nombre d'itérations devrait être proportionnel à x^n, n étant le nombre de paramètres, ce qui devient rapidement prohibitif lorsque n est important. En réalité, on fait toujours un compromis entre le nombre de paramètres, l'étendue de leur domaine... et le temps de calcul qu'on s'autorise !

Validation d'un déplacement

La fonction est évaluée après chaque pas de façon à calculer le ΔE correspondant.

Si $\Delta E \leq 0$, le déplacement est toujours accepté. On se positionne au nouveau point.

Si $\Delta E > 0$, on tire un nombre aléatoire q, entre 0 et 1, et on le compare à la probabilité de transition $P = e^{-\Delta E/T}$:
- si q > P, le déplacement est rejeté, on revient à la position précédente
- si q ≤ P, le déplacement est accepté, on se positionne au nouveau point.

A chaque déplacement accepté, l'état précédent est oublié.

Utilisation pratique du recuit simulé

Le *recuit simulé* est d'utilisation relativement lourde, mais il peut être une aide efficace dans certaines situations, par exemple :

- on n'a aucune idée des valeurs de certains paramètres, et on n'arrive pas à s'approcher manuellement des données expérimentales. Le recuit simulé peut alors être utilisé comme un outil de recherche "objective" d'une solution possible

- on a déjà trouvé une, ou plusieurs, solutions mais on se demande si il n'y en a pas d'autres, éventuellement meilleures

- on n'arrive pas à un ajustement correct avec un modèle donné, et on voudrait s'assurer qu'on n'est pas passé à côté d'une solution avant de rejeter le modèle.

Son utilisation demande un peu de pratique et il est impossible de donner une méthode générale pour s'en servir. Le lecteur trouvera cependant quelques conseils dans le Manuel de *Sa*, (Tome 1, chapitre VII, p.42) et des exemples (Tome 2, Exemples 1 et 2).