Projet de fin d'étude

Master 2 - Mathématiques appliquées et modélisation

sujet : "Optimisation : algorithmes locaux et globaux, implémentation et applications"

Julien Guégan

Tuteur Polytech : Cédric BOULBE
Tuteurs Exact Cure : Frédéric DAYAN

Table des matières

1 Contexte

ExactCure est une start-up spécialisée dans la médecine digitale et personnalisée. Leur ambition est de proposer aux patients ainsi que leur écosystème de santé des solutions de personnalisation de l'effet des médicaments, basées sur des technologies d'intelligence artificielle et de biomodélisation. En effet, chaque individu étant différent (âge, sexe, poids, taille, patrimoine génétique ...), la dose de médicaments ou encore l'heure auquel il est pris doit s'adapter en fonction de chaque patient. ExactCure propose donc une solution pour simuler précisément la réponse d'un médicament de façon à individualiser avec une précision essentielle dans le monde d'aujourd'hui.

L'étude de la biologie des système permet de résoudre de tel problème puisque c'est le domaine d'étude qui cherche à comprendre le fonctionnement d'un système biologique en étudiant les relations et interactions entre différents niveaux biologiques (organites, cellules, protéines ...). Le but est alors de décrire un modèle de fonctionnement de la totalité du système. On commence à s'intéresser à un certain périmètre d'un réseau de réaction biochimique, le but est de modéliser la dynamique des quantités de concentrations des espèces présentes. On s'intéresse, à titre d'exemple, à une protéine A qui se dégraderait au cours du temps à une vitesse cinétique k1 (réaction R1) et produit la protéine B qui à son tour se transforme en la protéine A (R2).

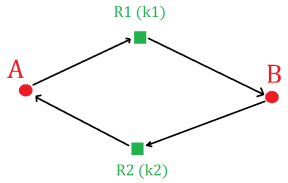


FIGURE 1.1 – Pathway biologique représentant les interactions entre les molécules

On modélise par le système dynamique :

$$\begin{cases} \frac{dA}{dt} = -k_1 A + k_2 B \\ \frac{dB}{dt} = k_1 A - k_2 B \\ A(t=0) = A_0 \ ; \ B(t=0) = B_0 \end{cases}$$

En résolvant ce système, on peut ainsi exprimer les quantité de protéines au cours du temps. Dans notre exemple, on obtient $\begin{cases} A(t) = A_0 + e^{-(k_1+k_2)t} + \frac{k_1}{k_1+k_2}(A_0+B_0)(1-e^{-(k_1+k_2)t}) \\ B(t) = A_0 + B_0 - A(t) \end{cases}$

Calibration de modèle

Pour des réseaux complexes, les paramètres cinétiques (k_1 et k_2 dans l'exemple) ne sont pas connus mais on est capable de mesurer une série de valeurs expérimentalement. La calibration du modèle est le processus qui nous permet d'estimer les paramètres cinétiques du modèle (jusqu'alors inconnus) pour que celui-ci corresponde aux observations et donc avoir une meilleure prédiction.

Si on note $y_{obs}(t_i)$ les observations et $\Psi(t_i, s_0, p)(= y(t))$ la solution à notre problème. On appelle J la fonction qui mesure la somme des écarts entre la simulation du système pour une valeur de paramètre p fixé et les données observées $y_{obs}(t_i)$. Alors le problème de calibration se formule ainsi :

$$J(p) = \sum_{i=1}^{n} \|\Psi(t_i, s_0, p) - y_{obs}(t_i)\|^2 \underset{p\in\mathbb{R}^q}{\longrightarrow} min$$

L'objectif du projet est d'étudier un ensemble d'algorithmes d'optimisation, les implémenter pour ensuite les valider sur des benchmarks classiques pour résoudre ce genre de problème de calibration. Dans un premier temps, une étude sera faite sur certaines méthodes à direction de descente, ensuite sur des méthodes appelées "stratégies d'évolution", et enfin des techniques sur l'optimisation avec contraintes.

2 Méthodes à direction de descente

Le principe des méthodes à direction de descente est de générer une suite d'itérés x_k en suivant une direction de descente d_k donnée : $\boxed{x_{k+1} = x_k + \alpha_k d_k}$. On peut par exemple citer les méthodes les plus connues :

Gradient : $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$ Gradient conjugué : $\begin{cases} x_{k+1} = x_k - \alpha_k d_k \\ d_{k+1} = \nabla f(x_k) + \beta d_k \end{cases}$

Newton : $\begin{cases} x_{k+1} = x_k - \alpha_k d_k \\ d_{k+1} = \nabla^2 f(x_k)^{-1} \nabla f(x_k) \end{cases}$ Quasi-Newton : $\begin{cases} x_{k+1} = x_k - \alpha_k d_k \\ d_{k+1} = M^{-1} \nabla f(x_k) \end{cases}$

2.1 Gradient à pas fixe

En utilisant un développement de Taylor à un itération x_{k+1} :

$$f(x_{k+1}) = f(x_k) + \nabla f(x_k)(x_{k+1} - x_k) + o(.)$$
$$f(x_{k+1}) - f(x_k) = \nabla f(x_k)(x_{k+1} - x_k)$$

Notre but est de trouver un x_{k+1} tel que la fonction f décroît, on veut donc que le terme $\nabla f(x_k)(x_{k+1} - x_k)$ soit le plus petit possible, on choisit pour cela $x_{k+1} - x_k = -\alpha \nabla f(x_k)$ avec α suffisamment petit.

Algorithm 1 Gradient à pas fixe

1) Initialisation

On choisit un $x^0 \in \mathbb{R}^n$: un x initial, un point de départ
un $\alpha \in \mathbb{R}$: le pas

2) Itération k [1]

$x_{k+1} = x_k - \alpha \nabla f(x_k)$

Cet algorithme est donc utilisé pour des fonctions différentiables, car on effectue à chaque itération un déplacement dans la direction opposé au gradient. La question qui se pose maintenant est comment bien choisir le pas qui va déterminer le déplacement effectué le long de notre direction de descente. Ce qui nous mène à l'algorithme à pas variable et à la technique de la recherche linéaire.

2.2 Gradient à pas variable et recherche linéaire

Le pas α joue un rôle important dans la convergence des méthodes à direction de descente. En effet, si on choisit une longueur de pas le long de notre direction de descente qui est trop petit, l'algorithme peut converger vers le minimum de façon très lente et d'un autre côté si on choisit un pas trop grand l'algorithme peut itérer à l'infini et osciller entre 2 valeurs sans jamais trouver le minimum (cf figure 2.1). Il faut donc trouver un pas de tel sorte que l'algorithme converge vers le minimum.

1. J'ai utilisé dans mes algorithmes comme critère d'arrêt : $\|x_{k+1} - x_k\| < \epsilon$ et ne pas dépasser un nombre d'itérations maximum

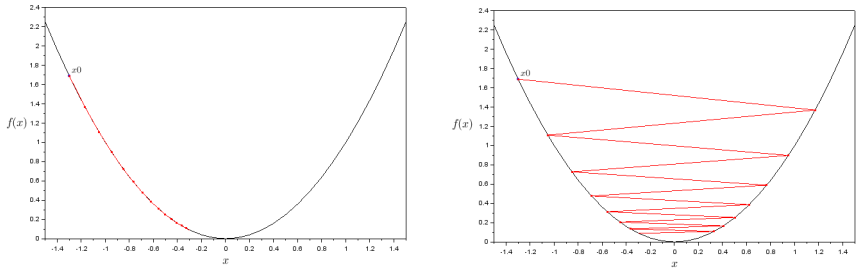


FIGURE 2.1 – Exemple de non convergence vers le minimum. A gauche, un pas trop petit et à droite, un pas trop grand.

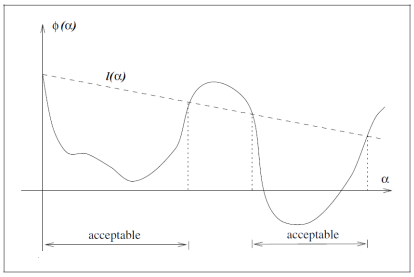
La recherche linéaire est donc l'étape qui consiste à trouver un pas convenable pour la recherche du minimum. Le but est donc de choisir un pas α_k qui minimise $f(x_{k+1})$, c'est-à-dire, à chaque itération k, vérifier que le pas choisi réalisera effectivement une diminution de la fonction coût tout en assurant la convergence. Une simple condition qu'on pourrait imposer serait $f(x_k + \alpha_k d_k) \leqslant f(x_k)$ qui fournit une réduction de f. En fait, ce n'est pas vraiment approprié car un pas α très petit peut vérifier cette condition. Les approches classiques reposent sur 2 conditions, la 1ère assure une décroissance suffisante (Armijo) et la 2ème empêche que le pas α_k tende vers 0 (Goldstein ou Wolfe).

Armijo :

$$f(x_{k+1}) \leqslant f(x_k) + c_1 \nabla f(x_k) \alpha_k . d_k$$

avec $c_1 \in [0, 1]$

$$\Rightarrow \phi(\alpha) \leqslant l(\alpha)$$



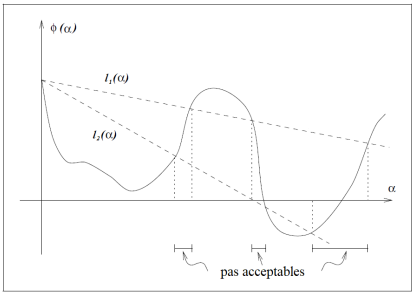
Cette condition dite d'Armijo signifie que la réduction de f doit être proportionnelle à la taille du pas α_k et à la dérivée directionnelle $\nabla f(x_k) d_k$. Cette condition est illustrée ci-contre avec la fonction $l(.)$ qui a une pente négative $c_1 \nabla f(x_k) d_k$, mais comme $c_1 \in [0, 1]$ le graphe de l est forcément au-dessus celui de ϕ pour des petites valeurs de α_k petit mais est aussi inférieur à $f(x_k)$. En effet si $c_1 = 0$, on a simplement que la droite l est horizontale et vaut $f(x_k)$ (l'ordonnée à l'origine de ϕ sur la figure) et si $c_1 = 1$, on a que $l(\alpha)$ vaut exactement $\phi'(0)$, la pente de ϕ en $\alpha = 0$. En pratique, on choisit c_1 très petit, généralement $c_1 = 10^{-4}$. Cette condition suffisante de décroissance n'est pas à utiliser seule puisque elle est satisfaite pour toutes les valeurs α suffisamment petites, c'est pourquoi on la combine avec l'une des 2 conditions suivantes.

Goldstein :

$$f(x_{k+1}) \geqslant f(x_k) + c_2 \nabla f(x_k) \alpha_k . d_k$$

avec $c_1 \leqslant c_2$

$$\Rightarrow \phi(\alpha) \geqslant l_2(\alpha)$$



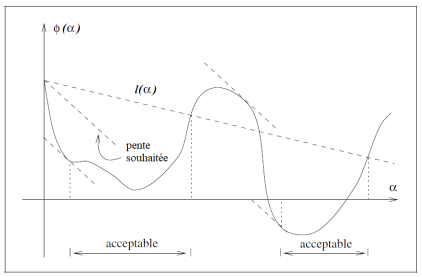
La condition de Goldstein permet d'assurer que le pas α choisit n'est pas trop petit. Le terme de droite de l'inégalité est représenté par la droite l_2 sur la figure ci-contre et la droite l_1 représente le terme de droite de la condition d'Armijo. Donc, en choisissant c_1 et $c_2 \in [0, 1]$ avec $c_2 \leqslant c_1$, on trouve un pas α qui permet une réduction suffisante de f, sur la figure c'est les α dont la courbe ϕ est comprise entre les 2 droites l_1 et l_2.

Wolfe :

$$\nabla f(x_{k+1}).d_k \geqslant c_2 \nabla f(x_k).d_k$$

avec $c_1 \leqslant c_2$

$$\Rightarrow \phi'(\alpha) \geqslant c_2 \phi'(0)$$



Comme la conditions de Goldstein, la condition de Wolfe permet d'empêcher α d'être trop petit. On note que la partie gauche est simplement la dérivée $\phi'(x_k)$, donc cette condition assure que la pente de $\phi(\alpha)$ est c_2 fois plus grande que la pente en $\phi(0)$ (qui est négative). C'est plutôt logique parce que si la pente $\phi'(\alpha)$ est fortement négative, on peut alors obtenir une réduction de f significative en continuant dans cette direction alors que si on est sur une pente très peu négative, presque plate, alors c'est un signe qu'on ne peut pas espérer obtenir une forte réduction de f dans cette direction.

L'algorithme du gradient à pas variable est identique à celui de gradient mais à chaque itération, on calcule un nouveau pas α_k qui respecte la condition d'Armijo et de Goldstein/Wolfe (par exemple, l'algorithme de Fletcher-Lemaréchal résoud ce problème de recherche linéaire en un nombre fini d'étapes).

Algorithm 2 Gradient à pas variable

1) Initialisation

On choisit un $x^0 \in \mathbb{R}^n$

2) Itération k

Calculer un pas α_k avec un algorithme de recherche linéaire

$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$

L'inconvénient des algorithmes de gradient simple est qu'ils ont tendance à avoir un comportement de zigzag le long des vallées étroites (cf figure 2.2) et donc la vitesse de convergence vers le minimum peut être ralentie. En effet, il a été montrer que l'algorithme peut nécessiter de nombreuses itérations pour converger vers un minimum local quand la courbure de la fonction coût est très différente dans des directions différentes. En plus, la recherche du pas α optimal, effectuée par une recherche linéaire, peut se révéler assez longue. Alors que si on utilise un pas α fixe peut parfois conduire à des résultats non satisfaisants.

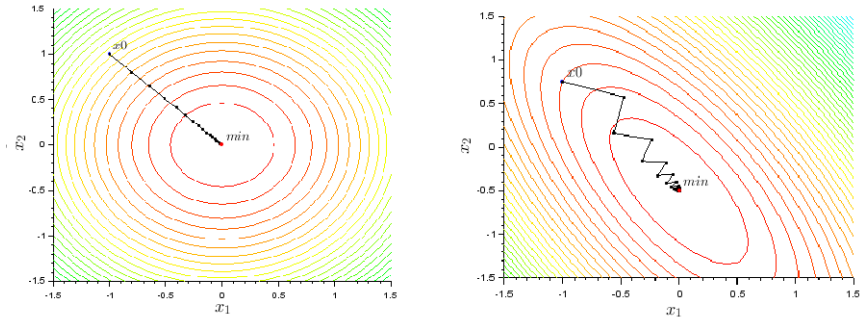


FIGURE 2.2 – Comportement de l'algorithme du gradient. A gauche, on converge directement vers le minimum pour une fonction sphérique et à droite la suite d'itérés zigzague dans une vallée étroite.

2.3 Gradient conjugué

La particularité de la méthode du gradient conjugué est qu'on prend en compte l'information des itérés précédents puisque les directions de descente d_k sont conjuguées les unes aux autres. En effet, la direction d_k est construite à partir du gradient actuel (en x_k) et de toutes les directions de recherche précédentes. L'opération de conjugaison se fait de façon orthonormal, le procédé ressemble donc à l'orthonormalisation de Gram-Schmidt (algorithme de construction d'une base orthonormée à partir d'une famille libre), à chaque itération on calcule une direction de descente : $d_{k+1} = \nabla f(x_k) + \beta_k d_k$. Il existe actuellement plusieurs façons pour évaluer le coefficient β_k, l'une des plus connue est la méthode dite de "Fletcher-Reeves" qui donne $\beta_k = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_{k-1})^T \nabla f(x_{k-1})}$ mais il en existe d'autres comme celles de Polak-Ribière, Dai-Yuan, Hesteness-Stiefel. Dans le cas linéaire, ces expressions sont équivalentes les unes aux autres mais dans les cas non linéaires, ces expressions ne sont plus tout à fait équivalentes et les chercheurs examinent toujours sur le meilleur choix possible.

Algorithm 3 Gradient conjugué

1) Initialisation

On choisit un $x_0 \in \mathbb{R}^n$

On choisit un $d_0 = \nabla f(x_0)$

2) Itération k

$x_{k+1} = x_k - \alpha_k d_k$

$\beta_k = \nabla f(x_k)^T \nabla f(x_k) / \nabla f(x_{k-1})^T \nabla f(x_{k-1})$

$d_{k+1} = \beta_k \nabla f(x_k)$

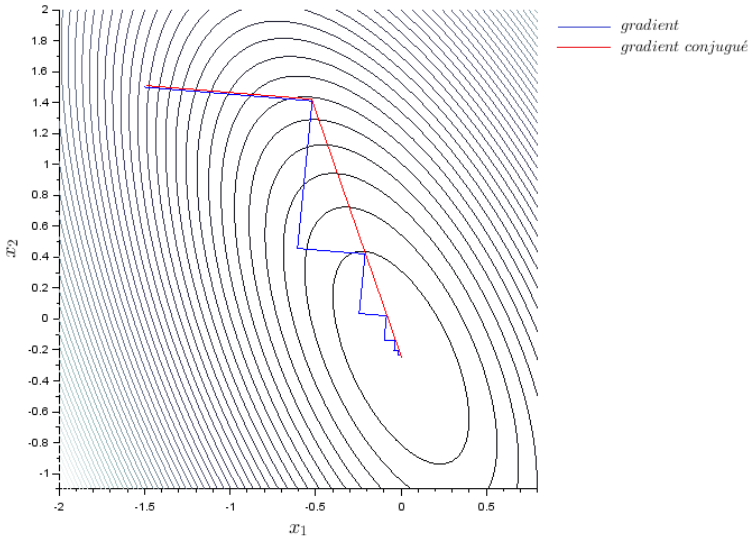


FIGURE 2.3 – Comparaison du comportement du gradient à pas variable et du gradient conjugué pour une fonction quadratique

Remarques : S'il y a N variables à minimiser, pour une fonction purement quadratique, le minimum sera atteint en au plus N itérations mais une fonction non quadratique sera minimiser plus lentement. La méthode du gradient conjugué peut suivre des vallées étroites où la méthode de gradient simple a tendance à ralentir et zigzaguer avec un petit pas. Cependant moins la fonction f est similaire à une fonction quadratique et plus les directions perdent leur conjugaison.

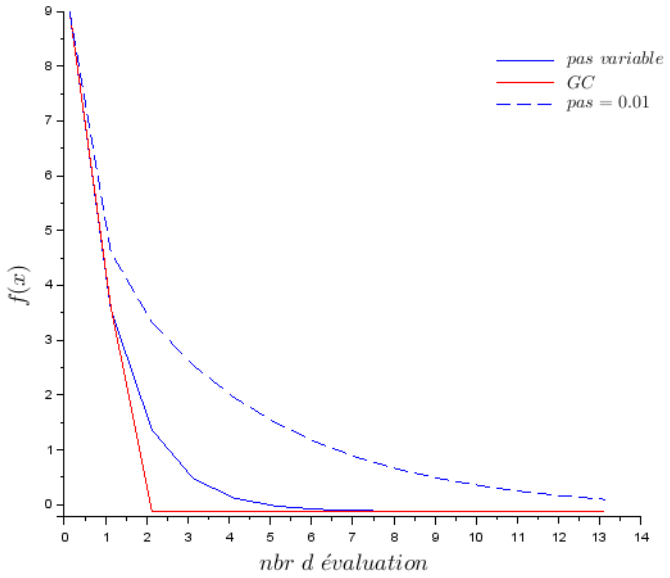


FIGURE 2.4 – Convergence des différentes méthodes de gradient pour une fonction quadratique

J'ai étudié, ici, le comportement et la convergence des méthodes de gradient avec notamment l'algorithme du gradient conjugué qui conjugue les directions de descentes pour converger plus rapidement vers le minimum. Cependant ses méthodes sont surtout efficace pour des fonctions plutôt quadratiques et la méthode du gradient conjugué préconditionné est d'ailleurs très connu pour résoudre les systèmes linéaires dont la matrice est symétrique définie positive. Il pourrait être intéressant d'étudier l'algorithme de Newton qui utilise le gradient et l'inverse de la hessienne de la fonction objectif comme direction de descente et de comparer sa convergence à la méthode du gradient conjugué. Et parmi les algorithmes à directions de descente connus les plus efficaces, la méthode BFGS (Broyden-Fletcher-Goldfarb-Shanno) est souvent utilisée, elle consiste à calculer en chaque étape une matrice qui multipliée au gradient permet d'obtenir une meilleure direction.

Les méthodes de descentes ont été appliquées pendant plusieurs années de part leur efficacité : preuve de convergence, vitesse de convergence rapide (peu d'itérations), faible dépendance à la dimension n du problème. Cependant elles se limitent à l'optimisation de fonctions différentiables et convexes et permettent de trouver seulement un minimum local.

3 Méthodes inspirées de la biologie

Les méthodes bio-mimétiques sont des méthodes d'optimisation qui ont été créées en se basant sur les grands principes de la nature comme la génétique et la théorie de l'évolution ou même les comportements d'êtres vivants (essaims de particules, colonies de fourmis) pour approcher un minimum de façon stochastique et elles sont des stratégies régulièrement employées dans le monde de l'ingénierie actuelle.

3.1 (μ,λ)-ES

Les algorithmes stratégies d'évolution se basent sur les opérateurs de mutation et de sélection de population qui tendent à trier les individus étant les plus adaptés à survivre, c'est-à-dire ici nos minimiseurs de notre fonction coût. Par exemple, la méthode de type (μ,λ)-ES repose sur une population de μ parents générant un ensemble de λ enfants par mutation. Ensuite une étape de sélection aboutit à la génération suivante de μ parents. Il existe quelques variantes comme $(\mu/\rho,\ \lambda)$-ES ou $(\mu+\lambda)$-ES.

Algorithm 4 (μ,λ)-ES

1) Initialisation

$\bar{x}^{(0)}$: barycentre de la population initiale

$\bar{\sigma}^{(0)}$: écart-type de la population initiale

$k \leftarrow 0$

2) Mutation : création de λ enfants

for $i = 1, ..., \lambda$ **do**

$\sigma_i \leftarrow \bar{\sigma}^{(k)} e^{\tau \mathcal{N}(0, I_d)}$

$x_i \leftarrow \bar{x}^{(k)} + \sigma_i \mathcal{N}(0, I_d)$

end for

3) Sélection : les μ meilleurs enfants = les μ parents de la génération $k+1$

4) Mise à jour :

$$\bar{x}^{(k+1)} \leftarrow \frac{1}{\mu} \sum_{i=1}^{\mu} x_i$$

$$\bar{\sigma}^{(k+1)} \leftarrow \frac{1}{\mu} \sum_{i=1}^{\mu} \sigma_i$$

$k \leftarrow k+1$

A chaque génération k, la population est caractérisée par sa moyenne $\bar{x}^{(k)}$ et son écart moyen $\bar{\sigma}^{(k)}$. La nouvelle génération $x^{(k+1)}$ est créée par des perturbations aléatoire autour de la moyenne $\bar{x}^{(k)}$ en utilisant une loi normale centrée réduite multidimensionnelle mais chaque nouvel individu x_i dépend également de son propre écart type σ_i qui est donc transmis à la prochaine génération. L'évolution de cet écart-type donne à l'algorithme un caractère *auto-adaptatif* et se révèle important dans l'efficacité de la méthode.

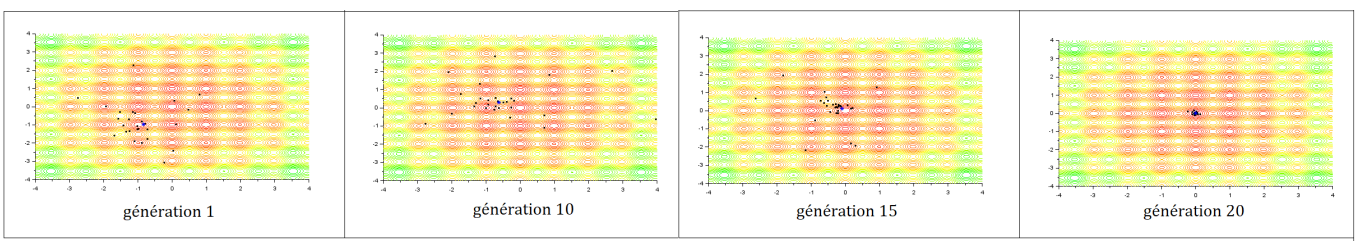


FIGURE 3.1 – Convergence de (μ,λ)-ES vers le minimum global de la fonction de Rastrigin.

3.2 CMAES

L'algorithme CMAES (Covariance Matrix Adaptation Evolution Strategie) est un algorithme de type stratégies d'évolution qui repose sur l'adaptation, au cours des itérations, de matrice de covariance de la distribution normale utilisée. Cette méthode a été proposée par A. Gawelczyk, N. Hansen, et A. Ostermeier à la fin des années 1990 et est aujourd'hui parmi les meilleures algorithmes métaheuristiques pour les problèmes continus.

La principale différence avec l'algorithme (μ,λ)-ES classique est lors de la création de λ enfants par une loi normale, on adapte à chaque itération la matrice de covariance de la distribution en prenant en compte un "chemin d'évolution" de nos itérations pour se diriger plus efficacement vers le minimum.

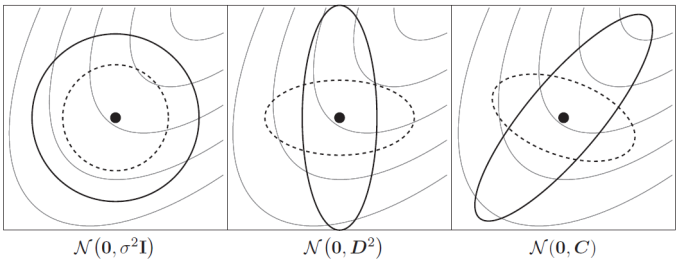


FIGURE 3.2 – Représentation de l'utilisation de matrice de covariance de la distribution.

Algorithm 5 CMAES [1]

1) Initialisation

$\bar{x}^{(0)}$: barycentre initiale
$\bar{\sigma}^{(0)}$: écart-type initiale
$k \leftarrow 0$

2) Itération

génération de λ enfants
sélection des μ meilleurs
mettre à jour $\bar{x}$
mettre à jour les chemins d'évolution de C et σ
calculer la matrice de covariance C
calculer la longueur de pas $\bar{\sigma}$
$k \leftarrow k + 1$

La figure 3.3 ci-dessous illustre bien l'efficacité de l'utilisation de la matrice de covariance pour la fonction de Rozenbrock où une vallée étroite fait évoluer lentement l'algorithme (μ,λ)-ES alors que l'algorithme CMAES progresse sans difficultés grâce à l'adaptation de la matrice de covariance.

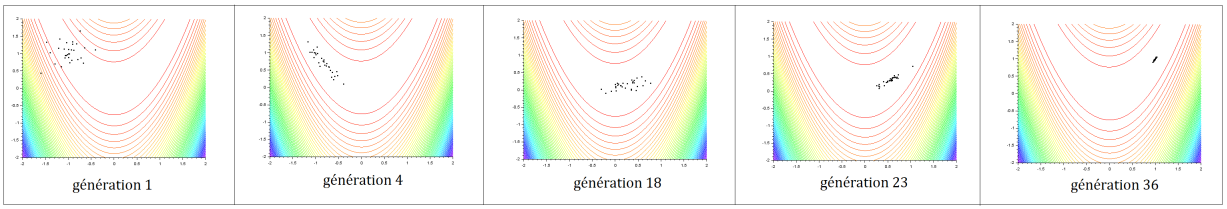


FIGURE 3.3 – Convergence de CMAES vers le minimum de la fonction de Rozenbrock.

1. Pour cette méthode un peu trop technique, j'ai choisi de ne pas détailler toutes les étapes de l'algorithme. Une étude approfondie peut se trouver en [12] et un code opensource se trouve sur `https://en.wikipedia.org/wiki/CMA-ES`

Dans de nombreux cas, la méthode CMAES s'est montré utile notamment dans des cas de fonctions non convexe, bruité ou mal conditionnée. L'avantage non négligeable de cette méthode, de part son caractère auto-adaptatif, est la faible dépendance aux paramètres, en effet en faisant varier les paramètres d'entrée (μ, λ, x_0, σ_0 ...) le nombre d'évaluation de la fonction ne change quasiment pas. Les cas où cette méthode peut se montrer moins efficace que d'autres algorithmes sont ceux par exemple où la fonction objectif est : de dimension peu élevée (n<5), quadratique convexe et peut être minimiser avec un faible nombre d'évaluation car l'inconvénient principal de cette méthode est que le nombre d'évaluation de f peut vite être élevé.

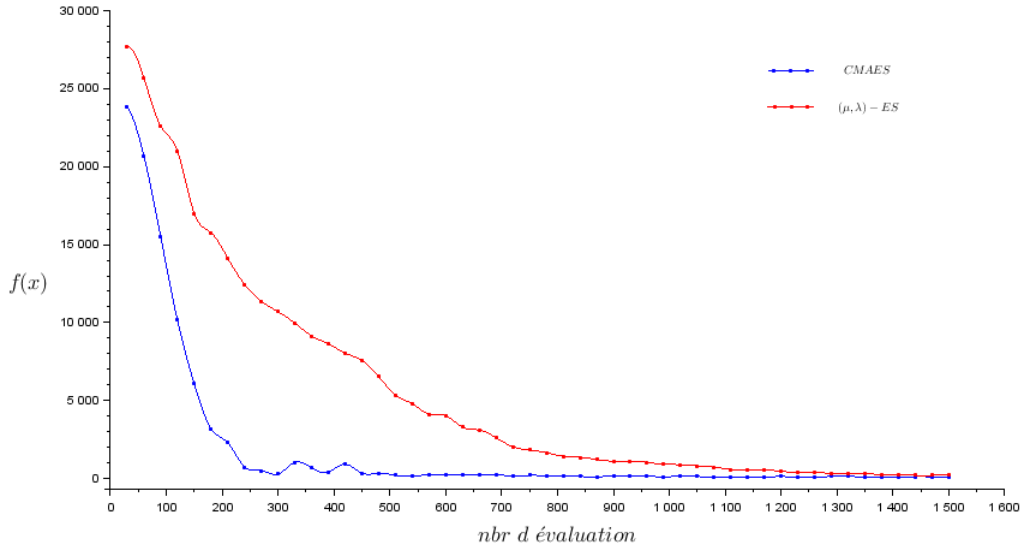


FIGURE 3.4 – Comparaison des vitesses de convergence des algorithmes CMAES et (μ,λ)-ES pour la fonction de Rastrigin en dimension 20.

4 Optimisation avec contraintes

4.1 Uzawa

On souhaite résoudre le problème $(\mathcal{P})$ de minimisation sous contraintes d'égalités et d'inégalités :

$$\min_{x \in K} J(x) \tag{$\mathcal{P}$}$$

$$K = \{x \in \mathbb{R}^n \ h(x) = 0, \ g(x) \leqslant 0\} \text{ avec } h = (h_i)_{1 \leqslant i \leqslant p} \text{ et } g = (g_j)_{1 \leqslant j \leqslant q}$$

On introduit le lagrangien $\mathcal{L}$ associé à ce problème :

$$\mathcal{L}(x, \lambda, \mu) = J(x) + \sum_{i=1}^{p} \lambda_i h_i(x) + \sum_{j=1}^{q} \mu_j g_j(x)$$

L'intérêt du Lagrangien est de ramener le problème $(\mathcal{P})$ de minimisation sous contraintes $x \in K$ à un problème sans contraintes. L'idée générale de l'introduction du Lagrangien est de trouver son point-selle, c'est-à-dire le point (x^*, λ^*, μ^*) tel que $\forall (x, \lambda, \mu) \in \mathbb{R}^n \times \mathbb{R}^p \times (\mathbb{R}_+)^q$, on a

$$\mathcal{L}(x^*, \lambda, \mu) \leqslant \mathcal{L}(x^*, \lambda^*, \mu^*) \leqslant \mathcal{L}(x, \lambda^*, \mu^*)$$

Le problème revient à d'une part, maximiser $\mathcal{L}$ par rapport aux variables λ et μ et d'autre part, minimiser $\mathcal{L}$ par rapport à sa première variable x. On résout alors 2 problèmes d'optimisation duaux.

Algorithm 6 Uzawa

1) Initialisation

$k = 0$: on choisit $\lambda^0 \in \mathbb{R}^p$ et $\mu^0 \in (\mathbb{R}_+)^q$

2) Itération k Trouver x^*, λ^* et μ^*

$\lambda^k \in \mathbb{R}^p$ et $\mu^k \in (\mathbb{R}_+)^q$ sont connus.

(a) Calcul de $x^k \in \mathbb{R}^n$

$$x^{k+1} = x^k - \alpha_x \nabla_x \mathcal{L}(x^k, \lambda^k, \mu^k)$$

(b) Calcul de $\lambda^{k+1} \in \mathbb{R}^p$ et $\mu^{k+1} \in (\mathbb{R}_+)^q$

$$\lambda^{k+1} = \lambda^k + \alpha_\lambda \nabla_\lambda \mathcal{L}(x^{k+1}, \lambda^k, \mu^k)$$

$$\mu^{k+1} = \mu^k + \alpha_\mu \nabla_\mu \mathcal{L}(x^{k+1}, \lambda^k, \mu^k)$$

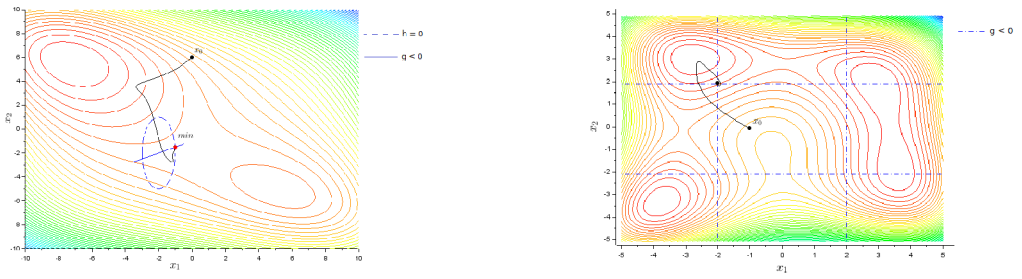


FIGURE 4.1 – Convergence de l'algorithme d'Uzawa. A droite, avec contraintes d'égalité et d'inégalité. A gauche, avec contraintes d'égalité.

4.2 Stratégie d'évolution

Dans l'algorithme (μ,λ)-ES, dans l'étape de sélection, on choisit les μ points qui sont tels que notre fonction f est la plus petite. Dans le cas où on veut prendre en compte des contraintes, on sélectionne seulement les points tels que f est minimum et qui vérifient notre contraintes d'inégalités.

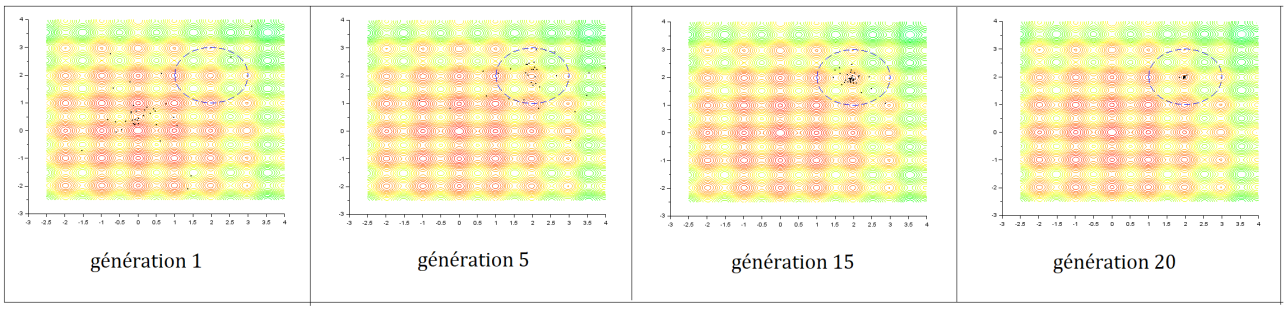


FIGURE 4.2 – Convergence de l'algorithme (μ,λ)-ES avec une contrainte d'inégalité quadratique.

4.3 Gradient projeté

La méthode du gradient projeté est une méthode pour les problèmes d'optimisation et est utile dans les cas où la contrainte est simple, par exemple quand les variables sont bornées.

$$\min_{a \leqslant x \leqslant b} J(x) \tag{1}$$

Le principe est d'utiliser l'algorithme du gradient et de projeter sur l'ensemble des contraintes : $x_{k+1} = \Pi_C(x_k - \alpha_k \nabla f(x_k))$ avec Π_C la projection sur l'espace C. En pratique, on utilise une matrice de projection qui est $r = -P\nabla f$. On définit M la matrice jacobienne des contraintes, la matrice de projection peut être calculée en utilisant $P = I - M(M^TM)^{-1}M^T$. Une contrainte d'inégalité $g(x) \leqslant 0$ est dite active en x^* si $g(x^*) = 0$ et inactive si $g(x^*) < 0$. Une contrainte d'égalité $h(x) = 0$ est dite active en x^* si $h(x^*) = 0$.

Algorithm 7 Gradient projeté [1]

1) Initialisation

$k = 0$: on choisit $x^0 \in \mathbb{R}^n$

2) Itération k

$r = -(I - M(M^TM)^{-1}M^T)\nabla f$
tant que $(r = 0)$
  $u = -(M^TM)^{-1}M^T\nabla f$
  si $\min_i u_i < 0$
    supprimer g_i des contraintes actives
    mettre à jour M et recalculer r
  sinon
    retourner x_k
fin tant que
$x_{k+1} = x_k + \alpha r$
$k = k + 1$

1. Par manque de temps, je n'ai pas pu finir l'implémentation de cette méthode mais il me semblait intéressant d'en parler dans le rapport

5 Perspectives

Pour approfondir mon étude, il serait intéressant d'étudier, tester et comparer un algorithme appartenant à la classe des méthodes dites à surface de réponse qui consistent à construire un modèle de la fonction objectif à partir d'évaluations. Ces approches ont récemment été popularisées car elles permettent de tirer un maximum d'informations sur la fonction à chaque itération et donc elles convergent rapidement. Une optimisation globale est possible avec ce genre de méthodes mais elles sont difficiles à mettre en place et fonctionnent bien seulement pour une centaine de paramètres.

Un point également à retravailler dans ce projet pourrait être l'algorithme d'Uzawa. En effet, le but est de minimiser et maximiser le lagrangien par rapport à certaines variables et j'ai utiliser pour ce faire la méthode du gradient avec des pas fixés, on pourrait chercher à trouver le meilleur pas pour x et pour λ et μ par recherche linéaire (ce qui supposerait de pouvoir facilement manipuler le lagrangien du problème associé) ou encore chercher à utiliser le gradient conjugué ou voir si une autre méthode d'optimisation pourrait marcher.

D'autre part, pendant la fin de mon projet je me suis concentré à explorer les différentes solutions du calcul parallèle dans le but de pouvoir améliorer par la suite le temps de calcul des méthodes stratégie d'évolution qui ont besoin d'un grand nombre d'évaluation de la fonction pour converger vers le minimum global notamment en utilisant des tirages aléatoires indépendants. Si la fonction objectif est coûteuse à évaluer alors cette étape de tirages aléatoires peut mettre du temps en séquentiel mais est améliorable si réaliser en parallèle. En effet, en distribuant ces tirages aléatoires et donc ces λ évaluations de la fonction objectif sur plusieurs processeurs en mémoire distribuée ou partagée selon la procédure de notre choix alors ce temps de calcul serait réduit. J'ai donc effectué des recherches et ai pu tester l'efficacité de la procédure OpenMP en C++ ou encore l'environnement parallel pool avec la commande parfor en Matlab qui se sont montrer efficace pour effectuer plus rapidement qu'en séquentiel un grand nombre de tirage aléatoire sur ma machine. Cependant mon tuteur m'a orienté vers les solutions proposées en Scilab puisque mes codes sont rédigés en Scilab et qu'il pense continuer avec ce langage.

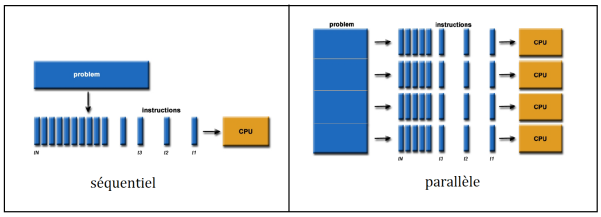


FIGURE 5.1 – Différence entre les principes de calcul parallèle et calcul séquentiel.

Il existe plusieurs méthodes proposées par Scilab [1] et après avoir essayé d'utiliser le module PVM et MPI sans succès, je me suis arrêté sur la commande parallelrun qui s'exécute seulement sous Linux mais qui m'a permis d'obtenir une amélioration de temps de calcul sur un exemple simple de plusieurs tirages aléatoires indépendants. La prochaine étape serait de pouvoir utiliser la commande parallelrun dans un algorithme stratégie d'évolution et ensuite tester et comparer ces différentes solutions proposées sur un cas concret de calibration de modèle pour la société ExactCure où la fonction coût demande la résolution d'un système dynamique à n variables et donc son évaluation est chère en temps de calcul.

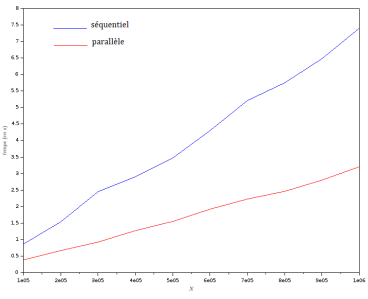


FIGURE 5.2 – Temps de calcul pour un exemple simple.

1. https://wiki.scilab.org/Documentation/ParallelComputingInScilab

6 Conclusion

L'optimisation est un domaine très large, en effet de nombreuses études ont déjà été faites et il existe une multitude de méthode pour réaliser la minimisation d'un fonction objectif. Dans ce projet, je me suis concentré seulement sur certaines méthodes simples de gradient et des méthodes de type stratégie d'évolution. Les méthodes de gradient sont parmi les plus efficaces en terme de vitesse de convergence cependant elles se limitent à une catégorie restreinte de problèmes puisque qu'elles convergent seulement vers le minimum local le plus proche et qu'il faut pouvoir calculer un gradient. A l'inverse, les méthodes stratégie d'évolution sont capables de résoudre des problèmes d'optimisation global pour des fonctions complexes mais demandent un plus grand nombre d'évaluations de la fonction.

A propos du déroulement du projet, je pense que le fait de travailler seul chez moi n'aura pas été très efficace puisque j'ai perdu pas mal de temps à changer de sujet, par exemple à passer du temps à essayer de comprendre les méthodes SQP ou à région de confiance qui finalement ne m'auront pas servies. J'ai également passer beaucoup de temps à faire des recherches sur le calcul parallèle et à essayer de mettre des exemples en place par exemple avec différents langages comme Julia, Python, C++ sans résultats probants. La principale difficulté était donc d'être autonome et organisé pour trouver les bonnes informations, implémenter les algorithmes demandés et les tester. Mais ceci m'aura permis de gagner en autonomie et en expérience pour mes prochains projets. Je remercie donc grandement Frédéric Dayan pour avoir proposé ce projet de fin d'étude et Sylvain Benito pour avoir pris le temps de faire un point avec moi une fois par semaine pendant les horaires du projet.

7 Bibliographie

[1] Nocedal, J., Wright, S., Numerical Optimization. *Springer* **1999**.

[2] Shewchuk J.R., An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. **1994**.

[3] Wilson D.R., Martinez T.R., The general inefficiency of batch training for gradient descent learning. *Elsevier* **2001**.

[4] Griva I., Nash S., Sofer A., Linear and Nonlinear Optimization. *Siam* **2009**.

[5] Chiong R., Nature-Inspired Algorithms for Optimisation. *Springer* **2008**.

[6] Hendrix E.,Toth B., Introduction to nonlinear and global optimization. *Springer* **2008**.

[7] Boyd S., Vandenberghe L., Convex Optimization. *Cambridge University Press* **2004**.

[8] Rao S., Engineering Optimization. *Wiley* **2009**.

[9] Brownlee J., Clever Algorithms : Nature-Inspired Programming Recipes. **2011**.

[10] Vanden Berghen F., CONDOR : a constrained, non-linear, derivative-free parallel optimizer for continuous, high computing load, noisy objective functions. *Thèse Université de Bruxelles* **2004**.

[11] H. Langouet, Optimisation sans dérivées sous contraintes. *Thèse Université de Nice-Sophia Antipolis* **2011**.

[12] N. Hansen, The CMA Evolution Strategy : A Tutorial.*archive HAL-inria* **2005**.

[13] J.F. Bonnans · J. Charles Gilbert, C. Lemaréchal, C. A. Sagastizabal, Numerical Optimization : Theoretical and Practical Aspects. *Springer* **2006**.

ANNEXES

Logiciels

- Dakota : application open source, licence LGPL.

- OPT++ : librairie C++ gratuite, licence LGPL.

- MIDACO : package gratuit, licence BY-NC-ND.

- AMPL : logiciel propriétaire.

- IPOPT : librairie C++ gratuite, licence CPL.

- Optimization Algorithm Toolkit (OAT) : logiciel gratuit, licence LGPL.

- Google Optimization Tools (OR-Tools) : logiciel open source gratuit, licence Apache 2.0

Algorithmes

- Algorithmes Génétiques

- Algorithme par séparation et évaluation (branch and bound)

- Algorithme de Bruss (ou algorithme des 'odds')

- Recherche harmonique

- Méthode du point intérieur

- Méthode BFGS

- Algorithme de Gauss–Newton

- Méthode de Nelder-Mead

- Recuit simulé

- Méthode du Lagrangien augmenté

- Méthode SQP

- Algorithme NSGA

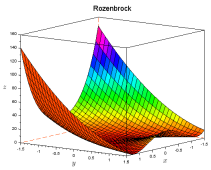
- Algorithme de Levenberg-Marquardt

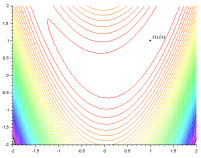
Fonctions Test pour l'optimisation

Il est important d'utiliser des fonctions Test pour évaluer les caractéristiques des algorithmes d'optimisation telles que la vitesse de convergence, la précision, la robustesse. Autrement dit, ceux sont des bancs d'essai qui servent à valider les algorithmes et comparer leurs performances selon les caractéristiques de notre problèmes.

Rosenbrock :

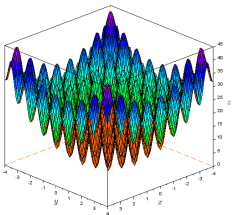
$$f(x) = \sum_{i=1}^{n} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

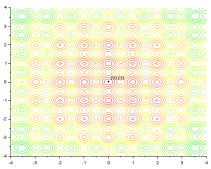




Rastrigin :

$$f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i)\right]$$

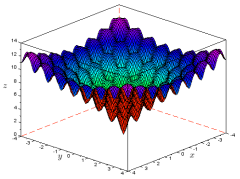


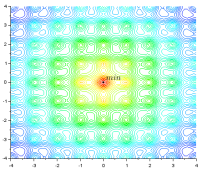


Ackley :

$$f(x) = -20\exp\left[-0.2\sqrt{0.5(x^2 + y^2)}\right]$$

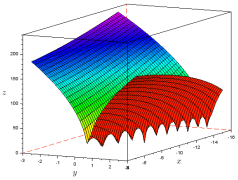
$$-\exp\left[0.5(\cos 2\pi x + \cos 2\pi y)\right] + e + 20$$

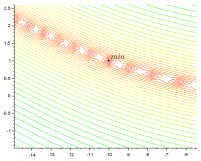




Bukin :

$$f(x) = 100\sqrt{|y - 0.01x^2|} + 0.01\,|x + 10|$$





Himmelblau :

$$f(x) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$