Optimisation coûteuse boîte noire à variables mixtes

Revue de littérature

Sanaa Zannane

EDF R&D - PRISME

Octobre 2020

Sommaire

Section 1

Introduction

Contexte

La conception de centrales photovoltaïques est guidée par :

- le respect de contraintes spécifiques (surface max, puissance installée min, etc.)
- la maximisation de la rentabilité espérée du projet.

Les écueils méthodologiques de ce problème d'optimisation sont :

- les outils de simulation de la production d'une centrale sont coûteux,
- les nombreuses variables d'optimisation (plusieurs dizaines), continus (emplacements des panneaux, inclinaison, orientation, etc.), entiers / ordinaux (nombre de modules installés, répartition en tables, etc.) ou catégoriels (choix technologiques du module, type d'onduleur électrique, etc.)
- l'espace des variables est fortement contraint par les liens de compatibilité entre choix technologiques.

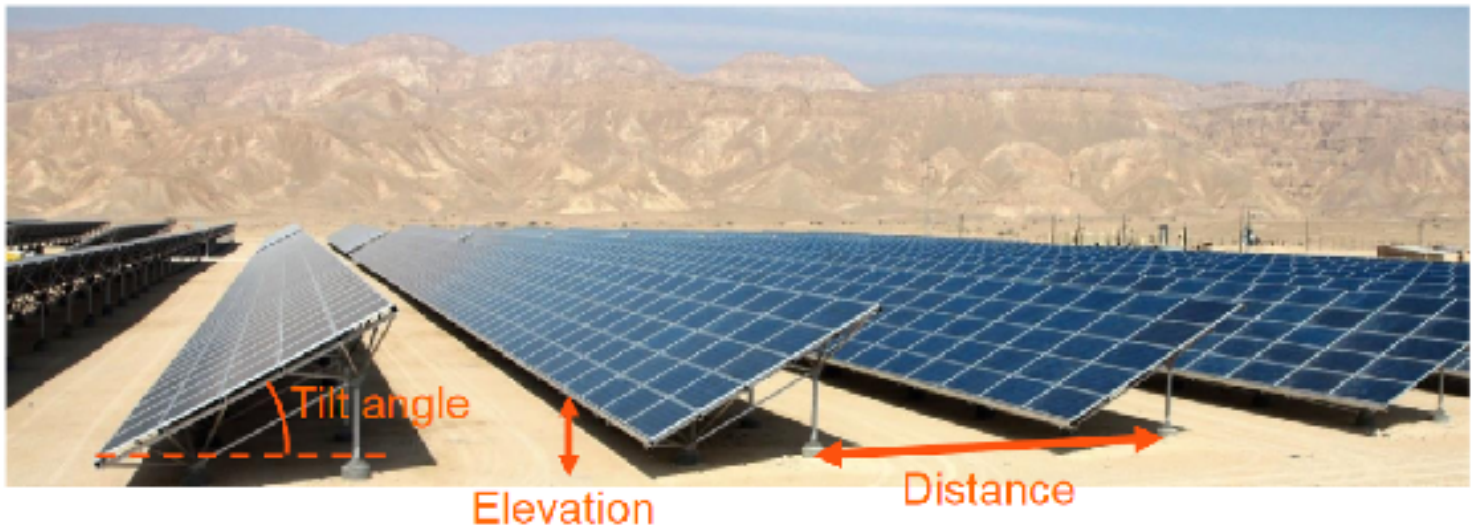


Figure: *Optimisation de la conception d'une centrale PV. Variables continues. Source : [1].*

Cas industriel simplifié

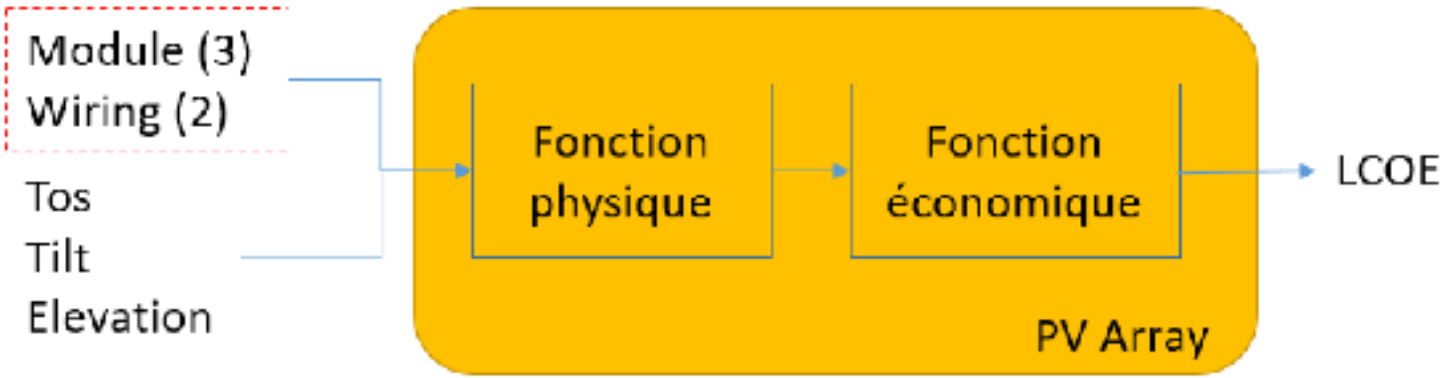


Figure: *Simulateur du LCOE d'un parc photovoltaïque*

Les entrées envisagées sont de dimension 5, avec trois variables continues :

1. $\texttt{TOS} \in [40\%, 90\%]$: taux d'occupation du sol, autrement dit la densité de couverture du terrain occupé par les panneaux photovoltaïques;
2. $\texttt{Tilt} \in [20°, 45°]$: angle d'orientation des panneaux par rapport au sol;
3. $\texttt{Elevation} \in [0.5m, 1.5m]$: hauteur des panneaux par rapport au sol.

On dispose également de deux variables d'entrée catégorielles, de modalités 3 et 2 respectivement :

1. `Module` : correspond à 3 choix de technologie de modules photovoltaïques ;
2. $\texttt{Wiring} \in \{0, 1\}$: correspond à 2 choix de géométrie du cablâge électrique (linéaire ou en «U»).

Cas jouet n°1

- On envisage une première fonction test, appelée f_1 et définie sur $G = [-5, 5] \times [-5, 5] \times \{a, b\} \times \{0, 1\}$, s'écrit comme suit :

$$f_1(x_1, x_2, x_3, x_4) = \begin{cases} x_1^2 + x_2^2 + 10x_4, & \text{si } x_3 = a, \\ x_1 + x_2 - 10x_4, & \text{si } x_3 = b, \end{cases} \tag{1}$$

- où $(x_1, x_2, x_3, x_4) \in G$.

Cas jouet n°1

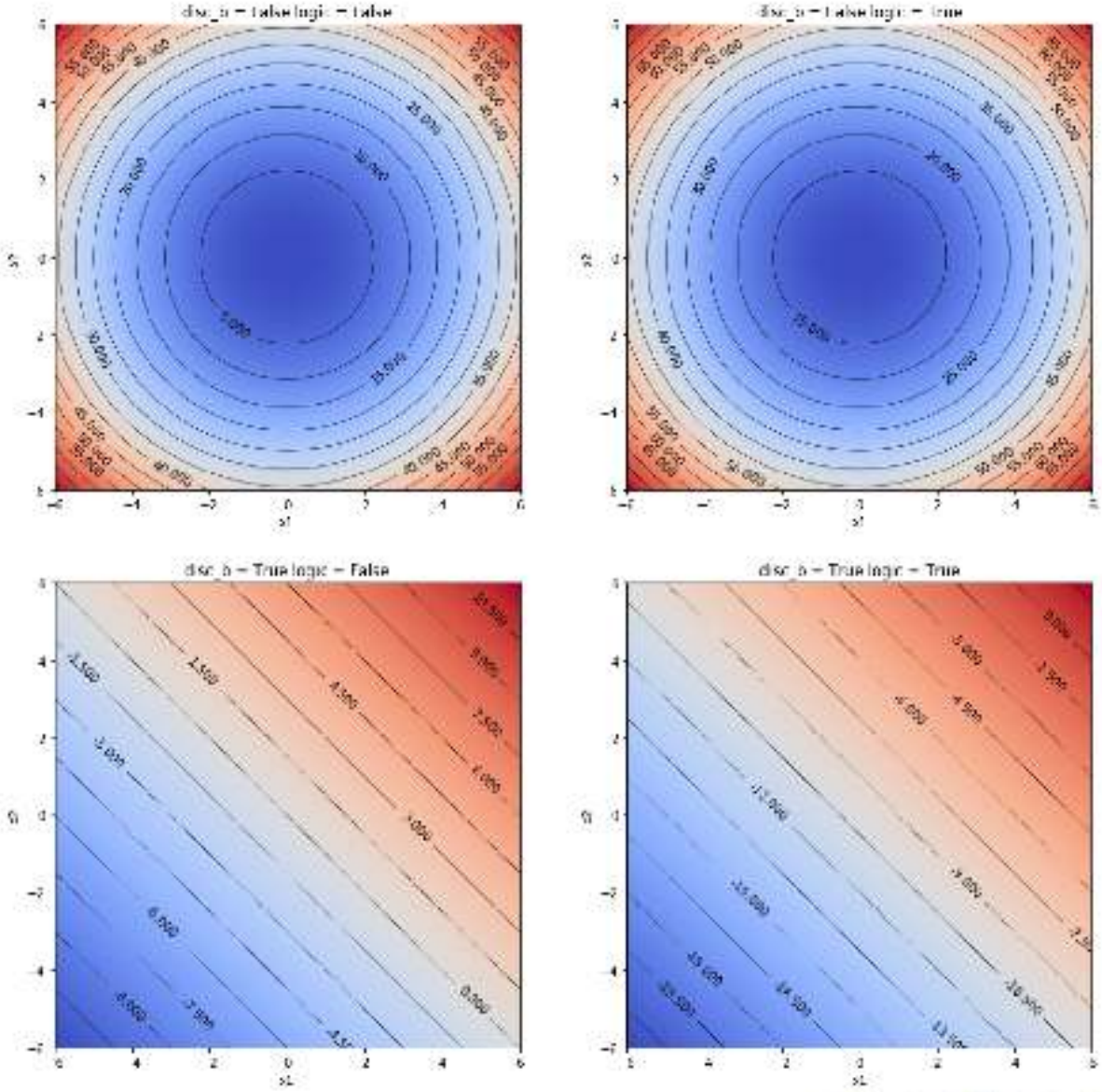


Figure: *Première fonction jouet. Optimum -20 atteint en* $[-5, -5, b, Vrai]$

Cas jouet n°2

La seconde fonction, plus complexe que la première en raison des nombreux minima locaux, est définie sur $G = [-5, 5] \times [-5, 5] \times \{1, 2, 3, 4, 5\}$. Elle correspond à la fonction dite Swiler2014 et s'écrit comme suit, en notant $x = (x_1, x_2, x_3)$:

$$f_2(x) = \begin{cases} res_1(x), & \text{si } x_3 = 1 \\ res_1(x) + 12 res_2(x), & \text{si } x_3 = 2, \\ res_1(x) + 0.5 res_2(x), & \text{si } x_3 = 3, \\ res_1(x) + 8 res_2(x), & \text{si } x_3 = 4, \\ res_1(x) + 3.5 res_2(x) & \text{si } x_3 = 5, \end{cases} \tag{2}$$

où :

$$\begin{cases} res_1(x_1, x_2, x_3) = \sin(2\pi x_2 - \pi) + 7 \sin(2\pi x_1 - \pi)^2 \\ res_2(x_1, x_2, x_3) = \sin(2\pi x_2 - \pi). \end{cases} \tag{3}$$

Cas jouet n°2

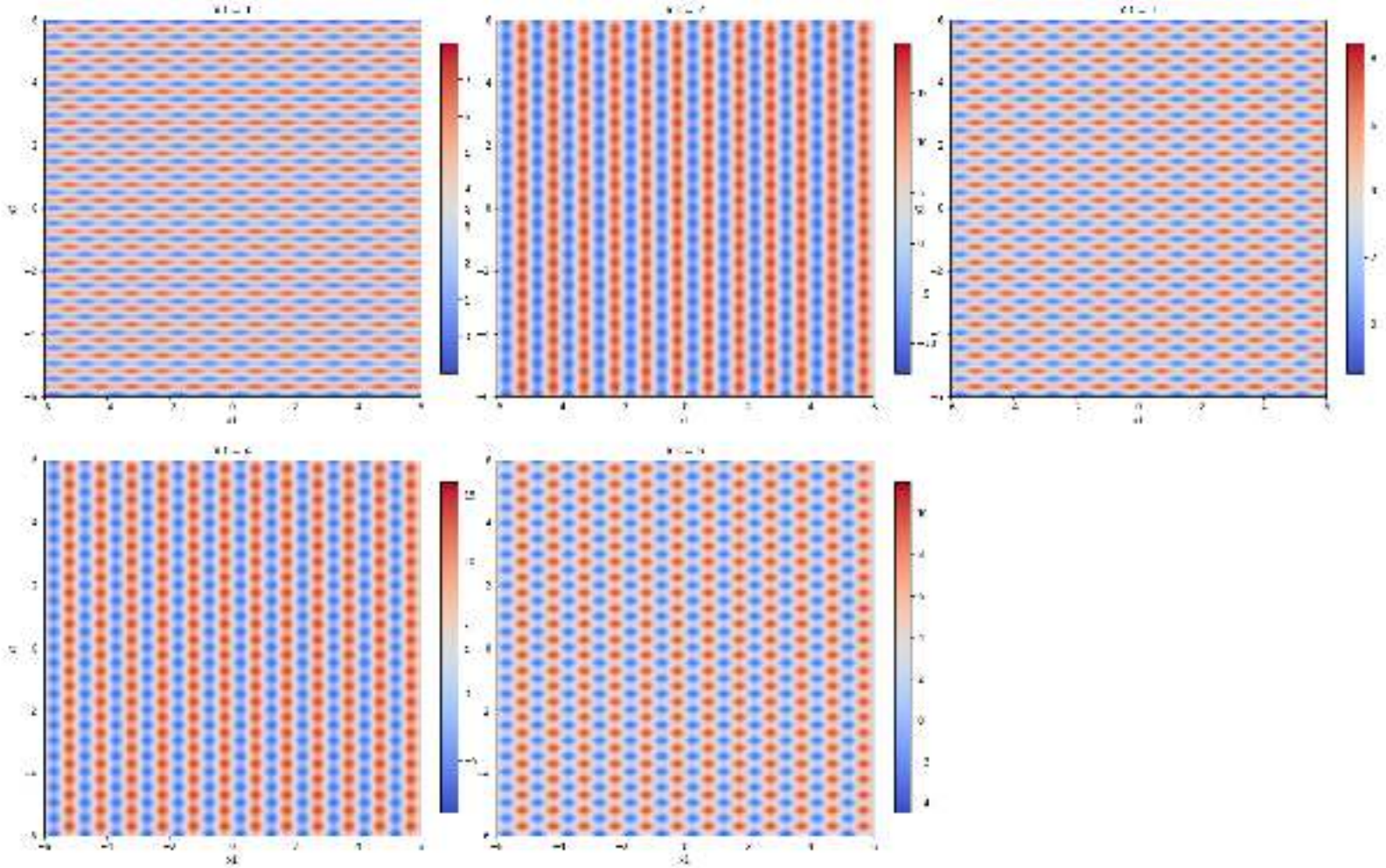


Figure: *Seconde fonction jouet. Optimum* $\simeq -13$ *atteint par exemple en* $[0, 0.25, 2]$

Cas jouet n°2

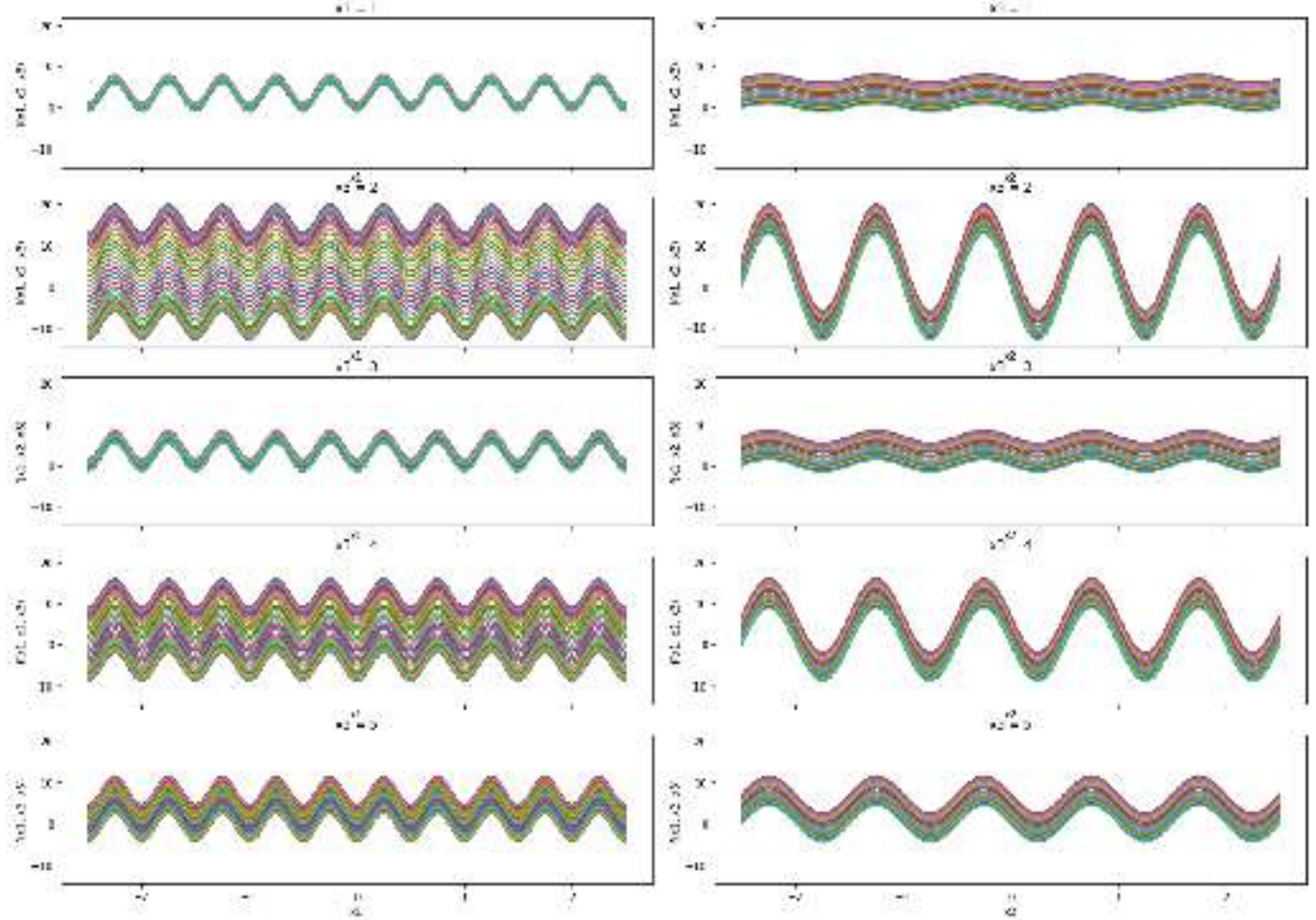


Figure: Le graphique en ligne i et colonne 1 représente $x_1 \longrightarrow f_2(x_1, x_2, i)$ pour différentes valeurs x_2. Celui en ligne i et colonne 2 représente $x_2 \longrightarrow f_2(x_1, x_2, i)$ pour différentes valeurs x_1.

Notations

- On considère le problème d'optimisation de dimension d suivant :

$$x^* \in \arg\min_{x \in G} F(x) \tag{4}$$

- où x représente le vecteur des variables de décision, F la fonction objectif coûteuse, et G le domaine admissible. On peut considérer, sans perte de généralité que :

$$G \subseteq \mathbb{R}^{d_{cont}} \times \mathbb{N}^{d_{ent}} \times \{0,1\}^{d_{cat}} \quad \text{tq.} \quad d_{ent} + d_{cont} + d_{cat} = d \tag{5}$$

Optimisation par métamodèle

On peut schématiser le fonctionnement d'un algorithme d'optimisation à base de métamodèle (dit EGO - Efficient Global Optimization) comme suit :

1: $D_0 \leftarrow \emptyset$ (ou un plan d'expérience initial)
2: **for** $n \in [|1 : n_{max}|]$ **do**
3: Optimisation du critère de remplissage : $x^* \leftarrow argmax\ \mathcal{C}(\hat{F}_{D_n}, \tilde{F})(x)$
4: Évaluation coûteuse : $F(x^*)$
5: Enrichissement du plan d'expérience : $D_{n+1} \leftarrow D_n \cup (x^*, F(x^*))$
6: Estimation d'un nouveau métamodèle : $\hat{F}_{D_{n+1}}$
7: **end for**
8: **return** $\tilde{x}^* = argmin D_{n_{max}}$

Notations

- On notera :

$$D_n = (x_i, F(x_i))_{i\in[|1:n|]} \quad \text{tq.} \quad (x_i)_{i\in[|1:n|]} \in G \tag{6}$$

  un plan d'expérience associé à la fonction F.
- On notera, le cas échéant, $\hat{F}_{D_n}$ la fonction métamodèle de F, c'est à dire une fonction approchant F au sens d'une certaine distance et construite par apprentissage statistique en utilisant le plan d'expérience D_n.
- Les valeurs de la fonction F étant inconnues en dehors des points calculés contenus dans D_n, on peut considérer par hypothèse que $(F(x))$ est une variable aléatoire pour laquelle on dispose d'un tirage aléatoire $(F(x_i))_{i\in[|1:n|]}$ associé à des variables explicatives $(x_i)_{i\in[|1:n|]}$ observées.
- $\hat{F}_{D_n}$ étant un modèle de régression statistique, $(\hat{F}_{D_n}(x))$ est aussi une variable aléatoire, et on peut poser la loi de probabilité conditionnelle suivante : $p(\hat{F}_{D_n}(x)|x)$.

Notations

- Enfin, on note $\mathcal{C}$ le «critère de remplissage» envisagé. Ce critère est central pour les algorithmes d'optimisation à base de métamodèles [3]. Il traduit l'arbitrage entre l'exploitation des évaluations coûteuses déjà effectuées et l'exploration des régions «mal estimées » de l'espace admissible.
- Un exemple de critère de remplissage est l' Expected Improvement (`EI`) ou amélioration espérée, défini par :

$$EI(x) := \mathbb{E}_{\hat{F}_{D_n}(x)}[max(F(x^*_{D_n}) - \hat{F}_{D_n}(x), 0)] \tag{7}$$

où $\mathbb{E}_{\hat{F}_{D_n}(x)}$ désigne l'espérance sous la loi de $\hat{F}_{D_n}(x)$ conditionnée par x; et où $x^*_{D_n}$ correspond à la meilleure évaluation coûteuse présente dans le plan d'expérience à l'itération n.

Section 2

TPE

Description de l'algorithme TPE

L'algorithme Tree-structured Parzen Estimator (TPE)[5] consiste à rechercher itérativement l'optimum en suivant le schéma général et en maximisant le critère EI.

- Soit $y := \hat{F}_{D_n}(x)$ la réponse d'un métamodèle $\hat{F}$. En supposant que y admet une densité de probabilité notée $p(y)$, on peut écrire que :

$$EI(x) = \int_{-\infty}^{F(x^*_{D_n})} (F(x^*_{D_n}) - y) p(y|x) dy \tag{8}$$

- L'idée centrale de TPE est de non plus estimer un modèle de probabilité portant sur les sorties y mais plutôt d'en estimer un sur l'espace des entrées. Il repose pour cela sur le théorème de Bayes et introduit l'inversion suivante :

$$\forall x \in G : \qquad p(y|x) = \frac{p(x|y)p(y)}{p(x)}. \tag{9}$$

Description de l'algorithme TPE

- Si on suppose de plus que la paramétrisation suivante est licite, où $l(x)$ et $g(x)$ sont deux densités de probabilité inconnues :

$$\forall y \in \mathbb{R} : \qquad p(x|y) = \begin{cases} l(x), & \text{si } y < F(x^*_{D_n}) \\ g(x), & \text{sinon.} \end{cases} ; \tag{10}$$

- Alors on peut établir, comme démontré dans [5], que :

$$EI(x) \propto \left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)^{-1} ; \tag{11}$$

où $\gamma = p(y < F(x^*_{D_n})$ est une constante par rapport à x.

- Il suffit donc, pour maximiser le critère EI, de maximiser la quantité $\frac{l(x)}{g(x)}$, qui peut se traduire par la recherche du point maximisant la probabilité d'être dans «la bonne classe» (i.e. les points dont la valeur est proche de l'optimum) tout en minimisant la probabilité d'apprtenir à «la mauvaise classe».

Description de l'algorithme TPE

En pratique, l'algorithme TPE estime les densités de (10) en séparant le jeu de données en cours D_n en deux classes : celle des points x_i dont la valeur $F(x_i)$ est proche de l'optimum en cours $F(x^*_{D_n})$ à une marge près (typiquement les 20% meilleures valeurs), et celle contenant tous les autres points.

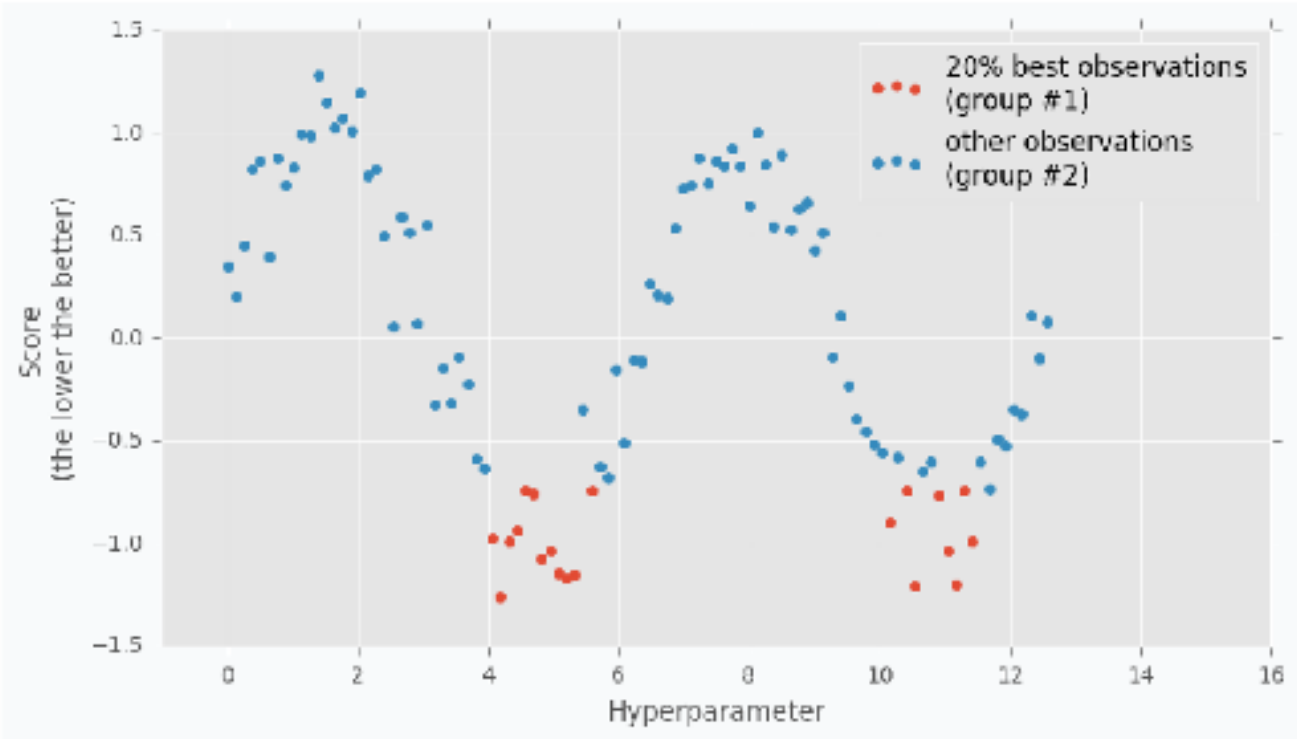
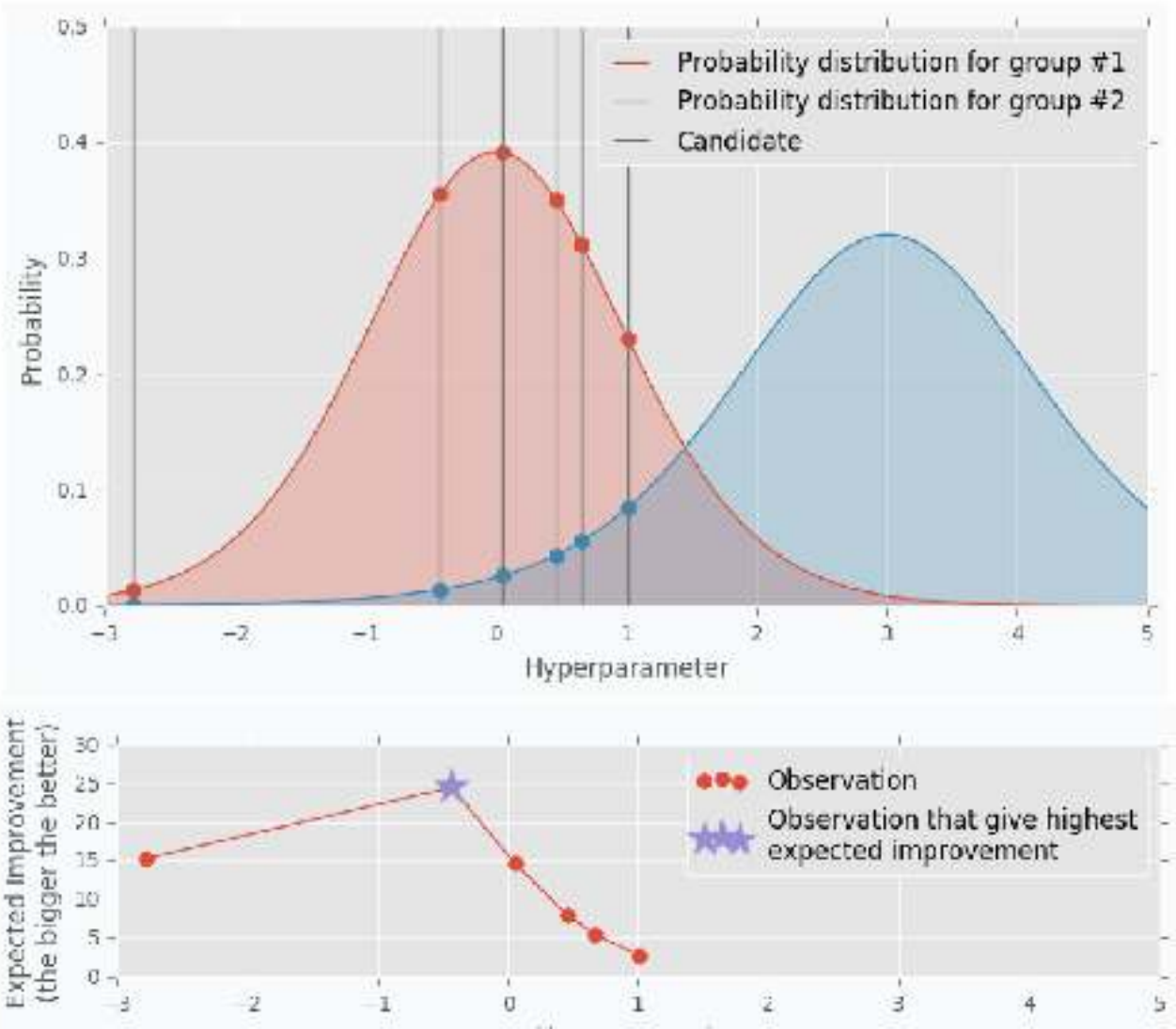


Figure: Séparation du jeu de données en deux sous-classes. Source : [6]

Description de l'algorithme TPE

La densité $x \to l(x)$ est estimée semble-t-il non paramétriquement (d'où le nom Parzen) en utilisant les points de la première classe, et $g(x)$ en utilisant ceux de la seconde. Toutefois, [5] est quelque peu évasif sur les propriétés de l'estimateur mis en oeuvre.



Implémentation de l'algorithme

L'implémentation Python Hyperopt a été utilisée pour cette étude [4].
L'API Hyperopt est très fluide et simple à utiliser, comme le montre l'extrait du code d'optimisation de la fonction $f = f_2$ ci-dessous :

```
from hyperopt import hp, fmin,  tpe,  Trials
import numpy as np

tpe_trials = Trials()
space = [hp.uniform('x1',-5,5), hp.uniform('x2', -5,5),
         hp.choice('x3',[0,1]), hp.choice('x4',[0,1])]

best = fmin(f, space, algo = tpe.suggest,
            max_evals = 100, trials = tpe_trials,
            rstate= np.random.RandomState(1234))

print('optimum : ' + str(tpe_trials.best_trial['result']['loss']));
print('obtenu en : ' +str(best))
```

Résultats pour la fonction jouet 2

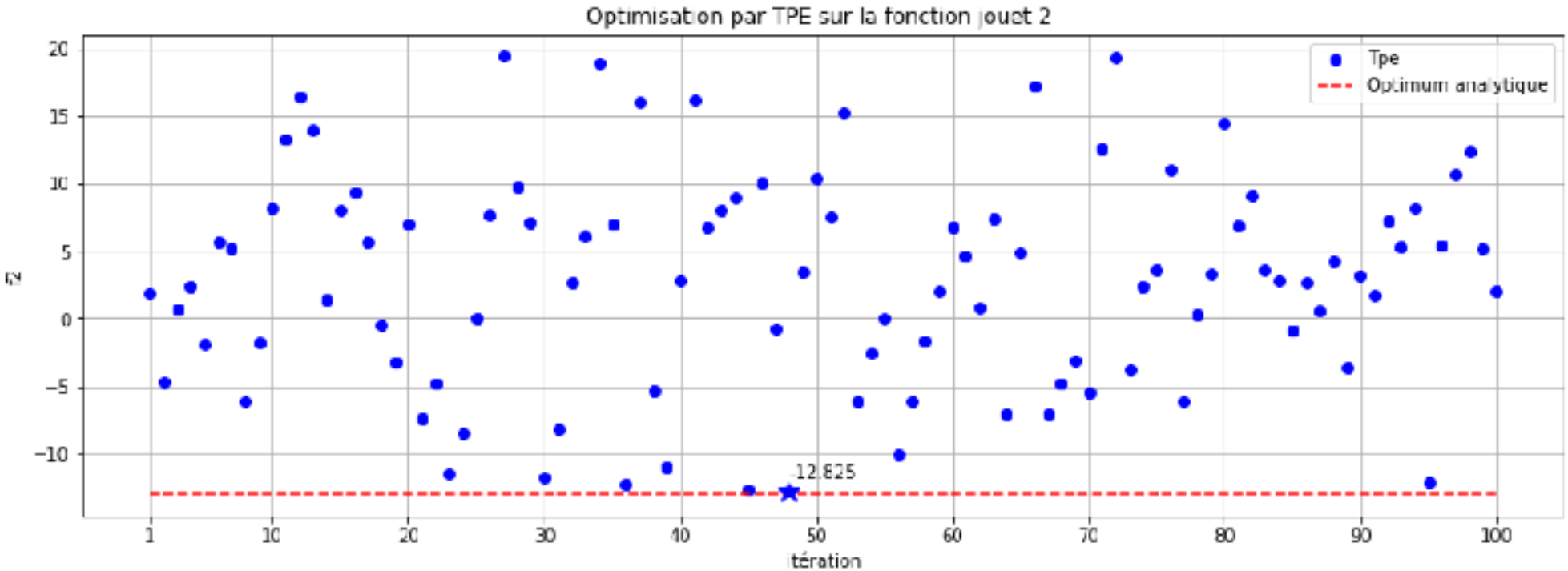
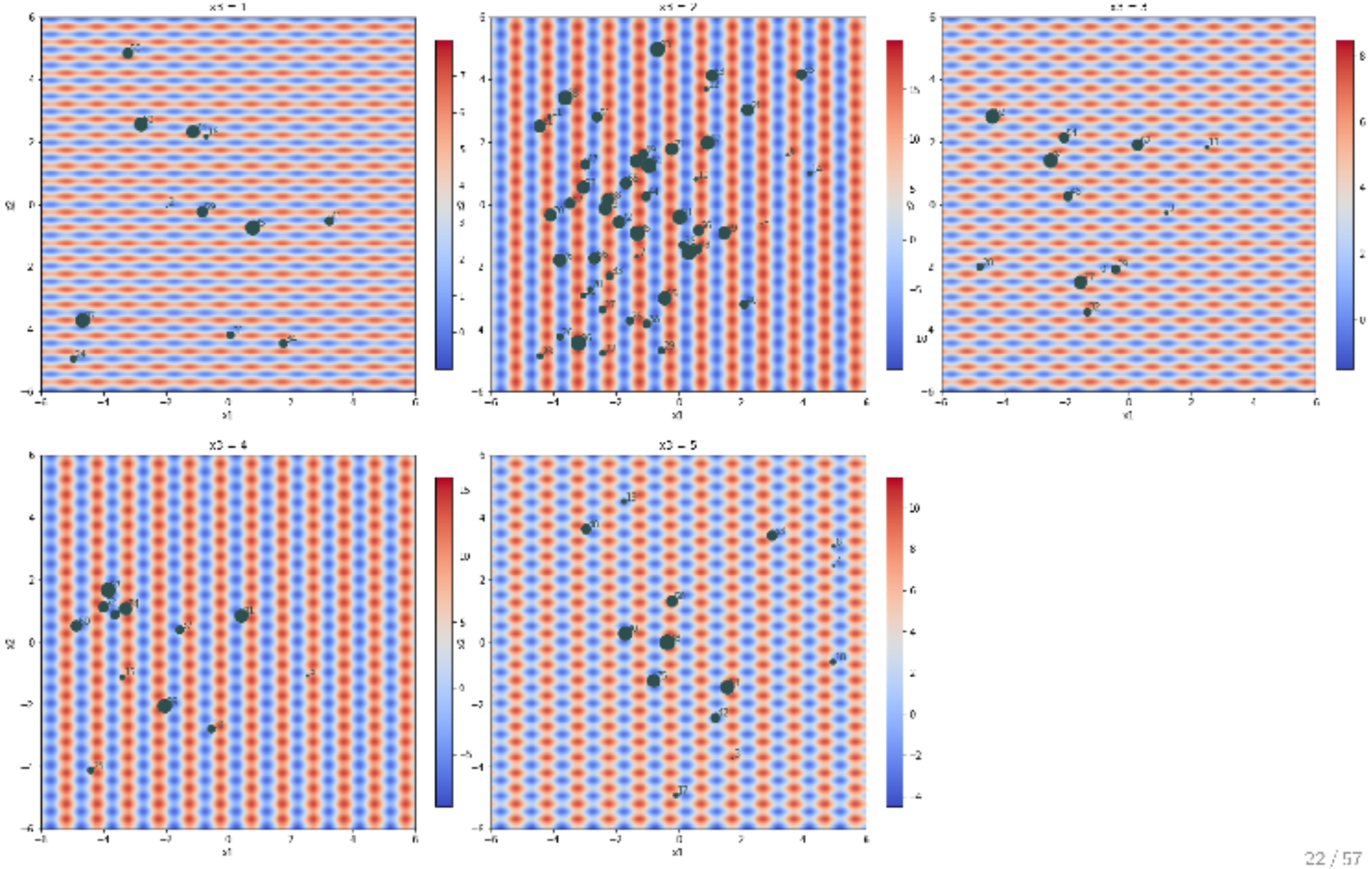


Figure: *Valeurs calculées pour chaque itération de l'algorithme TPE. Meilleure valeur obtenue : -12.8*

Résultats pour la fonction jouet 2



Conclusion sur TPE

Cet algorithme, couplé avec une phase d'optimisation continue, nous semble prometteur :

- Il a permis expérimentalement de proposer des solutions dont les performances sont meilleures que la solution de référence pour le cas industriel;
- Il approche relativement bien l'optimum analytique des fonctions jouets envisagées;
- Le formalisme de TPE, développé à la faveur des travaux sur l'optimisation des hyperparamètres de modèles d'apprentissage automatique, est adapté aux situations où les variables de décision sont interdépendantes et obéissent à une logique séquentielle. L'extrait de code ci-dessous décrit un espace de recherche à 3 paramètres : a, pouvant valoir uniformément $case1$ ou $case2$, $c1$ une variable continue activée ssi. $a = case1$ et $c2$ une autre variable continue activée ssi. $a = case2$.

```
from hyperopt import hp
space = hp.choice('a',
    [
        ('case 1', 1 + hp.lognormal('c1', 0, 1)),
        ('case 2', hp.uniform('c2', -10, 10))
    ])
```

Section 3

EGO avec forêts aléatoires

Description de l'algorithme

L'algorithme `EGO-RF` utilisé lors de cette étude dérive du schéma général décrit précédemment, en prenant comme :

- Métamodèle : un métamodèle de type régression par `Random forest`.
- Critère de remplissage : deux critères de remplissage ont été testés :
  1. Le critère de l'`EI` , estimé en considérant que l'ensemble des prédictions associées à chacun des arbres `CART` constitutifs du `Random Forest` est un tirage suivant la loi $\hat{F}(x)$. On prend comme estimateur empirique de l'espérance l'estimateur usuel de la moyenne empirique.
  2. Le second critère envisagé est le critère dit `confidence bound`(CB), qui s'exprime comme suit (où $\lambda > 0$) :

$$CB(x) = \hat{\mu}(x) - \lambda\hat{\sigma}(x). \tag{12}$$

La minimisation de ce critère traduit l'arbitrage entre l'exploitation des résultats prédictifs du métamodèle (λ proche de 0) et l'exploration/ enrichissement du jeu de données afin de réduire la variance du métamodèle (λ proche de 1).

Implémentation de l'algorithme

Nous avons utilisé deux implémentations :

- l'implémentation `R` de `mlrMBO` [9] avec le critère `CB`. Le package `reticulate` [1]a été utilisé afin de permettre l'appel `R` à du code `Python` ;
- une implémentation manuelle en `Python` utilisant `scikit-learn` [15] pour les Random Forests et l'algorithme `SADE` de `Pygmo` [21] pour l'optimisation du critère de remplissage. Cet algorithme est une heuristique génétique d'optimisation.

[1] https://cran.r-project.org/web/packages/reticulate/

Résultats de l'algorithme - cas jouet n°2

Résultats obtenus pour la fonction jouet f_2 :

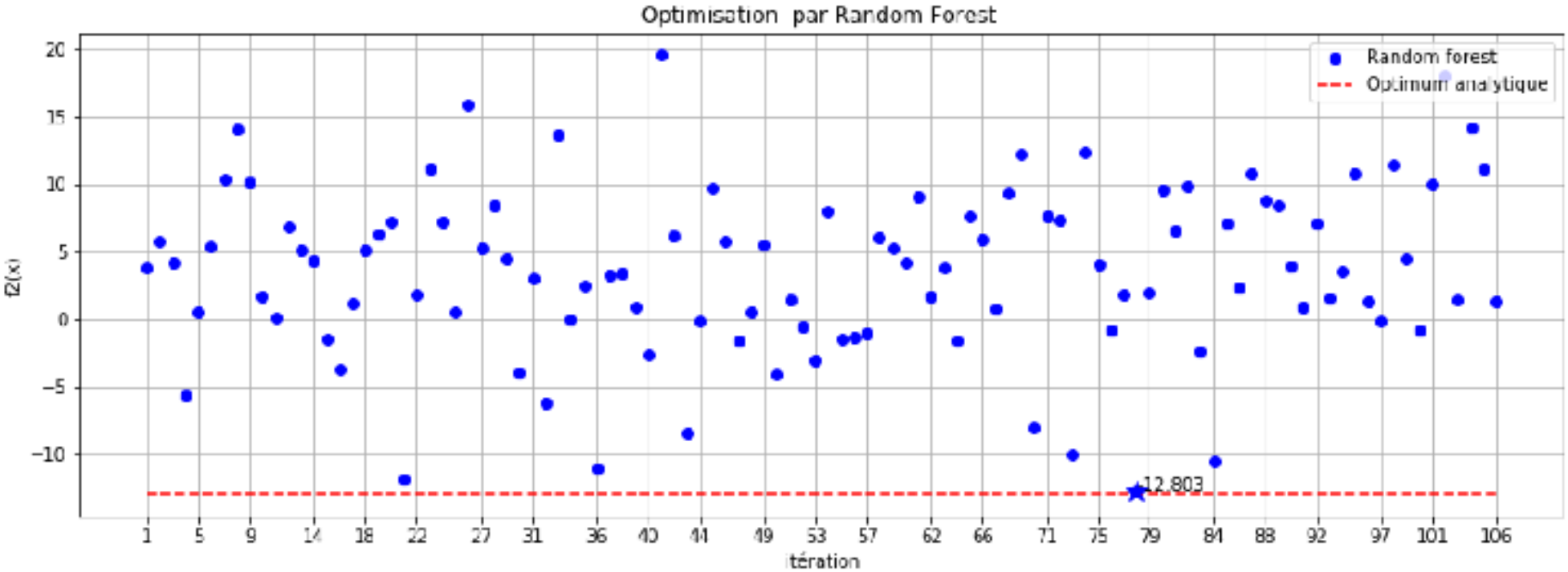
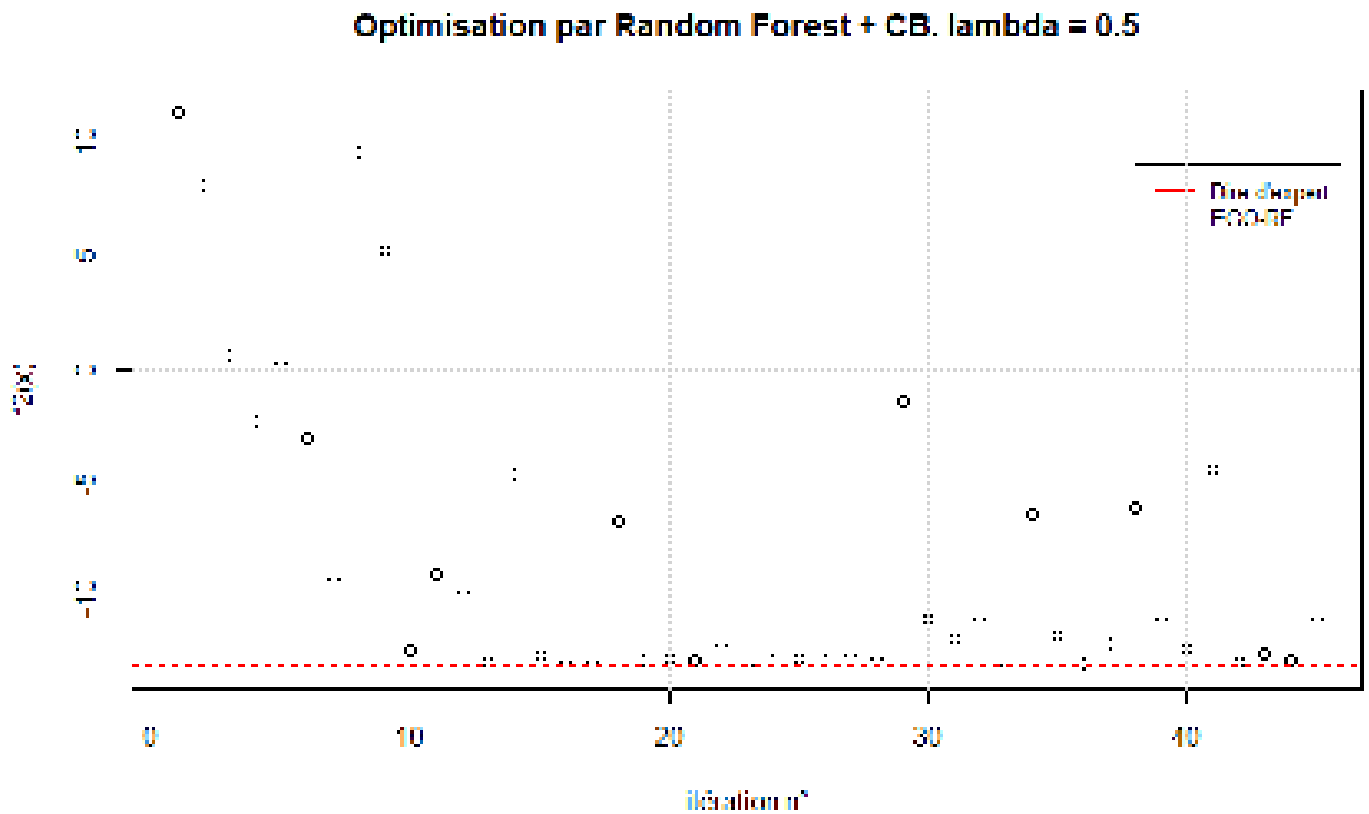


Figure: *Valeurs calculées pour chaque itération de l'algorithme EGO avec utilisation d'un RF. Meilleure valeur obtenue : -12.803. Implémentation manuelle, critère EI.*

Résultats pour le cas jouet 2, critère CB

Résultats obtenus pour la fonction jouet f_2 :



Conclusion sur EGO-RF

Cette méthode a permis d'améliorer sensiblement l'optimum en cours. Elle nécessite toutefois une étude ad hoc afin de tester et de caler convenablement ses nombreux hyperparamètres (caractérisation de la forêt aléatoire, paramètre λ, paramètres liés à la méthode d'optimisation non coûteuse, etc.).
Pistes d'amélioration possibles :

- Stratégie adaptative pour λ, en fonction du budget de calcul restant
- Amélioration de la méthode d'optimisation non coûteuse,
- Calibration du `RF` par validation croisée,
- Utiliser un critère EI pénalisé favorisant moins l'exploration.

Section 4

EGO avec régression quantile

Description de l'algorithme

`Gradient Tree Boosting` est réputé être l'algorithme «star » d'apprentissage machine.

- Contrairement à la régression `RF`, qui est construite autour d'une logique de moyennisation de la réponse de plusieurs arbres `CART`, cet algorithme-ci procède plutôt d'une logique additive.
- On ne peut donc plus envisager les réponses des différents arbres comme autant de tirages aléatoires de la loi sous-jacente à $F(x)$ et en dériver différentes statistiques d'intérêt, comme nous avons pu le faire précédemment.
- Nous allons plutôt utiliser le principe de la régression quantile, usuellement utilisée en apprentissage machine afin de calculer des approximations d'intervalles de prédiction. La régression quantile vise à estimer, non plus $\mathbb{E}(F(x)|x)$ [2], comme c'est le cas usuellement, mais plutôt la quantité :

$$q_\alpha(F(x)|x) = \inf\{y_\alpha \text{ tq. } \mathbb{P}(F(x) \leq y_\alpha|x) = \alpha\} \tag{13}$$

[2] avec les notations introduites précédemment. $F(x)$ est la réponse inconnue du vrai modèle en $x \in G$.

Description de l'algorithme

- La fonction de perte quantile, aussi appelée `pinball loss function`, est la fonction suivante :

$$\mathcal{L}_\alpha(F(x), \hat{F}(x)) := (F(x) - \hat{F}(x)) \times (\alpha - 1_{F(x) \leq \hat{F}(x)}) \tag{14}$$

- Un estimateur avec de «bonnes propriétés» (voir [14] pour une preuve) du quantile du niveau α est donné par :

$$\hat{q}_\alpha(F(x)|x) := \arg\min_\gamma \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_\alpha(F(x), \gamma) \tag{15}$$

- On peut donc assez simplement adapter l'algorithme `Gradient Tree Boosting` classique (i.e. minimisant le risque empirique quadratique) en modifiant la métrique et la fonction de coût sous-jacente.

Implémentation manuelle

L'algorithme envisagé ici dérive du schéma général également.

- On utilise l'implémentation `scikit-learn` de `Gradient Tree Boosting` [3], avec la fonction de perte `quantile`, et ce pour 100 différents niveaux uniformément répartis $\alpha \in [0.01, 0.99]$. On obtient à ce stade les prédicteurs $x \to \hat{F}_\alpha(x)$ pour $\alpha \in [0.01, 0.99]$
- on utilise ensuite le critère de remplissage `CB` introduit en (12), en prenant :
  - $\hat{\mu}(x) = \frac{1}{100} \sum_\alpha \hat{F}_\alpha(x)$
  - $\hat{\sigma}^2(x) = \frac{1}{100} \sum_\alpha \hat{F}_\alpha(x)^2 - \hat{\mu}^2(x)$
  - $\lambda = \frac{1}{2}$.

[3] https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_quantile.html#sphx-glr-auto-examples-ensemble-plot-gradient-boosting-quantile-py

Résultats de l'algorithme pour f1

Résultats obtenus pour la fonction jouet f_1 :

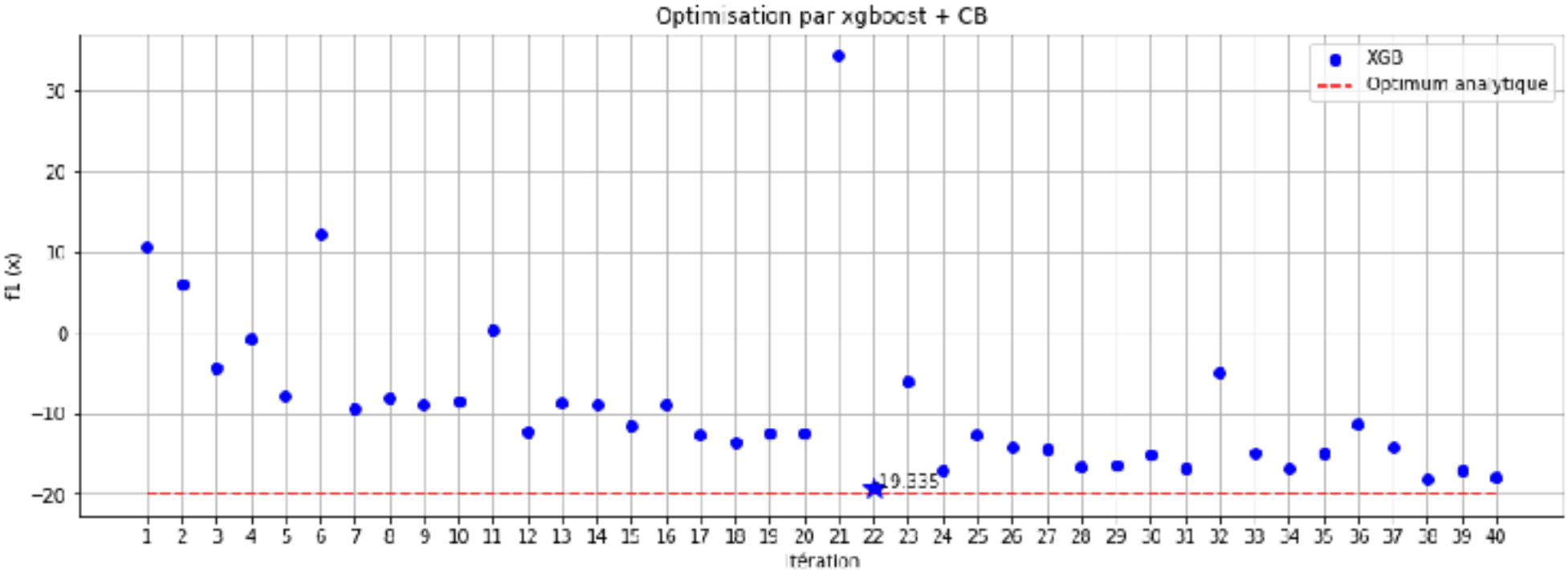


Figure: *Fonction jouet n°1. résultats pour* $\lambda = 0.5$ *. Plan initial en 6 points.*

Résultats de l'algorithme pour f2

Résultats obtenus pour la fonction jouet f_2 :

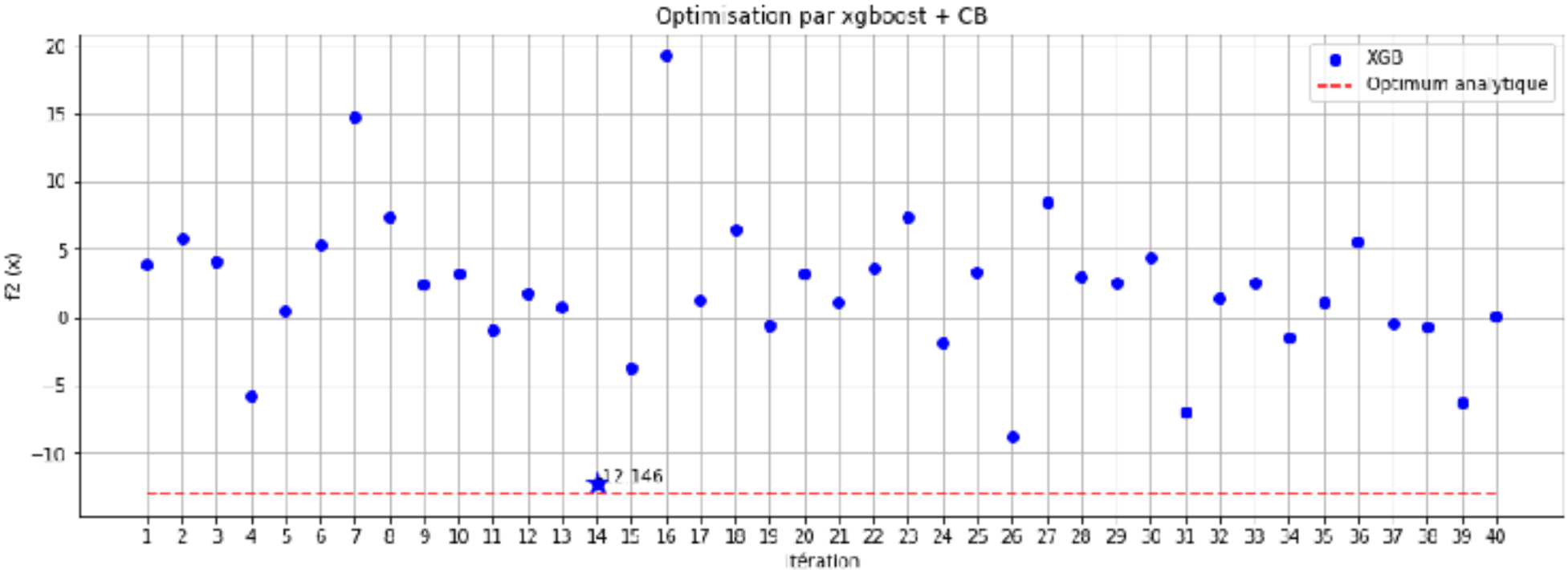


Figure: *Fonction jouet n°2. résultats pour $\lambda = 0.5$. Plan initial en 6 points.*

Implémentation `skopt`

La librairie `Python SKOPT` offre une implémentation de l'optimisation par métamodèle de Gradient Tree Boosting. La documentation afférente est relativement succincte, mais on note que `SKOPT` :

- Semble opérer une approximation gaussienne sur la distribution de $\hat{F}_{D_n}(x)$. Sous cette hypothèse, l'écart type de la variable aléatoire sous-jacente peut être estimé avec la formule classique suivante :

$$\hat{\sigma}(x) \approx \frac{q_{1-\alpha}(x) - q_{\alpha}(x)}{2} \quad \text{où} \quad \alpha = 16\% \quad \text{et où } q_{\alpha}(x) \text{ désigne le quantile de niveau } \alpha$$

  Cette approximation réduit grandement le problème d'estimation, puisqu'il s'agit d'estimer 3 modèles de régression (les deux régressions quantiles + la régression en moyenne) au lieu de 101 modèles comme précédemment.
- Utilise le critère de remplissage CB (d'autres critères sont disponibles dans l'API).
- Utilise l'algorithme `LBFGS`[31] pour l'optimisation non coûteuse du critère de remplissage. [4]

[4] Il n'est toutefois pas clair comment `SKOPT` tient compte de la contrainte d'intégrité des variables non continues, étant donné que c'est un algorithme d'optimisation continue.

Implémentation skopt

L'API de SKOPT est très simple à utiliser, comme le montre cet extrait de code optimisant la fonction $f = f_2$:

```
from skopt import  gbrt_minimize, space
SPACE = [space.Real(-5, 5, name='x1', prior='uniform'),
         space.Real(-5, 5, name='x2', prior='uniform'),
         space.Integer(1, 5, name='x3')]

res = gbrt_minimize(f, SPACE, acq_func='LCB',
                    n_calls=100, n_random_starts=10,
                    random_state=1, acq_optimizer ="lbfgs")

print('optimum : ' + str(res.fun)); print('obtenu en : ' + str(res.x))
```

Conclusion sur l'algorithme

`EGO-XGB` a permis de bien approximer l'optimum des fonctions analytiques et d'améliorer la solution initiale du code industriel. L'idée d'avoir recours à la régression quantile pour l'estimation de l'incertitude de prédiction, lorsqu'un estimateur de la variance du modèle de prédiction est difficile à construire, reste très générale et pourrait être appliquée à d'autres métamodèles (notamment des modèles de régression par lasso réputés bien adaptés aux grandes dimensions).
Par ailleurs, l'implémentation proposée ici reste très largement perfectible, notamment concernant :

- le choix de λ,
- la finesse de la grille des niveaux de quantiles, (l'apprentissage d'un modèle `Gradient Tree Boosting` n'est pas anodine en temps de calcul),
- l'implémentation de `Gradient Tree Boosting` : tester `xgboost`[28] et `LightGBM`[29], réputées plus performantes;

Section 5

EGO-RBF

Description de l'algorithme

L'algorithme `EGO-RBF` utilisé lors de cette étude dérive du schéma général décrit précédemment, en prenant comme :

- Métamodèle : un métamodèle de type `Radial Basis Function`, i.e. de la forme :

$$\hat{F}_{D_n}(x) = \sum_{i=1}^{n} \lambda_i \phi(||x - x_i||) + p(x) \tag{16}$$

où $\forall x \in G, \quad x \to \phi(x)$ est un noyau, $||.||$ est une norme sur G et $x \to p(x)$ un polynôme dans $\prod_{k-1}^{d}$, ensemble des polynômes de $d = dim(x)$ variables et de degré $\leq k-1$, respectant :

  - la condition d'interpolation : $\hat{F}_{D_n}(x_i) = F(x_i)$ pour tout $x_i \in D_n$
  - la condition d'orthogonalité : $\sum_{i=1}^{n} \lambda_i q(x_i) = 0$ pour tout $q \in \prod_{k-1}^{d}$.

Description de l'algorithme

- Critère de remplissage : le critère utilisé est le `SRBF` (`Stochastic RBF`). Cette stratégie consiste à générer des points candidats $\mathcal{P}_i$ en perturbant la meilleure solution en cours suivant une loi gaussienne de variance σ^2. Cette variance est recalculée à chaque itération en fonction du progrès réalisé. Le prochain point à évaluer sera le point dans $\mathcal{P}_i$ pour lequel la quantité suivante est minimale :

$$merit(x) = w\hat{F}(x) + (1 - w)(1 - dist(x)) \tag{17}$$

où $dist(x) = min\{||x - \xi||, \text{ pour } \xi \in D_n\}$ et où $w \in [0, 1]$. Le prochain point est donc celui qui - dans le voisinage du meilleur point en cours - minimise la réponse du métamodèle tout en étant le plus éloigné des points déjà explorés [13].

Implémentation de l'algorithme

Nous avons utilisé l'implémentation de `pySOT` [11]. D'autres implémentations `Python` existent, comme par exemple `RBFopt` [20]. En l'occurrence, nous avons utilisé un noyau cubique avec une norme euclidienne usuelle et une tendance linéaire.

Résultats pour la fonction jouet 1

Cet algorithme parvient sans grande difficulté à bien approximer l'optimum analytique pour la fonction simple f_1 :

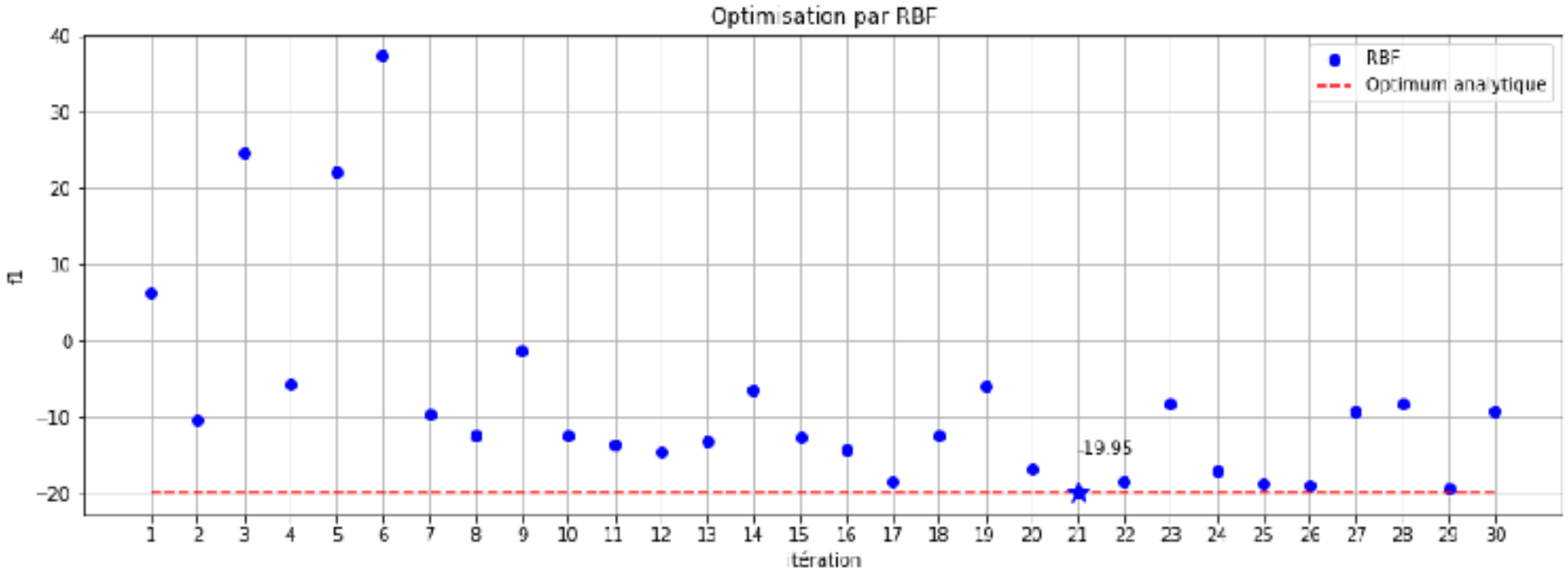


Figure: *Valeurs calculées pour chaque itération de l'algorithme EGO avec utilisation d'un RBF. Meilleure valeur obtenue à : -19,9. Implémentation pySOT. Plan initial en 6 points*

Résultats pour la fonction jouet 1

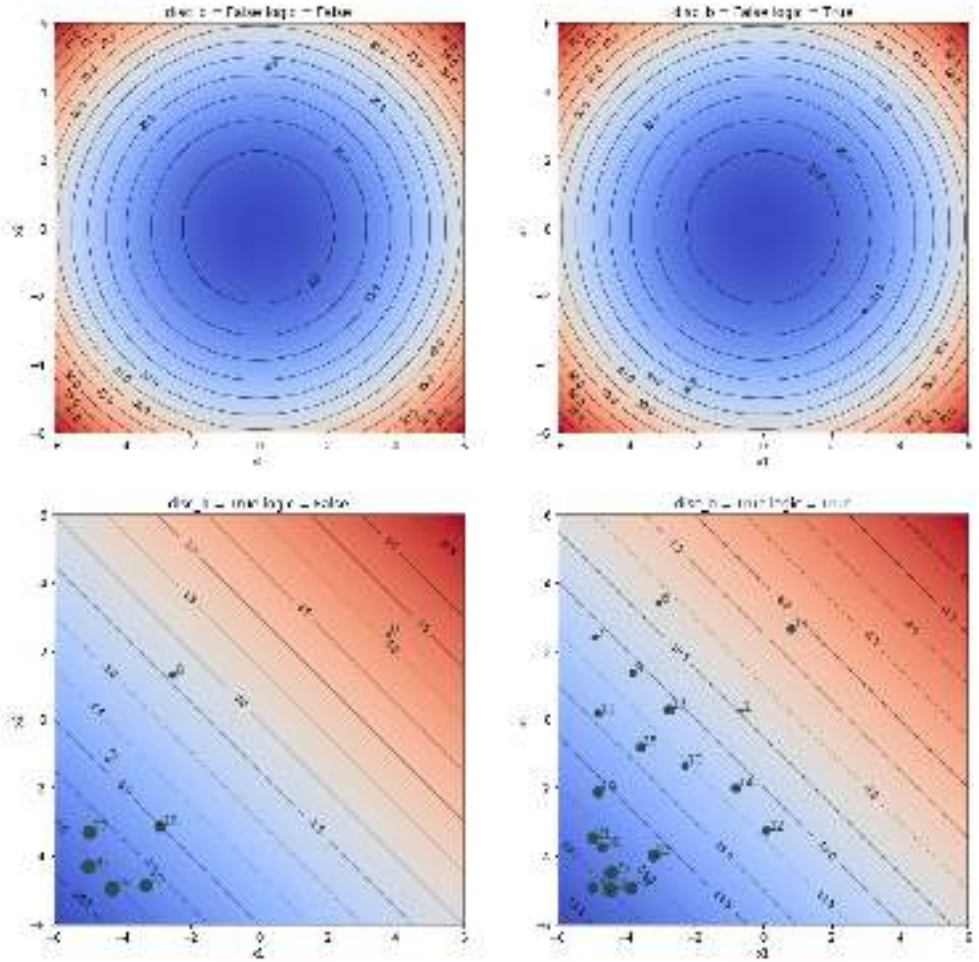


Figure: *Valeurs visitées à chaque itération de l'algorithme EGO avec utilisation d'un RBF. Plan initial en 6 points*

Résultats pour la fonction jouet 2

En revanche, la fonction jouet complexe f_2 est plus ardue à optimiser :

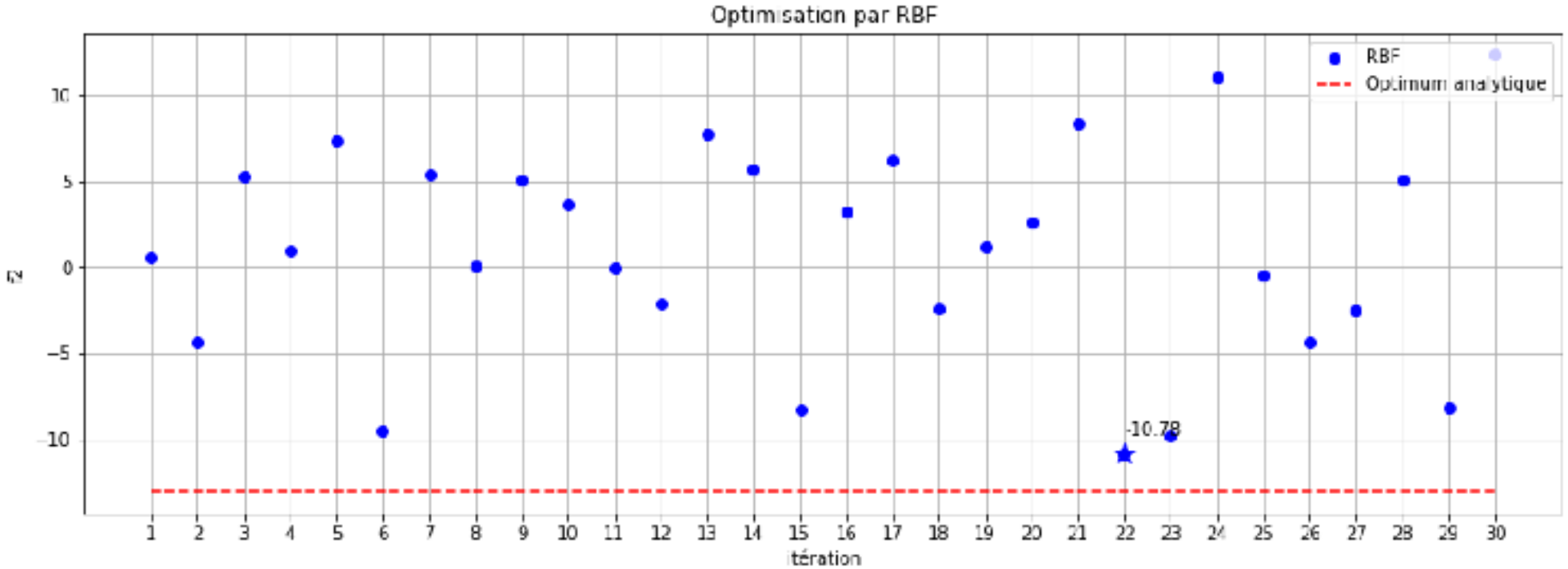


Figure: *Valeurs calculées pour chaque itération de l'algorithme EGO avec utilisation d'un RBF. Meilleure valeur obtenue : -11. Implémentation pySOT. Plan initial en 6 points*

Résultats pour la fonction jouet 2

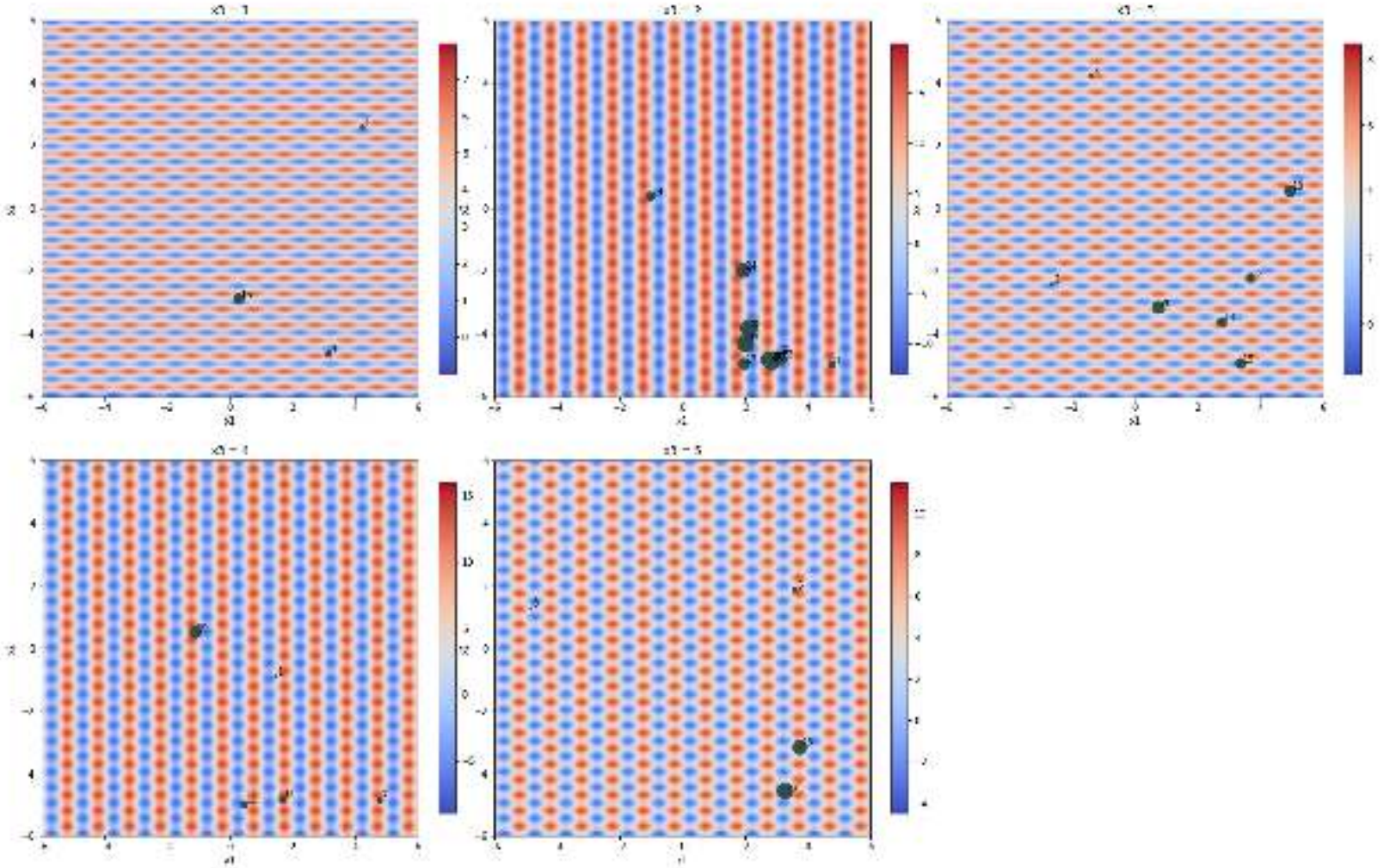


Figure: *Valeurs visitées à chaque itération de l'algorithme EGO avec utilisation d'un RBF.*

Conclusion sur EGO-RBF

Cet algorithme nous semble prometteur et laisse présager de nombreuses pistes d'améliorations :

- Tester avec la métrique de Gower, *a priori* mieux adaptée aux données mixtes (au lieu de la distance euclidienne utilisée par défaut)[32].
- Exploiter les possibilités de parallélisation nativement offertes par `pySOT`.

Section 6

Krigeage mixte

Krigeage mixte

Cette méthode n'a pas pu être testée de façon aboutie. Des premiers résultats ont toutefois pu être obtenus sur les fonctions jouet, et à ce titre nous avons jugé utile d'en citer les prémices.
Il s'agit de reprendre le schéma général décrit précédemment, en prenant comme :

- Métamodèle : Un processus gaussien à noyau mixte et à tendance linéaire. La librairie `R kerGP`[22] a été utilisée pour ce faire. L'extrait de code ci-dessous montre la paramétrisation retenue pour l'optimisation de la fonction f_2.

```
## Noyau de covariance
kContx1 <- covRadial(k1Fun1 = k1Fun1Matern5_2,d = 1, cov = "homo",
                                                    input = colnames(df)[1])
kContx2 <- covRadial(k1Fun1 = k1Fun1Matern5_2, d = 1, cov = "corr",
                                                    input = colnames(df)[2])
kCatx3 <- q1Symm(factor = as.factor(c(1,2,3,4,5)),
                                  input = colnames(df)[3], cov = "cor")

kMix <- covComp(formula = ~ kContx1() * kContx2() * kCatx3())

## Apprentissage du modele de krigeage
fit <-  gp(formula = y ~ x1 + x2, cov = kMix, data = df)
```

- Critère de remplissage : le critère `EI` (7), maximisé avec une méthode d'optimisation non coûteuse.

Krigeage mixte: Résultats obtenus

Pour la fonction jouet f_1, on obtient assez simplement une bonne approximation du minimum global. La méthode d'optimisation non coûteuse mise en oeuvre est la recherche aléatoire simple en 100 points :

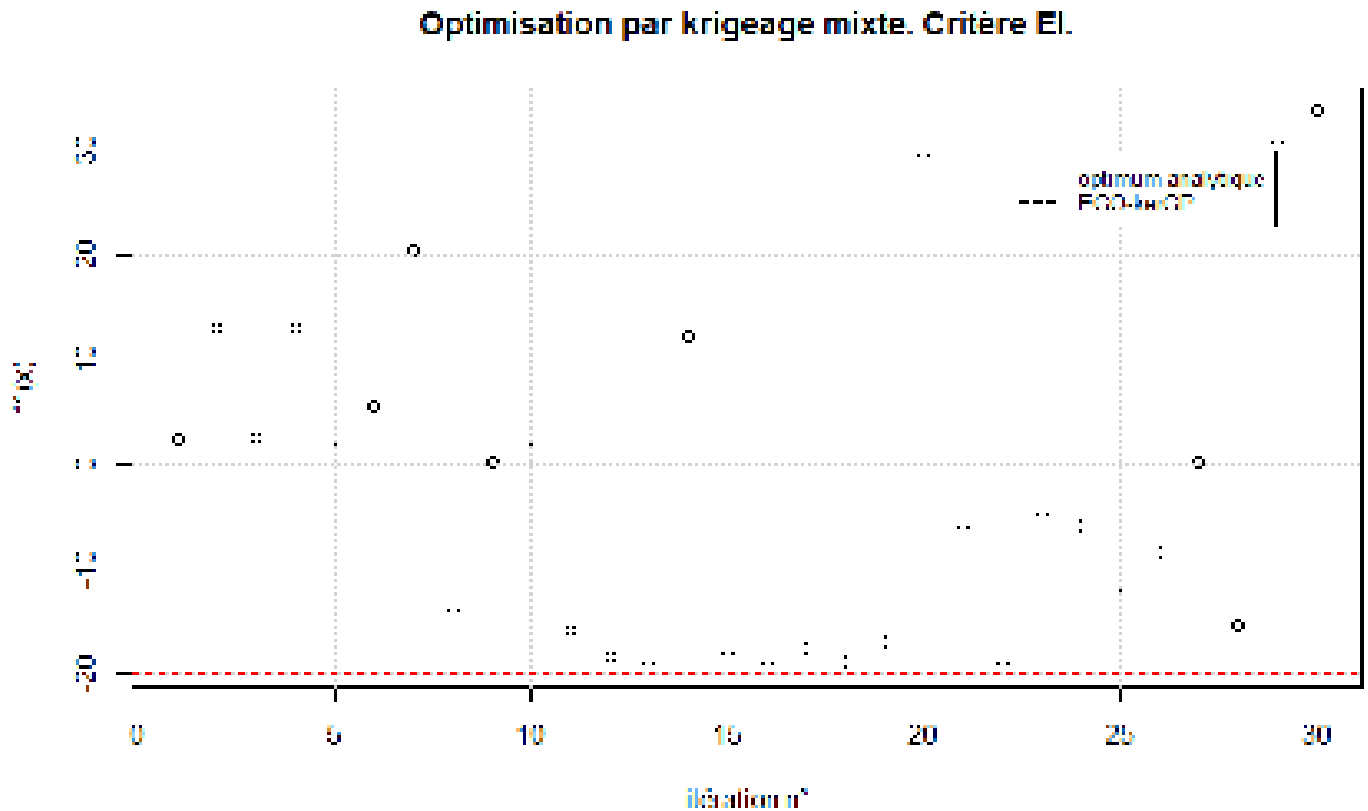


Figure: *Résultats pour f_1. Meilleur résultat -19.0, plan en 10 points*

Krigeage mixte: Résultats obtenus

Le recours à un algorithme d'optimisation stochastique adaptatif implémenté par la libraire `R CEoptim` [33] a permis d'améliorer sensiblement les performances de la méthode:

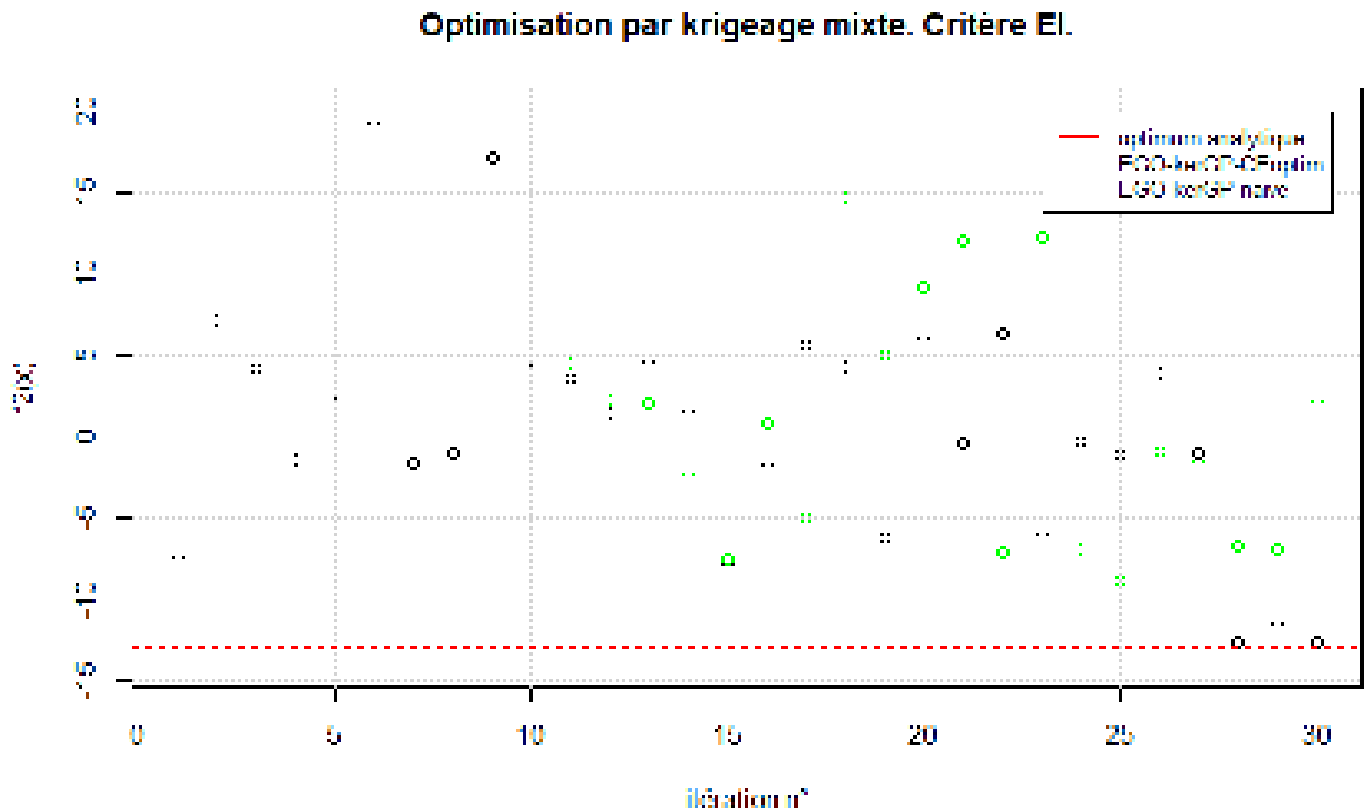


Figure: *Résultats pour f_2, optimisation non coûteuse avec CEoptim. Meilleur résultat -12.65, plan en 10 points*

Section 7

Perspectives

Perspectives

Plusieurs pistes d'amélioration se dégagent suite à cette étude. Citons notamment :

1. La réflexion autour du plan d'expérience initial (ici nous avons pris des plans où les points sont uniformément et indépendamment répartis).
2. Il conviendrait aussi de rationaliser et d'uniformiser les conditions de test de chaque méthode d'optimisation envisagée, en construisant un banc de test adéquat (en contrôlant le budget de calcul global, en contrôlant le pouvoir prédictif des métamodèles, en conservant le même plan d'expérience initial, etc.).

Perspectives

1. Les méthodes d'optimisation non coûteuses (appliquées aux critères de remplissage, notamment EI qui est une fonction complexe à optimiser numériquement) gagneraient à être améliorées. Les pistes sont :
   - Utilisation de méthodes prometteuses de recherche directe, notamment l'algorithme `MADS (Mesh Adapted Direct Search)` implémenté notamment dans `Nomad` [16]. `Nomad` devrait également être envisagée comme méthode de résolution frontale du problème initial (4).
   - en ajoutant une phase d'optimisation non coûteuse locale continue, par exemple avec l'algorithme `COBYLA` (disponible sous `OpenTurns`[10]).
2. Pour cette étude exploratoire nous nous sommes limités à des petites dimensions. La question du passage à l'échelle des méthodes envisagées ici reste entière. La piste *a priori* envisagée, outre celles développées ici, serait celle d'une régression quantile pénalisée avec un `LASSO` [24]. Notons que dans ce cas, il serait intéressant de tenter une optimisation analytique du critère de remplissage, *a priori* dérivable.

Matthieu Chiodetti, KTH MS. thesis. Bifacial PV plants: performance model development and optimization of their configuration (2015)

https://cran.r-project.org/web/packages/smoof/smoof.pdf

D.R. Jones. A taxonomy of global optimization methods based on response surfaces. (2001)

http://hyperopt.github.io/hyperopt/

https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html

https://www.automl.org/automated-algorithm-design/algorithm-configuration

http://fastml.com/tuning-hyperparams-automatically-with-spearmint/

B Bischl. mlrMBO : A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions (2017)

Michaël Baudin, Anne Dutfoy, Bertrand Iooss, Anne-Laure Popelin. OpenTURNS : An industrial software for uncertainty quantification in simulation (2015)

Tipaluck Krityakierne, Taimoor Akhtar and Christine A. Shoemaker. SOP : parallel surrogate global optimization with Pareto center selection for computationally expensive single objective problems. Journal of Global Optimization (2016)

Eriksson, Bindel, Shoemaker. pySOT and POAP : An event-driven asynchronous framework for surrogate optimization (2019)

Rommel G Regis and Christine A Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. (2007)

Pauline Givord - Xavier D'Haultfoeuille, La régression quantile en pratique (2014)

Scikit-learn : Machine Learning in Python, Pedregosa et al. (2011)

Nonlinear Optimization by Mesh Adaptive Direct Search. Le Digabel (2011)

Nikolaus Hansen. A CMA-ES for Mixed-Integer Nonlinear Optimization (2011)

Neveu et al. A Generic Interval Branch and Bound Algorithm for Parameter Estimation (2018)

Federico Zertuche. Assessment of uncertainty in computer experiments when working with multifidelity simulators. Université Grenoble Alpes, 2015.

Alberto Costa and Giacomo Nannicini. Rbfopt : an open-source library for black-box optimization with costly function evaluations. Optimization Online, 4538, (2014)

Biscani, Francesco; Izzo, Dario; Yam, Chit Hong. A global optimisation toolbox for massively parallel engineering optimisation (2010)

Y. Deville, D. Ginsbourger, O. Roustant, N. Durrande. Package 'kergp' : Gaussian Process Laboratory (2015)

C. Tuleau-Malot https://math.unice.fr/ malot/presCART.pdf

https://fr.wikipedia.org/wiki/Lasso_(statistiques)

Fabrice Rossi, Decision Trees course https://apiacoa.org/publications/teaching/

T. Hastie, R. Tibshirani and J. Friedman. Elements of Statistical Learning Ed. 2, Springer (2009)

https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting

https://xgboost.readthedocs.io/en/latest/

https://lightgbm.readthedocs.io/en/latest/

https://scikit-optimize.github.io/

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html

M. Halstrup, PhD Thesis : Black-box optimization of mixed discrete-continuous optimization problems (2016)

T. Benham and al. CEoptim: Cross-Entropy R Package for Optimization. Journal of Statistical Software (2015)