Chapitre 5

Tests et résultats

Dans ce chapitre, on cherche à comparer les performances de l'algorithme proposé Pca-Mads avec d'autres méthodes d'optimisation sans dérivées. Dans un premier temps, on effectue quelques tests exploratoires sur des fonctions simples pour observer le comportement de l'algorithme, ensuite on s'intéresse à des problèmes plus difficiles. Les performances des algorithmes d'optimisation sans dérivées sont comparées à l'aide de profiles de performances et de données décrites à la section 5.1. Les premiers tests exloiratoires sont présentés à la section 5.2. Ensuite, on utilise la plateforme *COCO* pour avoir une suite de problèmes plus difficiles, celle-ci est décrite à la section 5.3. Dans la section 5.5, l'influence des principaux paramètres de l'algorithme est étudiée. Les sections 5.6 et 5.7 comparent Pca-Mads avec d'autres algorithmes d'optimisation, sur une suite de tests et sur des boîtes noires réelles respectivement.

5.1 Profils de performances et profils de données

Les performances de différents algorithmes peuvent être comparées au moyen de profils de performances et de données. Ceux-ci ont été introduit par Moré et Wild [44] et sont construits de la façon suivante.

On considère un ensemble de problèmes $\mathcal{P}$, un ensemble d'algorithmes ou méthodes $\mathcal{S}$ et une mesure de performance $t_{p,s}$. Cette dernière a été choisie comme le nombre d'évaluations nécessaires pour satisfaire un test de convergence,

$$f(x_0) - f(x_k) \geq (1-\tau)(f(x_0) - f_L)$$

où x_i représente la i^e évaluation, τ est une tolérance et f_L la valeur de la meilleure solution trouvée pour un problème. A partir de cette mesure de performance, on définit un ratio de performance $r_{p,s}$,

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,a} : a \in \mathcal{S}\}}.$$

Le profil de performance d'un algorithme $s \in \mathcal{S}$ est la proportion de problèmes où le ratio de performance est au plus α :

$$\rho_s(\alpha) = \frac{|\{p \in \mathcal{P} : r_{p,s} \leq \alpha\}|}{|\mathcal{P}|}.$$

La valeur de $\rho_s(1)$ représente la proportion de problèmes pour lequel l'algorithme s a trouvé la meilleure solution et, pour α suffisamment grand, la valeur de $\rho_s(\alpha)$ représente la proportion de problèmes pour lesquels l'algorithme s a satisfait le test de convergence. Le profil de données d'un algorithme s est

$$d_s(\kappa) = \frac{\left|\left\{p \in \mathcal{P} : \frac{t_{p,s}}{n_p+1} \leq \kappa\right\}\right|}{|\mathcal{P}|},$$

où n_p est le nombre de variables du problème p. Le profil de données d'un algorithme représente la proportion de problèmes pour lesquels il a satisfait le test de convergence avec au plus κ groupes de $n_p + 1$ évaluations.

On note que ces profils donnent une performance relative de chaque algorithme par rapport aux autres, sur un ensemble de problèmes donnés. L'algorithme présentant la courbe au dessus des autres a de meilleures performances car il résout une plus grande proportion de problèmes que les autres pour un budget donné.

5.2 Premières implémentation et tests exploratoires

Dans un premier temps, nous comparons les performances de l'algorithme Pca-Mads proposé avec une implémentation similaire de Mads. Nous considérons la fonction de Rosenbrock à n variables suivante :

$$f(x) = \sum_{i=1}^{n-1} b(x_{i+1} - x_i^2)^2 + (a - x_i)^2, \tag{5.1}$$

où $a = 1$ et $b = 100$. Pour ces tests, on considère deux ensembles d'instances de la fonction de Rosenbrock. Le premier ensemble reprend cette fonction avec $n = 2, 3, \ldots, 20$ variables, tandis que le second ensemble reprend la fonction de de Rosenbrock à $n = 100, 200, \ldots, 500$ variables.

Les différences entre les deux méthodes comparées portent sur l'étape de recherche de Mads. Dans sa version basique, aucune étape de recherche n'est effectuée tandis que dans la version Pca-Mads, une étape de recherche basée sur une analyse en composante principale telle que décrite au chapitre 4 est effectuée.

Les deux méthodes sont initialisées avec

— $\Delta^0 = 1$;
— $D = [I_n - I_n]$;
— $\tau = 2$,$\omega^- = -1$ et $\omega^+ = 1$;
— $\epsilon_{stop} = 10^{-15}$;
— Sonde de $2n$ points générés à la manière de LTMADS, avec stratégie opportuniste

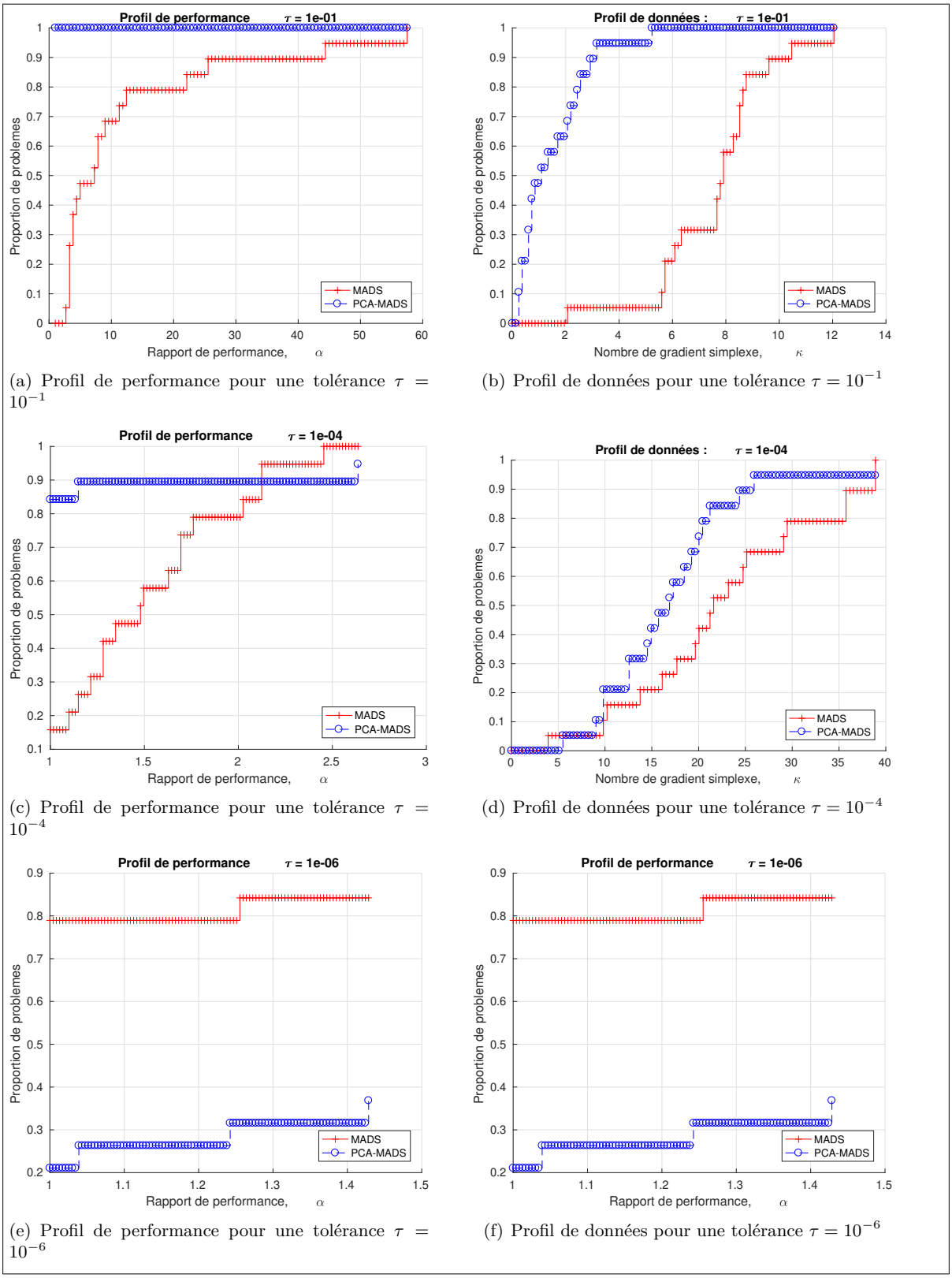
Pour l'étape de recherche de Pca-Mads, on utilise les paramètres suivants

— dimension $p = \lceil \frac{n}{5} \rceil$;
— nombre d'évaluations minimum pour effectuer une recherche $N_{min}^{search} = 2n$ généré par échantillonnage par hypercube latin ;
— nombre de points utilisées dans l'analyse en composante principale N_{pca}^{search} est l'ensemble des points évalués
— budget d'évaluations de l'optimisation du sous-problème $\frac{1}{20}$ du budget total ;
— transformation $x = x^k + Py$ (4.5) ;
— critère d'arrêt pour l'optimisation du sous-problème $\epsilon_{stop} = \frac{\Delta^k}{2}$;
— les autres paramètres de la méthode Mads optimisant le sous-problème sont les mêmes que pour la méthode Mads du problème original en dimension n.

Des profils de performance et de données ont été tracés pour une implémentation de Mads et de Pca-Mads. Ceux-ci sont représentés sur la figure 5.1. On remarque que lorsque la tolérance dur critère de convergence est assez grande, $\tau = 10^{-1}$ ou 10^{-4}, l'algorithme Pca-Mads semble avoir de meilleures performances que Mads, comme l'indiquent les figures 5.1(a), 5.1(b), 5.1(c) et 5.1(d). Avec une tolérance plus petite, le comportement des algorithmes semble s'inverser. Cela est visible sur les figures 5.1(e) et 5.1(f). Cela indique que Mads a tendance a trouvé des meilleures solutions que Pca-Mads, mais ce dernier trouve des solutions proches de la solution de Mads plus rapidement.

En regardant les profils pour un la même fonction en plus grande dimension, visibles sur la figure 5.2, on remarque un comportement différent. En effet, en plus grande dimension, l'algorithme Pca-Mads semble avoir de meilleures performances que l'implémentation Mads. Pour chacun des problèmes, Pca-Mads trouve de meilleures solutions et plus rapidement que Mads. Cela indique que la stratégie de réduction de dimension semble être plus efficace pour des problèmes grande dimension, aux alentours de quelques centaines de variables, par rapport aux problèmes en petite dimension, quelques dizaines de variables.

Le nombre et la variété de problèmes testés étant restreints, on ne peut conclure de manière définitive quant aux performances de chacun des algorithmes proposés. De plus, il y de nombreux paramètres et stratégies qui peuvent jouer sur la performance des algorithmes.



(a) Profil de performance pour une tolérance $\tau = 10^{-1}$

(b) Profil de données pour une tolérance $\tau = 10^{-1}$

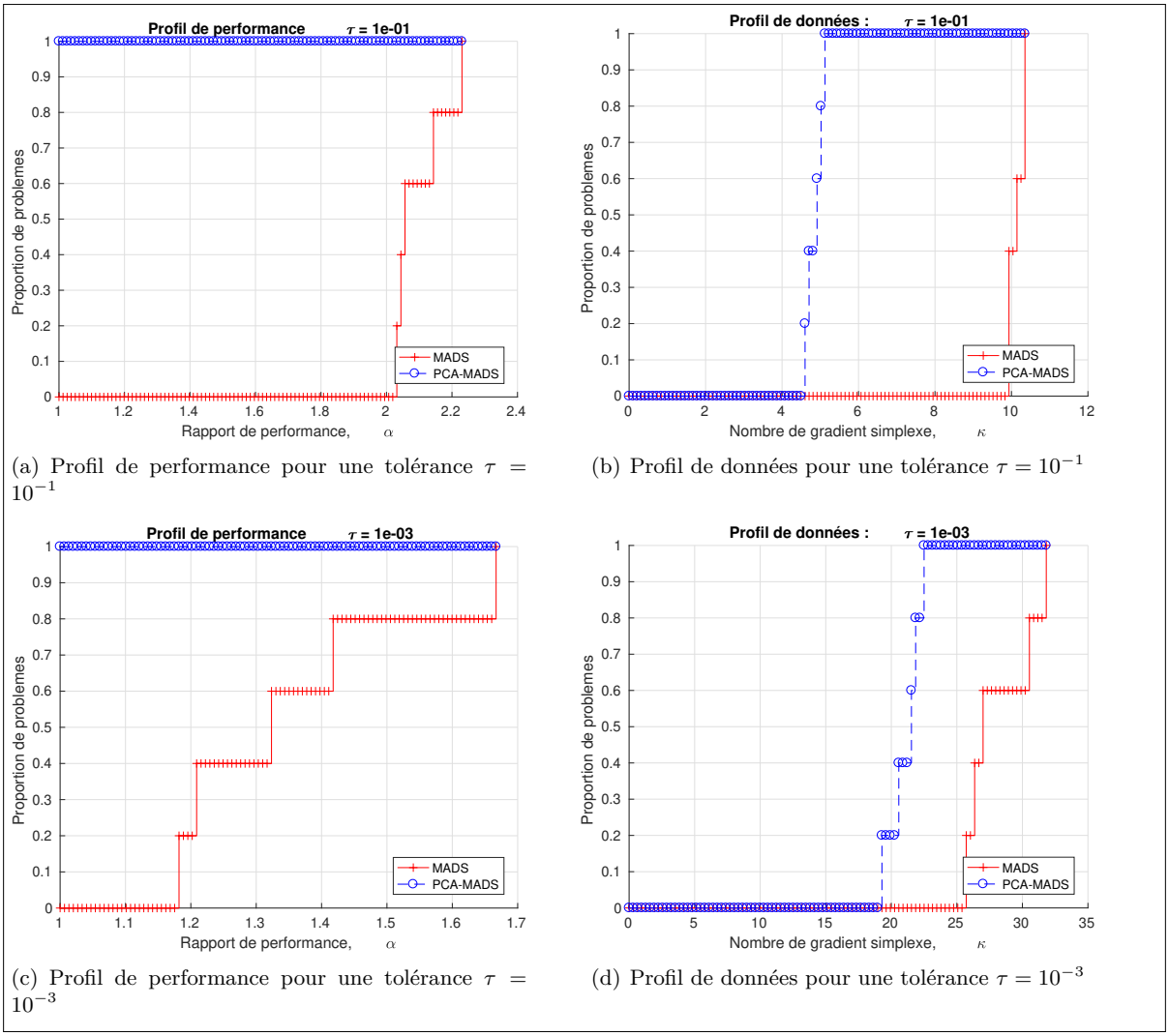
(c) Profil de performance pour une tolérance $\tau = 10^{-4}$

(d) Profil de données pour une tolérance $\tau = 10^{-4}$

(e) Profil de performance pour une tolérance $\tau = 10^{-6}$

(f) Profil de données pour une tolérance $\tau = 10^{-6}$

FIGURE 5.1 – Profils de performance et de données pour un ensemble de fonctions de Rosenbrock de dimension $2, 3, 4, \ldots, 20$ avec un budget de $100n$



(a) Profil de performance pour une tolérance $\tau = 10^{-1}$

(b) Profil de données pour une tolérance $\tau = 10^{-1}$

(c) Profil de performance pour une tolérance $\tau = 10^{-3}$

(d) Profil de données pour une tolérance $\tau = 10^{-3}$

FIGURE 5.2 – Profils de performance et de données pour un ensemble de fonctions de Rosenbrock de dimension $100, 200, \ldots, 500$ avec un budget de $50n$

5.3 Plateforme *COCO* et suite de fonctions *bbob*

COCO [31] est une plateforme dont le but est de comparer des méthodes d'optimisation de boîtes noires continues. Celle-ci procure plusieurs suites de problèmes à minimiser, de différentes dimensions et peuvent être bruités ou contraints. Les fonctions sont implémentées en C mais la plateforme offre des interfaces en C/C++, Java, Matlab/Octave ou Python. Le but de cette plateforme est d'automatiser la procédure de comparaison d'algorithme d'optimisation sans dérivées et de traitements des résultats.

Toutes ces suites se basent sur un ensemble de 24 fonctions définies dans [32]. Celles-ci peuvent être de différentes dimensions et être utilisées dans plusieurs instances. Toutes les fonctions peuvent être définies pour toute les dimension $n \geq 2$ et sont bornées par $[-5; 5]^n$. L'ensemble de fonctions peut être divisé en plusieurs sous-ensembles avec certaines caractéristiques, comme cela est décrit dans [32]. Ceux-ci sont

— 5 fonctions séparables ;
— 4 fonctions avec un conditionnement faible à modéré ;
— 5 fonctions avec un haut conditionnement et unimodales ;
— 5 fonctions multimodales avec une structure globale adéquate ;
— 5 fonctions multimodales avec une faible structure globale.

Pour chacune de ces fonctions, les solutions sont connues. La plateforme crée plusieurs instances de problème à partir de chaque et fourni les bornes et un point de départ pour chaque instance. Le budget d'évaluations total est défini par l'utilisateur via un facteur multipliant la dimension du problème.

La plateforme *COCO* traite également les résultats issus des expériences sur leur suite de fonctions. Ils utilisent comme élément central pour comparaison les différentes algorithmes, le nombre d'évaluation des fonctions pour atteindre un certain seuil, une certaine valeur de l'objectif. Celui-ci est appelé temps d'exécution. Nous préférons comparer les performances des algorithmes via des profiles de performances et de données décrits à la section 5.1.

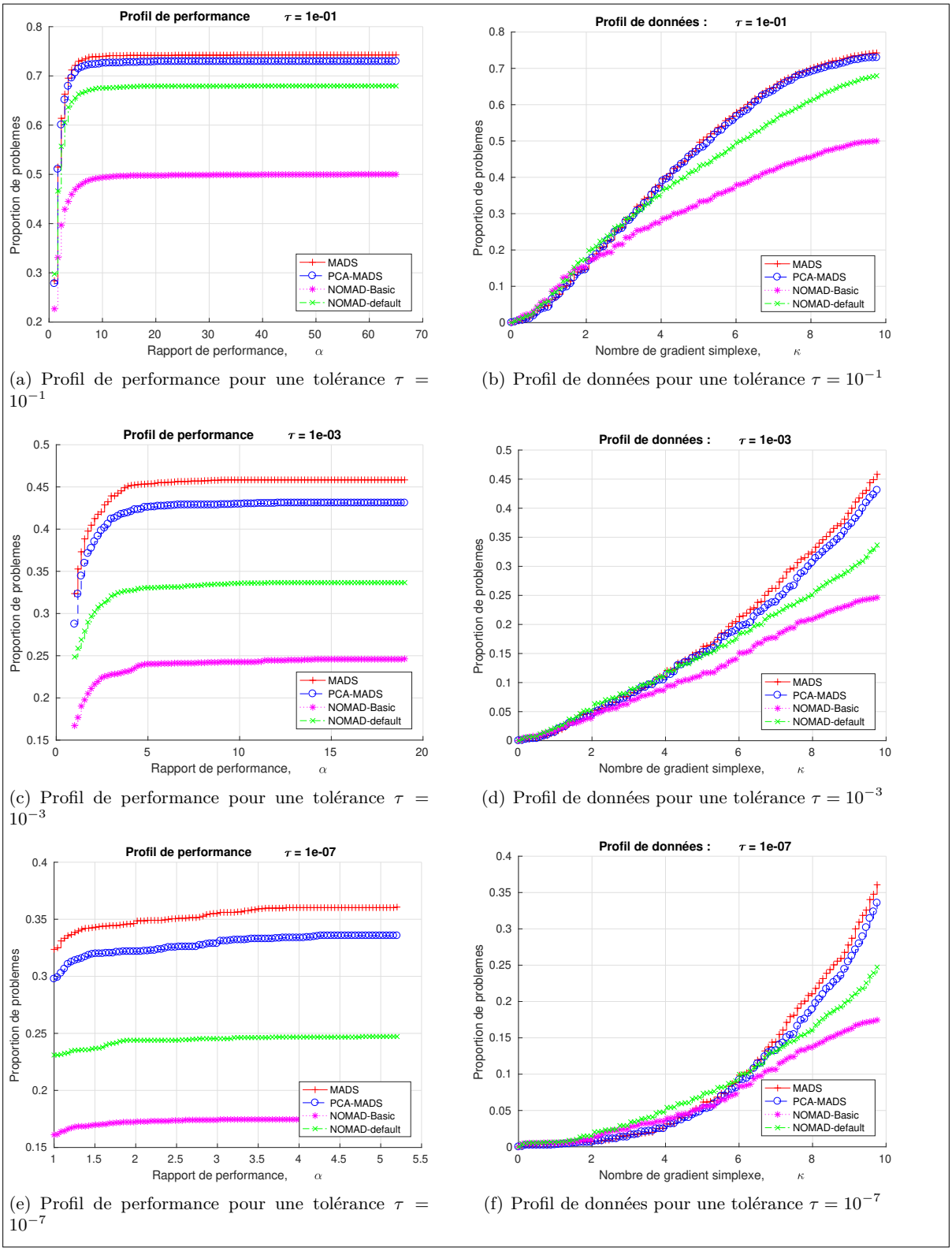
5.4 Tests sur la suite coco

Pour ces premiers tests sur la plateforme *COCO*, on veut comparer les performances de Pca-Mads et d'une version plus standart de Mads. Pour ce faire, on utilise une implémentation de Pca-Mads ainsi que la même implémentation avec l'étape de recherche désactivée. Cette dernière sera nommé simplement Mads. De plus, on voudrait également comparer les performances de ces algorithmes avec le *solver* `NOMAD3.9.1` [11, 1, 40]. On compare deux versions de `NOMAD3.9.1` . La première est une version basique où la plupart des fonctionnalités ont été désactivées comme l'étape de recherche, le treillis anisotropique ou l'utilisation de modèles. La seconde est la version de `NOMAD` , avec tous ses paramètres et options mis à leur valeur par défaut. La première version sera appelée `NOMAD-basique` tandis que la seconde sera appelée `NOMADdefault` .

L'ensemble de problèmes sur lesquels ces algorithmes seront comparés vient de la plateforme *COCO*. Dans un premier temps, on utilise la suite de fonctions *bbob*. Dans cette suite, on retrouve les 24 fonctions en dimension $2, 3, 5, 10, 20, 40$. Pour chaque fonction dans chaque dimension, 16 instances de problèmes sont créés. Cela donne un total de 2160 problèmes. On donne un budget de $10n$ évaluations pour chaque algorithme, où n est la dimension du problème.

La figure 5.3 présente des profils de performance et des profils de données pour ces tests. Grâce à ces profils, on remarque que la version par défaut de `NOMAD` a effectivement de meilleures performances que la version basique. Ensuite, les implémentations Mads et Pca-Mads ont des meilleures performances que `NOMAD` . Mads et Pca-Mads sont assez proches, avec un léger avantage à Mads tout de même. On note que ces tests ont été effectués sur une suite de fonctions de dimension assez petite et avec un budget assez restreint et ne permet de généraliser les résultats obtenus.

TODO : tests avec plus grand budget et sur la suite large scale pour comparer



(a) Profil de performance pour une tolérance $\tau = 10^{-1}$

(b) Profil de données pour une tolérance $\tau = 10^{-1}$

(c) Profil de performance pour une tolérance $\tau = 10^{-3}$

(d) Profil de données pour une tolérance $\tau = 10^{-3}$

(e) Profil de performance pour une tolérance $\tau = 10^{-7}$

(f) Profil de données pour une tolérance $\tau = 10^{-7}$

FIGURE 5.3 – Profils de performance et de données pour des implémentations de MADS et de PCA-MADS similaire ainsi que l'implémentation `NOMAD3.9.1` , une version basique et la version par défaut, sur la suite de fonctions *bbob* de *coco* avec un budget de $10n$

5.5 Influence des paramètres

Il existe plusieurs paramètres et stratégies pouvant influencer les performances de Pca-Mads. On voudrait donc étudier l'influence de ceux-ci pour pouvoir recommander des valeurs par défaut. Pour ce faire, on compare les performances des différentes versions de Pca-Mads au moyen de profils de performances et de données.

L'ensemble de fonctions utilisé pour ces tests est basé sur la suite de fonctions disponibles dans *coco*. On s'intéresse aux performances de l'algorithme sur des problèmes en relativement grande dimension, sans bruit et sans contraintes. C'est pourquoi nous avons choisi d'utiliser la suite *bbob-largescale*. Afin de limiter les temps d'exécution de ces tests, on se compte d'utiliser la première instance des 10 premières fonctions de la suite, et ce en dimension 80, 160, 320, 640. Cela donne un total de 40 problèmes au total.

5.5.1 Dimension du sous-problème p

Au cours de l'étape de recherche, l'algorithme Pca-Mads construit un sous-problème de plus faible dimension et cherche à l'optimiser. La dimension de ce sous-problème semble être un paramètre critique, qui peut influencer les performances de l'algorithme. On veut donc comparer plusieurs stratégies. On note n la dimension du problème original et p la dimension du sous-problème. Les premières stratégies consistent simplement par diviser la dimension du problème original par un facteur constant. Chaque sous-problème construit aura donc la même dimension. Une alternative à cette stratégie est d'utiliser un algorithme de classement pour séparer les directions issues de l'analyse en composante principales et de les diviser en deux groupes. On utilise l'algorithme $k - means$, avec $k = 2$, pour séparer les $(n + 1)$ directions en deux groupes. La dimension du problème p correspondra à la taille de l'ensemble des directions ayant les directions les plus influentes sur l'objectif.

Toutes les stratégies comparées sont reprises dans la liste ci-dessous.

1. Stratégie $p = \lceil \frac{n}{5} \rceil$;
2. Stratégie $p = \lceil \frac{n}{10} \rceil$;
3. Stratégie $p = \lceil \frac{n}{20} \rceil$;
4. Stratégie $k - means$.

Les performances de l'algorithme en fonction de ces stratégies sont comparées à l'aide de profils de données et de performances tels que décrit à la section 5.1. Ceux-ci sont repris à la figure 5.4.

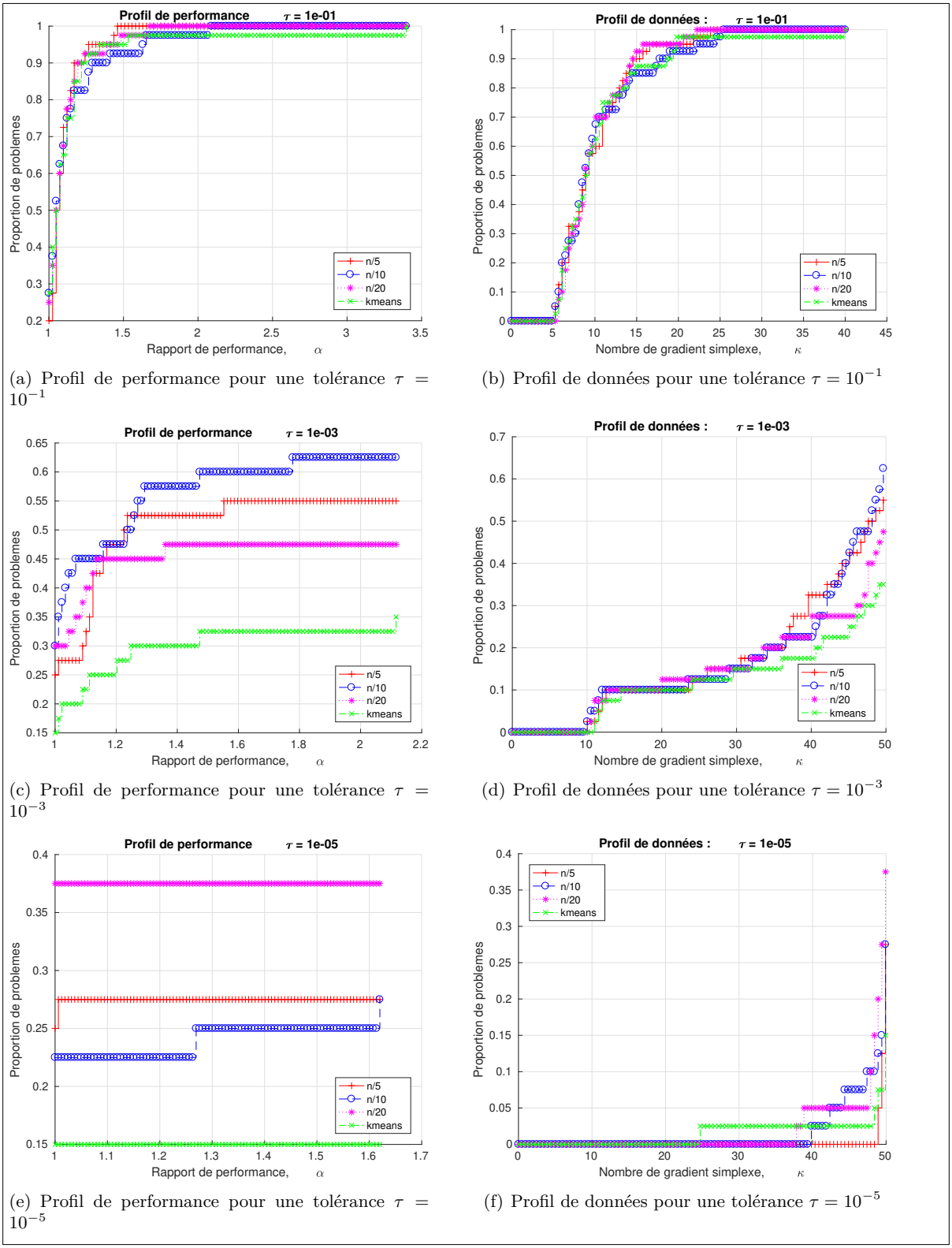
Sur les profiles 5.4(a) et 5.4(b), avec une assez faible précision, $\tau = 10^{-1}$, pour le critère de convergence, les différentes stratégies proposées semblent avoir des performances assez similaires. Lorsque le critère de convergence devient plus précis, par exemple $\tau = 10^{-3}$ sur les profiles 5.4(c) et 5.4(d), deux stratégies se distinguent. Il s'agit des stratégies $p = \lceil \frac{n}{5} \rceil$ et $p = \lceil \frac{n}{10} \rceil$. Cela peut s'expliquer par les dimensions des problèmes utilisés pour la comparaison. En effet, avec la stratégie $p = \lceil \frac{n}{5} \rceil$ (resp. $p = \lceil \frac{n}{10} \rceil$, on construit des sous-problèmes de taille 16 à 128 (resp. 8 à 64). Il s'agit de taille raisonnable pour lesquelles l'algorithme Mads a de bonnes performances. Lorsque le critère de convergence pour la création des profiles est encore plus strict, avec $\tau = 10^{-5}$ par exemple, la stratégie $p = \lceil \frac{n}{20} \rceil$ est la meilleure des stratégies comparées. Sur le profile de performance 5.4(e), on peut voir qu'aucune des autres stratégies ne parvient à s'approcher de la stratégie $p = \lceil \frac{n}{20} \rceil$.

On note également que la stratégie basée sur $k - means$ ne semble pas être un choix judicieux.

5.5.2 Budget d'évaluations pour l'optimisation du sous-problème

L'étape de recherche consiste à optimiser un nouveau problème d'optimisation de petite dimension avec un algorithme Mads. Le budget alloué à cette optimisation aura une influence sur la qualité de celle-ci. Dans le cadre de l'algorithme Pca-Mads, il y a une alternance entre cette minimisation et une étape de sonde en grande dimension. Le budget maximum d'évaluations pour l'optimisation de ce sous-problème devrait avoir une influence sur les performances de Pca-Mads. On note n la dimension du problème original, B le budget total d'évaluations et ev le nombre d'évaluations déjà effectuées.

On propose de comparer plusieurs stratégies :



(a) Profil de performance pour une tolérance $\tau = 10^{-1}$

(b) Profil de données pour une tolérance $\tau = 10^{-1}$

(c) Profil de performance pour une tolérance $\tau = 10^{-3}$

(d) Profil de données pour une tolérance $\tau = 10^{-3}$

(e) Profil de performance pour une tolérance $\tau = 10^{-5}$

(f) Profil de données pour une tolérance $\tau = 10^{-5}$

FIGURE 5.4 – Profils de performance et de données pour des implémentations de PCA-MADS avec différentes stratégies de réduction de dimension, sur une partie de la suite de fonctions *bbob-largescale* (première instance des fonctions 1 à 10) en dimension $80, 160, 320, 640$ de *coco* avec un budget de $50n$

1. Stratégie $\frac{B}{10}$: le budget maximum d'évaluations pour le sous-problème correspond à un dixième du budget total d'évaluations.
2. Stratégie $\frac{B}{20}$: le budget maximum d'évaluations pour le sous-problème correspond à un dixième du budget total d'évaluations.
3. Stratégie n : le budget maximum d'évaluations pour le sous-problème correspond au nombre de variables du problème original.
4. Stratégie $2n$: le budget maximum d'évaluations pour le sous-problème correspond à deux fois le nombre de variables du problème original.
5. Stratégie *inc* : à étape de recherche, on donne un budget de $\lceil(1 + \frac{5ev}{B})\frac{n}{2}\rceil$ évaluations. Cela donne un budget minimum de $\frac{n}{2}$ et maximum de $3n$ évaluations et le budget augmente à chaque nouvelle étape de recherche.
6. Stratégie *dec* : à étape de recherche, on donne un budget de $\lceil(1 - \frac{5ev}{6B})3n\rceil$ évaluations. Cela donne un budget minimum de $\frac{n}{2}$ et maximum de $3n$ évaluations et le budget diminue à chaque nouvelle étape de recherche.

Plusieurs profiles de données et de performances ont été tracés pour comparer ces stratégies. Ceux-ci sont repris à la figure 5.5. Avec une faible précision, dont les profils sont visibles sur les figures 5.5(a) et 5.5(b), la stratégie qui vise à diminuer la budget au fur à mesure des itérations semblent un peu plus intéressantes que les autres, bien que ces dernières soient relativement proches. Lorsque la précision augmente, la stratégie $2n$ s'isole comme une meilleure stratégie que les autres.

5.5.3 Ensemble de points utilisé pour l'analyse en composante principale

TODO : Combien ? Dernier points ? Points autour de la solution courante

5.5.4 Points de sonde

TODO : 2n, n+1 ou 2 points ?

5.5.5 Stratégie de construction et d'évaluation du sous-problème

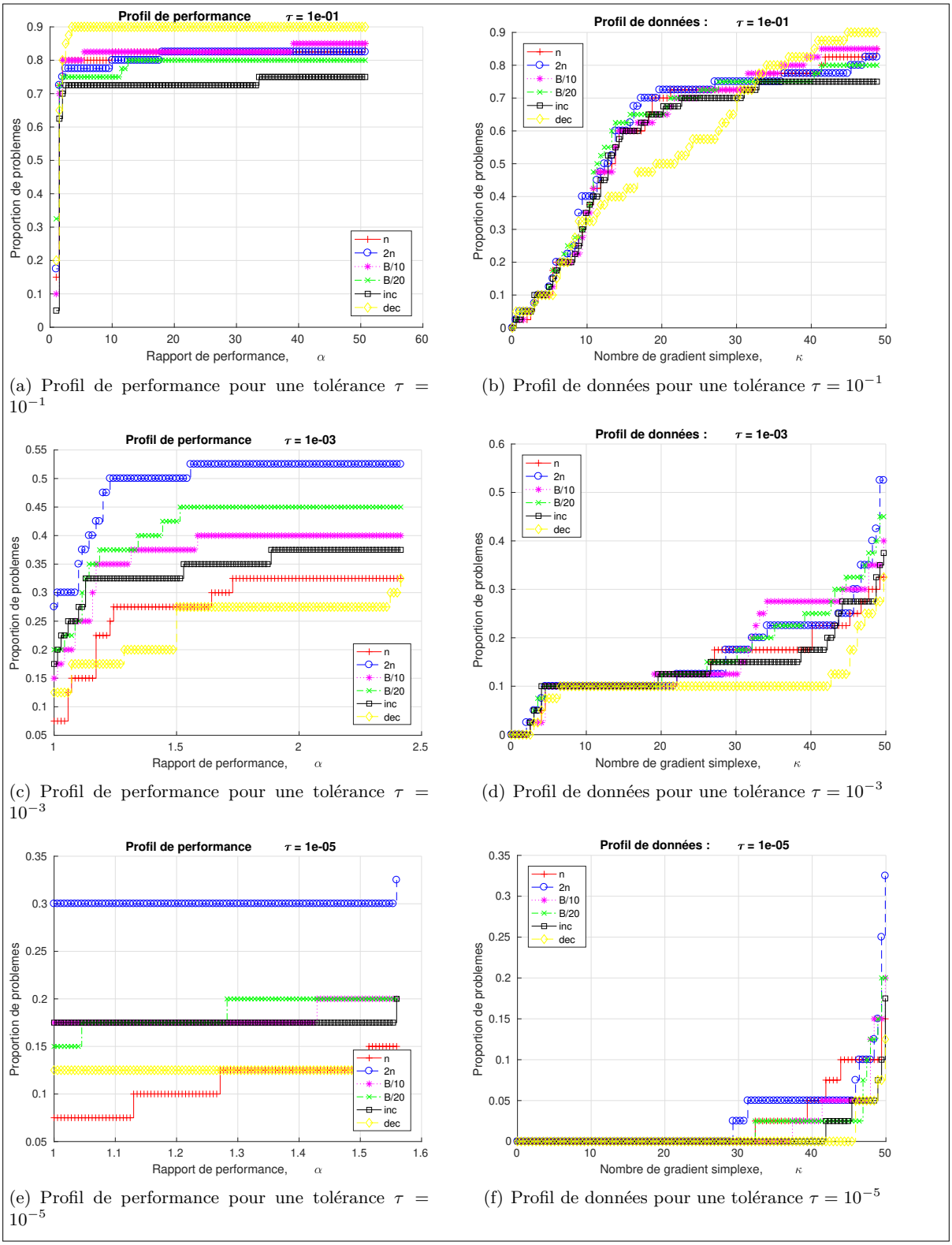
TODO : Fixer les paramètres et en faire varier qu'un seul et comparer les performances sur cococ largescale

5.6 Comparaison avec d'autres méthodes et algorithmes

TODO : Comparer avec nomad défaut et basic, stats mads et peut-etre d'autres méthodes (GPS ?, Algo génétique ?, Bayensian optimization ?,...) sur coco large scale

5.7 Boîtes noires réelles

TODO : Comparer les mêmes algos sur des boîtes noires réelles et checker s'il y a une différence



(a) Profil de performance pour une tolérance $\tau = 10^{-1}$

(b) Profil de données pour une tolérance $\tau = 10^{-1}$

(c) Profil de performance pour une tolérance $\tau = 10^{-3}$

(d) Profil de données pour une tolérance $\tau = 10^{-3}$

(e) Profil de performance pour une tolérance $\tau = 10^{-5}$

(f) Profil de données pour une tolérance $\tau = 10^{-5}$

FIGURE 5.5 – Profils de performance et de données pour des implémentations de PCA-MADS avec différentes budget d'évaluations pour l'optimisation du sous-problème, sur une partie de la suite de fonctions *bbob-largescale* (première instance des fonctions $2, 4, 7, 9, 11, 13, 16, 18, 21$ et 23) en dimension $80, 160, 320$ de *coco* avec un budget de $50n$