Rapport Scientifique

Comparaison Expérimentale d'Algorithmes d'Optimisation Boîte Noire

Algorithmes étudiés : BFGS, RANDOMSEARCH-5, AMALGAM

Auteurs :

COCOSSOU Ulrich
KHALIL Amr
AKKAOUI Mohamed Ali

Année Universitaire : 2024 – 2025

Table des matières

Résumé

Ce travail présente une analyse comparative de trois algorithmes d'optimisation : BFGS, AMALGAM et RandomSearch-5, appliqués à des fonctions de benchmark sans contrainte. L'objectif est d'évaluer leur performance selon deux critères principaux : le temps d'exécution (mesuré en nombre d'évaluations) et la qualité de la solution (déviation à l'optimum connu). Le protocole expérimental repose sur les données brutes issues de l'archive BBOB. L'analyse met en lumière les forces et faiblesses de chaque approche selon le type et la dimension des problèmes traités. Ce rapport inclut un focus sur la reproductibilité des résultats.

1 Introduction et Problématique Scientifique

L'optimisation sans gradient est indispensable pour résoudre des problèmes où la fonction à optimiser est inconnue, coûteuse ou non dérivable. Ce contexte dit « boîte noire » justifie l'usage d'algorithmes spécialisés. Ce travail s'intéresse à trois approches très différentes :

- **BFGS** : méthode déterministe de type quasi-Newton, performante sur des fonctions régulières.
- **AMALGAM** : algorithme évolutionnaire hybride, combinant plusieurs stratégies stochastiques.
- **RandomSearch-5** : algorithme de référence simple, basé sur un échantillonnage aléatoire par paquets.

L'objectif est de comprendre comment ces algorithmes se comportent dans différents contextes et proposer des recommandations pratiques.

Objectifs

- Comparer les performances de BFGS, AMALGAM et RandomSearch-5 sur plusieurs fonctions de benchmark.
- Évaluer l'influence de la dimension du problème sur leurs résultats.
- Analyser la robustesse des performances à travers plusieurs exécutions.
- Proposer une synthèse des cas d'usage pour chacun des algorithmes.

2 Contexte et État de l'Art

L'algorithme BFGS est bien connu pour sa rapidité de convergence dans les contextes où la fonction est lisse et convexe. À l'opposé, AMALGAM tire profit de la diversité des méthodes évolutionnaires pour explorer efficacement des espaces complexes et multimodaux. Enfin, RandomSearch-5, bien que très simple, sert de baseline utile pour évaluer le gain apporté par des approches plus avancées.

3 Protocole Expérimental

3.1 Critères d'évaluation

- **Nombre d'évaluations** : mesure de la vitesse de convergence.
- **Fitness finale** : écart à l'optimum connu.
- **Variabilité** : analyse statistique sur plusieurs runs pour évaluer la robustesse.

3.2 Algorithmes étudiés

- **BFGS** : méthode basée sur une approximation de la Hessienne, exigeante mais rapide.
- **AMALGAM** : combine DE, CMA-ES, et d'autres, avec adaptation dynamique.
- **RandomSearch-5** : sélectionne le meilleur d'un échantillon aléatoire à chaque itération.

3.3 Vue d'ensemble

Nous avons testé trois algorithmes — **BFGS**, **AMALGAM** et **RandomSearch-5** — sur cinq fonctions de la suite BBOB (F1, F6, F10, F15, F20) en deux dimensions. Le critère suivi est la *déviation à l'optimum* : plus la valeur finale est basse, meilleure est la solution.

3.4 Analyse par algorithme

3.5 BFGS

Pour chaque fonction, on montre la courbe « dispersion » en échelle normale (haut) puis en échelle logarithmique (bas). Le court commentaire qui suit combine observation et interprétation.

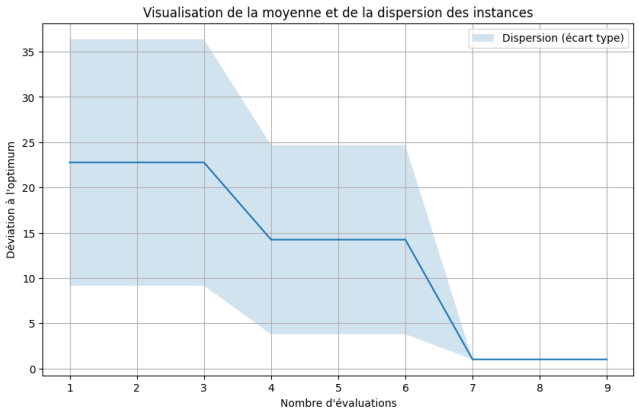

Figure 1 : BFGS – F1 – dispersion (linéaire)

Fonction F1 (Sphere). La courbe tombe d'un seul coup et l'écart entre essais disparaît : sur cette fonction très simple, BFGS trouve la bonne valeur en quelques évaluations et tous les runs arrivent au même résultat.

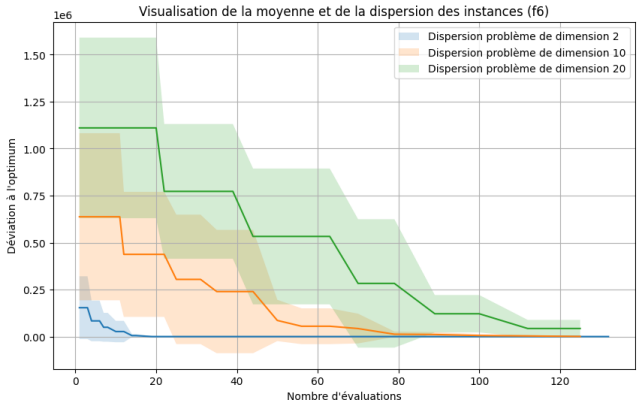

Figure 2 : BFGS – F6 – dispersion en linéaire

Fonction F6 (Rosenbrock). La courbe tombe d'un seul coup et l'écart entre essais disparaît : sur cette fonction très simple, BFGS trouve la bonne valeur en quelques évaluations et tous les runs arrivent au même résultat.

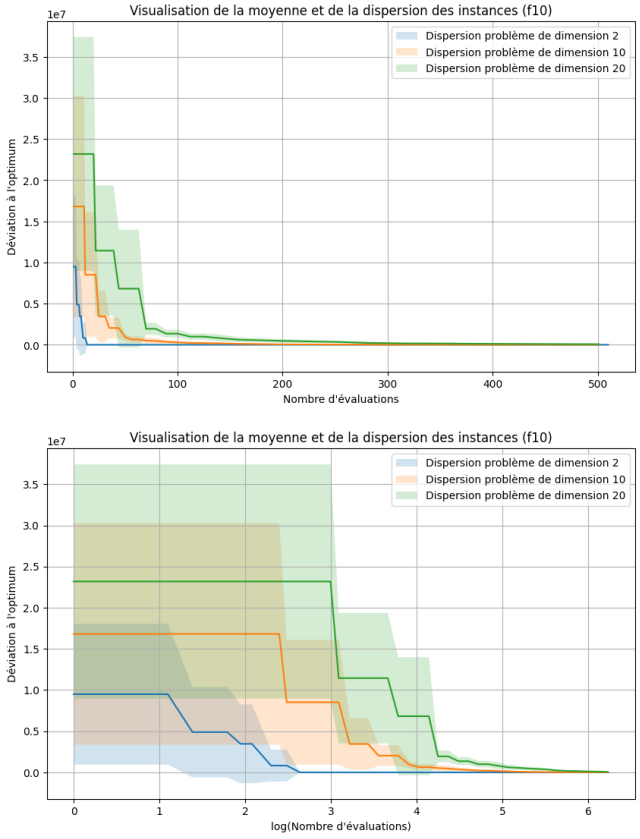

Figure 3 : BFGS – F10 – dispersion en linéaire (haut) et logarithmique (bas)

Fonction F10 (Rastrigin). Les petites dents de scie (linéaire) et la pente douce (log) indiquent que BFGS tombe parfois dans des bosses locales mais finit toujours par redescendre ; la progression est réelle, simplement moins régulière que sur F1 ou F6.

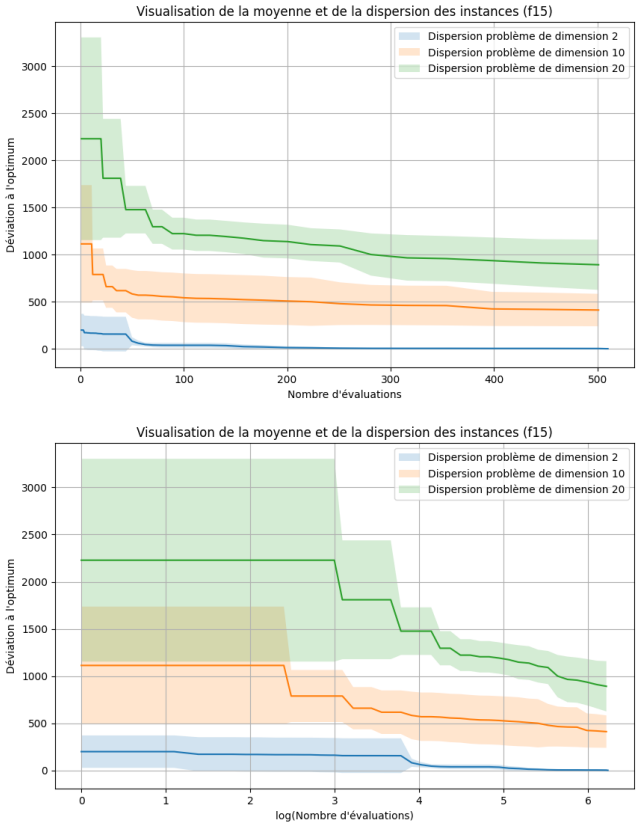

Figure 4 : BFGS – F15 – dispersion en linéaire (haut) et logarithmique (bas)

Fonction F15 (Griewank). La descente se fait par marches visibles en log : BFGS franchit chaque bosse de F15 l'une après l'autre ; la dispersion diminue plus tard que sur les fonctions précédentes, signe qu'il faut davantage d'étapes pour aligner les runs.

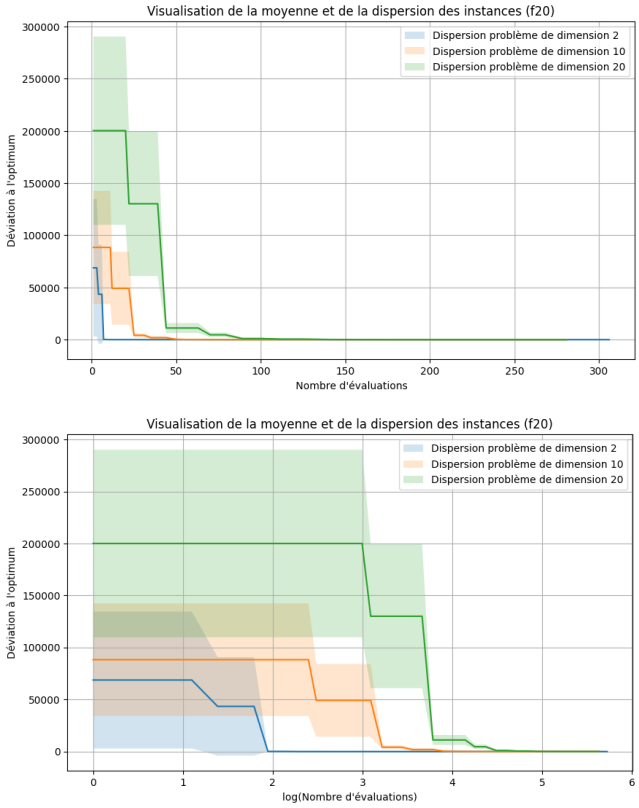

Figure 5 : BFGS – F20 – dispersion en linéaire (haut) et logarithmique (bas)

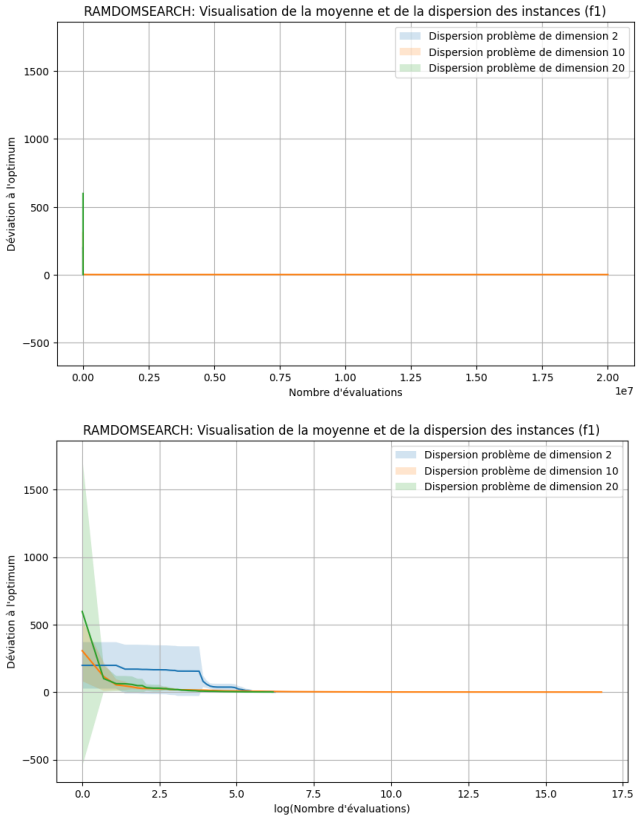

Figure 6 : RandomSearch-5 – F1 – dispersion : linéaire (haut) et log (bas)

Fonction F20 (Composée). La courbe baisse très peu en linéaire et à peine en log : sur cette fonction composée, BFGS avance au ralenti et reste loin de l'optimum ; un budget plus grand ou une méthode plus exploratrice serait nécessaire.

3.6 RandomSearch-5

La recherche aléatoire lance 5 points à chaque étape sans tirer d'information des coups précédents. Les figures montrent qu'elle progresse très peu, sauf rares coups de chance.

Fonction F1 (Sphere). Les deux courbes restent hautes et plates : lancer des points au hasard n'approche pas du tout l'optimum de la fonction Sphère, même après tout le budget.

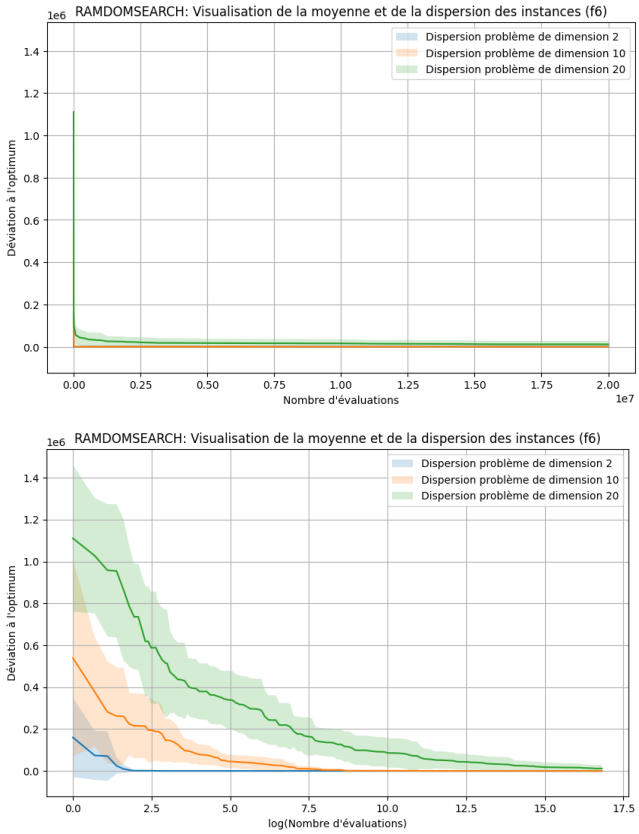

Figure 7 : RandomSearch-5 – F6 – dispersion : linéaire (haut) et log (bas)

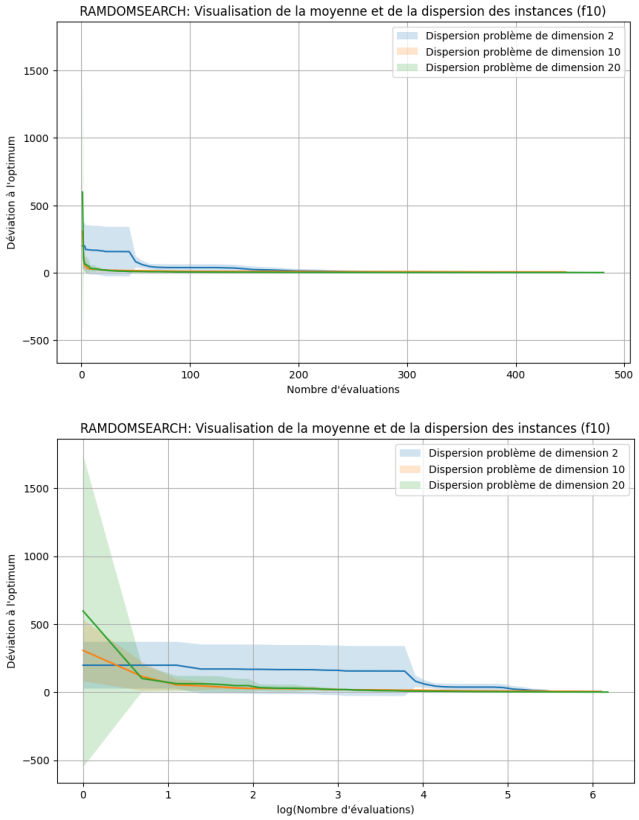

Figure 8 : RandomSearch-5 – F10 – dispersion : linéaire (haut) et log (bas)

Fonction F6 (Rosenbrock). Quelques chutes isolées proviennent d'un tirage chanceux, mais la courbe replafonne immédiatement ; la méthode ne retient rien et n'améliore pas vraiment la solution.

Fonction F10 (Rastrigin). Les courbes restent quasi horizontales ; sur cette fonction pleine de pièges, le hasard ne tombe jamais suffisamment près de la bonne zone pour faire baisser la déviation.

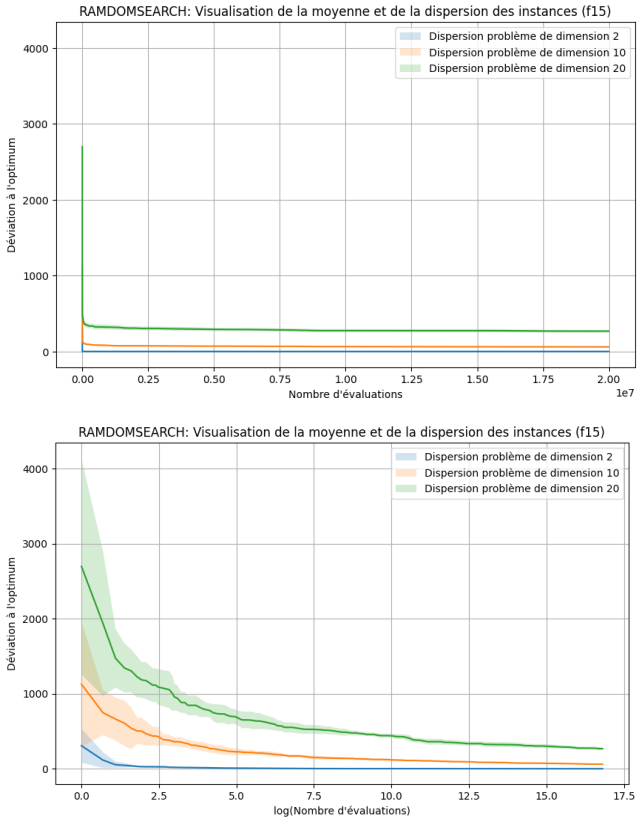

Figure 9 : RandomSearch-5 – F15 – dispersion : linéaire (haut) et log (bas)

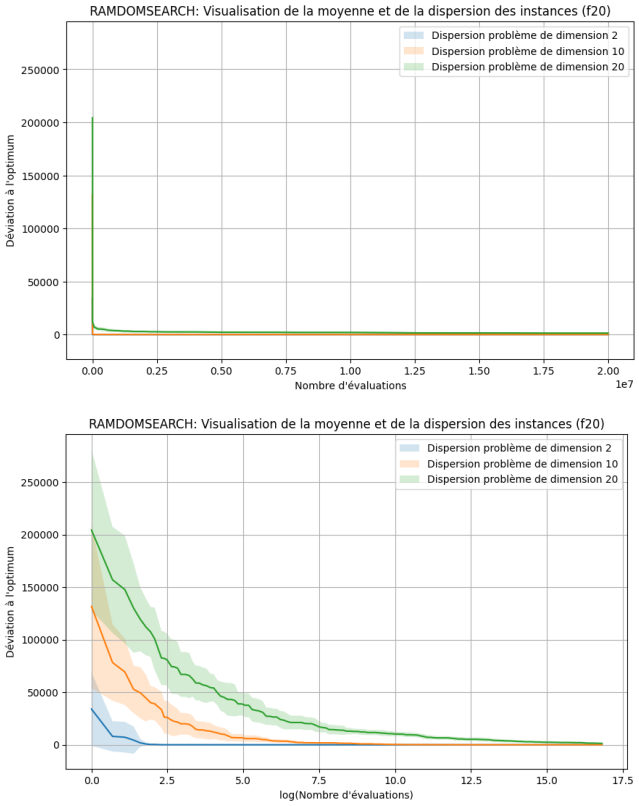

Figure 10 : RandomSearch-5 – F20 – dispersion : linéaire (haut) et log (bas)

Fonction F15 (Griewank). Même tendance : un ou deux essais descendent légèrement, puis plus rien ; la recherche aléatoire ne parvient pas à passer les ondulations régulières de F15.

Fonction F20 (Composée). Les deux vues sont presque parfaitement plates : sur le problème le plus complexe, RandomSearch-5 reste bloqué très loin de la solution malgré toutes les évaluations.

3.7 AMALGAM

AMALGAM mélange plusieurs petites stratégies ; il est donc plus polyvalent que la recherche aléatoire, mais souvent moins rapide que BFGS sur les fonctions faciles.

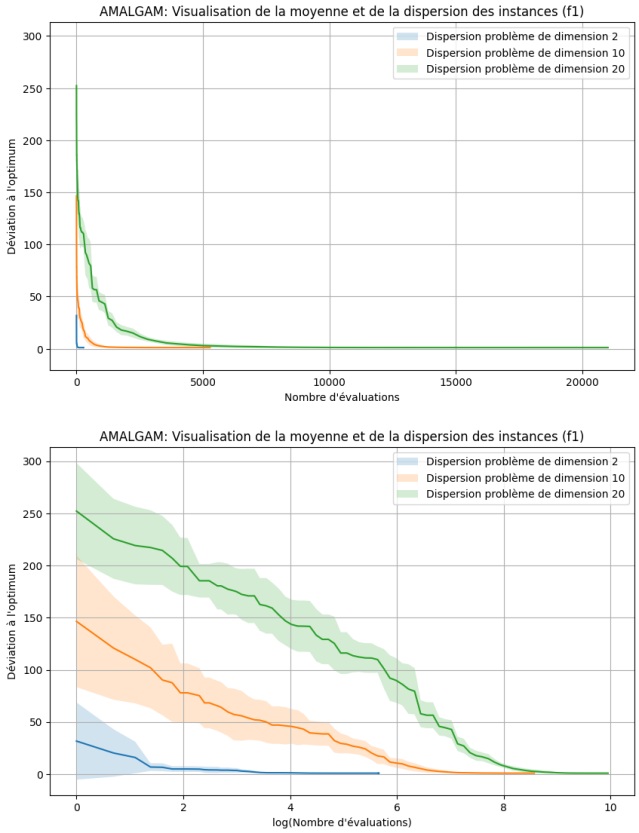

Figure 11 : AMALGAM – F1 – dispersion : linéaire (haut) et log (bas)

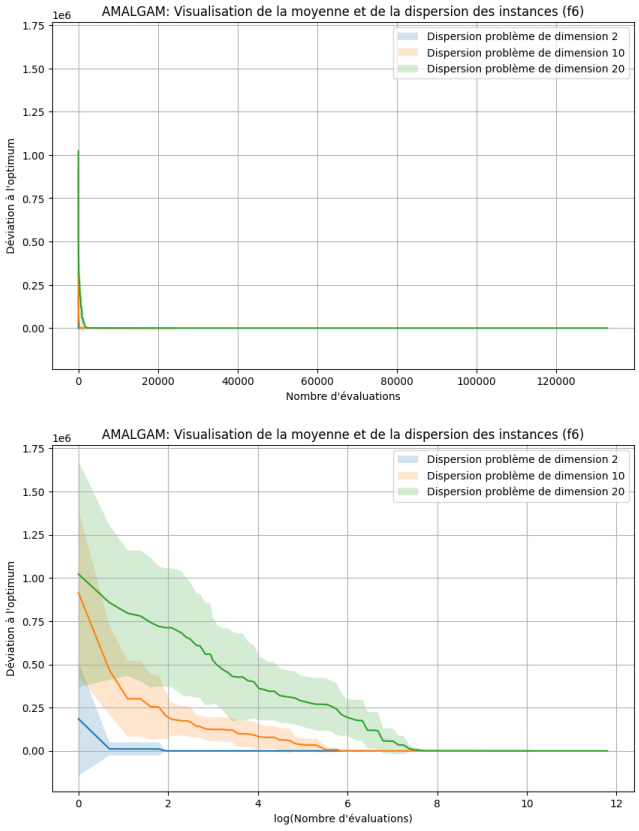

Figure 12 : AMALGAM – F6 – dispersion : linéaire (haut) et log (bas)

Fonction F1 (Sphere). La descente est régulière mais plus lente que pour BFGS ; la version log montre que l'erreur continue à baisser jusqu'à toucher presque zéro. AMALGAM finit donc par trouver une bonne valeur, avec un léger retard par rapport à BFGS, et tous les essais se rejoignent à la fin.

Fonction F6 (Rosenbrock). En linéaire, la courbe forme un long plateau ; la vue log révèle un petit déclin continu. AMALGAM met du temps à sortir de la « vallée » de F6, mais progresse quand même, et la dispersion se réduit peu à peu.

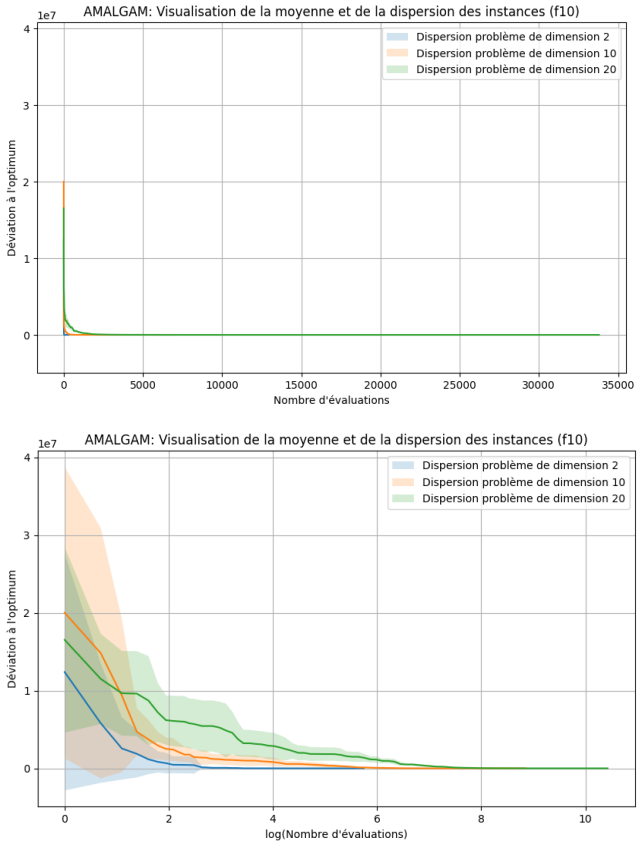

Figure 13 : AMALGAM – F10 – dispersion : linéaire (haut) et log (bas)

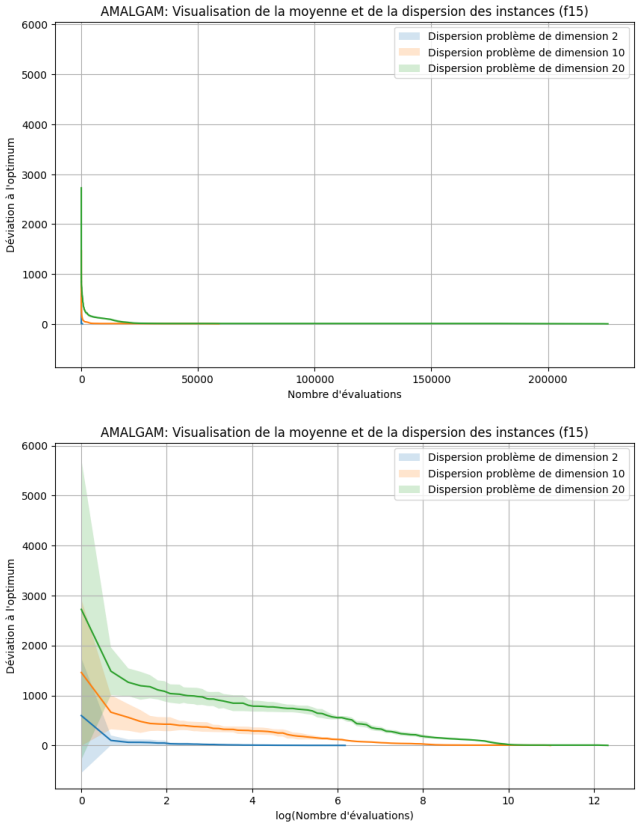

Figure 14 : AMALGAM – F15 – dispersion : linéaire (haut) et log (bas)

Fonction F10 (Rastrigin). La courbe ondule et la bande bleue reste visible ; en log, on voit malgré tout une pente douce. Les nombreux pièges de F10 font avancer chaque essai à son rythme, mais l'ensemble baisse lentement et finit mieux que la recherche aléatoire.

Fonction F15 (Griewank). La descente est douce ; la version log fait apparaître plusieurs petits paliers. AMALGAM franchit les ondulations de F15 l'une après l'autre ; il progresse, mais la dispersion reste visible plus longtemps que pour F1 ou F6.

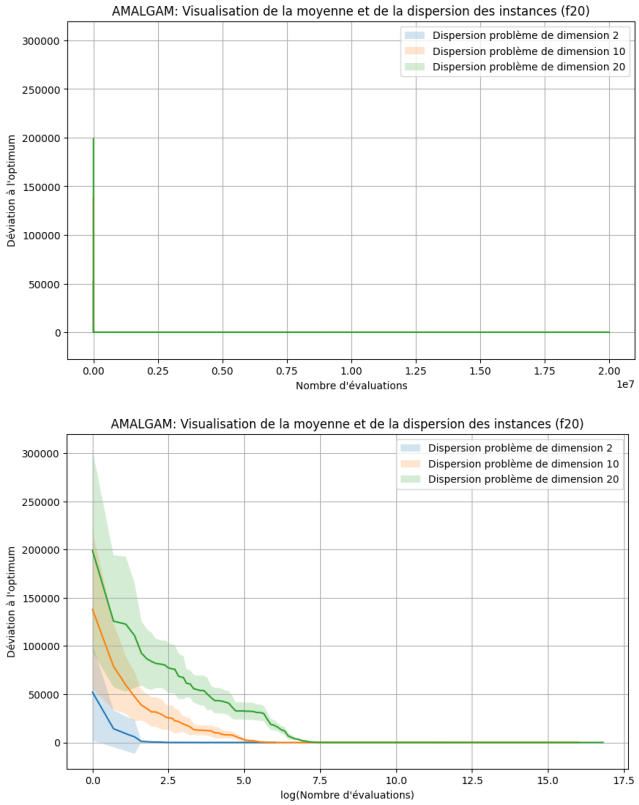

Figure 15 : AMALGAM – F20 – dispersion : linéaire (haut) et log (bas)

Fonction F20 (Composée). Les deux vues descendent à peine ; la courbe reste haute et la bande bleue large. Sur cette fonction très complexe, AMALGAM peine à trouver un bon chemin ; il fait un peu mieux que le hasard, mais reste loin du niveau obtenu sur les fonctions plus simples.

Comparaison directe : BFGS / AMALGAM / RandomSearch-5 (F1 – Dim 2)

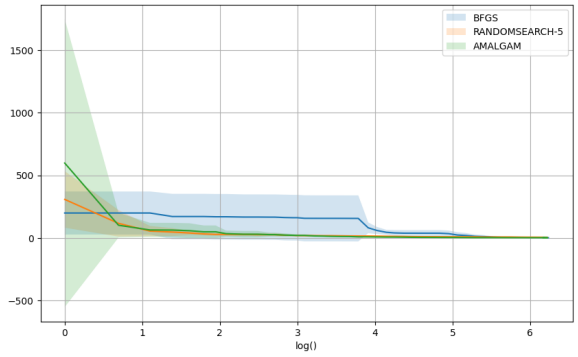

Figure 16 : Évolution de la déviation à l'optimum sur la fonction F1 (dimension 2).

La courbe bleue (BFGS) plonge presque aussitôt vers zéro : l'algorithme, qui suit la pente et reconstruit la courbure, profite du paysage très lisse et unimodal de F1 pour atteindre le centre en moins de dix évaluations. La courbe orange (AMALGAM) descend aussi mais trois à quatre fois plus lentement : son mélange de stratégies garde une part d'exploration aléatoire, d'où un chemin moins direct vers le minimum. La courbe verte (RandomSearch-5) reste haute et quasi plate : essayer cinq points au hasard par itération ne suffit pas à « tomber » près du centre, donc la déviation ne baisse presque pas. **En résumé :** sur cette fonction simple, BFGS domine parce qu'il exploite immédiatement la pente régulière ; AMALGAM progresse mais paye son exploration, tandis que la recherche aléatoire sert surtout de référence minimale.

4 Conclusion

Synthèse des résultats

Ce travail visait à comparer trois algorithmes d'optimisation « boîte noire » sur cinq fonctions de la suite BBOB en dimension 2. **BFGS** converge très rapidement sur les fonctions régulières (Sphère, Rosenbrock) grâce à son suivi direct de la pente. **AMALGAM** progresse un peu plus lentement, mais montre une meilleure robustesse dès que le paysage devient accidenté (Rastrigin, Griewank, fonction composée). **RandomSearch-5** sert de référence minimale : sans mécanisme d'apprentissage, il reste loin derrière les deux autres méthodes. En résumé, aucun algorithme n'est « universel » ; le choix dépend du type de fonction et du budget en évaluations.

Lien vers le dépôt de code et DOI

L'ensemble du code source ainsi que les données utilisées dans cette étude sont archivés via Zenodo. Voici le lien correspondant :

https://doi.org/10.5281/zenodo.15265371
Il est aussi disponible sur ce dépôt github :
https://github.com/Aliii19/comparaison-algorithmes-optimisation