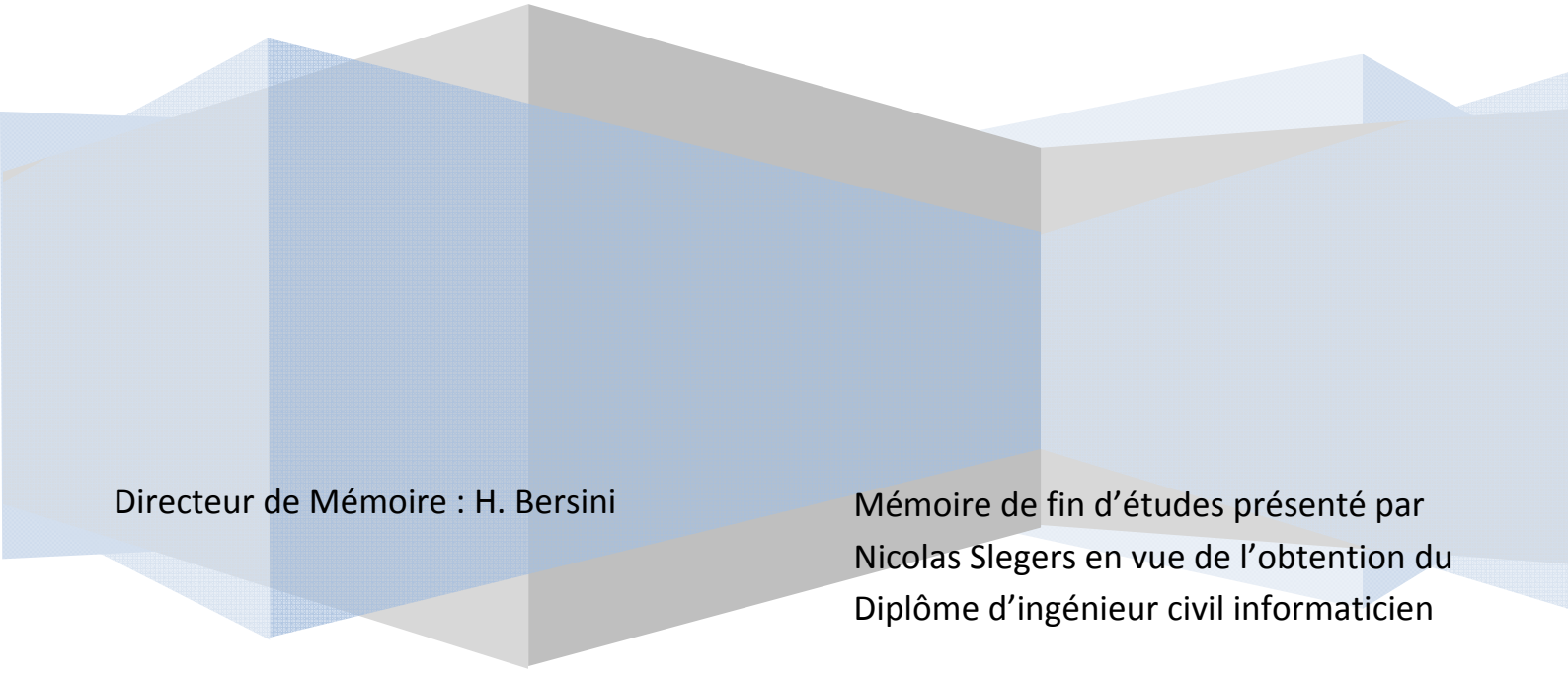
UNIVERSITE LIBRE DE BRUXELLES
Faculté des Sciences appliquées

Année académique 2007- 2008

Adaptation d'algorithme d'optimisation continue dans le domaine de la chimie de synthèse



Directeur de Mémoire : H. Bersini

Mémoire de fin d'études présenté par
Nicolas Slegers en vue de l'obtention du
Diplôme d'ingénieur civil informaticien

LECTURE RAPIDE

Ce travail consiste à trouver, comparer et adapter les meilleurs algorithmes d'optimisation capables de trouver le meilleur mélange des concentrations chimiques avec un minimum de nombre d'évaluation de la fonction objective. Cette fonction objective sera en réalité l'analyse d'un mélange chimique dans le laboratoire d'une société à Venise. Nous étudierons le fonctionnement de deux algorithmes d'optimisation utilisant un modèle d'approximation quadratique de la fonction objective, CONDOR et NEWUOA. Ces algorithmes seront appliqués sur une fonction objective mathématique qui servira d'exemple afin d'analyser les performances de ces deux algorithmes. Ensuite, nous chercherons les paramètres d'optimisation des algorithmes qui permettront de converger vers la meilleure solution en un minimum de temps de calcul et de nombre d'évaluation de la fonction objective. Enfin, des développements futurs seront proposés pour le meilleur des deux algorithmes.

TABLE DES MATIÈRES

1. INTRODUCTION

Depuis des années, les biologistes essaient de créer la vie artificielle à partir de matériel biochimique. Une telle étude permettrait aux chercheurs de guérir des maladies comme le cancer en injectant les cellules créées artificiellement dans le corps humain. Celles-ci étant chimiquement stables et ayant les propriétés de guérison requises se diviseraient et se multiplieraient dans le cops humain comme un antivirus. Cependant, de multiples tentatives ont été réalisées jusqu'à présent sans succès. Une nouvelle compagnie localisée au milieu de Venise, *European Center for Living Technology* (ECLT), continue le défi et cherche de nouvelles réponses au problème. Cette entreprise a été établie à Venise avec l'aide de l'Union Européenne, l'université de Ca' Foscari et de la ville de Venise. Elle emploie des scientifiques dans le domaine de la physique et dans le domaine pharmaceutique de renommées internationales. Elle complète également son personnel avec des étudiants doctorants ou stagiaires et des visiteurs.

Ce travail consiste à trouver, comparer et adapter les meilleurs algorithmes d'optimisation disponibles capables de trouver le meilleur mélange des concentrations chimiques avec un minimum de nombre d'évaluation de la fonction objective. Cette fonction objective sera en pratique l'analyse d'un mélange chimique en laboratoire. Ces expériences en laboratoire sont très couteuses en temps et en argent et doivent donc faire objet d'une optimisation approfondie afin d'en effectuer le moins possible et d'en obtenir le plus d'informations possible. Néanmoins, l'algorithme d'optimisation doit être capable de trouver le mélange optimal de concentrations même avec un faible nombre d'évaluations. Chaque expérience contient énormément d'informations et permettra à l'algorithme d'optimisation de prendre la meilleure décision sur les expériences à effectuer par la suite afin de converger au plus vite vers la solution optimale. Trois algorithmes seront étudiés et comparés dans ce travail : CONDOR (Constrained, Non-linear, Direct, parallel Optimization using trust Region method for high-computing load function), NEWUOA (NEW Unconstrained Optimization Algorithm) et un algorithme génétiques couplé à un modèle d'approximation quadratique de ECLT.

Un tel algorithme permettra de raccourcir la période d'étude en laboratoire lors de la conception d'un nouveau médicament grâce à de solides simulations informatiques. Les multiples simulations informatiques permettront également de valider le nouveau médicament plus facilement et de le vendre ainsi plus rapidement sur le marché.

2. DESCRIPTION DE ECLT

Dans ce chapitre, nous étudierons le travail de recherche d'ECLT et nous exposerons le nouveau problème au quel les Italiens font face à présent.

ECLT est un centre étudiant la vie artificielle. La vie artificielle est une technologie dont le caractère essentiel dérive des propriétés du vivant, comme la guérison, la reproduction et la capacité à évoluer. Les demandes en technologies qui utilisent ces principes de la vie sont nombreuses et variées, variant des systèmes biomoléculaires aux technologies de l'information. Ainsi, l'étude de la vie artificielle est utile dans beaucoup de domaines scientifiques, comme les nano-bio-technologies, les systèmes auto-organisateurs, les technologies de production d'information, ainsi que les systèmes complexes adaptatifs. Quelques exemples de résultats de la vie artificielle sont le développement des cellules artificielles et le développement du World Wide Web.

2.1. TRAVAIL D'ÉTUDE DE ECLT

Nous détaillerons à présent plus en détail le travail d'étude qu'effectue ECLT. Leur travail consiste à modéliser des plans d'expériences évolutionnaires pour des systèmes chimiques complexes.

Ils s'intéressent à la modélisation de systèmes chimiques à grandes dimensions et plus particulièrement à ceux dédiés à la génération et au développement de vésicules. Dans cette recherche de la technologie du vivant, ECLT souhaiterait comprendre et recréer la vie artificielle à partir d'une cellule biologique minimale. Afin de créer cette cellule biologique minimale, consistante d'un génome, d'un cytoplasme et d'une membrane, ECLT doit déterminer les structures à molécules les plus utiles dans ce problème. Le problème est de grandes dimensions, car nous aimerions prendre en compte un grand nombre de structures, de variables de processus et leurs interactions entre elles. Il s'est avéré également que les interactions au sein du système évoluaient au cours des expériences consécutives.

Un procédé de synthèse chimique et de mesure en laboratoire a été établi par ECLT et sera décrit plus loin. La turbidité a été déterminée comme la mesure optique la plus pertinente pour l'évaluation de la qualité de la vésicule obtenue. Nous nous intéressons aux interactions de premier ordre et second ordre entre les différents composants du mélange. Une mesure de la moyenne de la turbidité a été effectuée et est affichée sur la Figure 1 (voir [4]). La figure montre l'importance qu'ont certains liens d'interaction sur la valeur de la fonction objective, la turbidité.

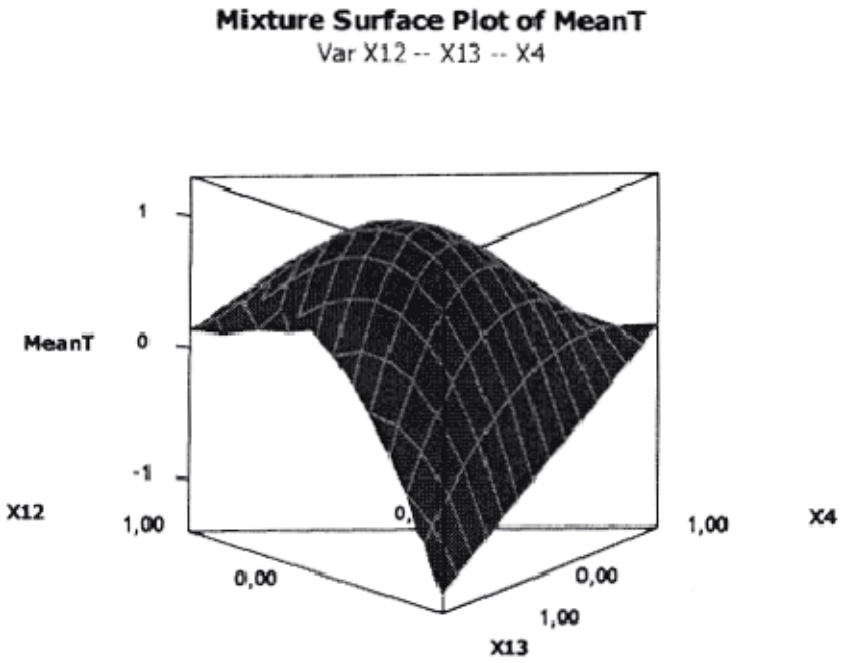


Figure 1: Moyenne de T en fonction des interactions entre composants du mélange

Dans un mélange, nous considérons non pas la quantité totale de la solution, mais la proportion entre les différents composants ainsi que le rôle que pourrait jouer un de ces composants dans une autre solution. Ainsi, la somme des concentrations des différents composants est maintenue à 1. Des bornes inférieures et supérieures peuvent être appliquées sur les concentrations des composants.

2.2. L'ALGORITHME D'OPTIMISATION

Lorsque les molécules amphiphiliques dominantes au problème ont été trouvées (voir [8]), un ensemble de mélanges différents sont proposés et étudiés dans le laboratoire. Cependant, le nombre de molécules existantes et de mélanges possibles est tellement grand qu'il est impossible d'imaginer une étude approfondie de toutes celles-ci en laboratoire. L'espace des solutions est tellement grand qu'ECLT a dû transformer son travail de recherche en l'élaboration d'un algorithme d'optimisation permettant de trouver les zones de solutions les plus prometteuses. L'algorithme qu'ECLT a choisi d'implémenter est un algorithme génétique couplé à un modèle d'approximation quadratique. Les mélanges obtenant les meilleurs résultats sont retenus et utilisés pour les expériences suivantes. Celles-ci sont stockées dans une base de données et forment un ensemble de solutions étudiées grandissantes. Cet ensemble permet d'acquérir une connaissance de l'espace de recherche des solutions et de pouvoir faire des prédictions sur les bonnes et mauvaises solutions à étudier prochainement en laboratoire. Cependant, avant d'effectuer l'évaluation de ces nouvelles solutions, celles-ci sont évaluées sur un modèle mathématique construit à partir des évaluations précédentes de manière à acquérir une certitude quant à la pertinence d'étudier en laboratoire ces solutions sélectionnées par l'algorithme génétique.

Au départ, cet algorithme génétique détermine les ensembles de solutions de manière totalement aléatoire. Par la suite, l'algorithme choisit les solutions en se basant en grande partie et uniquement sur les meilleures solutions obtenues précédemment. Plusieurs techniques, présentes dans les algorithmes génétiques, sont utilisées afin de converger vers la solution optimale. Nous étudierons dans le chapitre 3 comment fonctionnent les algorithmes génétiques et quels sont les inconvénients majeurs qui y sont liés. Plusieurs autres algorithmes d'optimisation seront proposés et étudiés. La suite du travail consistera à analyser les exigences du problème de synthèse chimique et à traduire celles-ci en contraintes d'optimisation dans les algorithmes.

3. LES ALGORITHMES GÉNÉTIQUES (G.A.)

Les algorithmes génétiques sont largement employés dans l'industrie et sont bien connus aux ingénieurs. Ils ont un processus très générique pour approcher le problème d'optimisation et sont faciles à comprendre et à utiliser. Cependant, ils ne permettent pas toujours d'obtenir la meilleure solution de manière efficace et rapide. Ils nécessitent également une étude et un ajustement soigné de leurs paramètres intrinsèques.

Un algorithme génétique utilise un ensemble de solutions qu'il met à jour de manière itérative. Il commence par générer un ensemble initial. Il constitue cet ensemble de solutions en échantillonnant de manière aléatoire tout l'espace de recherche. Lors d'un échantillonnage, la solution est envoyée à une fonction objective. Cette fonction objective retourne une valeur pour cette solution qui correspond à une mesure de la qualité de cette expérience en laboratoire. Ce premier ensemble constitue ce que l'on appelle la première population de solutions. ECLT a adopté une autre approche quant à la génération des nouveaux échantillons. Ils utilisent un générateur de solutions qui garantit que ces solutions sont techniquement réalisables en laboratoire.

Après la génération de cette première population, une nouvelle population est créée en effectuant des opérations sur la population précédente. Les opérations sont les suivantes : sélection, *'cross-over'*, et mutation.

L'opération de **sélection** consiste à sélectionner les meilleurs individus de la population précédente et à garder ceux-ci dans la nouvelle population. Cette idée de sélection est analogue au même processus qui existe dans la nature : la sélection naturelle. Les meilleurs individus restent vivants, tandis que les moins forts meurent.

Les ***'cross-over'*** ou **croisements** forment des enfants à partir de deux individus de la population précédente, considérés comme parents. Ceci nous assure que cet enfant aura les bonnes propriétés des deux parents.

L'opération de **mutation** est la substitution d'un individu par un autre. La mutation sert à éviter une convergence prématurée vers une solution qui n'est pas la meilleure et à ne pas tomber ainsi dans un minimum local de l'espace de recherche. Les probabilités de croisement et de mutation sont des paramètres à définir dans l'algorithme génétique.

Le critère d'arrêt d'un algorithme génétique est défini par l'utilisateur et est généralement une fonction qui évalue l'évolution de l'algorithme, la meilleure valeur de la fonction objective obtenue et du temps parcouru. Une combinaison de tous ces critères est possible.

Bien que les algorithmes génétiques permettent d'obtenir, en théorie, la meilleure solution globale de l'espace de recherche, ils définissent les prochaines solutions à évaluer en exploitant très peu les informations recueillies des individus des populations précédentes. Ainsi, la convergence vers l'optimum global est lente et se fait avec un nombre important d'évaluations de la fonction objective.

4. CONDOR

4.1. INTRODUCTION

Nous aimerions, ayant compris la problématique des Italiens, pouvoir leur fournir un travail d'analyse et d'expertise et ainsi leur fournir un meilleur algorithme d'optimisation afin de trouver le mélange de molécules amphiphiliques optimal permettant de créer la vie artificielle. Le premier algorithme vers le quel notre attention s'est portée est CONDOR, *Constrained, Non-linear, Direct, parallel Optimization using trust Region method for high-computing load function*. Celui-ci a été développé par Dr. Ir. Frank Vanden Berghen lors de son doctorat et a déjà pu se révéler extrêmement performant sur des problèmes industriels. Cet algorithme adopte une approche totalement différente pour trouver la solution optimale que celle adoptée par les algorithmes génétiques. Celui-ci ne détermine pas la prochaine solution à étudier de façon aléatoire, mais utilise un grand nombre d'évaluations précédentes afin de créer une approximation mathématique, un modèle, de l'espace des solutions. Chaque variable peut varier de manière continue sur les abscisses. L'image en tout point de l'espace de toutes les valeurs continues des variables forme ce que nous appelons le paysage, *'fitness'*, de la fonction objective. Ce modèle mathématique découvre ce paysage en l'échantillonnant à des endroits stratégiques. Ces endroits sont choisis de manière à recueillir un maximum d'informations pour améliorer le modèle. Ce modèle, que nous appellerons par la suite « la quadratique » ou « modèle quadratique », suit le meilleur chemin convergent vers un minimum local. Il fait ceci en découvrant la pente décroissante de ce paysage et en suivant celle-ci comme une bille roulante vers le creux d'une vallée (voir Figure 2).

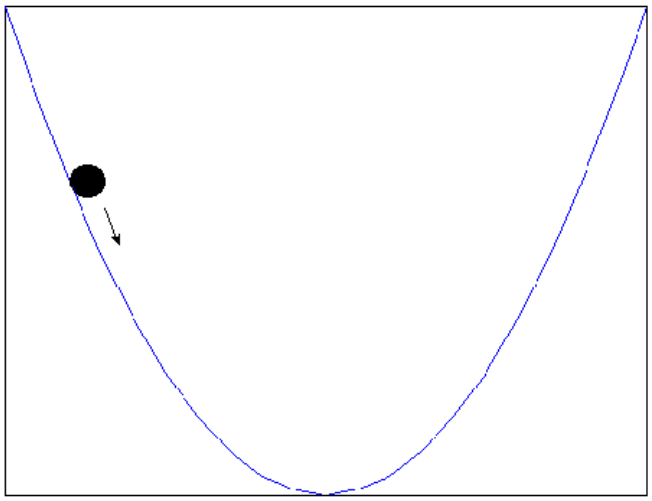


Figure 2 : Chemin menant vers l'optimum local

La fonction objective que nous utiliserons dans l'algorithme d'optimisation ne pourra pas être l'analyse du mélange proposé par l'algorithme dans les laboratoires. Nous utiliserons une fonction mathématique qui permettra de comparer les algorithmes entre eux. Cette fonction mathématique a été choisie de façon à ce que les évaluations de performances effectuées sur CONDOR puissent facilement refléter comment celui-ci se comporterait par rapport au problème chimique réel.

4.2. DESCRIPTION DE CONDOR

CONDOR est une extension d'UOBYQA, un algorithme de M.J.D. Powell, qui converge rapidement vers la solution finale grâce à la découverte de la courbure de la fonction objective. CONDOR est un outil d'optimisation direct, c'est-à-dire que les dérivées premières de la fonction objective ne lui sont pas nécessaires. Il échantillonne l'espace de recherche d'une façon qui lui permet de réduire l'influence du bruit sur la fonction objective. L'algorithme est donc particulièrement efficace lorsque l'évaluation d'un point (une expérience) est couteuse en temps et sujet à un bruit important. CONDOR essaie de minimiser le nombre total d'évaluations de la fonction objective, au prix d'un plus grand travail de calcul interne.

Nous pourrions donner un aperçu du fonctionnement itératif de l'algorithme d'optimisation comme suite :

1. Nous construisons une approximation, un modèle local, de la fonction objective autour du point courant.
2. Nous cherchons le minimum de ce modèle et nous nous déplaçons vers ce nouveau minimum. Ceci est appelé un pas d'optimisation ou *'newton step'*.
3. Nous évaluons la fonction objective en ce nouveau point. Nous reconstruisons le modèle local autour de ce nouveau point en utilisant cette nouvelle évaluation et les évaluations précédentes et nous nous préparons à chercher un nouveau minimum et à nous déplacer (étape 2).

CONDOR a une approche très systématique, rigoureuse et efficace quant à la manière de trouver l'optimum. Comme la plus part des algorithmes d'optimisation basés sur un modèle d'approximation, CONDOR utilise un modèle local construit de polynômes de deuxième degré. Il construit un modèle quadratique en utilisant la technique d'interpolation de Lagrange. Cette technique est particulièrement efficace lorsque les dimensions de l'espace de recherche sont petites (n<100). La courbure de la fonction objective, la dérivée première, est obtenue à partir de ce modèle d'approximation. Ce modèle d'approximation est construit tout en utilisant le moins d'évaluations possible.

CONDOR utilise l'algorithme de Moré et Sorenson pour le calcul de la taille de la *trust region*, que l'on pourrait traduire par « région de validité du modèle ». Cet algorithme permet d'obtenir des résultats numériques stables et très précis.

CONDOR met à jour la quadratique complète à chaque itération et s'assure de sa validité. Pour ce faire, il utilise un ensemble de $N = 1/2(n+1)(n+2)$ points dans le quel il remplace itérativement un point à la fois. Le problème mathématique que nous étudierons par la suite a pour dimensions 100. CONDOR utilisera donc un ensemble de 5151 points pour construire la quadratique complète. Nous verrons que CONDOR convergera rapidement vers un optimum local avec un faible nombre d'évaluations grâce à un modèle très précis, malgré que la construction de ce modèle requière un nombre important de points initiaux.

CONDOR permet également d'exécuter plusieurs évaluations de fonction objective en parallèle et de pouvoir ainsi profiter de toute la puissance de calcul disponible. Ceci lui permet de réduire le temps total d'exécution et d'améliorer la qualité de la quadratique. Avec cette fonctionnalité, CONDOR se place parmi les meilleurs algorithmes d'optimisation lorsque la fonction objective est exigeante en calcul et sujette à un bruit élevé.

Nous pouvons également définir des contraintes d'optimisation dans l'algorithme CONDOR. Il est possible d'expliciter des bornes inférieures et supérieures, des contraintes d'inégalités et d'égalité. La méthode utilisée pour les contraintes linéaires est la *'null space'* méthode. Celle-ci consiste à faire une projection de l'espace de solution de dimension n dans un nouvel espace de recherche de dimension $n-m_a$ sans contraintes, où m_a est le nombre de contraintes actives.

Une dernière fonctionnalité de CONDOR est le maintien d'une base de données des points de l'espace de recherche déjà échantillonnés. Lorsqu'un point est retourné par la fonction objective, sa valeur est enregistrée et stockée dans une base de données afin de ne plus devoir recalculer la valeur de la fonction objective en ce point de l'espace. Cette base de données est gardée pour les exécutions suivantes de l'algorithme à condition que la fonction objective ne change pas. Cette fonctionnalité permet d'améliorer encore plus les performances de CONDOR, car il ne doit pas recalculer les points déjà échantillonnés lors d'une exécution précédente de l'algorithme. Il sera donc possible de construire rapidement le premier modèle d'approximation avec des points échantillonnés lors d'une exécution précédente. Cette fonctionnalité est appelée *hot start* dans CONDOR.

Pour toutes ces raisons, CONDOR nous semble le meilleur algorithme pour le problème de synthèse chimique des Italiens. Ainsi au lieu de construire un modèle pour chaque population de solutions et d'utiliser celui-ci pour valider les prochains points à évaluer en laboratoire, émis par l'algorithme génétique, ECLT pourrait directement utiliser ce modèle pour prédire les prochains points à échantillonner comme le fait CONDOR.

4.3. ÉTUDE DÉTAILLÉE DE CONDOR

CONDOR est un algorithme d'optimisation pour des fonctions objectives non linéaires continues sujet à des bornes supérieures et inférieures, des contraintes linéaires et non linéaires. Supposons n les dimensions de l'espace de recherche. Supposons $f(x)$ la fonction objective à minimiser. Nous voulons trouver le $x^* \in \mathbb{R}^n$ qui satisfait :

$$f(x^*) = \min_x f(x) \text{ sujet à : } \begin{cases} b_l \le x \le b_u, & b_l, b_u \in \mathbb{R}^n \\ Ax \ge b, & A \in \mathbb{R}^{mxn}, b \in \mathbb{R}^m \\ c_i \ge 0, & i = 1, \ldots, l \end{cases} \quad (1)$$

Où x^* est l'optimum de $f(x)$ que nous cherchons sujet aux bornes b_l et b_u, aux contraintes linéaires $Ax \ge b$ et aux contraintes non-linéaires c_i.

Nous définissons la *distance d'échantillonnage* entre deux points de l'espace de recherche par la lettre ρ. Nous n'autorisons pas de grandes valeurs pour ρ car ceci nécessite beaucoup de temps ensuite pour réduire cette distance d'échantillonnage. Nous introduisons Δ, un autre paramètre, définissant la taille de la *trust region* et qui satisfait $\Delta \ge \rho$. L'avantage d'utiliser Δ est d'autoriser celui-ci à faire varier la taille de la *trust region* au dessus de la valeur de la distance d'échantillonnage. Ainsi, nous définissons ρ_{start} et ρ_{end} qui fixent la valeur de la *distance d'échantillonnage* au début et à la fin de l'exécution de l'algorithme. Nous choisissons également un x_{start} qui est la position de départ de l'algorithme.

Nous pouvons à présent donner une explication plus détaillée des étapes du processus itératif de CONDOR :

1. **Initialisation** : Dans un premier temps, il nous faudra recueillir $N = 1/2(n+1)(n+2)$ points de l'espace de recherche de manière à construire la première quadratique. Nous choisissons un point de départ x_{start}, une taille initiale de la *trust region* Δ_{start} et une distance d'échantillonnage initiale ρ_{start}. Au départ,

la taille de la *trust region* Δ_{start} et la distance d'échantillonnage initiale ρ_{start} sont confondues $\Delta_{start} = \rho_{start}$. Ainsi, les points d'échantillonnage sont séparés d'une distance ρ_{start} au départ. Nous définissons k comme l'indice de l'itération en cours de l'algorithme. L'approximation quadratique initiale de $f(x)$ autour de $x_k = x_{start}$ est selon la technique d'interpolation de Lagrange :

$$Q_k(\delta) = f(x_k) + g_k^t \delta + \frac{1}{2}\delta^t B_k \delta \qquad (2)$$

Après cette phase d'initialisation, l'algorithme continue à l'étape 3.

2. **Mise à jour du modèle local** Q_k : Cette mise à jour est uniquement effectuée lorsque nous détectons que Q_k, la quadratique, est dégénérée et qu'elle ne représente plus de manière précise la courbure réelle de la fonction objective $f(x)$. Cette mise à jour requiert que nous échantillonnions la fonction objective $f(x)$ autour de la position x_k pour que nous découvrions sa forme exacte. Les nouveaux points d'échantillonnages ont une distance maximum du point courant x_k de $2\rho_k$.

3. **Calcul du pas** : Nous calculons le pas à effectuer δ_k qui nous conduit vers le minimum du modèle local $Q_k(\delta)$. La grandeur du pas est comprise entre $\frac{1}{2}\rho_k$ et Δ_k. Donc :

$$Q(\delta_k) = \min_{\delta} Q_k(\delta) \text{ et } \frac{\rho_k}{2} < \|\delta_k\|_2 < \Delta_k \qquad (3)$$

4. **Calcul du « degré de conformité »** τ_k **entre** $f(x)$ **et** Q **:** Cette étape permet de vérifier la différence entre la valeur retournée par le modèle local Q_k pour la fonction objective en ce nouveau point et la valeur réelle de la fonction objective $f(x)$. Ce degré de conformité τ_k sera utile pour en déduire, à quel point notre modèle est valide.

$$\tau_k = \frac{f(x_k) - f(x_k + \delta_k)}{Q_k(0) - Q_k(\delta_k)} \qquad (4)$$

5. **Mise à jour de x_k et Δ_k grâce à τ_k** : 3 situations sont définies dans CONDOR correspondant à 3 degrés de conformité du modèle quadratique. Suivant le niveau de conformité de la quadratique par rapport à la valeur réelle de la fonction objective $f(x)$, nous adapterons les paramètres x_k et Δ_k. Voici le tableau représentant les différents niveaux et les changements de paramètres correspondant :

Mauvaise itération	Bonne itération	Très bonne itération
τ_k <0,01	0,01< τ_k <0,9	0,9< τ_k
$x_{k+1} = x_k$ $\Delta_{k+1} = \frac{\Delta_k}{2}$	$x_{k+1} = x_k + \delta_k$ $\Delta_{k+1} = \Delta_k$	$x_{k+1} = x_k + \delta_k$ $\Delta_{k+1} = 2\Delta_k$

Tableau 1 : Degrés de conformité du modèle quadratique

6. **Mise à jour de ρ_k** : Si $\|\delta_k\|_2 < \rho_k$, les pas d'optimisation deviennent petits et nous nous rapprochons du point optimum. Nous devons augmenter la précision de la quadratique $Q_k(\delta)$ afin de converger plus précisément vers cet optimum. Pour ce faire, ρ_k est mis à jour comme suite : $\rho_{k+1} = 0{,}1.\rho_k$. Si toutefois $\|\delta_k\|_2 > \rho_k$, nous sommes toujours à la recherche de la zone contenant un optimum local potentiel et la précision, la distance d'échantillonnage ρ_k, n'est pas modifiée.

7. Nous augmentons k et vérifions si la condition d'arrêt de l'algorithme $\rho_k \leq \rho_{end}$ est atteinte. Si la condition n'est pas satisfaite, l'algorithme continue à l'étape 2.

Dans cette suite d'étapes, le modèle quadratique Q_k nous permet de calculer les pas δ_k à effectuer pour converger vers le minimum x*de la fonction objective $f(x)$. Le pas maximal que l'on peut effectuer à chaque itération est limité par Δ_k. Ce dernier donne le rayon de la *trust region,* la région dans laquelle nous faisons confiance à la validité du modèle d'approximation. Effectivement, le modèle est construit sur une approximation locale et nous ne pouvons certifier sa conformité loin du point courant x_k. Δ_k est ajusté dynamiquement comme décrit à l'étape 5. L'idée étant d'accroitre la région de

confiance, la *trust region*, lorsque le modèle Q_k reflète de manière correcte la fonction objective $f(x)$. Car dans ce cas le modèle nous indique le bon chemin à suivre pour converger vers la solution optimale.

4.4. PERPENDICULAR STEP

Les évaluations sur la fonction objective $f(x)$ sont utilisées à 2 fins différentes :

- Pour faire progresser la recherche *vers* le minimum de $f(x)$ et de trouver ainsi l'optimum local. La quadratique Q_k est *exploitée* afin de progresser dans la bonne direction.
- Pour accroitre la qualité de la quadratique Q_k. L'espace de recherche doit être *exploré* pour éviter que la quadratique Q_k ne dégénère.

Ces 2 utilités des évaluations de la fonction objective sont paradoxales. La quadratique Q_k est à la fois utilisée et améliorée. Il faudra trouver une balance entre l'exploitation de la quadratique Q_k et l'exploration.

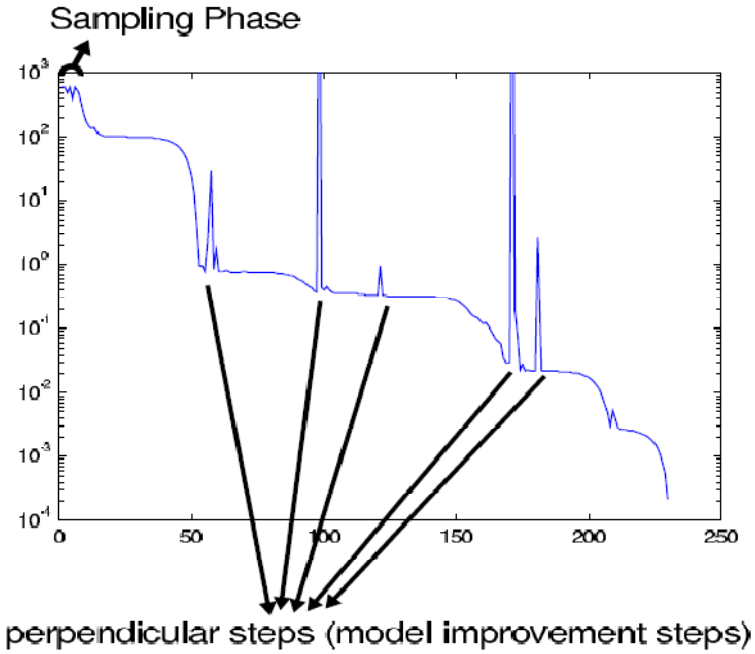


Figure 3 : Perpendicular steps

Voici un exemple d'exécution de CONDOR (voir Figure 3). Sur l'axe x nous observons le nombre d'évaluations et sur l'axe y nous observons la valeur de la fonction objective.

CONDOR commence par échantillonner la fonction objective de façon à construire la quadratique initiale autour de x_{start} . Pendant cette phase, CONDOR ne suit pas un chemin vers l'optimum, mais reste en x_{start} . C'est ainsi que la valeur de la fonction objective reste plus ou moins constante pendant la phase d'échantillonnage.

Une fois cette phase achevée, l'algorithme commence à se déplacer vers l'optimum. Les informations assemblées durant la phase d'échantillonnage sont à présent utilisées pour trouver le chemin vers l'optimum. Ceci explique pourquoi immédiatement après la phase d'initialisation il y a une brusque diminution de la valeur de la fonction objective. À cet instant, la quadratique Q_k se déplace au minimum de la quadratique construite initialement.

Comme dit précédemment, cette phase requiert $N = 1/2(n+1)(n+2)$ évaluations de la fonction objective. Cette phase est donc extrêmement longue lorsque les dimensions sont grandes.

Après cette phase d'initialisation, lorsque la quadratique se déplace dans l'espace de recherche, il nous faudra de temps à autre effectuer un '*perpendicular* step'. Cet échantillonnage permet de s'assurer de la validité de la quadratique.

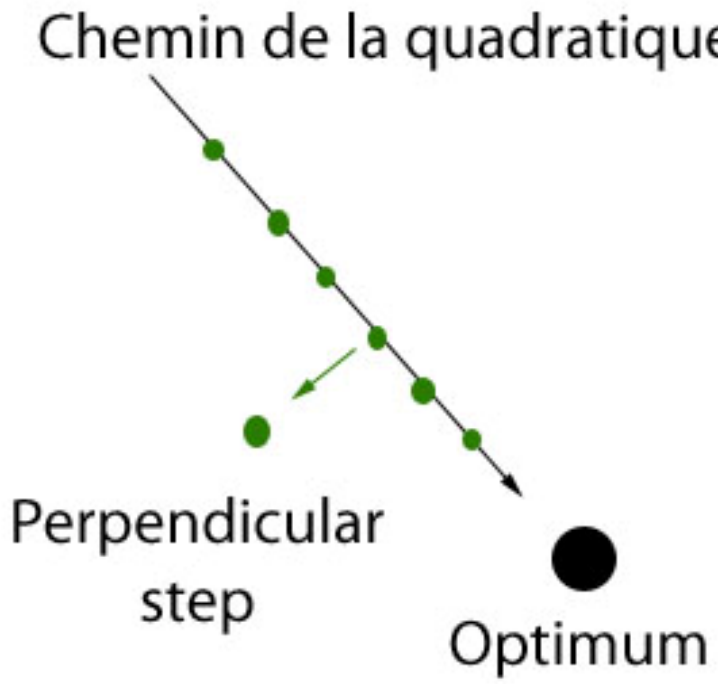


Figure 4: Perpendicular step

La Figure 4 nous explique que lors de la recherche de l'optimum, la quadratique se déplace vers l'optimum sur un chemin plus au moins rectiligne. Si ce chemin converge trop longtemps en ligne droite vers l'optimum, l'ensemble des points faisant partie du jeu d'approximation ne sera presque plus qu'une droite.

À ce moment-là, la quadratique sera l'approximation d'une droite et n'aura plus aucune connaissance de l'espace de recherche environnant. C'est pour éviter cette divergence de la quadratique que l'algorithme effectue de temps à autre un échantillonnage à côté du chemin menant vers l'optimum et continue ainsi à converger de manière optimale vers l'optimum.

4.5. VIOLATION DE CONTRAINTES

Les évaluations de la fonction objective se font 99 % du temps dans l'espace faisable. Cet espace de recherche faisable est délimité par l'ensemble des contraintes. Quelques fois dans CONDOR il faudra explorer l'espace infaisable.

Nous devons acquérir une connaissance de la valeur de la fonction objective autour de l'espace faisable de manière à pouvoir approcher de très près les bords de cet espace si l'optimum s'y trouvait. La quadratique est un nuage de point, *'interpolation set'*, qui se déplace vers l'optimum et dont le centre x_k est toujours situé dans l'espace faisable.

4.6. DISCUSSION DES PARAMÈTRES ET DE LEUR INFLUENCE

ρ_{start}

Les points d'évaluations initiaux sont séparés d'une distance ρ_{start}. Si cette distance ρ_{start} est prise trop petite, nous construisons un modèle quadratique qui ne sera que l'approximation du bruit de la fonction objective.

x_{start}

Les premières évaluations commenceront autour du point de départ x_{start}. Afin de converger vers le point optimum x^*, x_{start} doit être placé suffisamment prêt de l'optimum x^*.

Si le point de départ x_{start} est placé loin de l'optimum x^* nous mettrons beaucoup de temps à trouver l'optimum et nous risquons de tomber dans un minimum local de la fonction objective avant l'optimum x^*.

Néanmoins lorsque nous démarrons l'algorithme très prêt du point optimum x^*, il faudra spécifier une distance d'échantillonnage initiale ρ_{start} petite, autrement l'algorithme fera de grands pas au démarrage qui l'éloigneront de l'optimum x^*.

ρ_{end}

La valeur de la distance d'échantillonnage à la fin de l'algorithme est ρ_{end}, c'est la condition d'arrêt. Nous devons arrêter l'algorithme une fois qu'il n'échantillonne plus que le bruit présent sur la fonction objective. À ce moment-là, l'algorithme est suffisamment près de la solution et ne peut plus trouver une meilleure valeur numérique pour l'optimum à proximité.

4.7. STRUCTURE DU FICHIER DE CONFIGURATION XML POUR CONDOR

Tous les paramètres d'optimisation de l'algorithme sont regroupés au sein d'un fichier XML. L'algorithme d'optimisation lit ce fichier lors du démarrage et s'initialise ainsi. Chaque requête d'évaluation de fonction objective est envoyée à un exécutable spécifié dans ce fichier de configuration. Le fichier XML utilisé dans les différents tests effectués avec CONDOR se trouve en annexe, pour plus d'explications (voir [10]).

4.8. LA FONCTION À OPTIMISER

L'algorithme d'optimisation sera exécuté sur un problème test (voir annexe 'SuperImposedGaussians') de manière à relever les performances de CONDOR. Ce problème test est appelé dans la littérature une '*fitness function*'. Il s'agit d'une super imposition de trois gaussiennes, où la moyenne est définie par la matrice m, où les hauteurs sont définies par le vecteur a, et où les déviations sont définies par le vecteur s.

Cette super imposition est la *maximisation* des trois gaussiennes de dimensions 100. La valeur de la fonction est la somme de la valeur réelle et d'un bruit avec une distribution normale. Ce niveau de bruit est défini par la variable *noise*. Nous voulons que certaines dimensions aient une contribution nulle à certaines des gaussiennes. À cette fin, nous attribuons à ces dimensions la valeur 'NA' dans la matrice des moyennes m. Par exemple, les 3 premières dimensions ont une contribution à une gaussienne tandis que les 97 autres n'ont aucune contribution. Chaque dimension représente la proportion avec laquelle ce composant est présent dans la solution. Les valeurs des coordonnées des points de l'espace de recherche sont comprises entre 0 et 21. Pour obtenir les concentrations de ces composants, il nous faudra diviser leurs coordonnées par 21.

Cette fonction à optimiser présente la même idée que l'optimisation du problème de synthèse chimique du centre de recherche des technologies du vivant. En analysant les performances des algorithmes d'optimisation proposés sur cette fonction exemple, nous pourrons nous forger une bonne idée sur le comportement de l'algorithme dans l'application réelle. Chaque dimension peut être assimilée à la concentration d'un composant dans la solution. Nous autorisons 100 composants à participer potentiellement dans la solution. La quête de l'algorithme d'optimisation sera alors de chercher les valeurs numériques à donner aux différentes concentrations des composants de manière à obtenir la meilleure turbidité.

4.9. GRAPHIQUES DE LA FONCTION

Nous aimerions maintenant étudier cette fonction dans un espace sans bruit. Ces informations nous permettront de comparer les résultats fournis par l'algorithme d'optimisation par rapport aux valeurs réelles prises par la fonction. Nous étudierons ci-après la maximisation de la troisième gaussienne en détail. Pour ce faire, nous varierons la valeur d'une dimension de 0 à 21 en laissant toutes les autres dimensions à une valeur 0. Nous relèverons ainsi les dimensions prédominantes à la *maximisation* de la fonction objective lorsqu'aucune interaction entre dimensions n'est présente.

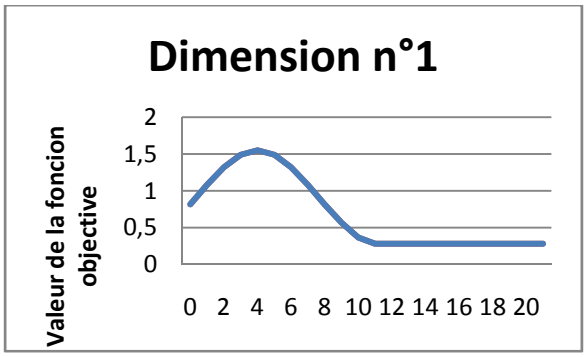


Figure 5:Dimension n°1

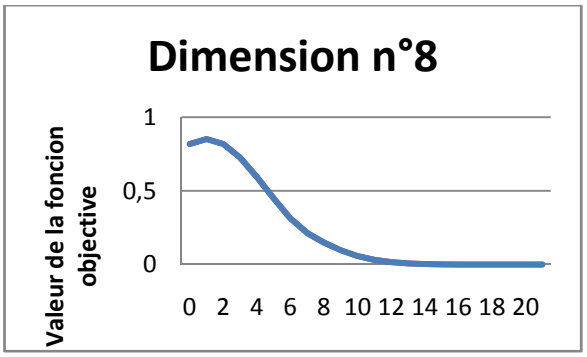


Figure 6:Dimension n°8

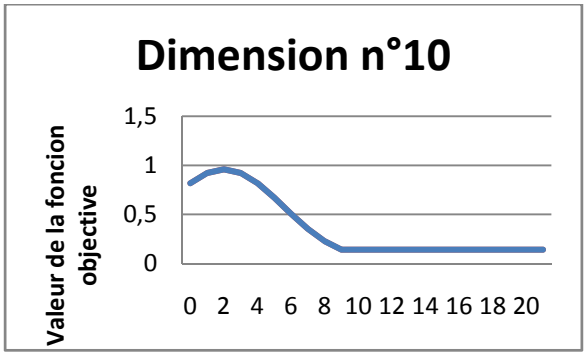


Figure 7 : Dimension n°10

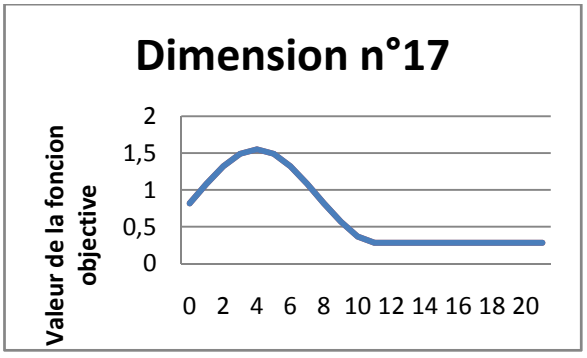


Figure 8 : Dimension n°17

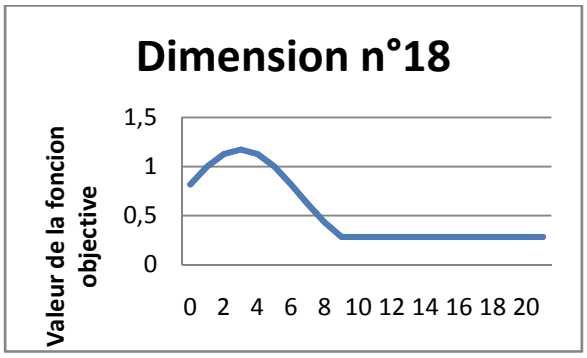


Figure 9 : Dimension n°18

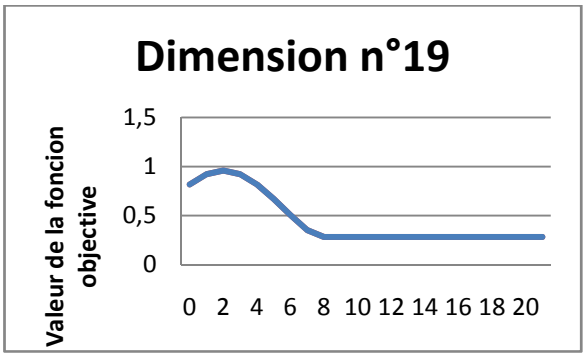


Figure 10 : Dimension n°19

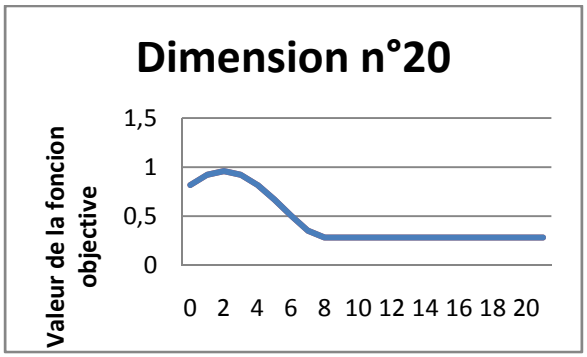


Figure 11 : Dimension n°20

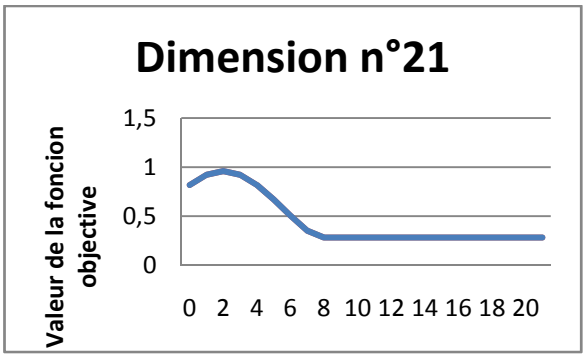


Figure 12 : Dimension n°21

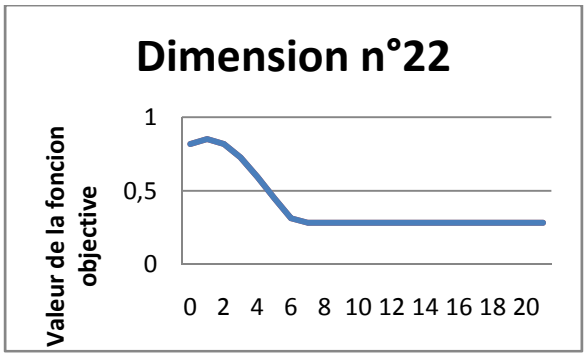


Figure 13 : Dimension n°22

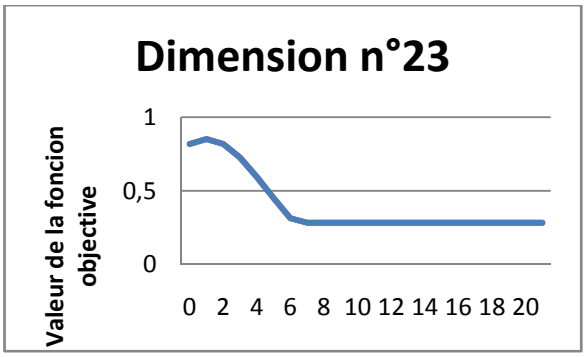


Figure 14 : Dimension n°23

Dans cette fonction exemple, 24 dimensions sont actives. Toutes les autres ont une valeur 'NA'. Toutes les dimensions n'ont pas une influence égale sur la fonction objective. Les dimensions non présentées ont une valeur constante pour la fonction objective de 0,8. Leur présence seule n'a aucune influence sur la valeur de la fonction objective. Nous pouvons voir sur les graphiques ci-dessus que 10 dimensions parmi les 24 ont une influence sur la fonction objective. Au plus, leur valeur est importante, au plus elles tendent à faire diminuer la valeur de la fonction objective.

Qu'elle serait la valeur de la fonction objective lorsque nous faisons varier de manière continue toutes les dimensions ? À ce moment-là, des interactions de second ordre et supérieures entre les dimensions apparaitront.

Si nous ajoutons une contrainte sur la somme des dimensions, ces dimensions se comporteront comme des composants d'une réaction chimique. Ces composants ensemble auront tendance parfois à soit augmenter la valeur de la fonction objective ou à la faire diminuer. CONDOR nous permettra de déceler les dimensions ayant une contribution positive lorsqu'elles sont seules ou lorsqu'elles ont une interaction de second ordre avec les autres dimensions.

4.10. PREMIÈRES SIMULATIONS EFFECTUÉES AVEC CONDOR

Nous effectuerons d'abord quelques simulations sur le problème de manière à comprendre le fonctionnement de CONDOR. Ces simulations nous permettront également d'avoir un ensemble d'optimums locaux qui serviront de référence dans la recherche vers l'optimum.

Nous avons porté notre intérêt sur la possibilité d'exécuter plusieurs clients en parallèle de CONDOR. 4 instances de l'algorithme d'optimisation ont été exécutées sur 2 machines différentes. Cette fonctionnalité s'est avérée extrêmement efficace dans la recherche de l'optimum global. Le processus hôte se charge de répartir la phase initiale d'exploration sur les 4 clients disponibles. Une fois cette phase d'exploration terminée, le premier modèle d'approximation quadratique est construit par le processus hôte.

Les premières simulations ont été lancées avec les paramètres d'optimisation suivants :

$$x_{start} = (1,1,1, rep(97,0))$$

$$\rho_{start} = 0,1 \qquad (5)$$

$$\rho_{end} = 1e-3$$

Où $rep(97,0)$ représente une succession de 97 zéros.

Ces premières simulations ont permis d'obtenir une valeur pour la fonction objective $f(x)$ de $0,36$. Nous observions que l'algorithme se déplaçait que peu du point de départ x_{start}. Il s'avérait donc que les simulations s'arrêteraient dans un optimum local proche du point de départ x_{start}.

Quel paramètre de l'algorithme pourrions-nous changer afin de converger vers une meilleure solution ? Nous pouvons envisager soit que le point de départ x_{start} ne fût pas

bien choisi, soit que nous n'autorisions pas l'algorithme à se déplacer suffisamment au départ à cause d'une distance d'échantillonnage ρ_{start} trop petite ou encore que nous rencontrions le critère d'arrêt ρ_{end} trop vite.

Nous avons d'abord envisagé d'augmenter ρ_{end} dans le but d'augmenter le temps de recherche total et d'obtenir un résultat plus proche de l'optimum global. Ce choix nous a permis d'obtenir une meilleure valeur pour la fonction objective $f(x)$. En contrepartie, l'algorithme nécessitait plus de temps pour converger vers la solution finale.

L'algorithme convergeait à présent vers une valeur de $7{,}09$ pour la fonction objective avec les paramètres d'optimisation suivants :

$$x_{start} = (1,1,1, rep(97,0))$$

$$\rho_{start} = 0{,}1 \qquad (6)$$

$$\rho_{end} = 1e-7 \ \rho_{start}$$

La quadratique initiale Q_k est constituée de 5151 points d'échantillonnage. Ces évaluations sont effectuées en parallèle sur toutes les instances de CONDOR. Ensuite, les résultats sont rapatriés vers le processus hôte. Ce dernier se charge de construire la quadratique initiale Q_k en utilisant la technique d'interpolation de Lagrange. La création de ce premier modèle est exécutée sur l'ordinateur hôte de manière séquentielle et nécessite 3 h de calcul.

Le temps total d'exécution de l'algorithme est principalement consommé par la construction de la quadratique initiale Q_k et non à l'échantillonnage de l'espace de recherche. L'évaluation d'un point x_k de l'espace de recherche est dans notre problème test peu exigeant en temps de calcul. En contrepartie, le travail de gestion en arrière-plan demande beaucoup de ressource.

4.11. SIMULATIONS APPROFONDIES EFFECTUÉES AVEC CONDOR

Nous voulons à présent trouver d'autres optimums locaux ayant une meilleure valeur pour la fonction objective. Nous étudierons également plus précisément l'influence des différents paramètres et les valeurs précises à donner à ceux-ci. À cette fin, nous étudierons l'influence du changement d'un des paramètres suivants : x_{start} et ρ_{start}. La valeur du critère d'arrêt sera maintenue égal à $\rho_{end} = 1e-3$ dans toutes les prochaines simulations.

Toutes les prochaines simulations ne seront plus exécutées en parallèle sur plusieurs processus de CONDOR comme précédemment. Nous avons détecté que lors des premières simulations des erreurs se produisaient lors de la communication entre les différents processus. Beaucoup d'évaluations exécutées sur les processus clients échouaient. Nous avons donc choisi pour nous assurer d'obtenir des résultats corrects de ne plus exécuter CONDOR que sur un seul processus.

4.11.1. INFLUENCE DU POINT DE DÉPART

Nous chercherons à présent le *point de départ* x_{start} le plus apte à nous amener vers l'optimum global ou vers toute solution meilleure que celles trouvées précédemment. Les dimensions de l'espace sont maintenues à 100, la fonction objective $f(x)$ utilisée est la fonction de super imposition dans la quelle nous utilisons la troisième matrice des moyennes et le niveau de bruit sur la fonction objective est maintenu à 5 %.

Lors de chaque simulation, nous changerons le point de départ x_{start} de manière systématique de la façon suivante : puisque seules les 24 premières dimensions ont une contribution à la maximisation de la fonction objective $f(x)$, nous distribuerons les 21 unités disponible sur ces dimensions. À chaque simulation, la valeur 10 sera attribuée à deux autres dimensions dans le but de voir leur tendance à fournir une meilleure solution finale. Les résultats des simulations sont les suivants :

Essai	Point de départ	Max FO	NEF
1	(rep(100,0))	5,73	5221
2	(10,10, rep(98,0))	4,86	5455
3	(rep(2,0), 10, 10, rep(96,0))	3,19	5185
4	(rep(4,0), 10, 10, rep(94,0))	6,55	5430
5	(rep(6,0), 10, 10, rep(92,0))	3,03	5293
6	(rep(8,0), 10, 10, rep(90,0))	6,90	5192
7	(rep(10,0), 10, 10, rep(88,0))	1,10	5347
8	(rep(12,0), 10, 10, rep(86,0))	1,52	7630
9	(rep(14,0), 10, 10, rep(84,0))	4,14	6134
10	(rep(16,0), 10, 10, rep(82,0))	4,35	5153

Tableau 2 : Influence du point de départ

Nous trouvons dans le tableau ci-dessus (voir Tableau 2) le point choisi comme point de départ x_{start} (*Point de départ*), la valeur maximale trouvée pour la fonction objective (*Max FO*) et le nombre d'évaluations effectuées (*NEF*) lors de chaque simulation.

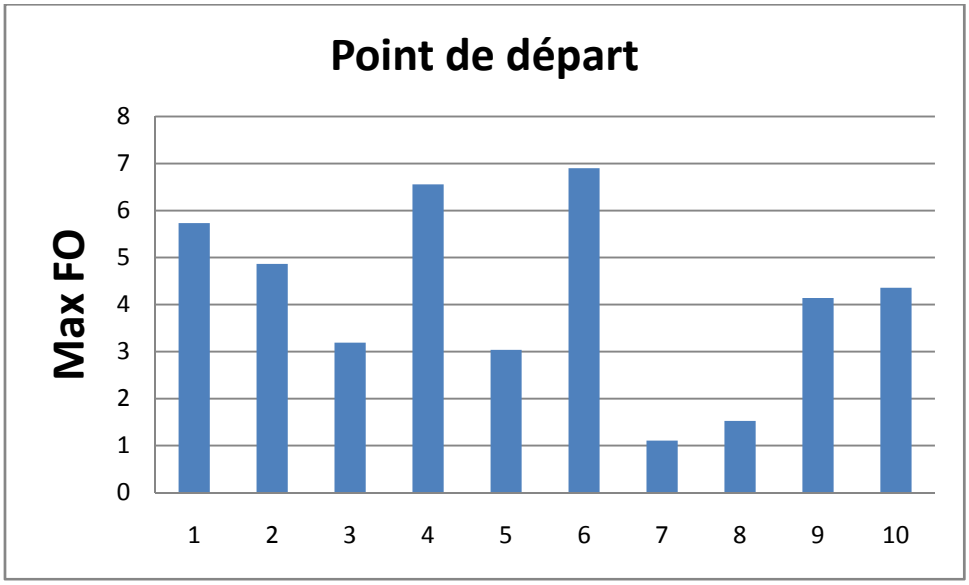


Figure 15: Influence du point de départ

La Figure 15 représente la valeur maximale trouvée pour $f(x)$ au cours de chaque simulation. En abscisse, nous trouvons le numéro de chaque simulation correspondant au Tableau 2.

Nous remarquons que dans les simulations ci-dessus (voir Figure 15) tous les points de départ x_{start} ne permettent pas d'atteindre le même résultat final. Le choix du point de départ x_{start} est donc important. Le point de départ x_{start} doit être choisi non loin de l'optimum. Les simulations 4 et 6 donnent des points finaux x^* et des valeurs maximales pour la fonction objective $f(x)$ semblables.

Le bruit qui fait partie de la fonction objective produit un résultat différent alors que ces deux simulations donnent approximativement le même point optimal final x^*. Le niveau de bruit présent est égal à 5 % de la valeur de la fonction objective $f(x)$ avant ajout du bruit. Par exemple, 5 % de bruit sur une valeur de 6,9 peut provoquer une différence de 0,345 (voir simulation 6).

Une deuxième raison des différences entre les deux simulations est la suivante : une fois qu'un point x_k est envoyé pour évaluation à la fonction objective $f(x)$, sa valeur est reçue en retour et enregistrée pour plus tard afin d'éviter de recalculer la valeur de la fonction objective en ce point x_k. La valeur de ce point x_k et le bruit présent sur cette valeur sont donc « figés » une fois calculés. La valeur enregistrée peut-être avantageuse si le bruit présent avait une valeur positive ou tout au contraire désavantageuse si le bruit réduisait la valeur de la fonction objective. De ce fait, une simulation peut avoir un meilleur résultat numérique qu'une autre alors que qu'elles obtiennent le même point final x^*.

Parfois, la présence du bruit de la fonction objective $f(x)$ fait en sort que l'algorithme ne suit pas le même chemin, même si nous partons du même point de départ x_{start}. Le modèle d'approximation quadratique interpole la valeur de la fonction objective aux différents points échantillonnés de l'espace de recherche. Si ces valeurs sont différentes à cause du bruit, nous construisons des modèles d'approximation différents lors de chaque simulation. De ce fait, l'algorithme ne suivra pas le même chemin et trouvera des optimums locaux distincts.

4.11.2. INFLUENCE DE LA DISTANCE D'ÉCHANTILLONNAGE INITIALE EN DÉMARRANT PRÈS DE L'OPTIMUM

Dans cette section, tout comme dans la précédente, nous étudierons le changement d'un paramètre d'optimisation. Celui-ci sera la *distance d'échantillonnage initiale* ρ_{start}. La meilleure valeur connue pour la distance d'échantillonnage initiale jusqu'à présent était $\rho_{start} = 0{,}1$. Si nous lui donnons une valeur trop petite, nous risquons d'échantillonner que le bruit de la fonction objective $f(x)$, de construire un mauvais modèle d'approximation et de nous arrêter dans un minimum local. Si nous prenons une valeur trop grande pour la distance d'échantillonnage ρ_{start}, nous n'aurons qu'un aperçu de l'ensemble de l'espace de recherche et nous ne pourrons connaitre le chemin qui mène vers l'optimum global. Dans le meilleur des cas, il nous faudra un nombre important d'évaluations de la fonction objective et un long temps d'exécution avant d'atteindre l'optimum global. L'ajustement de ce paramètre est donc très important dans la recherche vers l'optimum global.

La distance d'échantillonnage initiale ρ_{start} est un paramètre dont la valeur est relative par rapport à la valeur minimale, 0, et maximale, 21, qu'une dimension peut prendre. Par exemple, une valeur de $\rho_{start} = 0{,}1$ autorise un pas d'une distance maximale de $0{,}1.21 = 2{,}1$ au départ. Cette limitation est imposée pour prévenir que l'algorithme n'échantillonne des points trop loin de la zone que nous considérons comme prometteuse. Bien sûr, il conviendra à mesure que nous acquerrons de plus amples points d'échantillonnages valides, d'augmenter le rayon de la *trust region* Δ.

Le point de départ x_{start} sera maintenu constant pour toutes les simulations. Nous lui donnerons les coordonnées du point final x^* de la sixième simulation (voir Tableau 2) de la section précédente. Ce point est la meilleure solution trouvée jusqu'à présent et a les coordonnées suivantes :

Coordonnée	Valeur Simulation n°6
1	0
2	3,82
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	5,1
11	1,2
12	0
13	0
14	0
15	0
16	0
17	0
18	3,2
19	2,6
20	2,1
21	1,3
22	0
23	1,4
24	0
Valeur FO	6,90

Tableau 3 : Coordonnées du point final de la sixième simulation

Essai	Distance d'échantillonnage initiale	Max FO	NEF
1	0.05	7,16	5151
2	0.1	7,24	5586
3	0.2	8,30	5153
4	0.3	7,30	5687
5	0.4	7,23	5356
6	0.5	7,28	6014

Tableau 4 : Influence de la distance d'échantillonnage initiale

Ce tableau récapitule la valeur donnée au paramètre d'optimisation ρ_{start} (*Distance d'échantillonnage initiale*), la meilleure valeur trouvée pour la fonction objective (*Max FO*) et le nombre d'évaluations de fonction objective effectuées (*NEF*) pour chaque simulation.

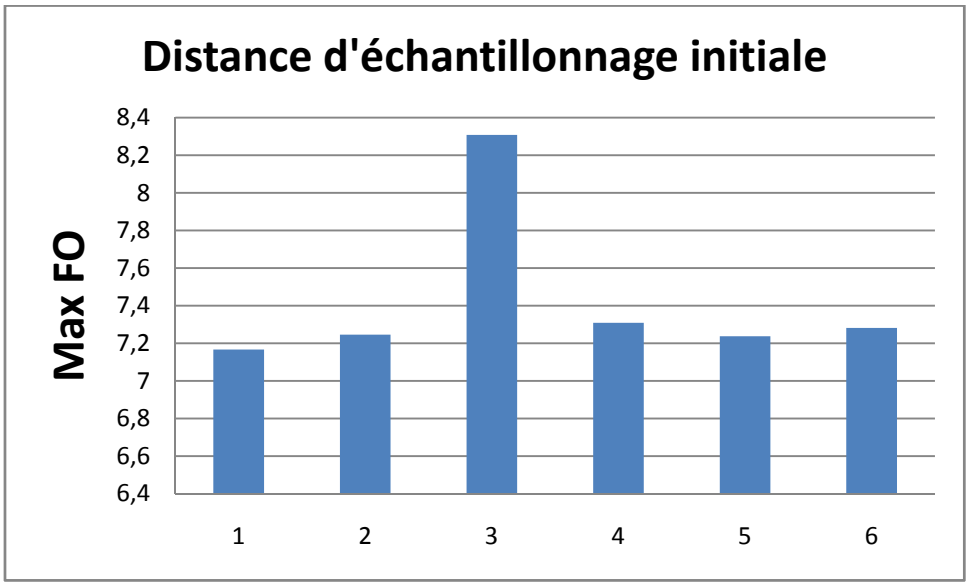


Figure 16 : Influence de la distance d'échantillonnage initiale

La Figure 16 représente la valeur maximale trouvée pour $f(x)$ au cours de chaque simulation. En abscisse, nous trouvons le numéro de chaque simulation correspondant au Tableau 4

Nous observons dans le tableau ci-dessus (voir Tableau 4) que toutes les simulations (sauf simulation nº 3) obtiennent des valeurs maximales similaires pour la fonction objective. Lors de la première simulation, l'algorithme ne s'est pas déplacé du tout du point de départ x_{start}, car la distance d'échantillonnage $\rho = 0,05$ était trop petite et l'algorithme reste ainsi coincé dans l'optimum local trouvé précédemment. En effectuant d'autres simulations avec des valeurs plus importantes pour la distance d'échantillonnage, nous nous sommes rendu compte que là aussi l'algorithme ne se déplaçait pas de la solution optimale trouvée précédemment. D'une part, cela certifie que l'algorithme a trouvé un point optimal x^* et ne souhaite plus se déplacer à présent de ce point, d'autre part il ne nous est donc plus possible de démarrer une exploration de l'espace de recherche à partir de cet optimum local. Enfin, nous avons découvert qu'en démarrant l'algorithme à partir d'un optimum local, il était possible d'obtenir des valeurs numériques plus précises des coordonnées de l'optimum local.

La simulation nº 3 a pu obtenir de meilleurs résultats par chance en échantillonnant un point de l'espace de recherche avec une valeur de bruit avantageuse. Ce point a été découvert et enregistré durant la phase de construction initiale de la quadratique. Plus aucun autre point de l'espace de recherche n'a pu avoir une meilleure valeur pour la fonction objective par la suite.

4.11.3. Influence de la distance d'échantillonnage initiale en démarrant loin de l'optimum

Nous voulons à présent découvrir quelle valeur pour la distance d'échantillonnage initiale ρ_{start} nous permet à partir d'un point de départ quelconque de l'espace de recherche de trouver une solution aussi bonne que celle trouvée jusqu'à présent ou une solution meilleure. L'algorithme pourrait également retrouver le même point optimal x^* pour une certaine valeur de la distance d'échantillonnage initiale ρ_{start}.

Nous choisissons comme point de départ x_{start} le point zéro de l'espace, qui est le même que le point de départ x_{start} de la première simulation de la section 4.11.1. Ce point est loin de tout optimum local x^*, car toutes ses dimensions ont la valeur zéro. L'algorithme

devra choisir les dimensions pertinentes à optimiser et changer leurs valeurs pour maximiser la fonction objective $f(x)$. Nous espérons trouver une valeur pour la fonction objective au moins égale à 5,73, qui est la meilleure valeur de la fonction objective trouvée parmi les simulations de la section 4.11.1.

Essai	Distance d'échantillonnage initiale	Max FO	NEF
1	0.05	6,01	5272
2	0.1	7,18	5699
3	0.2	5,31	5623
4	0.3	8,32	6340
5	0.4	7,07	5613
6	0.5	4,84	5341

Tableau 5 : Influence de la distance d'échantillonnage initiale

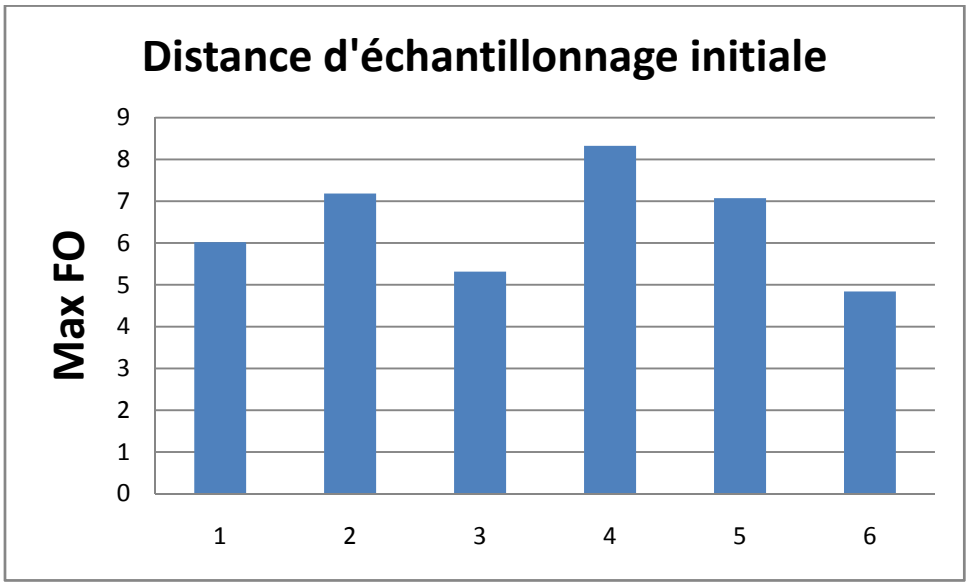


Figure 17 : Influence de la distance d'échantillonnage initiale

La Figure 17 représente la valeur maximale trouvée pour la fonction objective au cours de chaque simulation. En abscisse, nous trouvons le numéro de chaque simulation.

Toutes les simulations ne permettent pas d'atteindre la solution optimale trouvée précédemment. Nous observons que l'essai 2 et l'essai 4 obtiennent les meilleurs résultats (voir Tableau 5). En changeant la distance d'échantillonnage ρ_{start} l'algorithme d'optimisation échantillonne d'autres points de l'espace de recherche et construit ainsi des modèles d'approximation différents. Une valeur pour la distance d'échantillonnage initiale entre $\rho_{start} = 0,1$ et $\rho_{start} = 0,3$ permet donc d'obtenir les meilleurs résultats. Si nous prenons une valeur trop petite, comme dans le premier essai, l'algorithme construit une approximation locale et n'échantillonne que le bruit de la fonction objective $f(x)$. À l'inverse, si la valeur est prise trop grande, l'algorithme construira un modèle d'approximation de l'ensemble de l'espace et nécessitera beaucoup de temps pour converger vers la solution optimale x^*. De plus, l'algorithme ne découvre pas l'optimum x^* trouvé précédemment lorsque la distance d'échantillonnage initiale est trop grande et s'arrêtera dans un optimum local comme dans le cinquième essai.

La quatrième simulation est la simulation obtenant le meilleur résultat final pour la fonction objective et ayant choisi comme point de départ x_{start} le point zéro de l'espace. Le Tableau 6 contient les coordonnées du point x^* ayant la meilleure valeur pour la fonction objective trouvée jusqu'à présent et les coordonnées du nouveau point optimum x^* trouvé lors de ces nouvelles simulations. Lors de ces derniers essais, le point optimum x^* avec la meilleure valeur pour la fonction objective a été obtenu lors de la quatrième simulation.

Coordonnée	Ancienne Valeur	Nouvelle Valeur simulation n°4
1	0	0,01
2	3,82	3,03
3	0	0
4	0	0
5	0	0
6	0	0,09
7	0	0,02
8	0	0
9	0	0,80
10	5,1	0
11	1,2	2,12
12	0	0
13	0	0
14	0	0,02
15	0	0
16	0	0
17	0	0
18	3,2	4,54
19	2,6	3,37
20	2,1	1,92
21	1,3	1,62
22	0	0,85
23	1,4	1,24
24	0	1,07
Valeur FO	6,90	8,32

Tableau 6 : Coordonnées du point final de la quatrième simulation

La deuxième simulation effectuée dans cette section est la même que la première de la section 4.11.1. Néanmoins, la valeur maximale obtenue pour la fonction objective est

fort différente entre ces deux simulations. Ceci est donc uniquement dû au bruit présent dans la valeur de la fonction objective. Les deux simulations ont reçu les mêmes paramètres d'optimisation et pourtant elles obtiennent des optimums locaux x^* différents. Les points d'évaluations des deux simulations sont distincts et ainsi par interpolation de ces points, les deux modèles d'approximation ne sont pas égaux et ne permettent pas de trouver les mêmes résultats.

Dans la prochaine section, nous étudierons la stabilité de la valeur finale de la fonction objective en gardant les mêmes paramètres d'optimisation. Nous découvrirons ainsi quelle sera la tendance de l'algorithme à trouver dans chaque simulation le même résultat.

4.11.4. ROBUSTESSE DE L'ALGORITHME PAR RAPPORT AU BRUIT

Il convient à présent de mesurer quelle est l'influence du bruit présent dans la fonction objective $f(x)$ sur la valeur finale de la fonction objective. L'algorithme d'optimisation sera lancé plusieurs fois depuis un même point de départ x_{start} et tous les paramètres d'optimisation seront gardés constants.

Lors de chaque simulation, les valeurs échantillonnées aux différents points de l'espace pour la fonction objective $f(x)$ varieront. Le modèle d'approximation construit par interpolation de ces points diffèrera pour chaque simulation et fournira ainsi d'autres valeurs maximales pour la fonction objective $f(x)$. Néanmoins, nous espérons que cette variance sur la valeur maximale de la fonction objective $f(x)$ est minime puisque nous démarrons chaque simulation avec les mêmes paramètres d'optimisation.

Les paramètres d'optimisation choisis pour les simulations sont les suivants :

$$x_{start} = (rep(100, 0))$$

$$\rho_{start} = 0{,}1 \qquad (7)$$

$$\rho_{end} = 1e-3$$

Essai	Max FO	NEF
1	7,01	5196
2	5,50	5284
3	7,27	7603
4	6,35	7622

Tableau 7 : Robustesse de l'algorithme

Figure 18 : Robustesse de l'algorithme

Le tableau ci-dessus nous indique que chaque simulation arrive à trouver le bon chemin vers l'optimum (voir Tableau 7). Mais toutes les simulations convergent vers des optimums locaux ayant des coordonnées différentes et ont des valeurs pour le maximum de la fonction objective $f(x)$ fort différentes. Toutes les simulations n'obtiennent pas les mêmes résultats numériques pour les optimums locaux x^*. Elles semblent s'arrêter dans un point x_k proche de l'optimum x^* trouvé jusqu'à présent (voir section 4.11.3).

Pour ce faire, il faudrait relancer l'algorithme à la fin de chaque simulation à partir du point optimal x^* trouvé en fin de simulation afin d'affiner les coordonnées de ce point. Lorsque nous relançons l'algorithme, il faut prendre soin d'effacer la base de données contenant les points calculés lors d'une simulation précédente, sans quoi nous répéterons la simulation avec les mêmes valeurs pour la fonction objective.

Coordonnée	Ancienne Valeur	Ancienne Valeur	Nouvelle Valeur simulation n°3
1	0	0,01	0,33
2	3,82	3,03	4,04
3	0	0	0,04
4	0	0	0
5	0	0	0
6	0	0,09	0
7	0	0,02	0,32
8	0	0	0
9	0	0,80	0,80
10	5,1	0	0,38
11	1,2	2,12	1,34
12	0	0	0
13	0	0	0
14	0	0,02	0
15	0	0	0,01
16	0	0	0
17	0	0	0
18	3,2	4,54	3,85
19	2,6	3,37	1,40
20	2,1	1,92	0,95
21	1,3	1,62	1,22
22	0	0,85	0,56
23	1,4	1,24	0,75
24	0	1,07	0,82
Valeur FO	6,90	8,32	7,27

Tableau 8 : Coordonnées du point final de la troisième simulation

Le Tableau 8 compare les valeurs numériques des coordonnées des optimums locaux x^* trouvés précédemment avec les coordonnées du point final de la meilleure simulation de

cette section, la troisième simulation. Ces coordonnées des différents optimums varient peu numériquement. Nous constatons que ce sont les mêmes dimensions (2, 11, 18, 19, 20, 21, 23) pour toutes les simulations affichées dans le Tableau 8 qui ont un rôle prépondérant dans la maximisation de la valeur de la fonction objective.

Certaines de ces dimensions avaient une contribution positive lorsqu'aucun autre composant et aucune interaction n'étaient présents (voir section 4.9). Ce sont les dimensions 10, 18, 19, 20, 21 et 23. D'autres composants, qui avaient aussi une influence positive sur la valeur de la fonction objective lorsqu'ils étaient seul présent dans la solution, n'ont plus aucune contribution positive à présent, lorsqu'ils sont en interaction avec d'autres composants et ont une valeur zéro. Ces composants sont les suivants ; 1, 8, 17.

4.11.5. ÉVOLUTION DE LA VALEUR DE LA FONCTION OBJECTIVE

La Figure 19 montre l'évolution typique de l'algorithme CONDOR lors de l'optimisation de la fonction objective $f(x)$. En abscisse, nous avons le nombre total d'évaluations effectuées de la fonction objective $f(x)$ depuis le démarrage de l'algorithme et en ordonnée la valeur maximale trouvée jusqu'à présent pour la fonction objective.

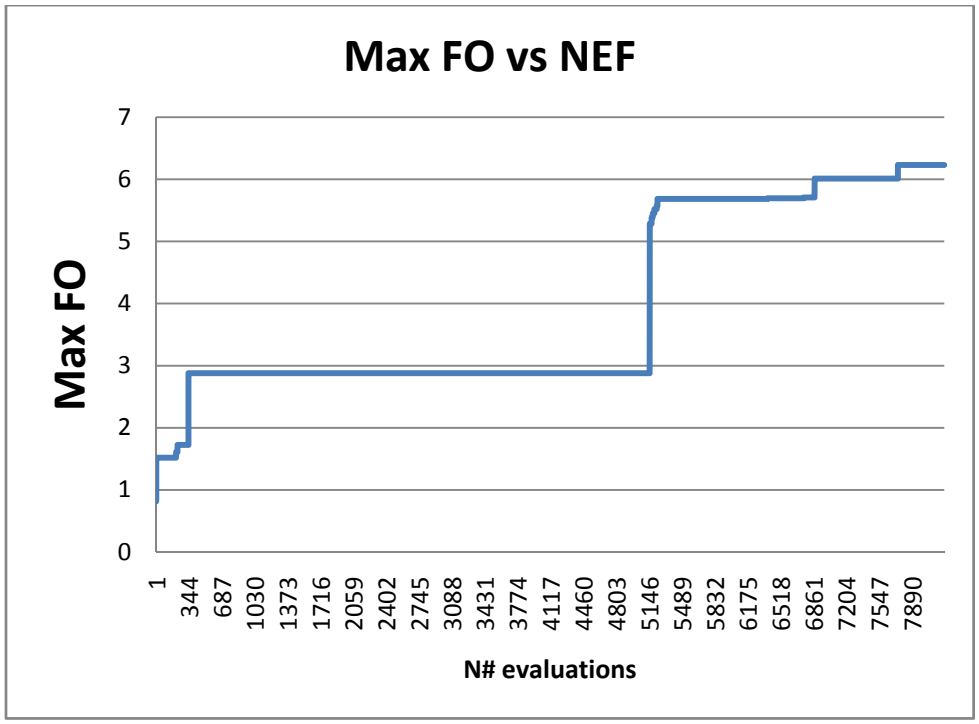


Figure 19 : Évolution de la valeur de la fonction objective avec le nombre d'évaluations

Nous constatons qu'au cours des 5100 premières évaluations, l'algorithme a une valeur constante pour la valeur de la fonction objective. Ceci est la phase de démarrage initiale de l'algorithme. Durant cette phase, CONDOR découvre l'espace de recherche. Il échantillonne un ensemble de points x_k de l'espace de manière à construire la première quadratique. Il est donc logique que l'algorithme ne se concentre pas à trouver de meilleures valeurs pour la fonction objective. Au début, l'algorithme trouvera des bonnes valeurs pour la fonction objective par hasard.

Après 5151 évaluations, l'algorithme a construit la première quadratique et commence à l'exploiter dès cet instant-là. Nous remarquons que l'algorithme se déplace au minimum de cette quadratique. Ce *minimum* de la quadratique correspond dans notre cas exemple à une *maximisation* de la valeur de la valeur de la fonction objective. Il se produit donc un saut de discontinuité représentant l'instant où l'algorithme commence à exploiter les informations retirées des évaluations précédentes. Ensuite, l'algorithme se déplace encore, certes peu, à la recherche d'un optimum local x^*. Il trouve rapidement un optimum local grâce à la qualité de la quadratique construite lors de la première phase.

Bien que cette phase initiale soit avantageuse afin d'obtenir un modèle quadratique complet, elle requiert un grand nombre d'évaluations avant que l'algorithme ne puisse commencer sa recherche vers l'optimum. Ceci est une limitation importante dans le cas de notre fonction exemple qui a 100 dimensions. À 100 dimensions, il nous faut 5151 évaluations avant que l'algorithme ne puisse prendre une décision sur le chemin à suivre et nous fournir des résultats.

Lors des simulations sur machine avec CONDOR, cette phase initiale se décomposait en deux étapes : l'échantillonnage des 5151 points de l'espace de recherche et l'exploitation de ces informations afin de construire la première quadratique. La première étape nécessite 3 h de temps de calcul et la deuxième étape consomme 3 h de plus avant de trouver un optimum local.

Nous étudierons et comparerons dans le paragraphe suivant un autre algorithme alternatif, NEWUOA, qui n'a pas l'inconvénients de nécessiter un modèle quadratique *complet* avant de procéder à l'exploitation de cette quadratique. Il nous faudra rechercher les avantages et inconvénient de l'utilisation de cette quadratique « incomplète » et l'évolution de la valeur de la fonction objective au cours des évaluations.

5. NEWUOA

5.1. DESCRIPTION DE NEWUOA

NEWUOA, *NEW Unconstrained Optimization Algorithm*, est un algorithme développé par M.J.D. Powell en 2004 après CONDOR. Cet algorithme était écrit en Fortran et a été converti en langage C pour ce travail. NEWUOA est un algorithme efficace pour les fonctions objectives à grandes dimensions. Il permet une convergence rapide des algorithmes d'optimisation itératifs sans contraintes. Tout comme CONDOR, il cherche le minimum d'une fonction objective $f(x)$. L'algorithme crée un modèle d'approximation, une quadratique Q_k, de la fonction objective et met ce modèle à jour itérativement. Cette quadratique interpole un ensemble de points x_k échantillonnés de la fonction objective. La différence avec CONDOR est que NEWUOA ne requiert pas un nombre précis de points pour la construction de cette quadratique. CONDOR nécessite un modèle quadratique complet de $1/2(n+1)(n+2)$ points. Nous avons pu constater que ce grand nombre d'évaluations initiales était très couteux en temps et en calcul avec CONDOR. NEWUOA laisse ce choix à l'utilisateur. Le nombre de points de la quadratique doit cependant être compris dans l'intervalle :

$$n+2 \le m \le \frac{1}{2}(n+1)(n+2) \qquad (8)$$

L'algorithme propose une valeur par défaut de $m = 2n+1$ points, où n est le nombre de dimensions et m le nombre de points constituant la quadratique. Nous choisissons comme dans CONDOR, un point de départ x_{start} et la taille initiale et finale de la distance d'échantillonnage, respectivement ρ_{beg} et ρ_{end}.

Ce nouvel algorithme arrive à construire un modèle quadratique avec moins de données en minimisant la norme de Frobenius des changements à $\nabla^2 Q$. L'algorithme remplace un point x dans l'ensemble des points d'interpolation par itération. Le cout de travail total par itération est de l'ordre $(m+n)^2$. Ceci nous permet d'appliquer l'algorithme sur notre problème à grandes dimensions.

A chaque itération, un nouveau point x_k est calculé au minimum de la valeur de la fonction objective retourné par le modèle d'approximation quadratique. Si la différence entre la valeur réelle de la fonction objective et la valeur approchée par la quadratique Q_{k-1} est petite, cela signifie que le modèle avait fait une bonne prédiction sur la valeur de la fonction objective. Si toutefois cette différence était grande, ce nouveau point x_k permettrait alors d'améliorer le modèle.

La valeur du rayon de la *trust region* Δ est mise à jour à chaque itération. Cette valeur de Δ ne peut décroitre en dessous de la distance d'échantillonnage ρ car autrement la distance entre les points d'interpolation entrainerait une dégénérescence de la quadratique Q_k . La distance d'échantillonnage ρ n'est modifiée que lorsque cette contrainte empêche l'algorithme de progresser plus prêt de l'optimum de la fonction objective. Dans ce cas, l'algorithme change ρ en le divisant d'un facteur dix tout comme CONDOR. La condition d'arrêt est également définie sur la distance d'échantillonnage ρ et doit valoir la distance d'échantillonnage finale imposée par l'utilisateur $\rho = \rho_{end}$.

Comme dit précédemment, NEWUOA est un algorithme permettant l'optimisation de fonction objective sans contraintes et sans les dérivées de la fonction objective. Notre fonction test contient cependant plusieurs contraintes. Chaque dimension ne peut avoir une valeur inférieure à 0 ou supérieure à 21 :

$$0 \leq x_i \leq 21 \qquad (9)$$

La deuxième contrainte est que la somme des dimensions est égale à 21 à chaque instant :

$$\sum x_i = 21 \qquad (10)$$

Il faudra donc adapter l'algorithme de manière à prendre en compte ces différentes contraintes sur la fonction objective. Nous détaillerons ceci plus en détail dans la section suivante.

5.2. LES CONTRAINTES DANS NEWUOA

NEWUOA n'intègre pas la possibilité de définir des contraintes dans l'optimisation de la fonction objective. Souvent, les contraintes d'une application à optimiser finissent par se vérifier d'elles-mêmes et ne devraient pas être prises en compte par l'algorithme d'optimisation. Cela diminue aussi la charge de calcul imposé à l'algorithme d'optimisation.

Dans notre cas, ceci n'a pas pu être vérifié, car la somme des valeurs des coordonnées dépassait toujours la valeur 21 que nous avons imposée. Les valeurs des coordonnées des points échantillonnées par l'algorithme d'optimisation étaient souvent aussi choisies négatives. Or aucun composant ne peut être présent avec une concentration négative dans la solution. Beaucoup de points étaient échantillonnés en dehors de l'espace admissible. Il nous a donc fallu intégrer dans NEWUOA les mêmes contraintes définies précédemment dans NEWUOA.

L'idée suivante nous est venue : nous allons appliquer une transformation aux coordonnées d'un point x_k lorsque celui-ci est envoyé à la fonction objective pour évaluation. Cette transformation du point lui donnera des nouvelles coordonnées y_k qui satisferont les différentes contraintes.

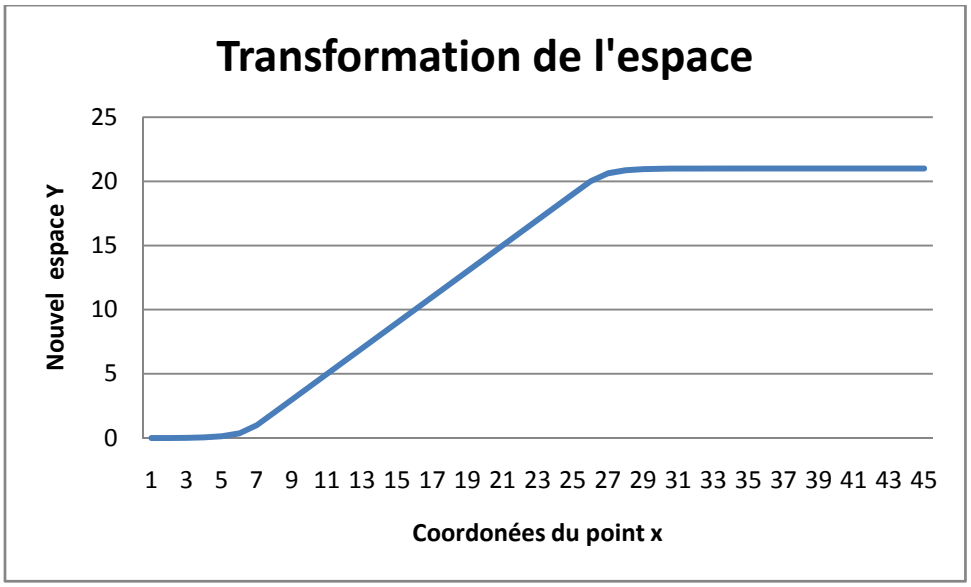


Figure 20 : Transformation de l'espace

Nous appellerons le point envoyé pour évaluation à la fonction objective le point x. Chaque coordonnée i du point x subira les transformations suivantes :

- Si la coordonnée i du point x est inférieure ou égale à 0, nous lui attribuerons la valeur 0 dans le nouvel espace selon la formule ci-dessous. Ceci permet de satisfaire la contrainte sur la borne inférieure de la valeur des dimensions.

$$y_i = \frac{e^{x_i}}{e} \text{ si } x_i < 1 \qquad (11)$$

- Si à l'inverse la coordonnée i dépasse la borne supérieure, nous lui imposons la valeur maximale autorisée qui est 21.

$$y_i = 21 - \frac{e^{(21-x_i)}}{e} \text{ si } x_i > 20 \qquad (12)$$

- Si aucun autre des 2 cas n'est satisfait, cela veut dire que la coordonnée i du point x est comprise entre la borne inférieure et supérieure. Dans ce cas, la valeur de la coordonnée reste inchangée.

$$y_i = x_i \qquad (13)$$

Les formules exponentielles employées permettent d'éviter des discontinuités lors de l'application des contraintes sur la valeur des coordonnées du point x. Une représentation graphique est donnée dans la Figure 20. En abscisse est affichée la valeur de la coordonnée i d'un point x et en ordonnée la valeur que nous donnerons à cette coordonnée dans le nouvel espace en fonction de sa valeur dans l'ancien espace.

Après cette première transformation, tous les points de l'espace satisfont les bornes inférieures et supérieures imposées sur leurs différentes coordonnées. Nous devons encore appliquer une deuxième transformation à ces points pour satisfaire la deuxième contrainte, qui est la suivante :

$$\sum x_i = 21 \qquad (14)$$

Pour ce faire, nous calculerons la somme des 100 coordonnées du point. Si cette somme est supérieure à 21, nous rapporterons la somme des coordonnées à 21 par une division :

$$S = \sum x_i \qquad (15)$$

$$y_i = x_i * \frac{21}{S} \text{ si } S > 21 \qquad (16)$$

La somme des coordonnées satisfait toujours la contrainte imposée maintenant.

Enfin, ce nouveau point y_k ayant subi les 2 transformations et satisfaisant les 2 contraintes sera envoyé à la fonction objective $f(x)$ pour évaluation.

5.3. SIMULATIONS APPROFONDIES EFFECTUÉES AVEC NEWUOA

Dans cette section, nous étudierons l'influence du changement des différents paramètres d'optimisation de NEWUOA comme nous l'avons fait précédemment pour CONDOR. Nous voulons à présent découvrir les mêmes optimums locaux ou des optimums locaux meilleurs avec NEWUOA en utilisant moins d'évaluations et donc moins de temps de calcul.

Les paramètres d'optimisation de NEWUOA sont les mêmes que ceux de CONDOR, car les deux algorithmes sont des algorithmes d'optimisation basés sur un modèle d'approximation.

Les paramètres d'optimisation choisis après quelques simulations sont les suivants :

$$x_{start} = (rep(100, 0))$$

$$\rho_{start} = 5 \qquad (17)$$

$$\rho_{end} = 1e-8$$

Il est important de noter que les valeurs pour la distance initiale et finale d'échantillonnage sont exprimées de manière absolue dans NEWUOA et non pas en valeur relative par rapport à la valeur maximale d'une dimension, comme c'était le cas dans CONDOR. Une valeur pour la distance initiale d'échantillonnage $\rho_{start} = 5$ vaut en valeur relative $5 / 21 = 0,24$ et correspond donc à un $\rho_{start} = 0,24$ dans CONDOR.

Nous avons pris une distance d'échantillonnage finale $\rho_{end} = 1e-8$ car l'algorithme converge rapidement dans le temps. Nous en profitons donc pour obtenir des résultats numériques plus précis pour l'optimum.

Il s'ajoute à ces paramètres un paramètre d'optimisation supplémentaire dans l'algorithme NEWUOA : le nombre de points m formant l'ensemble des points d'interpolation de la quadratique. Le nombre de points sera maintenu à la valeur recommandée : $m = 2n+1$, où n est le nombre de dimensions. Le nombre de dimensions est égal à 100 et nous avons donc 201 points qui constituent la quadratique.

5.3.1. INFLUENCE DU POINT DE DÉPART

Ce test consiste à étudier l'influence du point x de l'espace de recherche choisi comme point de départ x_{start}. La valeur du point final x^* de la fonction objective sera portée en graphique en fonction du point de départ x_{start} choisi. Le point de départ variera comme pour le test effectué avec CONDOR. C'est-à-dire que lors de chaque simulation les valeurs de 2 dimensions du point de départ x_{start} seront changées. Ainsi, nous parcourrons l'ensemble des dimensions actives dans le problème à maximiser.

Dans les simulations préliminaires effectuées avec NEWUOA, cet algorithme semblait beaucoup plus sujet à la valeur du bruit présente dans la fonction objective. Quelques fois, il atteignait le même point optimal x^*, d'autres fois il ne l'atteignait pas du tout. Ceci était donc dû au bruit, car tous les paramètres d'optimisation étaient maintenus constants. L'algorithme ne suit pas le même chemin lors de chaque simulation uniquement à cause du bruit présent dans l'espace de recherche.

Nous avons ainsi envisagé 3 solutions pour atténuer l'influence du bruit sur l'algorithme :

- La première serait de lancer l'algorithme plusieurs fois séquentiellement. On enregistrerait la valeur maximale de la fonction objective obtenue lors de chaque itération. Ensuite, on pourrait prendre la moyenne de ces valeurs enregistrées. Cependant, en prenant la moyenne sur plusieurs itérations, nous risquons de ne pas pouvoir étudier l'influence du paramètre d'optimisation que nous étudions, ex. le point de départ x_{start}. La prise d'une moyenne cacherait toute l'influence de la variation de ce paramètre.

- La deuxième idée consisterait exécuter l'algorithme plusieurs fois séquentiellement et de prendre la valeur maximale obtenue lors des différentes itérations. Ceci nous donnerait une meilleure idée sur l'influence du paramètre d'optimisation que nous faisons varier.

- Enfin, une dernière possibilité serait de lancer l'algorithme plusieurs fois en repartant du point optimal x^* trouvé lors de l'itération précédente. La première itération servirait d'exploration de l'espace de recherche jusqu'à un optimum local. Les itérations suivantes permettent alors de sortir de cet optimum local et de partir à la recherche d'un optimum meilleur à proximité du précédent. Lors de chaque itération, la distance d'échantillonnage ρ_{start} est décroissante. Nous espérons ainsi trouver l'optimum global après plusieurs itérations.

La valeur de la distance d'échantillonnage initiale ρ_{start} a été fixée à 5 pour les simulations ci-dessous. Lors des simulations, nous avons pris la moyenne des valeurs maximales trouvées pour la fonction objective après 5 itérations. Les autres solutions proposées seront étudiées par la suite.

Essai	Point de départ	Max FO	NEF
1	(rep(100,0))	5,99	2252
2	(10,10, rep(98,0))	6,00	2028
3	(rep(2,0), 10, 10, rep(96,0))	5,08	2038
4	(rep(4,0), 10, 10, rep(94,0))	4,94	2220
5	(rep(6,0), 10, 10, rep(92,0))	5,45	2168
6	(rep(8,0), 10, 10, rep(90,0))	5,22	2100
7	(rep(10,0), 10, 10, rep(88,0))	6,21	2231
8	(rep(12,0), 10, 10, rep(86,0))	5,47	2059
9	(rep(14,0), 10, 10, rep(84,0))	5,26	2147
10	(rep(16,0), 10, 10, rep(82,0))	5,69	2147

Tableau 9 : Influence du point de départ

Le Tableau 10 affiche les mêmes simulations, mais lorsque nous prenons la valeur maximale atteinte pour la fonction objective lors des 5 itérations et non pas la moyenne des 5 itérations.

Essai	Point de départ	Max FO	NEF
1	(rep(100,0))	6,90	2154
2	(10,10, rep(98,0))	5,53	2220
3	(rep(2,0), 10, 10, rep(96,0))	5,11	2036
4	(rep(4,0), 10, 10, rep(94,0))	5,85	2070
5	(rep(6,0), 10, 10, rep(92,0))	5,37	2119
6	(rep(8,0), 10, 10, rep(90,0))	6,30	2049
7	(rep(10,0), 10, 10, rep(88,0))	6,83	2040
8	(rep(12,0), 10, 10, rep(86,0))	5,95	2033
9	(rep(14,0), 10, 10, rep(84,0))	6,19	2056
10	(rep(16,0), 10, 10, rep(82,0))	5,92	2036

Tableau 10 : Influence du point de départ

Lorsque nous juxtaposons les 2 ensembles de simulations sur un graphique, nous obtenons le graphique ci-dessous.

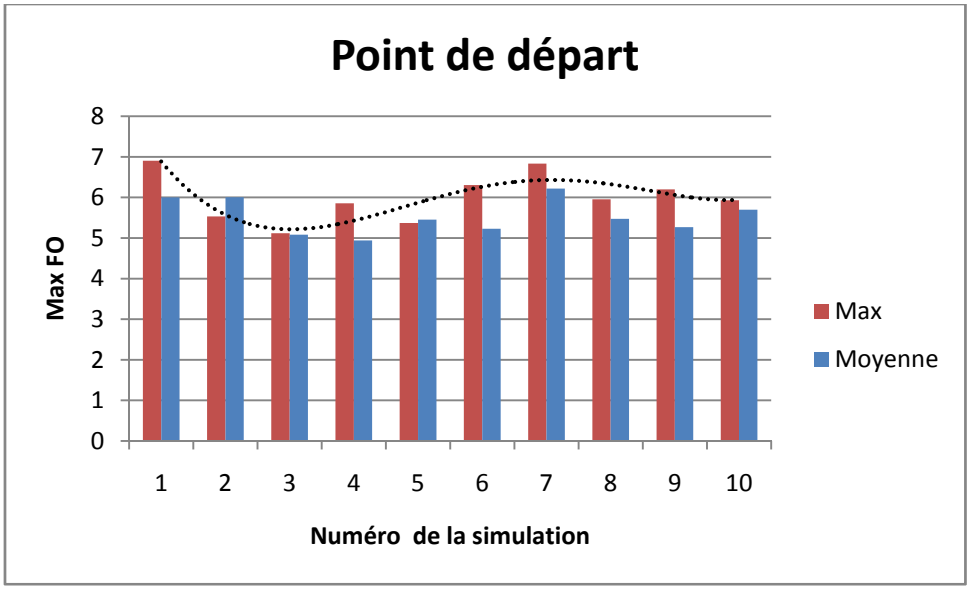


Figure 21 : Influence du point de départ

La Figure 21 montre que les informations obtenues lors des simulations dans les quelles nous effectuons une moyenne sur les 5 itérations et les simulations dont nous prenons la valeur maximale de la fonction objective des 5 itérations sont bien différentes. Alors

qu'avec la moyenne des itérations il est difficile de s'apercevoir de la variation qu'induit le choix du point de départ, cette différence en nettement plus visible lorsque nous prenons la valeur maximale des différentes itérations. Néanmoins, les 2 ensembles de simulations s'accordent à dire que le point de départ x_{start} choisi lors de la septième simulation fournit le meilleur optimum local.

NEWUOA ne mémorise pas la valeur de la fonction objective en un point. Lorsqu'il évalue un point déjà évalué précédemment, il l'envoie à la fonction objective $f(x)$ pour évaluation. CONDOR était optimisé de manière à ne pas devoir évaluer un point déjà évalué précédemment. De ce fait, NEWUOA peut potentiellement acquérir de meilleurs résultats numériques, car chaque fois qu'il évalue un point x_k la valeur du bruit est également chaque fois recalculée. NEWUOA obtient donc des valeurs maximales pour la fonction objective supérieure à CONDOR.

Nous retiendrons la valeur du point de départ x_{start} de la septième simulation comme meilleur point de départ.

5.3.2. Influence de la distance d'échantillonnage initiale en démarrant près de l'optimum

À présent, nous allons faire varier la distance d'échantillonnage initiale ρ_{start} et mesurer quelle est la variation sur la valeur maximale de la fonction objective obtenue lors des simulations. À nouveau comme avec CONDOR, une petite ou grande distance d'échantillonnage initiale donne une tout autre tournure à l'évolution de l'algorithme d'optimisation et au chemin qu'il choisira pour atteindre l'optimum. Une certaine valeur pour la distance d'échantillonnage initiale ρ_{start} permettra d'atteindre le meilleur optimum local. Le point de départ x_{start} pour ces simulations est l'optimum trouvé dans la section précédente, le point final x^* de la septième simulation. La condition d'arrêt de l'algorithme est maintenue comme suite : $\rho_{end} = 1e-8$

Essai	Distance d'échantillonnage initiale	Max FO	NEF
1	2	4,85	1886
2	4	6,14	2043
3	6	6,05	2165
4	8	5,09	1938
5	10	3,53	1857
6	12	3,90	2003
7	14	3,53	1957
8	16	3,28	1869
9	18	3,35	1919
10	20	4,50	2129

Tableau 11 : Influence de la distance d'échantillonnage initiale

Nous observons dans le Tableau 11 qu'une valeur pour la distance d'échantillonnage initiale ρ_{start} entre 4 et 6 procure les meilleurs résultats. La valeur que nous avions trouvée, $\rho_{start} = 5$ était donc bien choisie. Cette valeur correspond à une valeur relative pour distance d'échantillonnage initiale de $\rho_{start} = 0,24$.

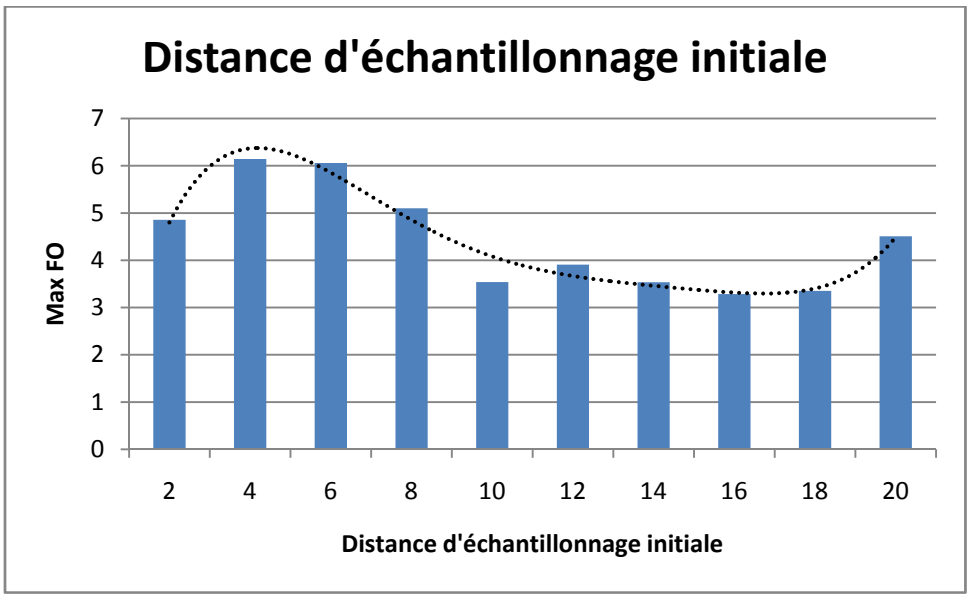


Figure 22 : Influence de la distance d'échantillonnage initiale

Lorsque nous portons ces résultats sur un graphique, cette constatation s'avère être encore plus claire (voir Figure 22). En abscisse est affichée la distance d'échantillonnage initial ρ_{start} et en ordonnée est portée la valeur maximale pour la fonction objective atteinte lors de chaque simulation.

5.3.3. Influence de la distance d'échantillonnage initiale en démarrant loin de l'optimum

Alors que précédemment nous avons démarré les simulations depuis le meilleur point trouvé de l'espace de recherche x^*, nous voulons à présent regarder l'influence de ce même paramètre, mais lorsque nous ne démarrons pas l'algorithme depuis un optimum x^*, mais depuis un point x situé au loin. Nous choisissons comme point de départ x_{start} le point zéro de l'espace de recherche. Toutes les distances d'échantillonnage initiales ne permettront pas de découvrir le chemin qui mène vers l'optimum trouvé précédemment ou un meilleur optimum.

Essai	Distance d'échantillonnage initiale	Max FO	NEF
1	2	3,99	1849
2	4	6,30	2061
3	6	5,70	2098
4	8	2,90	1941
5	10	3,84	1888
6	12	3,80	1923
7	14	3,50	1899
8	16	3,12	1855
9	18	3,30	1909
10	20	3,51	1991

Tableau 12 : Influence de la distance d'échantillonnage initiale

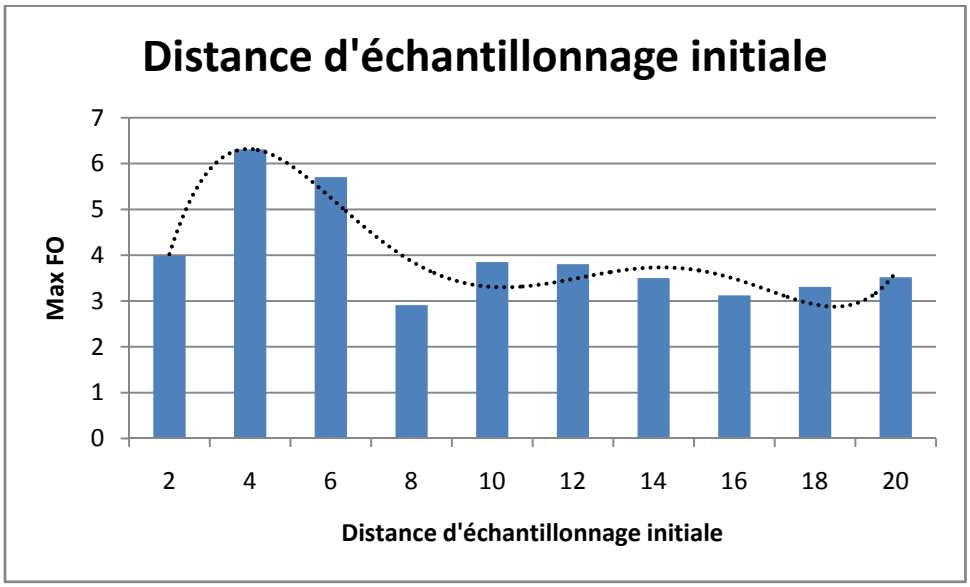


Figure 23 : Influence de la distance d'échantillonnage initiale

Ici à nouveau nous découvrons qu'une distance d'échantillonnage initiale ρ_{start} entre 4 et 6 procure les meilleurs résultats (voir Tableau 12). Il s'avère donc qu'en prenant une distance d'échantillonnage trop petite, l'algorithme ne peut découvrir l'ensemble de l'espace de recherche. Tandis que lorsque cette valeur est prise trop grande l'algorithme

aura une bonne idée de l'ensemble de l'espace de recherche, mais ne pourra trouver de chemin menant vers l'optimum.

5.3.4. Robustesse de l'algorithme par rapport au bruit

Tout comme nous l'avions fait avec CONDOR, nous voulons mesurer la robustesse de l'algorithme par rapport au bruit. Pourra-t-il contourner le bruit de la fonction objective et quand même trouver l'optimum local trouvé précédemment ? Est-ce que l'algorithme obtient systématiquement les mêmes valeurs maximales de la fonction objective même lorsque nous ne changeons pas les paramètres d'optimisation ? NEWUOA est-il plus sujet à la valeur du bruit que CONDOR ? Nous répondrons à ces questions en lançant plusieurs fois la même simulation.

Les paramètres d'optimisation choisis sont les suivants :

$$x_{start} = (rep(100, 0))$$

$$\rho_{start} = 5 \qquad (18)$$

$$\rho_{end} = 1e-8$$

Nous démarrons l'algorithme du point zéro de l'espace avec une distance d'échantillonnage initiale $\rho_{start} = 5$. Chaque simulation n'est cette fois-ci qu'une seule itération.

Essai	Max FO	NEF
1	5,48	2100
2	5,11	2032
3	4,86	2079
4	5,48	2100
5	5,25	2056
6	5,20	2240
7	5,63	2059
8	5,63	2059
9	5,33	2034
10	5,24	2207

Tableau 13 : Robustesse par rapport au bruit

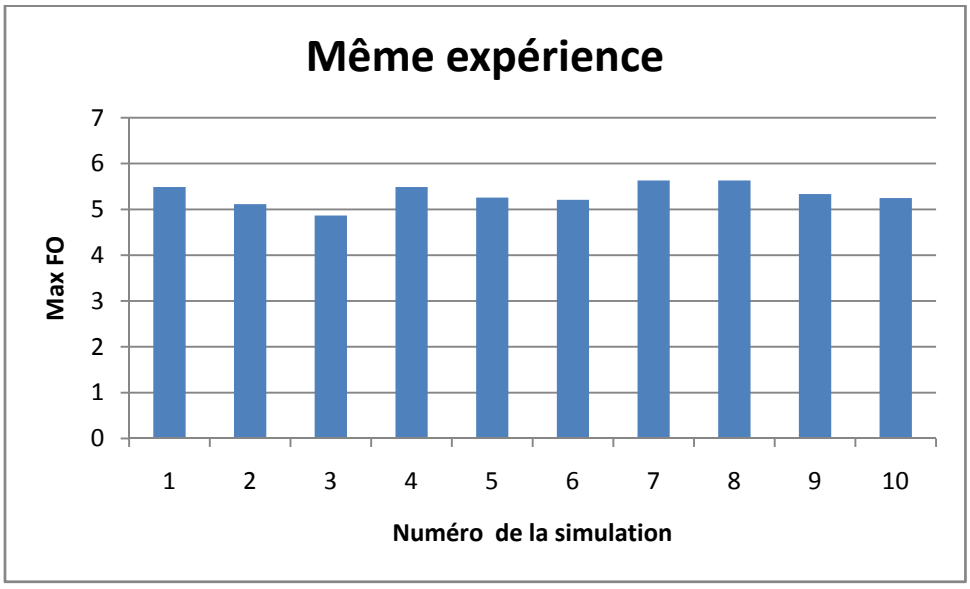


Figure 24 : Robustesse par rapport au bruit

Nous observons que l'algorithme est très régulier lorsque nous lançons plusieurs fois la même simulation. L'algorithme trouve lors de chaque simulation le même optimum et fournit plus au moins la même valeur numérique pour la valeur maximale trouvée de la fonction objective. L'algorithme est donc robuste par rapport au bruit et trouve le bon chemin menant vers l'optimum à condition que les paramètres d'optimisation aient été

bien choisis. NEWUOA semble d'après la Figure 24 moins influencé par le bruit présent dans la fonction objective que CONDOR (voir Figure 18)

5.3.5. ÉVOLUTION DE LA VALEUR DE LA FONCTION OBJECTIVE

La Figure 25 donne un aperçu de l'évolution au cours du temps de l'algorithme NEWUOA. En abscisse, nous avons le nombre total d'évaluations de la fonction objective effectuée et en ordonnée la valeur maximale de la fonction objective au cours du temps.

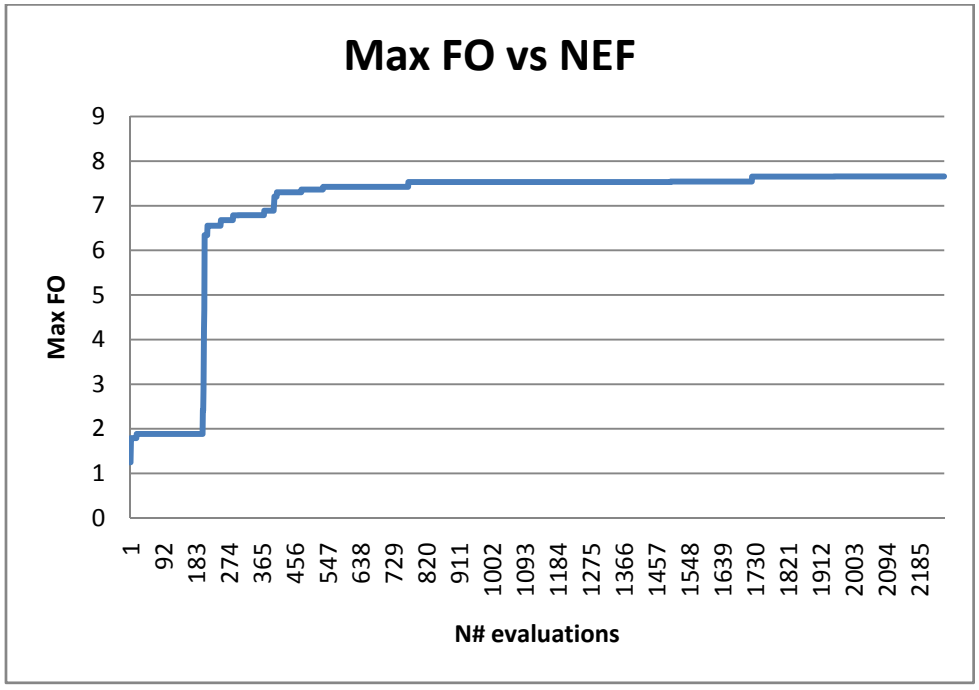


Figure 25 : Evolution de la valeur de la fonction objective avec le nombre d'évaluations

La Figure 25 est la représentation graphique du parcours de la septième simulation de la section 5.3.1.

Durant une phase initiale de 201 évaluations, l'algorithme construit sa quadratique initiale. Pendant cette phase, la valeur maximale atteinte de la fonction objective reste constante, dans notre exemple à une valeur de 2.

Une fois cette phase initiale terminée, l'algorithme commence à exploiter la quadratique construite et se déplace ainsi vers le minimum de cette quadratique. Ceci permet à l'algorithme de trouver une meilleure valeur pour la fonction objective. Nous observons donc sur le graphique un saut dans la valeur de la fonction objective après 201 évaluations.

Ensuite, l'algorithme met un nombre d'évaluations plus important avant de converger vers une valeur numérique stable. Ceci est essentiellement dû au fait que la quadratique construite dans NEWUOA est moins complète que CONDOR. Bien que la quadratique construite par NEWUOA soit utilisable pour prédire le chemin vers l'optimum, elle ne permet pas de s'y déplacer immédiatement comme dans le cas de CONDOR. Cependant, NEWUOA ne requiert que 201 évaluations pour construire sa première quadratique alors que CONDOR en requiert 5150 pour un problème à 100 dimensions.

NEWUOA se montre aussi plus performant en temps de calcul. En effet la charge de calcul dédié à la machine est nettement moindre lors de la mise à jour de la quadratique que dans l'algorithme CONDOR. Ainsi, NEWUOA arrive à l'optimum en seulement une quinzaine de secondes tandis que CONDOR nécessite 3 h pour atteindre ce même optimum à cause de sa longue phase d'initialisation.

Nous aimerions à présent porter en graphique l'évolution de la valeur maximale de la fonction objective en fonction du nombre d'évaluations lorsque nous essayons d'atteindre un meilleur optimum à l'aide des meilleurs paramètres d'optimisation découvert dans les sections précédentes.

Les paramètres d'optimisation choisis sont les suivants :

$$x_{start} = (rep(10,0),10,10,rep(88,0))$$

$$\rho_{start} = 5 \qquad (19)$$

$$\rho_{end} = 1e-7$$

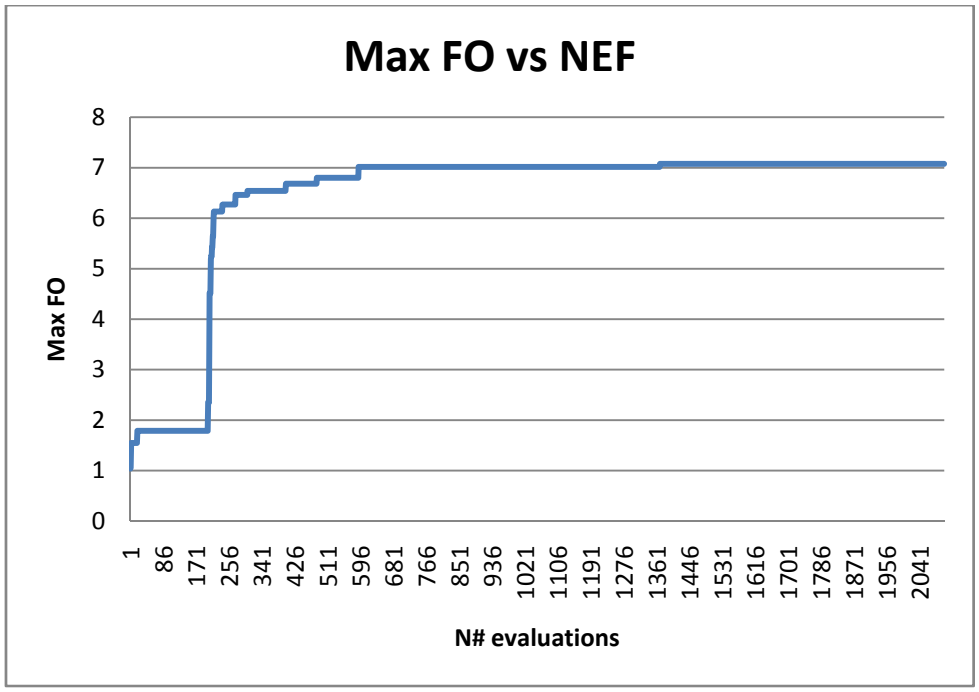


Figure 26 : Recherche de l'optimum global

L'évolution de l'algorithme que nous observons dans la Figure 26 est exactement la même que l'évolution observée précédemment dans la Figure 25. La phase d'initialisation est tout aussi visible sur le graphique, ainsi que le saut après cette phase d'initialisation. Nous voyons également que l'algorithme atteint déjà la valeur maximale après approximativement 700 évaluations contre 5300 évaluations pour CONDOR. NEWUOA atteint donc un optimum tout aussi bon que CONDOR avec nettement moins d'évaluations de la fonction objective. La résolution des degrés de liberté restants lors de la construction de la quadratique par la technique de minimisation de la norme de Frobenius fourni donc une quadratique tout aussi complète que celle construite par CONDOR.

Dans le tableau ci-dessous nous comparons les points optimums x^* trouvés par NEWUOA et CONDOR pour la fonction objective donnée en exemple.

Coordonnée	CONDOR	NEWUOA
1	0,01	0,09
2	3,03	3,42
3	0	0,02
4	0	0,04
5	0	1,20
6	0,09	0,01
7	0,02	0,03
8	0	0,02
9	0,80	0,09
10	0	0,20
11	2,12	1,84
12	0	0,06
13	0	0,09
14	0,02	0,12
15	0	0,01
16	0	0,11
17	0	0,12
18	4,54	3,12
19	3,37	2,04
20	1,92	0,76
21	1,62	0,52
22	0,85	0,09
23	1,24	0,20
24	1,07	0,07
Valeur FO	8,32	7,65

Tableau 14 : Comparaison des optimums locaux trouvés avec les 2 algorithmes

Nous comparons ci-dessus les 2 meilleurs points de l'espace de recherche trouvé x^* respectivement avec chaque algorithme d'optimisation. L'optimum local affiché dans la colonne de gauche du Tableau 14 est le point de l'espace ayant obtenu la meilleure valeur pour la fonction objective avec CONDOR. Ce point a été obtenu lors de la quatrième simulation de la section 4.11.3. Celui sélectionné pour l'algorithme NEWUOA est l'optimum local trouvé lors de la septième simulation de la section 5.3.1.

Les 2 algorithmes s'accordent à dire que les mêmes dimensions ont la plus grande contribution à la maximisation de la valeur de la fonction objective. Tous deux ont sélectionné les dimensions suivantes comme déterminants : 2, 11, 18, 19, 20 et 21. La différence numérique entre les coordonnées des 2 points optimums x^* comparés est liée au fait que CONDOR est beaucoup plus apte à déterminer les dimensions actives dans la maximisation de la fonction objective. L'algorithme sélectionne ces dimensions et continue la recherche de l'optimum en ne variant ensuite plus que les dimensions sélectionnées. NEWUOA ne détecte à aucun moment les dimensions ayant une réelle contribution à la solution. Ainsi, la valeur numérique d'une coordonnée n'est jamais égale à 0. Ceci représente un désavantage pour NEWUOA étant donné que la valeur maximale de la somme des dimensions est fixée à 21. La valeur des dimensions ayant une réelle contribution à la solution sera ainsi limitée par la valeur des dimensions n'ayant aucune contribution.

6. COMPARAISON DES DEUX ALGORITHMES D'OPTIMISATION

Les paragraphes précédents nous ont permis d'étudier de manière approfondie les deux algorithmes d'optimisation et leurs résultats obtenus sur la fonction objective prise en exemple. Nous avons pu constater que bien que les deux algorithmes soient tous les deux des algorithmes d'optimisation basés sur la construction d'un modèle d'approximation qu'ils ne la construisent pas et ne l'exploitent pas de la même manière. CONDOR a choisi de construire un modèle quadratique complet reflétant mieux les valeurs réelles prises par la fonction objective, tandis que NEWUOA préfère construire un modèle quadratique initial de moins bonne qualité afin de l'exploiter plus vite et ainsi de partir plus vite à la recherche de l'optimum.

NEWUOA s'est avéré dans l'ensemble beaucoup plus intéressant que CONDOR. Il peut en effet atteindre les mêmes résultats en utilisant moins d'évaluations, moins de ressources système et moins de temps de calcul. Il est tout à fait capable d'optimiser la fonction objective même lorsque les dimensions de l'espace sont très grandes, comme dans notre cas exemple. Il peut en effet démarrer très vite et finir rapidement la phase d'initialisation pour se lancer à la recherche de l'optimum local. Il lui faut en effet que 201 évaluations pour construire son premier modèle d'approximation.

NEWUOA présente cependant quelques désavantages par rapport à CONDOR notamment en nombre de fonctionnalités implémentées. L'algorithme ne permet pas la prise en compte directe des contraintes linéaires et non linéaires, ni des bornes inférieures ou supérieures. Il faudra pour chaque contrainte modifier le code source de l'algorithme afin de restreindre l'espace de recherche de l'algorithme. Il n'est pas possible non plus avec NEWUOA de lancer plusieurs instances de l'algorithme en parallèle et d'utiliser ainsi toutes les ressources de calcul disponible. Cependant, le nombre d'évaluations nécessaire avant d'atteindre l'optimum est tellement inférieur par rapport à CONDOR, que cette fonctionnalité ne s'est pas montrée indispensable. Une exécution de l'algorithme ne prenait qu'une quinzaine de secondes.

ECLT a pu nous fournir également des résultats en guise de comparaison à nos deux algorithmes d'optimisation étudiés ci-dessus. La fonction objective étudié est la même que celle décrit dans la section 4.8. Les mêmes paramètres ont été choisi, c'est-a-dire la même matrice de moyenne m , les mêmes hauteurs défini par a , et les mêmes déviations définis par s. La quantité de bruit sur la valeur de la fonction objective est également définie à 5%.

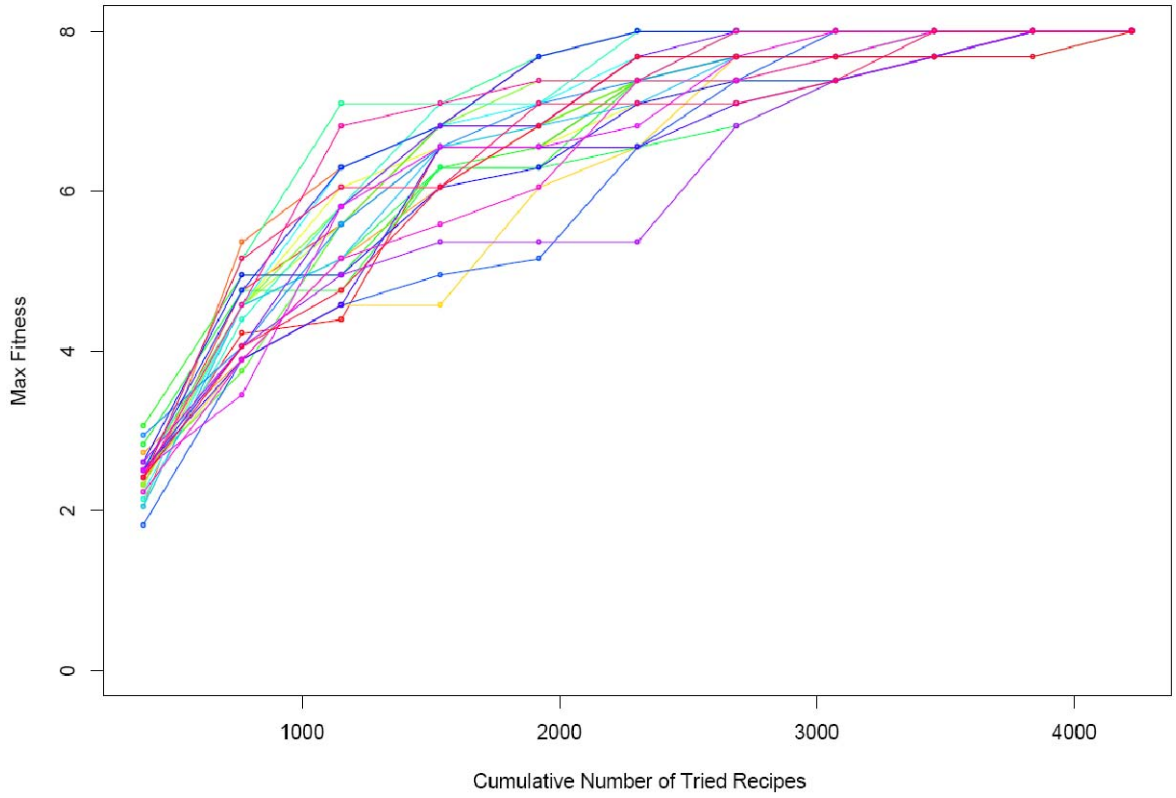


Figure 27 : Comparaison des performances avec l'algorithme génétique

Nous observons qu'il faut en moyenne plus de 3000 évaluations pour l'algorithme génétique avant d'atteindre une valeur supérieure à 7 pour la fonction objective $f(x)$ (voir Figure 27). Lors des simulations avec NEWUOA, il nous fallait seulement 600 évaluations avant d'atteindre la même valeur. Les 4000 évaluations effectuées par l'algorithme génétique sont faites en générant 11 populations de 384 points d'échantillonnages.

7. ÉVOLUTION FUTURE DU DÉVELOPPEMENT

7.1. ESPACE D'ÉCHANTILLONNAGE DISCRET

Au cours des simulations, un problème a été soulevé par les Italiens. Le problème est le suivant : les algorithmes d'optimisation utilisée dans ce travail sont conçus de manière à procéder à la recherche d'un optimum local dans un espace de recherche continu où les valeurs de la fonction objective sont définies en tout point de l'espace. Cette condition n'est pas vérifiée pour l'application de synthèse chimique étudiée à Venise. En effet, il n'est pas possible techniquement d'avoir une précision infinie des concentrations des différents composants en réalité. Les robots de laboratoire ont une bonne précision de mesures des concentrations, mais limitée. Les valeurs prises par les différents composants sont de ce fait discret avec une certaine précision et non continu dans l'espace.

Nous avions envisagé une adaptation de NEWUOA de façon à prendre en compte cette limitation rencontrée dans le laboratoire, mais cette adaptation s'est avérée impossible, car elle demanderait le changement des concepts essentiels de l'algorithme. L'espace de recherche est supposé continu dès le départ de la conception d'un algorithme basé sur un modèle d'approximation.

Plusieurs idées ont émergé pour résoudre cette problématique. La première idée consiste à échantillonner les points de l'espace que nous validons d'abord comme « faisable en laboratoire ». C'est-à-dire que lors de l'évaluation d'un point de l'espace, nous recherchons le point de l'espace le plus proche qui contient des valeurs pour les concentrations mesurables en laboratoire. Ceci revient à trouver le point faisable de l'espace le plus proche par la méthode des moindres carrés.

La première idée donne une réponse favorable à la requête de l'application chimique, mais entraine une dégénérescence assurée de la quadratique. Tous les points de l'espace que l'algorithme veut échantillonner sont déplacés et modifiés. L'interpolation des

points de l'espace échantillonnés ne permet plus de construire un modèle quadratique valide.

La deuxième idée envisagée entraine de modifications moins lourdes et permet de maintenir l'algorithme entièrement fonctionnel. La routine de l'algorithme d'optimisation n'est pas modifiée. Lorsque l'algorithme d'optimisation termine l'optimisation de la fonction objective et trouve l'optimum local, nous rechercherons le point valide à proximité de cet optimum local. Ce point valide est un point dont les concentrations ont des valeurs numériques qui sont mesurables en laboratoire. La distance entre le point optimum et le point valide le plus proche sera déterminée par la méthode des moindres carrés par exemple.

7.2. ÉCHANTILLONNAGES EN PARALLÈLE DE LA FONCTION OBJECTIVE

Il y a une deuxième différence pratique et technique entre les algorithmes d'optimisation proposés dans ce travail et l'application de synthèse chimique réelle. Les expériences en laboratoire sont effectuées avec des populations de solutions d'une taille bien déterminée. Cette taille des populations est variable en fonction des machines et des instruments utilisés pour effectuer les expériences, par exemple des bancs de 96 ou 384 éprouvettes sont utilisés. Ainsi si par exemple si le nombre d'éprouvettes disponibles est de 384, l'algorithme d'optimisation devrait fournir un ensemble de 384 expériences à effectuer en laboratoire. Lorsque les 384 expériences ont été analysées en laboratoires, les informations collectées sont retournées à l'algorithme afin qu'il procède à une mise à jour du modèle d'approximation. Ensuite, l'algorithme choisit un nouvel ensemble de 384 expériences à effectuer en laboratoire.

Cette contrainte pratique ne convient pas du tout à un algorithme d'optimisation tel que NEWUOA où à chaque itération seulement un point est envoyé au laboratoire pour être analysé. Ces algorithmes ont besoin de la mesure effectuée en laboratoire sur cet échantillon avant de pouvoir continuer leur recherche vers la solution optimale. Cette mesure est nécessaire, car l'algorithme d'optimisation l'utilise afin de savoir quel est le niveau de conformité actuelle de la quadratique. En fonction de ce niveau de conformité,

la taille de la *trust region* sera mise à jour. Les deux algorithmes ne définissent pas de *population* de solutions à analyser en laboratoire, mais seulement une *seule* solution à chaque itération.

Afin de remédier à ces limitations des algorithmes d'optimisation basés sur un modèle d'approximation quadratique et d'utiliser pleinement le matériel de laboratoire disponible, il faudra ajouter des fonctionnalités à l'algorithme d'optimisation.

La première fonctionnalité est une fonctionnalité qui pourrait démarrer l'algorithme avec plusieurs modèles d'approximation local. Ces différents modèles démarreraient à des endroits différents de l'espace de recherche. Leurs points de départ seront suffisamment distanciés de manière à ce que l'ensemble des modèles d'approximation quadratique générée couvre l'ensemble de l'espace de recherche intéressant. Ces différents modèles communiquent entre eux de manière à décider si un modèle d'approximation est plus prometteur qu'un autre et afin de prendre communément des décisions sur le chemin à suivre pour chaque modèle dans l'espace de recherche. Au fur et à mesure que l'algorithme progresse, les modèles d'approximation convergeront tous ensemble vers un optimum global choisi par l'ensemble des modèles. Cette fonctionnalité permettrait d'appeler l'algorithme, un algorithme d'optimisation avec des *micro* - et *macro* modèles d'approximation (voir [5]).

La deuxième fonctionnalité qui permettrait d'adapter l'algorithme afin d'échantillonner plusieurs points en parallèle consiste à échantillonner des points supplémentaires quand on évalue un point de l'espace de recherche. Les points supplémentaires qui seront choisis à être évalués seront pris à proximité du point courant. Ils permettront de créer un modèle local de la valeur que prend la fonction objective autour de ce bruit. De ce fait, nous construisons un petit modèle local autour du point courant x_k qui permet d'éliminer l'influence du bruit sur ce point courant. Nous améliorons ainsi la qualité de la mesure en x_k. Les points sont choisis sur une grille autour du point x_k. Le modèle local autour de x_k est construit par régression linéaire des points échantillonnés sur cette grille (voir .

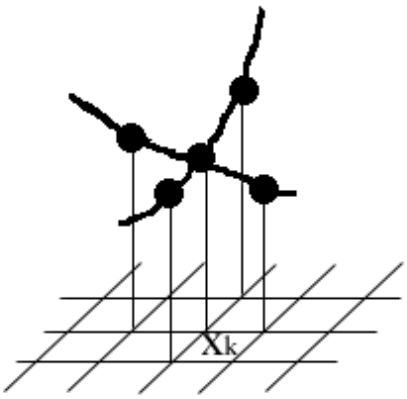


Figure 28 : Amélioration de la qualité de la mesure en X_k

Nous pouvons également paralléliser la phase d'initialisation de l'algorithme. Durant cette phase, l'algorithme parcourt une grille et échantillonne des points de l'espace prédéfinis. Tous les points à échantillonner sont connus d'avance grâce au point de départ x_{start} et de la distance d'échantillonnage initiale ρ_{start}. Deux points d'interpolation sont initialement séparés d'une distance de ρ_{start}. Il est donc tout à fait possible d'échantillonner tous les points à la fois en parallèle et d'utiliser tout le matériel du laboratoire disponible durant cette phase d'initialisation.

Une dernière adaptation possible de l'algorithme afin qu'il effectue des échantillonnages en parallèle est de combiner l'exploration de l'espace de recherche et exploitation des données recueillies. Pendant que l'algorithme prépare les prochains points à échantillonner en exploitant le modèle d'approximation quadratique Q_k, il pourrait définir des points supplémentaires à échantillonner en parallèle de manière à augmenter la qualité du modèle quadratique et à maintenir ainsi la validité du modèle tout en convergeant précisément vers l'optimum local x^*.

8. CONCLUSION

Au cours de ce travail, nous avons étudié le problème de synthèse chimique d'ECLT et l'algorithme d'optimisation utilisé dans leurs laboratoires. Cet algorithme génétique présentait l'inconvénient de ne pas utiliser l'ensemble des expériences précédentes effectué par l'algorithme d'optimisation. Les individus de chaque population étaient déterminés uniquement en effectuant des opérations sur la population précédente. L'algorithme créait un modèle d'approximation afin de vérifier la validité des individus proposés par l'algorithme, mais ne l'utilisait pas afin de déterminer les meilleurs individus à analyser.

Nous avons donc choisi d'appliquer CONDOR sur cette application. CONDOR est un algorithme d'optimisation qui utilise un modèle d'approximation quadratique afin de déterminer itérativement le prochain point à évaluer. La valeur en chaque point et calculée par une fonction objective. Afin de pouvoir comparer les différents algorithmes entre eux, nous avons utilisé une fonction objective exemple. Cet algorithme construit un modèle d'approximation quadratique en utilisant un grand nombre de points. Il est ainsi capable de prédire avec exactitude le meilleur point suivant à évaluer. Nous avons également étudié la possibilité d'exécuter plusieurs instances de CONDOR en parallèle. Les contraintes imposées sur les valeurs des différents composants de la solution ont été intégrées dans CONDOR afin de définir l'espace faisable.

Cependant, CONDOR nécessitait un nombre important de points à échantillonner pour construire la quadratique initiale, car les dimensions de l'espace de recherche sont extrêmement grandes. L'algorithme nécessitait donc beaucoup de temps avant de pouvoir exploiter le modèle d'approximation qu'il construit dans une phase d'initialisation. Nous avons donc choisi un autre algorithme d'optimisation, NEWUOA qui permet d'étudier les problèmes d'optimisations à grandes dimensions plus efficacement. Les résultats numériques obtenus lors des simulations avec NEWUOA étaient concluants et meilleurs que ceux obtenus avec CONDOR. Nous avons donc choisi d'utiliser NEWUOA et d'apporter les changements nécessaires afin de prendre en compte les contraintes du problème d'optimisation.

ANNEXES

A. PROBLÈME TEST

SUPERIMPOSEDGAUSSIANS.R

```
###----GAUSSIAN 1--PARAMETERS-------###
a=c(8,3,1)
s=c(5,5,5)
m= matrix(
        c(
        c(16,3,2,rep(NA,96)),
        c(NA,11,7,3,rep(NA,95)),
        c(rep(NA,4), 10,7,4,rep(NA,92))
        ),
        nrow=3,byrow=T)
noise=0.05
###-----------------------------------###

###----GAUSSIAN 2--PARAMETERS-------###
a=c(8,3,1)
s=c(5,5,5)
m= matrix(
        c(
        c(10,4,3,3,1,rep(NA,95)),
        c(6,6,NA,NA,NA,9,rep(NA,94)),
        c(4,17,rep(NA,98))),
        nrow=3,byrow=T)
noise=0.05
###-----------------------------------###

###----GAUSSIAN 3--PARAMETERS-------###
a=c(1,3,8)
s=c(5,5,5)
m= matrix(
        c(
        c(1,1,2,2,2,2,2,3,3,3,rep(NA,90)),
        c(rep(NA,7),5,3,2,3,2,2,1,1,1,1,rep(NA,83)),
        c(NA,4,rep(NA,6),1,NA,2,rep(NA,6),4,3,2,2,1,1,1,rep(NA,76)
        )
        ),
        nrow=3,byrow=T)

noise=0.05
###-----------------------------------###

x=c(10,11,rep(0,98))
```

```
rec=x
heights =a
sdev=s
coords=m
noiselevel=noise

###---FITNESS FUNCTION DEFINITION---###
SuperImposedGaussians = function(rec, heights, sdev, coords, noiselevel)
{
	nDims = length(rec)
	nPks = length(heights)

	if (any(dim(coords) - c(nPks, nDims)) != 0) stop("Inconsistent size inputs")

	ret = rep(0, nPks)

	for (i in 1:nPks) {
		tmp = coords[i, ]
		distance = rec - tmp
		distance[is.na(tmp)] = 0
		distSum=sum((distance/sdev[i])^2)
		ret[i] = heights[i] * exp(-distSum)
	}

	fit = max(ret)
	perturb = rnorm(1, 0, abs(noiselevel*fit))
	result = fit + perturb
	return(result)
}
###---------------------------------###
```

B. FICHIER DE CONFIGURATION XML CONDOR

OPTIMFITNESSFUNCTION.XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configCONDOR>
   <!--  name of all design variables (tab separated) -->
   <varNames dimension="100">

            </varNames>
   <objectiveFunction nIndex="1">
      <!-- blackbox objective function -->
      <executableFile>
          FitnessFunction.exe
      </executableFile>

      <!-- objective function: input file -->
      <inputObjectiveFile>
         fit_data0.in
      </inputObjectiveFile>

      <!-- objective function: output file -->
      <outputObjectiveFile>
         fit_data0.out
      </outputObjectiveFile>

   </objectiveFunction>

   <startingPoint>
                        0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0

            </startingPoint>
            <constraints>
            <!-- lower bounds for x -->
                  <lowerBounds>
                        0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
            0       0       0       0       0       0       0       0       0       0       0       0
```

```
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0

        </lowerBounds>
        <!-- upper bounds for x -->
        <upperBounds>
                21      21      21      21      21      21      21      21      21      21
21      21      21      21      21      21      21      21      21      21      21      21
21      21      21      21      21      21      21      21      21      21      21      21
21      21      21      21      21      21      21      21      21      21      21      21
21      21      21      21      21      21      21      21      21      21      21      21
21      21      21      21      21      21      21      21      21      21      21      21
21      21      21      21      21      21      21      21      21      21      21      21
21      21      21      21      21      21      21      21      21      21      21      21
21      21      21      21      21      21

        </upperBounds>
    <!-- Here would be the matrix for linear inequalities definition if they were needed -->
    <linearInequalities>
    <eq> -1  -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1
    -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1
    -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1
    -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1
    -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1
    -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1
    -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1
    -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1
    -1      -1      -1      -1      -21
    </eq>
    </linearInequalities>
    <!-- non-Linear Inequalities
    <nonLinearInequalities>
    <eq>
    </eq>
    </nonLinearInequalities> -->

    </constraints>

<optimizationParameters
   rhostart    =".1"
   rhoend      ="1e-3"
   timeToSleep =".001"
   maxIteration="6000"
/>
 <!-- all the datafile are optional
binaryDatabaseFile: the filename of the full DB data (WARNING!! BINARY FORMAT!)
asciiDatabaseFile: data to add to the full DB data file (in ascii format)
traceFile: the data of the last Essai are inside a file called (WARNING!! BINARY FORMAT!) -->
<dataFiles
   binaryDatabaseFile="db_fitness_function.dat"
   asciiDatabaseFile="db_fitness_function.dat.txt"
   traceFile="trace_fitness_function.dat"
```

```
/>
<scalingFactor auto/>
<!-- name of the save file containing the end result of the optimization process -->
<resultFile>
   results_fitness_function.txt
</resultFile>

<!-- <cluster>

<node address="127.0.0.1" file="fitness_client1.xml">

   <executableFile>
      FitnessFunction.exe
   </executableFile>

   <inputObjectiveFile>
      fit_data1.in
   </inputObjectiveFile>

   <outputObjectiveFile>
      fit_data1.out
   </outputObjectiveFile>

</node>

<node address="127.0.0.1" port="1305" file="fitness_client2.xml">

   <executableFile>
      FitnessFunction.exe
   </executableFile>

   <inputObjectiveFile>
      fit_data2.in
   </inputObjectiveFile>

   <outputObjectiveFile>
      fit_data2.out
   </outputObjectiveFile>

   <optionalParameters>

         </optionalParameters>
</node>

</cluster> -->
```

C. CODE NEWUOA

MAIN.C

```
#include <stdio.h>
#include <sys/timeb.h>
#include <math.h>
#include "f2c.h"
#include "globals.h"

/* Table of constant values */
#define multiple_searchs_avg 1
//#define multiple_searchs_max 1
//#define single_search 1

/* Local functions */
void fromXtoY(double *y,double *x);
void initRandom();
double gasdev(long *idum);
double ran1(long *idum);
void InitializeMatrix1 ( int m[nPks][nDims], int a[nPks] ) ;
void InitializeMatrix2 ( int m[nPks][nDims], int a[nPks] ) ;
void InitializeMatrix3 ( int m[nPks][nDims], int a[nPks] ) ;

extern void calfun_(integer *, doublereal *, doublereal *);

/* Local variables */
long rstart = -100;
static integer c__1 = 1;
int sdev[nPks]  = {5,5,5};
int coords[nPks][nDims] = {0};
int heights[nPks]  = {0};
double noiselevel = 0.05;
FILE *ff;
FILE *trace;


/*     The Chebyquad test problem (Fletcher, 1965) for N = 2,4,6 and 8, */
/*     with NPT = 2N+1. */

/* Main program */ int main(int argc, char **argv)
{

   /* Format strings */
   static char fmt_20[] = "(//4x,\002Results with N =\002,i2,\002 and NPT "
               "=\002,i3)";
```

```
   /* System generated locals */
//    integer i__1;

   /* Builtin functions */
   integer s_wsfe(cilist *), do_fio(integer *, char *, ftnlen), e_wsfe(void);
   /* Subroutine */ int s_stop(char *, ftnlen);

   /* Local variables */
   static integer i__, n;
            static doublereal s=0, w[100000], x[nDims] = {0};
            static doublereal f=0.0;
   static integer npt;
   static doublereal rhobeg, rhoend;
   static integer maxfun;

   extern /* Subroutine */ double newuoa_(integer *, integer *, doublereal *,
              doublereal *, doublereal *, integer *, integer *, doublereal *);
   static integer iprint;

   static cilist io___9 = { 0, 6, 0, fmt_20, 0 };



            double y[nDims],f_old,f_avg=0,f_max=0;
   int i,loop=0;
   iprint = 2; //Amount of printing.
   maxfun = 50000;
   n = nDims; //Number of dimensions.
   rhobeg = 20;
   rhoend = 1e-7;
            f=0.0;f_old=0.0;

            InitializeMatrix3(coords,heights);
            initRandom();

            x[0]=0;x[1]=0;x[2]=0;x[3]=0;x[4]=0;x[5]=0;x[6]=0;x[7]=0;x[8]=0;x[9]=0;
            x[10]=0;x[11]=0;x[12]=0;x[13]=0;x[14]=0;x[15]=0;x[16]=0;x[17]=0;x[18]=0;x[19]=0;
            x[20]=0;x[21]=0;x[22]=0;x[23]=0;x[24]=0;x[25]=0;x[26]=0;x[27]=0;x[28]=0;x[29]=0;

   npt = (n << 1) + 1;  //NPT = 2N+1. Number of initial points to compute first quadratic
                                                                  //Binary shift = fast way to perform X 2 multiplication.
            s_wsfe(&io___9);
            do_fio(&c__1, (char *)&n, (ftnlen)sizeof(integer));
            do_fio(&c__1, (char *)&npt, (ftnlen)sizeof(integer));
            e_wsfe();
            ff=fopen("results.txt","w");
            trace=fopen("trace.txt","w");
#ifdef refined_search
            do {
```

```
		loop++;
		if (iprint >=2)
			fprintf(ff,"\nNEW START %d\n\n",loop);
		f_old=f;*/
		f=newuoa_(&n, &npt, x, &rhobeg, &rhoend, &iprint, &maxfun, w);
		calfun_(&n, &x[0], &f); //to get final objective value
		fromXtoY(y,x);//convert back to normal X coordinates
		rhobeg=10/(loop+1);//define a new smaller rho begin
		rhoend = 1e-8;
		if (iprint >=2){
			for (i=0;i<nDims;i++) fprintf(ff,"y[%i]= %g\n",i,y[i]);
			s=0;
			for (i=0;i<nDims;i++) s+=y[i];
			fprintf(ff,"\n\nSum concentration: %Lg\nObj.Function=%Lg  %Lg\n",s,f_old,f);
			fflush(ff);
		}
#endif
#ifdef single_search
		f=newuoa_(&n, &npt, x, &rhobeg, &rhoend, &iprint, &maxfun, w);
		calfun_(&n, &x[0], &f); //to get final objective value
		fromXtoY(y,x);//convert back to normal X coordinates
#endif
#ifdef multiple_searchs_avg
		for (loop=0 ; loop<5;loop++)
		{
			for (i=0;i<nDims;i++) {
				x[i]=0;
				y[i]=0;
			}
			f=newuoa_(&n, &npt, x, &rhobeg, &rhoend, &iprint, &maxfun, w);
			f_avg+=f;
			calfun_(&n, &x[0], &f); //to get final objective value
			fromXtoY(y,x);//convert back to normal X coordinates
		}
		f_avg/=5;
#endif
#ifdef multiple_searchs_max
		for (loop=0 ; loop<5;loop++)
		{
			for (i=0;i<nDims;i++) {
				x[i]=0;
				y[i]=0;
			}
			f=newuoa_(&n, &npt, x, &rhobeg, &rhoend, &iprint, &maxfun, w);
			if (f<f_max) f_max = f;
			calfun_(&n, &x[0], &f); //to get final objective value
			fromXtoY(y,x);//convert back to normal X coordinates
		}
#endif
```

```
		//} while(f < f_old - 0.01);
		/* L30: */
		// Final results print out
		fprintf(ff,"** Final results **\n");
		for (i=0;i<nDims;i++) fprintf(ff,"y[%i]= %g\n",i,y[i]);
		s=0;
		for (i=0;i<nDims;i++) s+=y[i];
#ifdef multiple_searchs_max
		fprintf(ff,"\n\nSum concentration: %Lg\nObj.Function=%Lg\n",s,f_max);
#elif  multiple_searchs_avg
		fprintf(ff,"\n\nSum concentration: %Lg\nObj.Function=%Lg\n",s,f_avg);
#else
		fprintf(ff,"\n\nSum concentration: %Lg\nObj.Function=%Lg\n",s,f);
#endif
		fflush(ff);
		fflush(trace);
		fclose(ff);
		fclose(trace);
		return 0;
} /* MAIN__ */

void initRandom()
{
		struct timeb t;
		ftime(&t);
		rstart=-t.millitm;
}
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
double ran1(long *idum) {
		/*"Minimal" random number generator of Park and Miller with Bays-Durham shuffe and added
		safeguards. Returns a uniform random deviate between 0.0 and 1.0 (exclusive of the endpoint
		values). Call with idum a negative integer to initialize; thereafter, do not alter idum between
		successive deviates in a sequence. RNMX should approximate the largest floating value that is
		less than 1.*/
		int j;
		long k;
		static long iy=0;
		static long iv[NTAB];
		double temp;
		if (*idum <= 0 || !iy) { //Initialize.
			if (-(*idum) < 1) *idum=1; //Be sure to prevent idum = 0.
```

```
                else *idum = -(*idum);
                for (j=NTAB+7;j>=0;j--) { //Load the shuffe table (after 8 warm-ups).
                        k=(*idum)/IQ;
                        *idum=IA*(*idum-k*IQ)-IR*k;
                        if (*idum < 0) *idum += IM;
                        if (j < NTAB) iv[j] = *idum;
                }
                iy=iv[0];
        }
        k=(*idum)/IQ;                                           //Start here when not initializing.
        *idum=IA*(*idum-k*IQ)-IR*k;                     //Compute idum=(IA*idum) % IM without over-
        if (*idum < 0) *idum += IM;             //flows by Schrage's method.
        j=iy/NDIV;                                              //Will be in the range 0..NTAB-1.
        iy=iv[j];                                               //Output previously stored value and rell the
        iv[j] = *idum;                                          //shuffe table.
        if ((temp=AM*iy) > RNMX) return RNMX; //Because users don't expect endpoint values.
        else return temp;
}

double gasdev(long *idum) {
        /*Returns a normally distributed deviate with zero mean and unit variance, using ran1(idum)
        as the source of uniform deviates.*/
        double ran1(long *idum);
        static int iset=0;
        static double gset;
        double fac,rsq,v1,v2;
        if (*idum < 0) iset=0; //Reinitialize.
        if (iset == 0) { //We don't have an extra deviate handy, so
                do {
                        v1 = 2.0 * ran1(idum)-1.0;                      //pick two uniform numbers in the
square extending
                        v2 = 2.0 * ran1(idum)-1.0;                      //from-1 to +1 in each direction,
                        rsq=v1*v1+v2*v2;                                        //see if they are in the unit
circle,
                } while (rsq >= 1.0 || rsq == 0.0);             //and if they are not, try again.
                fac = sqrt(-2.0*log(rsq)/rsq);
                //Now make the Box-Muller transformation to get two normal deviates. Return one and
                //save the other for next time.
                gset = v1*fac;
                iset = 1; //Set flag.
                return v2*fac;
        } else { //We have an extra deviate handy,
                iset=0; //so unset the flag,
                return gset; //and return it.
        }
}

//a=c(8,3,1)
//s=c(5,5,5)
```

```
//m= matrix(
//c(
//c(16,3,2,rep(NA,96)),
//c(NA,11,7,3,rep(NA,95)),
//c(rep(NA,4), 10,7,4,rep(NA,92))
//),
//nrow=3,byrow=T)
//noise=0.05

void InitializeMatrix1 ( int m[nPks][nDims], int a[nPks] ) {
        int i = 0;
        a[0]  = 8;
        a[1]  = 3;
        a[2]  = 1;
        // Line 1
        m[0][0] = 16;
        m[0][1] = 3;
        m[0][2] = 2;
        // Line 2

        for ( i = 3 ; i < nDims ;i++)
                m[1][0] = 0xfffffff;

        m[1][1] = 11;
        m[1][2] = 7;
        m[1][3] = 3;
        for ( i = 4 ; i < nDims ;i++)
                m[1][i] = 0xfffffff;  //NaN
        // Line 3
        m[2][0] = 0xfffffff;  //NaN
        m[2][1] = 0xfffffff;  //NaN
        m[2][2] = 0xfffffff;  //NaN
        m[2][3] = 0xfffffff;  //NaN
        m[2][4] = 10;
        m[2][5] = 7;
        m[2][6] = 4;
        for ( i = 7 ; i < nDims ;i++)
                m[2][i] = 0xfffffff;  //NaN

}

//a=c(8,3,1)
//s=c(5,5,5)
//m= matrix(
//c(
//c(10,4,3,3,1,rep(NA,95)),
//c(6,6,NA,NA,NA,9,rep(NA,94)),
//c(4,17,rep(NA,98))),
//nrow=3,byrow=T)
```

```
//noise=0.05
void InitializeMatrix2 ( int m[nPks][nDims], int a[nPks] ) {
	int i = 0;
	a[0]  = 8;
	a[1]  = 3;
	a[2]  = 1;
	// Line 1
	m[0][0] = 10;
	m[0][1] = 4;
	m[0][2] = 3;
	m[0][3] = 3;
	m[0][4] = 1;
	for ( i = 5 ; i < nDims ;i++)
		m[0][0] = 0xffffffff;

	// Line 2
	m[1][0] = 6;
	m[1][1] = 6;
	m[1][2] = 0xffffffff;  //NaN
	m[1][3] = 0xffffffff;  //NaN
	m[1][4] = 0xffffffff;  //NaN
	m[1][5] = 9;
	for ( i = 6 ; i < nDims ;i++)
		m[1][i] = 0xffffffff;  //NaN

	// Line 3
	m[2][0] = 4;  //NaN
	m[2][1] = 17;  //NaN
	for ( i = 2 ; i < nDims ;i++)
		m[2][i] = 0xffffffff;  //NaN
}
//a=c(1,3,8)
//s=c(5,5,5)
//m= matrix(
//c(
//c(1,1,2,2,2,2,2,3,3,3,rep(NA,90)),
//c(rep(NA,7),5,3,2,3,2,2,1,1,1,1,rep(NA,83)),
//c(NA,4,rep(NA,6),1,NA,2,rep(NA,6),4,3,2,2,1,1,1,rep(NA,76)
//)
//),
//nrow=3,byrow=T)
//
//noise=0.05

void InitializeMatrix3 ( int m[nPks][nDims] , int a[nPks] ) {
	int i = 0;
	a[0]  = 1;
	a[1]  = 3;
	a[2]  = 8;
```

```
// Line 1
m[0][0] = 1;
m[0][1] = 1;
m[0][2] = 2;
m[0][3] = 2;
m[0][4] = 2;
m[0][5] = 2;
m[0][6] = 2;
m[0][7] = 3;
m[0][8] = 3;
m[0][9] = 3;
for ( i = 10 ; i < nDims ;i++)
        m[0][i] = 0xffffffff;  //NaN
// Line 2
for ( i = 0 ; i < 7 ;i++)
        m[1][i] = 0xffffffff;  //NaN
m[1][7] = 5;
m[1][8] = 3;
m[1][9] = 2;
m[1][10] = 3;
m[1][11] = 2;
m[1][12] = 2;
m[1][13] = 1;
m[1][14] = 1;
m[1][15] = 1;
m[1][16] = 1;
for ( i = 17 ; i < nDims ;i++)
        m[1][i] = 0xffffffff;  //NaN
// Line 3
m[2][0] = 0xffffffff;  //NaN
m[2][1] = 4;  //NaN
m[2][2] = 0xffffffff;  //NaN
m[2][3] = 0xffffffff;  //NaN
m[2][4] = 0xffffffff;
m[2][5] = 0xffffffff;
m[2][6] = 0xffffffff;
m[2][7] = 0xffffffff;
m[2][8] = 1;
m[2][9] = 0xffffffff;
m[2][10] = 2;
m[2][11] = 0xffffffff;  //NaN
m[2][12] = 0xffffffff;  //NaN
m[2][13] = 0xffffffff;
m[2][14] = 0xffffffff;
m[2][15] = 0xffffffff;
m[2][16] = 0xffffffff;
m[2][17] = 4;
m[2][18] = 3;  //NaN
m[2][19] = 2;  //NaN
```

```
        m[2][20] = 2;
        m[2][21] = 1;
        m[2][22] = 1;
        m[2][23] = 1;
        for ( i = 24 ; i < nDims ;i++)
                m[2][i] = 0xffffffff;  //NaN
}
```

CALFUN.C

```
#include <stdio.h>
#include <math.h>
#include "globals.h"
#include "f2c.h"

extern long rstart;
extern double gasdev(long *idum);
extern int sdev[nPks];
extern int coords[nPks][nDims];
extern int heights[nPks];
extern double noiselevel;

const double bupper=21;
extern FILE *trace;


void fromXtoY(double *y,double *x)
{
   double s=0;
   int i;
   for ( i = 0; i < nDims; i++) {
      if (x[i]<1) y[i]=exp(x[i])/exp(1);
      else if (x[i]>bupper-1) y[i]=bupper-exp(bupper-x[i])/exp(1);
      else y[i]=x[i];
   }
   for (i=0;i<nDims;i++) s+=y[i];
   if (s>21)
   {
      s=21/s;
      for (i=0;i<nDims;i++) y[i]*=s;
   }
}

/* Subroutine */ int calfun_(integer *n, doublereal *x, doublereal *f)
{
   double y[nDims];
            double ccoords[nDims],distance[nDims];
            double distSum[nPks] = {0},ret[nPks],fit = 0,tmp[nDims];
   int i,j ;

   fromXtoY(y,x);
            for ( i = 0; i < nPks; i++) {
                     for ( j = 0; j < nDims ; j++) {
                              ccoords[j] = coords[i][j];
                              if (ccoords[j] == 0xffffffff)  //NA
                                       distance[j]=0;
```

```
                        else
                                distance[j] = y[j] - ccoords[j];
                        tmp[j] = pow((double)distance[j]/sdev[i],2);  //distSum=sum((distance/sdev[i])^2)
                        distSum[i] += tmp[j];
                }
                ret[i] =  heights[i] * exp(-distSum[i]);
        }
        for (i = 0; i < nPks; i++) {
                if (ret[i] > fit) //fit = max(ret)
                        fit = ret[i];
        }
        *f = -(fit + gasdev(&rstart)*abs(noiselevel*fit));

        for (i=0;i<nDims;i++) fprintf(trace,"%g\t",y[i]);
        fprintf(trace,"%g\n",*f);
        return 0;
} /* calfun_ */
```

BIBLIOGRAPHIE

[1] Bersini H., *S.T.E.P.:* The Easiest Way to Optimize a Function, TR/IRIDIA/94-7, Proceedings of the IEEE World Congress on Computational Intelligence, 1994

[2] Buchanan A., Gazzola G., Evolutionary design of a model of self-assembling chemical structures.

[3] Forlin M. Poli, I., Evolving complex biochemical experimentation, European center for Living Technology, 2007

[4] Forlin M., Poli I., Modelling evolutionary experimental designs for complex chemical systems, European center for Living Technology, 2006

[5] J. Suykens, J. Vandewalle and B. De Moor, Intelligence and Cooperative Search by Coupled Local, Katholieke Universiteit Leuven, Int. J. Bifurcation and Chaos, Vol.11, No.8, pp.2133-2144, 2001

[6] J. Leha, G. Zimmermann, A. Krueger, Chemical combination effects predict connectivity in biological systems, Molecular Systems Biology 3, Article number 80, 2007

[7] Powell M.J.D., The NEWUOA software for unconstrained optimization without derivatives, Department of Applied Mathematics and Theoretical Physics, 2004

[8] Theis M., Gazzola G., Forlin M., Optimal Formulation of Complex Chemical systems with a Genetic Algorithm, Protolife SPRL, 2006

[9] Torczon V., Model-Assisted Pattern Search Methods for Optimizing Expensive Computer Simulations, Department of Computer Science

[10] Vanden Berghen F., CONDOR user's guide, Constrained non-linear, derivative-free, parallel optimization of continuous high computing load noisy objective functions, IRIDIA, 2004

[11] Vanden Berghen F., CONDOR, an new Parallel, Constrained extension of Powell's UOBYQA algorithm. Experimental results and comparison with the DFO algorithm, IRIDIA, 2004.

[12] Vanden Berghen F., Constrained non-linear, derivative-free, parallel optimization of continuous high computing load noisy objective functions, IRIDIA, 2004

[13] Vanden Berghen F., Constrained, Non-linear, Derivative-free, parallel, Multi-Objective Optimization of continuous, high computing load, noisy objective functions, 2004