

Optimisation sans (?) dérivées

Clément Royer

Certificat Chef de Projet IA - Université Paris Dauphine-PSL

18 novembre 2021



$$\text{minimiser}_{\mathbf{x} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \max\{1 - b_i \mathbf{a}_i^T \mathbf{x}, 0\} + \frac{\lambda}{2} \|\mathbf{x}\|^2.$$

- $\mathbf{a}_i \in \mathbb{R}^d$, $b_i \in \{-1, 1\}$;
- Reformulation en programme quadratique possible.

$$\text{minimiser}_{\mathbf{x} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \max\{1 - b_i \mathbf{a}_i^T \mathbf{x}, 0\} + \frac{\lambda}{2} \|\mathbf{x}\|^2.$$

- $\mathbf{a}_i \in \mathbb{R}^d$, $b_i \in \{-1, 1\}$;
- Reformulation en programme quadratique possible.

Appliquer le gradient stochastique

- Intéressant dans le cas de données massives ($n \gg 1$);
- **Problème** : la fonction n'est pas partout dérivable !

Un réseau de neurones pour la régression

$$\text{minimiser}_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) := \frac{1}{2n} \|\mathbf{W}_3 (\mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{a}_i + \mathbf{v}_1) + \mathbf{v}_2) - \mathbf{b}_i\|^2.$$

- $\mathbf{a}_i \in \mathbb{R}^{d_a}$, $\mathbf{b}_i \in \mathbb{R}^{d_b}$.
- σ : Activation non linéaire appliquée à chaque composante d'un vecteur.
- \mathbf{x} concaténation des paramètres $\mathbf{W}_3, \mathbf{W}_2, \mathbf{v}_2, \mathbf{W}_1, \mathbf{v}_1$.

Un réseau de neurones pour la régression

$$\text{minimiser}_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) := \frac{1}{2n} \|\mathbf{W}_3 (\mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{a}_i + \mathbf{v}_1) + \mathbf{v}_2) - \mathbf{b}_i\|^2.$$

- $\mathbf{a}_i \in \mathbb{R}^{d_a}$, $\mathbf{b}_i \in \mathbb{R}^{d_b}$.
- σ : Activation non linéaire appliquée à chaque composante d'un vecteur.
- \mathbf{x} concaténation des paramètres $\mathbf{W}_3, \mathbf{W}_2, \mathbf{v}_2, \mathbf{W}_1, \mathbf{v}_1$.

Entraînement via le gradient stochastique

- Si σ est \mathcal{C}^1 (ex: tanh), tout est \mathcal{C}^1 : formule pour $\nabla f(\mathbf{x})$?
- Si σ n'est pas \mathcal{C}^1 (ex: ReLU), que faire ?

$$\text{minimiser}_{\mathbf{x} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \max\{1 - b_i \mathbf{a}_i^T \mathbf{x}, 0\} + \frac{\lambda}{2} \|\mathbf{x}\|^2.$$

- $\mathbf{a}_i \in \mathbb{R}^d$, $b_i \in \{-1, 1\}$.
- Régularisation en $\frac{\lambda}{2} \|\mathbf{x}\|^2$.

$$\text{minimiser}_{\mathbf{x} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \max\{1 - b_i \mathbf{a}_i^T \mathbf{x}, 0\} + \frac{\lambda}{2} \|\mathbf{x}\|^2.$$

- $\mathbf{a}_i \in \mathbb{R}^d$, $b_i \in \{-1, 1\}$.
- Régularisation en $\frac{\lambda}{2} \|\mathbf{x}\|^2$.

Régularisation

- Compromis entre un terme d'attache aux données et une structure souhaitée.
- Questions :
 - Quelle est la meilleure valeur de λ ?
 - Peut-on optimiser sur λ ?

Calcul des dérivées

- Différentiation automatique;
- Logiciels.

Pas de notion de dérivée

- Que faire dans ce contexte ?
- Quels algorithmes ?

Optimisation sans dérivées

- Quels problèmes ?
- Quelles méthodes ?

- 1 Différentiation
- 2 L'optimisation sans dérivées

1 Différentiation

- Calcul de gradient
- Analyse non lisse
- Application à la régularisation

2 L'optimisation sans dérivées

1 Différentiation

- Calcul de gradient
- Analyse non lisse
- Application à la régularisation

2 L'optimisation sans dérivées

Dans la suite, on considère une fonction $f : \mathbb{R}^d \rightarrow \mathbb{R}$ de classe \mathcal{C}^1 .

Gradient

Soit une fonction $f : \mathbb{R}^d \rightarrow \mathbb{R}$ de classe \mathcal{C}^1 . Le **gradient de f en \mathbf{x}** est donné par

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_i}(\mathbf{x}) \right]_{1 \leq i \leq d} \in \mathbb{R}^d.$$

On considère une fonction **lisse** (ou douce, ou *smooth*) $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$.

Matrice jacobienne

$f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ est dérivable en $\mathbf{x} \in \mathbb{R}^d$ si il existe une matrice $\mathbf{J}_f(\mathbf{x}) \in \mathbb{R}^{m \times d}$ telle que

$$\lim_{\substack{\mathbf{z} \rightarrow \mathbf{x} \\ \mathbf{z} \neq \mathbf{x}}} \frac{\|f(\mathbf{z}) - f(\mathbf{x}) - \mathbf{J}_f(\mathbf{x})(\mathbf{z} - \mathbf{x})\|}{\|\mathbf{z} - \mathbf{x}\|} = 0.$$

$\mathbf{J}_f(\mathbf{x})$ s'appelle la (matrice) **jacobienne** de f en \mathbf{x} .

On considère une fonction **lisse** (ou douce, ou *smooth*) $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$.

Matrice jacobienne

$f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ est dérivable en $\mathbf{x} \in \mathbb{R}^d$ si il existe une matrice $\mathbf{J}_f(\mathbf{x}) \in \mathbb{R}^{m \times d}$ telle que

$$\lim_{\substack{\mathbf{z} \rightarrow \mathbf{x} \\ \mathbf{z} \neq \mathbf{x}}} \frac{\|f(\mathbf{z}) - f(\mathbf{x}) - \mathbf{J}_f(\mathbf{x})(\mathbf{z} - \mathbf{x})\|}{\|\mathbf{z} - \mathbf{x}\|} = 0.$$

$\mathbf{J}_f(\mathbf{x})$ s'appelle la (matrice) **jacobienne** de f en \mathbf{x} .

Cas particuliers

- $m = 1$: la jacobienne se ramène à un vecteur $\nabla f(\mathbf{x}) = \mathbf{J}_f(\mathbf{x})^T$, que l'on appelle le **vecteur gradient**;
- $n = m = 1$: la jacobienne est équivalente à un scalaire $f'(x) = \nabla f(\mathbf{x}) = \mathbf{J}_f(x)$, que l'on appelle la dérivée de f en \mathbf{x} .

Théorème : Dérivée d'une fonction composée

Supposons que $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ s'écrive sous la forme

$$f(\mathbf{x}) = \phi_2 \circ \phi_1(\mathbf{x}) = \phi_2(\phi_1(\mathbf{x})).$$

avec $\phi_1 : \mathbb{R}^d \rightarrow \mathbb{R}^{m_1}$, $\phi_2 : \mathbb{R}^{m_2} \rightarrow \mathbb{R}^m$. Alors, si ϕ_1 et ϕ_2 sont \mathcal{C}^1 , f est aussi \mathcal{C}^1 avec

$$\mathbf{J}_f(\mathbf{x}) = \mathbf{J}_{\phi_2}(\phi_1(\mathbf{x}))\mathbf{J}_{\phi_1}(\mathbf{x}).$$

Théorème : Dérivée d'une fonction composée

Supposons que $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ s'écrive sous la forme

$$f(\mathbf{x}) = \phi_2 \circ \phi_1(\mathbf{x}) = \phi_2(\phi_1(\mathbf{x})).$$

avec $\phi_1 : \mathbb{R}^d \rightarrow \mathbb{R}^{m_1}$, $\phi_2 : \mathbb{R}^{m_2} \rightarrow \mathbb{R}^m$. Alors, si ϕ_1 et ϕ_2 sont \mathcal{C}^1 , f est aussi \mathcal{C}^1 avec

$$\mathbf{J}_f(\mathbf{x}) = \mathbf{J}_{\phi_2}(\phi_1(\mathbf{x}))\mathbf{J}_{\phi_1}(\mathbf{x}).$$

Extensions

- Compositions multiples : $f(\mathbf{x}) = \phi \circ \phi_L \circ \phi_{L-1} \circ \cdots \circ \phi_1(\mathbf{x})$;
- Fonctions progressives :

$$f(\mathbf{x}) = \phi(\phi_L(\mathbf{x}_L, \phi_{L-1}(\mathbf{x}_{L-1}, \dots, \phi_2(\mathbf{x}_2, \phi_1(\mathbf{x}_1)) \dots))).$$

Calcul à la main

- Risque d'erreurs;
- Doit être refait dès que le modèle change.

Différentiation symbolique

- Renvoie une expression mathématique (cf Wolfram Alpha);
- Doit être recalculée si le modèle change.

Différentiation numérique

- Exemple : Différences finies.
- Requiert plusieurs évaluations de fonction.

Principe

- Pour une itération de descente de gradient/de gradient stochastique, besoin de la valeur du gradient **en le point courant \mathbf{x}_k** .
- La **valeur** $\nabla f(\mathbf{x}_k)$ peut se calculer numériquement.

Principe

- Graphe de calcul d'une fonction;
- Différentiation automatique via le graphe de calcul et la règle de composition.

1 Différentiation

- Calcul de gradient
- Analyse non lisse
- Application à la régularisation

2 L'optimisation sans dérivées

Définition

Une fonction est dite **non lisse** si elle n'est pas dérivable partout.

NB: Non lisse \neq Discontinue.

Exemples de fonctions non lisses

- $w \mapsto |w|$ de \mathbb{R} dans \mathbb{R} ;
- $w \mapsto \|w\|_1 = \sum_{i=1}^d |w_i|$ de \mathbb{R}^d dans \mathbb{R} ;
- ReLU: $w \mapsto \max\{w, 0\}$ de \mathbb{R}^d dans \mathbb{R} .

- Un algorithme d'optimisation lisse (ex : descente de gradient) est basé sur des dérivées;
- Non lisse \Leftrightarrow Pas de dérivée en certains points;
- On utilise des notions plus générales de dérivée.

Alternatives

- Si possible, reformuler en un problème lisse :

Ex) minimiser $w \in \mathbb{R} \mid w$ se ré-écrit

$$\text{minimiser}_{w, t^+, t^- \in \mathbb{R}} t^+ - t^- \quad \text{subject to} \quad w = t^+ - t^-, t^+ \geq 0, t^- \geq 0.$$

- Si la *fonction* est lipschitzienne, elle possède un gradient en presque tous les points (**mais** souvent pas en les minima).

Ex) Fonction d'activation des réseaux de neurones

$$\text{ReLU}(\mathbf{w}) = [\max\{w_i, 0\}].$$

Définition

Soit $f : \mathbb{R}^d \rightarrow \mathbb{R}$ une fonction convexe. Un vecteur $\mathbf{g} \in \mathbb{R}^d$ est un **sous-gradient** de f en $\mathbf{x} \in \mathbb{R}^d$ si

$$\forall \mathbf{z} \in \mathbb{R}^d, \quad f(\mathbf{z}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{z} - \mathbf{x}).$$

L'ensemble des sous-gradients de f en \mathbf{x} s'appelle le *sous-différentiel* de f en \mathbf{x} : on le note $\partial f(\mathbf{x})$.

- Si f dérivable en \mathbf{x} , $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$;
- $0 \in \partial f(\mathbf{x}) \Leftrightarrow \mathbf{x}$ minimum de f !

Exemple : Soit $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(x) = |x|$.

$$\partial f(x) = \begin{cases} -1 & \text{si } x < 0 \\ 1 & \text{si } x > 0 \\ [-1, 1] & \text{si } x = 0. \end{cases}$$

Itération pour minimiser $\mathbf{w} f(\mathbf{w})$, f convexe nonsmooth

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbf{g}_k, \quad \mathbf{g}_k \in \partial f(\mathbf{w}_k).$$

- Dépend du choix du sous-gradient;
- Choix de α_k plus technique (f peut croître dans la direction d'un sous-gradient !).

À retenir

- Certaines méthodes dites “de gradient” se basent en fait sur des sous-gradients;
- Ceux-ci sont bien compris pour des problèmes simples, et dans le cas convexe.

1 Différentiation

- Calcul de gradient
- Analyse non lisse
- Application à la régularisation

2 L'optimisation sans dérivées

$$\text{minimiser}_{\mathbf{x} \in \mathbb{R}^d} \underbrace{f(\mathbf{x})}_{\text{perte}} + \underbrace{\lambda \Omega(\mathbf{x})}_{\text{régularisation}} .$$

où $\lambda > 0$ est un paramètre de régularisation.

Exemple : Régularisation *ridge* ou écrêtée

$$\text{minimize}_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) + \frac{\lambda}{2} \|\mathbf{x}\|^2.$$

Interprétations :

- Revient à ajouter une contrainte sur $\|\mathbf{x}\|^2 = \sum_{i=1}^d x_i^2$;
- Favorise les vecteurs \mathbf{x} avec des composantes uniformément faibles en amplitude;
- Réduit la variance des solutions par rapport aux données;
- Problème fortement convexe pour λ suffisamment grand.

Problème régularisé et exemples (2)

Régularisation ℓ_0

- Objectif : Trouver $\mathbf{x} \in \mathbb{R}^d$ qui colle aux données avec le plus de coefficients nuls possible;
- Problème idéal : $\min_{\mathbf{x}} f(\mathbf{x}) + \lambda \|\mathbf{x}\|_0$, avec $\|\mathbf{v}\|_0 := |\{i | [\mathbf{v}]_i \neq 0\}|$.
Mais la fonction $\|\cdots\|_0$ est non lisse, discontinue et introduit de la combinatoire.

Problème régularisé et exemples (2)

Régularisation ℓ_0

- Objectif : Trouver $\mathbf{x} \in \mathbb{R}^d$ qui colle aux données avec le plus de coefficients nuls possible;
- Problème idéal : $\min_{\mathbf{x}} f(\mathbf{x}) + \lambda \|\mathbf{x}\|_0$, avec $\|\mathbf{v}\|_0 := |\{i | [\mathbf{v}]_i \neq 0\}|$.
Mais la fonction $\|\cdots\|_0$ est non lisse, discontinue et introduit de la combinatoire.

Régularisation LASSO

LASSO=Least Absolute Shrinkage and Selection Operator

$$\min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) + \lambda \|\mathbf{x}\|_1, \quad \|\mathbf{x}\|_1 = \sum_{i=1}^d |x_i|.$$

- $\|\cdot\|_1$ convexe, continue, norme;
- Non lisse mais possède des sous-gradients.

Cadre : Optimisation composite

$$\underset{\mathbf{x} \in \mathbb{R}^d}{\text{minimize}} f(\mathbf{x}) + \lambda \Omega(\mathbf{x}).$$

- $f \in \mathcal{C}^{1,1}$;
- Ω convexe.

Approche proximale

- Classique en optimisation : remplacer un problème par une suite de sous-problèmes plus simples;
- Ici on exploite la “douceur” de f et la structure de Ω pour construire des problèmes que l’on peut résoudre **efficacement**.

Itération

$$\mathbf{x}_{k+1} = \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T (\mathbf{x} - \mathbf{x}_k) + \frac{1}{2\alpha_k} \|\mathbf{x} - \mathbf{x}_k\|_2^2 + \lambda \Omega(\mathbf{x}) \right\}.$$

- Si $\Omega \equiv 0$, la solution est $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$: **c'est l'itération de la descente de gradient !**
- En général, une itération coûte un calcul de gradient + **1 résolution de sous-problème.**

Propriétés

- Complexité/Vitesses de convergence;
- Règles de choix de longueur de pas;
- Variantes accélérées/avec sous-gradients.

Contexte

- Résoudre $\text{minimize}_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) + \lambda \|\mathbf{x}\|_1$;
- Problème classique en traitement du signal/de l'image;
- La méthode du gradient proximal a une **forme explicite**.

Exemple de méthode proximale : ISTA

Contexte

- Résoudre $\text{minimize}_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) + \lambda \|\mathbf{x}\|_1$;
- Problème classique en traitement du signal/de l'image;
- La méthode du gradient proximal a une **forme explicite**.

Itération ISTA : Iterative Soft-Thresholding Algorithm

Définit \mathbf{x}_{k+1} par coordonnées: pour tout $i \in \{1, \dots, d\}$,

$$[\mathbf{x}_{k+1}]_i = \begin{cases} [\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)]_i + \alpha_k \lambda & \text{si } [\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)]_i < -\alpha_k \lambda \\ [\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)]_i - \alpha_k \lambda & \text{si } [\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)]_i > \alpha_k \lambda \\ 0 & \text{si } [\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)]_i \in [-\alpha_k \lambda, \alpha_k \lambda]. \end{cases}$$

Mise à jour dans ISTA

- Part du pas de gradient $\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$;
- Applique l'opérateur de *soft-thresholding* $s_{\alpha_k \lambda}(\bullet)$ à chaque coordonnée, avec

$$s_{\mu}(t) = \begin{cases} t + \mu & \text{si } t < -\mu \\ t - \mu & \text{si } t > \mu \\ 0 & \text{sinon.} \end{cases}$$

- Favorise les composantes nulles.

Variantes de ISTA

- Changement de longueur de pas;
- Ajout de momentum : FISTA (la plus utilisée en pratique).

Problèmes lisses (avec dérivée)

- Différentiation automatique pour problèmes complexes;
- Outil-clé en apprentissage (backpropagation);
- On calcule juste une valeur !

Problèmes non lisses

- Notion de dérivée généralisée;
- Théorie et algorithmes associés;
- Permet notamment l'optimisation composite.

- 1 Différentiation
- 2 L'optimisation sans dérivées

Quand on parle d'optimisation sans dérivées, on parle de...

- 1 Derivative-free optimization (DFO);

Quand on parle d'optimisation sans dérivées, on parle de...

- ① Derivative-free optimization (DFO);
- ② Black-box optimization;

Quand on parle d'optimisation sans dérivées, on parle de...

- ① Derivative-free optimization (DFO);
- ② Black-box optimization;
- ③ Surrogate-based optimization;

Quand on parle d'optimisation sans dérivées, on parle de...

- ① Derivative-free optimization (DFO);
- ② Black-box optimization;
- ③ Surrogate-based optimization;
- ④ Response surface methodology/Design of experiments;

Quand on parle d'optimisation sans dérivées, on parle de...

- ① Derivative-free optimization (DFO);
- ② Black-box optimization;
- ③ Surrogate-based optimization;
- ④ Response surface methodology/Design of experiments;
- ⑤ Automated machine learning;

Quand on parle d'optimisation sans dérivées, on parle de...

- ① Derivative-free optimization (DFO);
- ② Black-box optimization;
- ③ Surrogate-based optimization;
- ④ Response surface methodology/Design of experiments;
- ⑤ Automated machine learning;
- ⑥ Hyperparameter tuning.

Quand on parle d'optimisation sans dérivées, on parle de...

- 1 **Derivative-free optimization (DFO);**
- 2 **Black-box optimization;**
- 3 Surrogate-based optimization;
- 4 Response surface methodology/Design of experiments;
- 5 Automated machine learning;
- 6 Hyperparameter tuning.

Nous allons parler de...

- **Tout cela** dans un même cadre;
- **Des avancées pour 1+2;**
- Du lien avec 5+6.

1 Différentiation

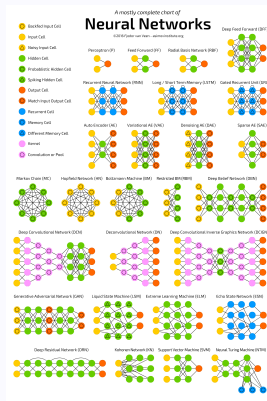
- Calcul de gradient
- Analyse non lisse
- Application à la régularisation

2 L'optimisation sans dérivées

- Exemples et définition
- Méthodes de recherche directe
- Méthodes basées sur des modèles

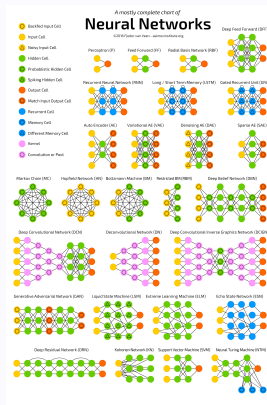
Entraînons un réseau de neurones...

- Quelle architecture ? (nombre de couches, types de couches, etc)
- Quel algorithme d'entraînement ? (Adam, RMSProp, SGD, etc)
- Quelles options pour l'algorithme (*learning rate*, etc)?



Entraînons un réseau de neurones...

- Quelle architecture ? (nombre de couches, types de couches, etc)
- Quel algorithme d'entraînement ? (Adam, RMSProp, SGD, etc)
- Quelles options pour l'algorithme (*learning rate*, etc)?



Calibration d'hyperparamètres

- Chaque test d'une configuration correspond à un nouvel entraînement (heures/jours en temps CPU + argent !);
- Énormément de choix possibles.

Calcul scientifique

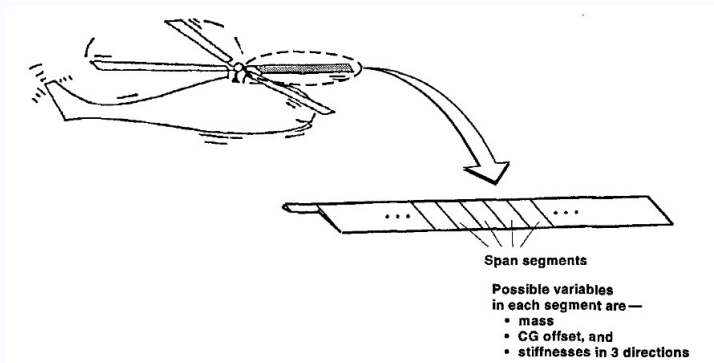
- Usage intensif de la simulation par ordinateur (CFD, CAO) dans les applications de type physique (aéronautique, automobile, météorologie);
- Beaucoup de paramètres à calibrer.



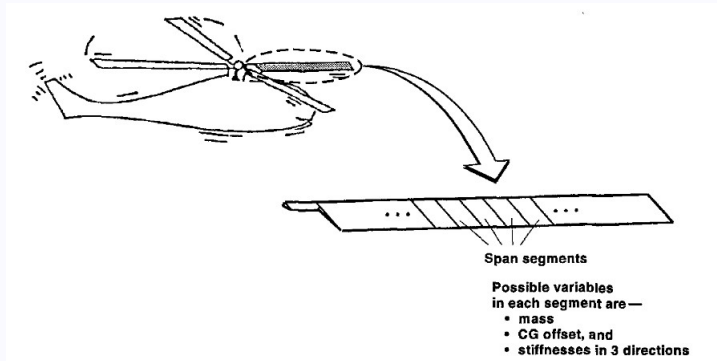
Optimisation de simulateurs

- Besoin : optimiser les paramètres de codes de simulations;
- Coût d'exécution des simulateurs élevé;
- Parfois plusieurs versions à coût variable (multifidélité).

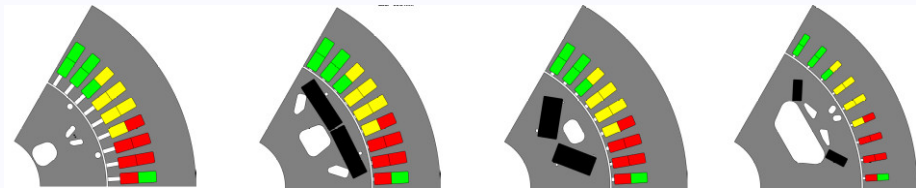
Exemple classique : Design de pale d'hélicoptère (Booker et al. 1998)



Exemple classique : Design de pale d'hélicoptère (Booker et al. 1998)



- 30 paramètres;
- 1 simulation : 2 semaines de calculs CFD;
- Échec de la simulation 60% du temps.
- Optimisation multi-disciplinaire : codes imbriqués;
- De la simulation numérique, beaucoup de calculs...
- qui peuvent échouer !



- Environ 50 paramètres (continus);
- Multiobjectif (3 objectifs), 6 fonctions de contraintes;
- La plupart des points ne sont pas réalisables !
- 1 simulation \approx 5 minutes;
- Optimisation (par algorithmes génétiques) : 3 semaines !

Une classe de problèmes en commun

Points communs : IA automatisée et optimisation de simulateurs

- Effort de calcul conséquent, basé sur la simulation;
- Choix des meilleurs paramètres non trivial;
- Préliminaire à la construction/au déploiement du système.



En termes d'optimisation

- **Le choix des meilleurs paramètres peut se formuler comme un problème d'optimisation;**
- La fonction objectif de ce problème est très coûteuse à évaluer (en temps de simulation).

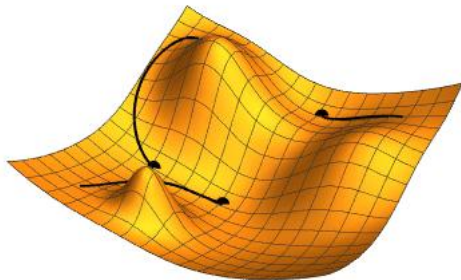
Optimisation sans dérivées

Certaines dérivées ne peuvent pas être exploitées pour optimiser.

Optimisation sans dérivées

Certaines dérivées ne peuvent pas être exploitées pour optimiser.

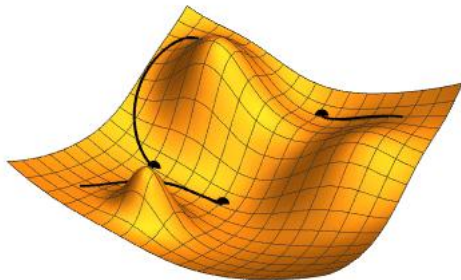
- L'optimisation est fortement basée sur les dérivées (gradient, conditions d'optimalité);
- **Certaines** : il en suffit d'une !



Optimisation sans dérivées

Certaines dérivées ne peuvent pas être exploitées pour optimiser.

- L'optimisation est fortement basée sur les dérivées (gradient, conditions d'optimalité);
- **Certaines** : il en suffit d'une !
- **ne peuvent pas** : qu'elles existent ou non !



Toujours préférable d'utiliser les dérivées si possible !

Toujours préférable d'utiliser les dérivées si possible !

- 1 Expression analytique \Rightarrow Dérivées explicites;

Toujours préférable d'utiliser les dérivées si possible !

- ❶ Expression analytique \Rightarrow Dérivées explicites;
- ❷ Approximation numérique par différences finies;

Toujours préférable d'utiliser les dérivées si possible !

- ❶ Expression analytique \Rightarrow Dérivées explicites;
- ❷ Approximation numérique par différences finies;
- ❸ Puissance de la différentiation automatique/symbolique.

Toujours préférable d'utiliser les dérivées si possible !

- ❶ Expression analytique \Rightarrow Dérivées explicites;
- ❷ Approximation numérique par différences finies;
- ❸ Puissance de la différentiation automatique/symbolique.

Dérivées non disponibles

- ❶ Systèmes complexes \Rightarrow Risque d'erreur dans le code à la main;
- ❷ Évaluations coûteuses/bruitées \Rightarrow Problème pour les différences finies;
- ❸ Code propriétaire \Rightarrow Pas de diff. auto.

Algorithmes génétiques/évolutionnaires

- Souvent inspirés par la nature;
- Efficaces avec peu de variables et des évaluations peu coûteuses/parallélisables;
- Beaucoup d'heuristiques inspirées par la nature.

Algorithmes génétiques/évolutionnaires

- Souvent inspirés par la nature;
- Efficaces avec peu de variables et des évaluations peu coûteuses/parallélisables;
- Beaucoup d'heuristiques inspirées par la nature.

Une méthode remarquable : CMA-ES

- Maintient une matrice de covariance;
- Efficace et populaire;
- Récemment interprétée comme une méthode de gradient appliquée à de l'optimisation sur des distributions (chercheurs en IA).

Notre formulation

$$\text{minimiser}_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) \quad \text{s. c.} \quad \mathbf{x} \in \mathcal{F}$$

- \mathbf{x} : variables;
- f : fonction objectif;
- \mathcal{F} : ensemble admissible.

Notre formulation

$$\text{minimiser}_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}) \quad \text{s. c.} \quad \mathbf{x} \in \mathcal{F}$$

- \mathbf{x} : variables;
- f : fonction objectif;
- \mathcal{F} : ensemble admissible.

Cadre de travail

- f minorée sur \mathcal{F} : $f(\mathbf{x}) \geq f_{\text{low}}$;
- Evaluations de f coûteuses :
 - Entraîner un réseau de neurones;
 - Prise de sang d'un patient;
 - Ramener un véhicule en usine.

Trouver les paramètres optimaux ?

- Sans dérivées, pas de garantie d'optimalité locale...
- Optimalité globale seulement garantie sous d'autres hypothèses (convexité) ou si l'on attend indéfiniment.

Notre objectif ?

Trouver les paramètres optimaux ?

- Sans dérivées, pas de garantie d'optimalité locale...
- Optimalité globale seulement garantie sous d'autres hypothèses (convexité) ou si l'on attend indéfiniment.

Trouver une meilleure configuration ?

- Toute amélioration est bonne à prendre;
- Les configurations de départ peuvent être fixées par des experts, et difficiles à améliorer.

Notre objectif ?

Trouver les paramètres optimaux ?

- Sans dérivées, pas de garantie d'optimalité locale...
- Optimalité globale seulement garantie sous d'autres hypothèses (convexité) ou si l'on attend indéfiniment.

Trouver une meilleure configuration ?

- Toute amélioration est bonne à prendre;
- Les configurations de départ peuvent être fixées par des experts, et difficiles à améliorer.

Fournir des garanties

- Validation de principe, guide pour choisir des méthodes;
- Métrique du moment : **complexité**.

Définition

A partir

- d'un critère de convergence/d'arrêt;
- d'une précision $\epsilon > 0$;
- d'un algorithme itératif $\{\mathbf{x}_k\}_k$;

borner le **nombre d'appels de fonction** requis dans le pire des cas pour satisfaire le critère avec précision ϵ .

Définition

A partir

- d'un critère de convergence/d'arrêt;
- d'une précision $\epsilon > 0$;
- d'un algorithme itératif $\{\mathbf{x}_k\}_k$;

borner le **nombre d'appels de fonction** requis dans le pire des cas pour satisfaire le critère avec précision ϵ .

La borne (en tant que fonction de ϵ) s'appelle la **complexité au pire cas** de l'algorithme.

Définition

A partir

- d'un critère de convergence/d'arrêt;
- d'une précision $\epsilon > 0$;
- d'un algorithme itératif $\{\mathbf{x}_k\}_k$;

borner le **nombre d'appels de fonction** requis dans le pire des cas pour satisfaire le critère avec précision ϵ .

La borne (en tant que fonction de ϵ) s'appelle la **complexité au pire cas** de l'algorithme.

Exemples de critère de convergence

- Gradient de f : $\|\nabla f(\mathbf{x}_k)\|$;
- Valeur de f : $f(\mathbf{x}_k)$.

1 Différentiation

2 L'optimisation sans dérivées

- Exemples et définition
- Méthodes de recherche directe
- Méthodes basées sur des modèles

But : Résoudre minimiser $\mathbf{x} \in \mathcal{F} f(\mathbf{x})$ avec accès à f uniquement, budget limité.

Algorithme de recherche basique

Start with: $\hat{\mathbf{x}}_0 = \mathbf{x}_0 \in \mathcal{F}$, $f = f(\mathbf{x}_0)$, $k = 0$.

- 1 Calculer \mathbf{x}_{k+1} et évaluer $f(\mathbf{x}_{k+1})$.

But : Résoudre minimiser $\mathbf{x} \in \mathcal{F} f(\mathbf{x})$ avec accès à f uniquement, budget limité.

Algorithme de recherche basique

Start with: $\hat{\mathbf{x}}_0 = \mathbf{x}_0 \in \mathcal{F}$, $f = f(\mathbf{x}_0)$, $k = 0$.

- 1 Calculer \mathbf{x}_{k+1} et évaluer $f(\mathbf{x}_{k+1})$.
- 2 Si $f(\mathbf{x}_{k+1}) < f(\hat{\mathbf{x}}_k)$ poser $\hat{\mathbf{x}}_{k+1} = \mathbf{x}_{k+1}$, sinon poser $\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k$.

But : Résoudre minimiser $\mathbf{x} \in \mathcal{F} f(\mathbf{x})$ avec accès à f uniquement, budget limité.

Algorithme de recherche basique

Start with: $\hat{\mathbf{x}}_0 = \mathbf{x}_0 \in \mathcal{F}$, $f = f(\mathbf{x}_0)$, $k = 0$.

- ➊ Calculer \mathbf{x}_{k+1} et évaluer $f(\mathbf{x}_{k+1})$.
- ➋ Si $f(\mathbf{x}_{k+1}) < f(\hat{\mathbf{x}}_k)$ poser $\hat{\mathbf{x}}_{k+1} = \mathbf{x}_{k+1}$, sinon poser $\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k$.
- ➌ Si budget dépassé terminer, sinon incrémenter k de 1.

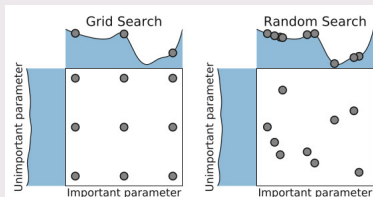
But : Résoudre minimiser $\mathbf{x} \in \mathcal{F} f(\mathbf{x})$ avec accès à f uniquement, budget limité.

Algorithme de recherche basique

Start with: $\hat{\mathbf{x}}_0 = \mathbf{x}_0 \in \mathcal{F}$, $f = f(\mathbf{x}_0)$, $k = 0$.

- 1 Calculer \mathbf{x}_{k+1} et évaluer $f(\mathbf{x}_{k+1})$.
- 2 Si $f(\mathbf{x}_{k+1}) < f(\hat{\mathbf{x}}_k)$ poser $\hat{\mathbf{x}}_{k+1} = \mathbf{x}_{k+1}$, sinon poser $\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k$.
- 3 Si budget dépassé terminer, sinon incrémenter k de 1.

- **Grille (Grid search) :** Valeurs des \mathbf{x}_k fixées a priori;
- **Aléatoire (Random search) :** Tirer \mathbf{x}_{k+1} au hasard.



Théorème

Soit $f^* = \min_{\mathbf{x} \in \mathcal{F}} f(\mathbf{x})$. Alors,

$$\mathbb{P}(f(\hat{\mathbf{x}}_K) \leq f^* + \epsilon) \geq p$$

après

$$K = \frac{\ln(p)}{\ln \left[\frac{\mu(\{\mathbf{x} \in \mathcal{F} | f(\mathbf{x}) > f^* + \epsilon\})}{\mu(\mathcal{F})} \right]}.$$

itérations.

Théorème

Soit $f^* = \min_{\mathbf{x} \in \mathcal{F}} f(\mathbf{x})$. Alors,

$$\mathbb{P}(f(\hat{\mathbf{x}}_K) \leq f^* + \epsilon) \geq p$$

après

$$K = \frac{\ln(p)}{\ln \left[\frac{\mu(\{\mathbf{x} \in \mathcal{F} | f(\mathbf{x}) > f^* + \epsilon\})}{\mu(\mathcal{F})} \right]}.$$

itérations.

- Plus : Valable pour f quelconque !
- Moins : Beaucoup d'itérations/d'évaluations de f ;
- Le budget est consommé en exploration.

Recherche exploratoire

- Variables en petite dimension;
- Algorithmes exploratoires.

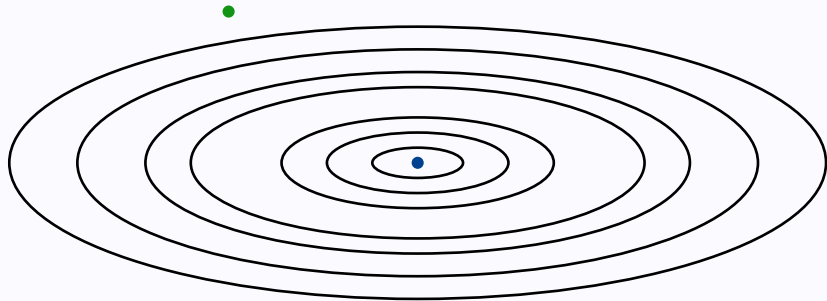
Recherche exploratoire

- Valables en petite dimension;
- Algorithmes exploratoires.

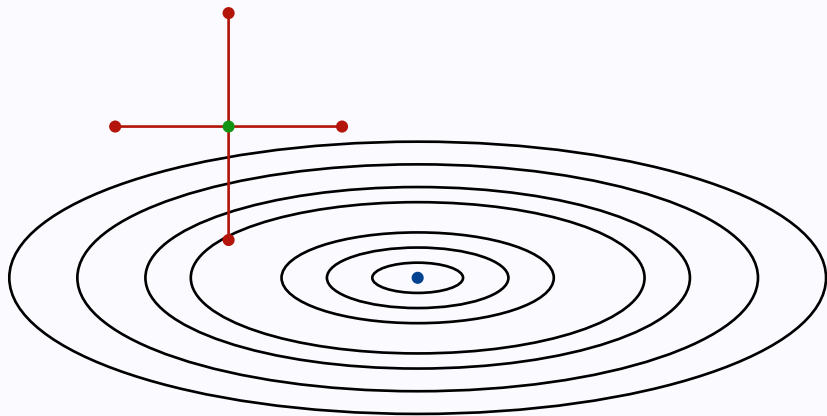
Recherche directe

- Origine : années 1960, théorie : années 1990;
- But : Aller au-delà de l'exploration pure.
- Intérêt : **simplicité, parallélisme**;
- La méthode du simplexe (Nelder-Mead, 1965) a plus de 125000 citations et est toujours la méthode sans dérivées de MATLAB!

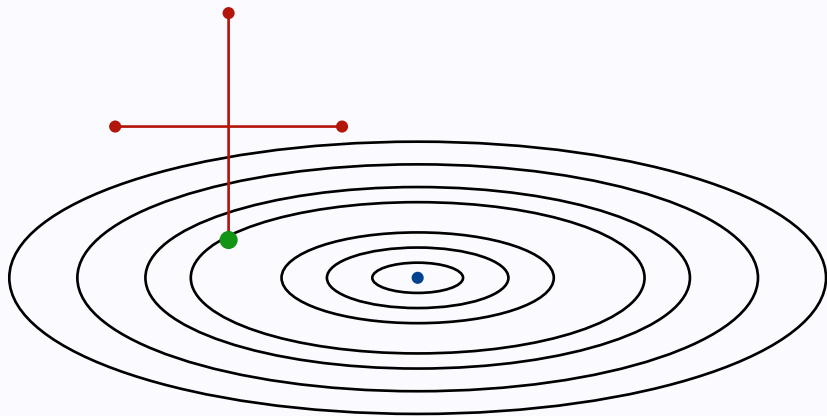
Exemple : Recherche par coordonnées



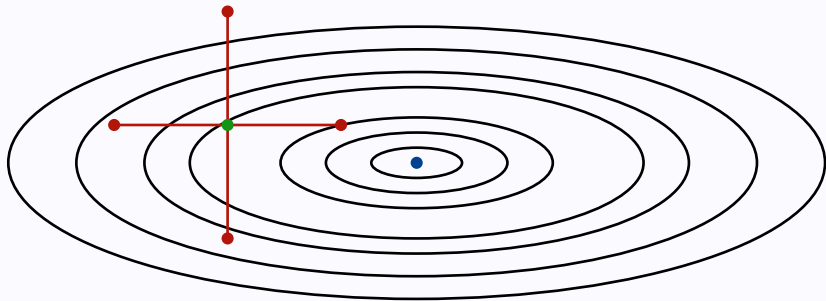
Exemple : Recherche par coordonnées



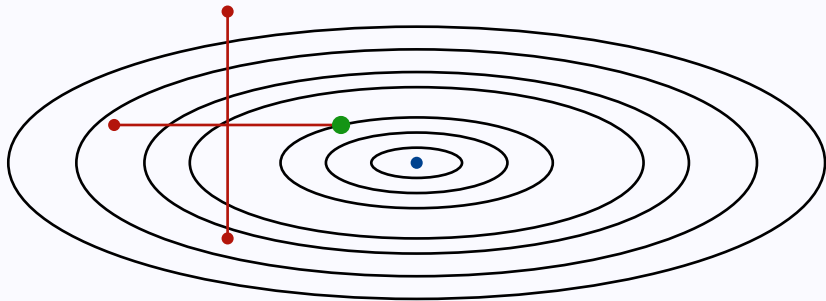
Exemple : Recherche par coordonnées



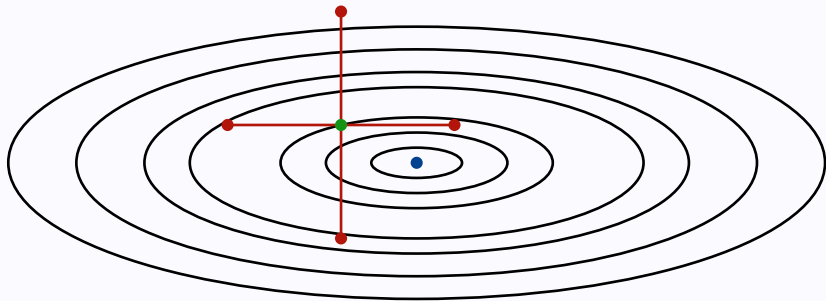
Exemple : Recherche par coordonnées



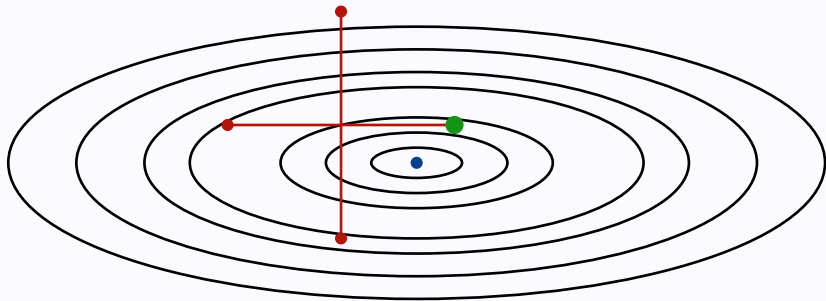
Exemple : Recherche par coordonnées



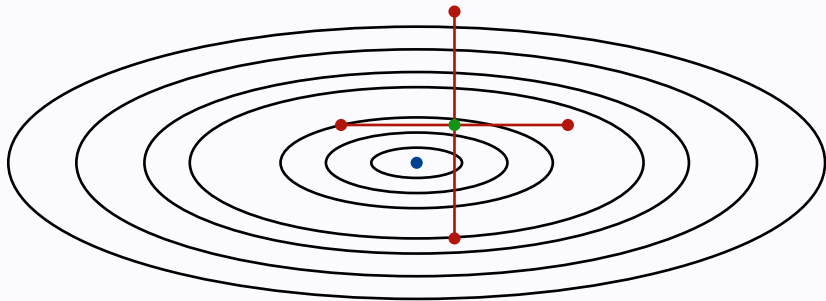
Exemple : Recherche par coordonnées



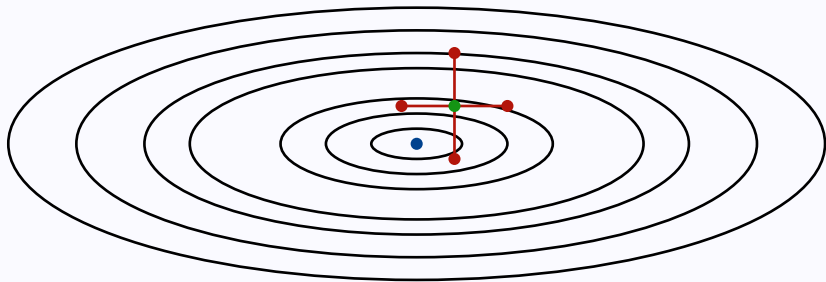
Exemple : Recherche par coordonnées



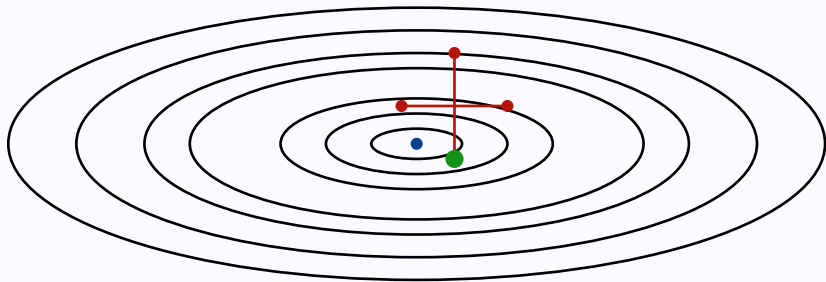
Exemple : Recherche par coordonnées



Exemple : Recherche par coordonnées



Exemple : Recherche par coordonnées



- ① Initialisation : $\mathbf{x}_0 \in \mathbb{R}^d, \alpha_0 > 0$.
- ② Pour $k = 0, 1, 2, \dots$
 - Choisir un ensemble $\mathcal{D}_k \subset \mathbb{R}^d$ de r directions.

① Initialisation : $\mathbf{x}_0 \in \mathbb{R}^d, \alpha_0 > 0$.

② Pour $k = 0, 1, 2, \dots$

- Choisir un ensemble $\mathcal{D}_k \subset \mathbb{R}^d$ de r directions.
- SI il existe $\mathbf{d}_k \in \mathcal{D}_k$ tel que

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k) - \alpha_k^2,$$

alors poser $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{d}_k$ et $\alpha_{k+1} \geq \alpha_k$ (itération réussie).

① Initialisation : $\mathbf{x}_0 \in \mathbb{R}^d, \alpha_0 > 0$.

② Pour $k = 0, 1, 2, \dots$

- Choisir un ensemble $\mathcal{D}_k \subset \mathbb{R}^d$ de r directions.
- SI il existe $\mathbf{d}_k \in \mathcal{D}_k$ tel que

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k) - \alpha_k^2,$$

alors poser $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{d}_k$ et $\alpha_{k+1} \geq \alpha_k$ (itération réussie).

- Sinon poser $\mathbf{x}_{k+1} := \mathbf{x}_k$ et $\alpha_{k+1} := 0.5\alpha_k$ (itération non réussie).

① Initialisation : $\mathbf{x}_0 \in \mathbb{R}^d, \alpha_0 > 0$.

② Pour $k = 0, 1, 2, \dots$

- Choisir un ensemble $\mathcal{D}_k \subset \mathbb{R}^d$ de r directions.
- SI il existe $\mathbf{d}_k \in \mathcal{D}_k$ tel que

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k) - \alpha_k^2,$$

alors poser $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{d}_k$ et $\alpha_{k+1} \geq \alpha_k$ (itération réussie).

- Sinon poser $\mathbf{x}_{k+1} := \mathbf{x}_k$ et $\alpha_{k+1} := 0.5\alpha_k$ (itération non réussie).

❶ Initialisation : $\mathbf{x}_0 \in \mathbb{R}^d, \alpha_0 > 0$.

❷ Pour $k = 0, 1, 2, \dots$

- Choisir un ensemble $\mathcal{D}_k \subset \mathbb{R}^d$ de r directions.
- SI il existe $\mathbf{d}_k \in \mathcal{D}_k$ tel que

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k) - \alpha_k^2,$$

alors poser $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{d}_k$ et $\alpha_{k+1} \geq \alpha_k$ (itération réussie).

- Sinon poser $\mathbf{x}_{k+1} := \mathbf{x}_k$ et $\alpha_{k+1} := 0.5\alpha_k$ (itération non réussie).

Aspects cruciaux

- Condition de décroissance;
- Choix de \mathcal{D}_k , valeur de r .

Positif

- **Exploitation** : On se base sur le point courant pour choisir le suivant;
- **Exploration** : Mouvement contrôlé par une longueur de pas α_k .

Négatif

- **Directions** : Au moins $d + 1$ évaluations par itération pour converger;
- **Dépendance en d** : Dans la complexité, mais aussi dans le nombre d'évaluations à chaque itération.

Positif

- **Exploitation** : On se base sur le point courant pour choisir le suivant;
- **Exploration** : Mouvement contrôlé par une longueur de pas α_k .

Négatif

- **Directions** : Au moins $d + 1$ évaluations par itération pour converger;
- **Dépendance en d** : Dans la complexité, mais aussi dans le nombre d'évaluations à chaque itération.

Pratique moderne

- Utiliser des directions aléatoires;
- S'applique au cas bruité.

$$\text{minimiser}_{\mathbf{x} \in \mathcal{F}} f(\mathbf{x})$$

Nouvelles hypothèses

- f seulement disponible via un oracle stochastique $\tilde{f}(\mathbf{x}; \xi)$;
- Le vecteur ξ est une quantité aléatoire dans un ensemble Ξ ;
- Typique : $\tilde{f}(\cdot; \xi)$ convexe en \mathbf{x} pour toute réalisation de ξ .
- Minimum de f atteint en \mathbf{x}_* .

minimiser $\mathbf{x} \in \mathcal{F}$ $f(\mathbf{x})$

Nouvelles hypothèses

- f seulement disponible via un oracle stochastique $\tilde{f}(\mathbf{x}; \xi)$;
 - Le vecteur ξ est une quantité aléatoire dans un ensemble Ξ ;
 - Typique : $\tilde{f}(\cdot; \xi)$ convexe en \mathbf{x} pour toute réalisation de ξ .
 - Minimum de f atteint en \mathbf{x}_* .
-
- Oracle stochastique \leftrightarrow “Bandit feedback”;
 - Liens modernes avec la littérature en IA sur les bandits et l’optimisation en ligne.

Cadre des bandits à plusieurs bras

- Ensemble de bras $\{1, \dots, A\}$;
- À l'itération k , on joue le bras \mathbf{x}_k , ξ_k est tirée, on obtient $f(\mathbf{x}_k; \xi_k)$;
- **Regret cumulé espéré :**

$$\mathbb{E} \left[\sum_{k=0}^{K-1} f(\mathbf{x}_k; \xi_k) \right] - Kf(\mathbf{x}_*),$$

Bandits avec infinités de bras (Auer, 2002)

- 1 On joue \mathbf{x}_k , ξ_k généré;
- 2 On observe $f(\mathbf{x}_k; \xi_k)$.

But (complexité) : $f(\bar{\mathbf{x}}_K) - f(\mathbf{x}_*) \leq \epsilon$, $\bar{\mathbf{x}}_K = \frac{1}{K} \sum_{k=0}^{K-1} \mathbf{x}_k$.

Méthodes à un point

Tirer $\mathbf{u}_k \sim \mathcal{U}(\mathbb{S}^{d-1})$ et utiliser

$$\frac{\tilde{f}(\mathbf{x}_k + \mu \mathbf{u}_k; \boldsymbol{\xi}_k)}{\mu} \mathbf{u}_k \quad \text{ou} \quad \frac{\tilde{f}(\mathbf{x}_k + \mu \mathbf{u}_k; \boldsymbol{\xi}_k^+) - \tilde{f}(\mathbf{x}_k - \mu \mathbf{u}_k; \boldsymbol{\xi}_k^-)}{\mu} \mathbf{u}_k$$

Méthodes deux/multi-pas

- Hypothèse : Le même $\boldsymbol{\xi}$ permet de faire plusieurs évaluations.
- Tirer $\mathbf{u}_k \sim \mathcal{U}(\mathbb{S}^{d-1})$ et prendre

$$\frac{\tilde{f}(\mathbf{x}_k + \mu_k \mathbf{u}_k; \boldsymbol{\xi}_k)}{\mu_k} \mathbf{u}_k \quad \text{or} \quad \frac{\tilde{f}(\mathbf{x}_k + \mu_k \mathbf{u}_k; \boldsymbol{\xi}_k) - \tilde{f}(\mathbf{x}_k - \mu_k \mathbf{u}_k; \boldsymbol{\xi}_k)}{\mu_k} \mathbf{u}_k$$

NOMAD/HyperNOMAD: <https://github.com/bbopt/HyperNOMAD>

- Dédié au départ aux problèmes physiques (HydroQuébec);
- C++/Matlab/Python;
- Gère de nombreuses difficultés non abordées ici :
 - Variables catégorielles, entières;
 - Contraintes plus ou moins relâchables.
- HyperNOMAD (2019) : Extension appliquée pour optimiser architectures et hyperparamètres de réseaux de neurones.

Méthode à base de bandits

- Hyperband (Jamieson et al 2016), BOHB (Falkner et al 2018): approches de bandits + optimisation bayésienne (cf ci-après);
- Garanties supérieures à la recherche aléatoire, applicables à un grand nombre de variables.

1 Différentiation

2 L'optimisation sans dérivées

- Exemples et définition
- Méthodes de recherche directe
- Méthodes basées sur des modèles

Résumé de ce qui précède

- Les méthodes de recherche directe explorent...
- ...et certaines **exploitent** localement.
- Utilisation de nouveaux points (notamment pour les méthodes aléatoires), pas de ré-utilisation d'information antérieure.

Résumé de ce qui précède

- Les méthodes de recherche directe explorent...
- ...et certaines **exploitent** localement.
- Utilisation de nouveaux points (notamment pour les méthodes aléatoires), pas de ré-utilisation d'information antérieure.

DFO basée sur des modèles

- Utilise des évaluations **passées** de la fonction pour en construire un modèle;
- Ré-utilise des points, beaucoup moins coûteux que des différences finies.

Algorithme : Régions de confiance sans dérivées

- But : minimiser $f(\mathbf{x})$ sur $\mathbf{x} \in \mathbb{R}^d$;
- Évaluations de f coûteuses.

Algorithme : Régions de confiance sans dérivées

- But : minimiser $f(\mathbf{x})$ sur \mathbb{R}^d ;
- Évaluations de f coûteuses.

Entrées : $\mathbf{x}_0 \in \mathbb{R}^d$, $\eta \in (0, 1)$, $\delta_0 > 0$.

Pour $k = 0, 1, 2, \dots$

- Calculer un modèle $\mathbf{s} \mapsto m_k(\mathbf{x}_k + \mathbf{s})$ de f en \mathbf{x}_k ;
- Calculer $\mathbf{s}_k \approx \operatorname{argmin}_{\|\mathbf{s}\| \leq \delta_k} m_k(\mathbf{x}_k + \mathbf{s})$;

Algorithme : Régions de confiance sans dérivées

- But : minimiser $f(\mathbf{x})$ sur $\mathbf{x} \in \mathbb{R}^d$;
- Évaluations de f coûteuses.

Entrées : $\mathbf{x}_0 \in \mathbb{R}^d$, $\eta \in (0, 1)$, $\delta_0 > 0$.

Pour $k = 0, 1, 2, \dots$

- Calculer un modèle $\mathbf{s} \mapsto m_k(\mathbf{x}_k + \mathbf{s})$ de f en \mathbf{x}_k ;
- Calculer $\mathbf{s}_k \approx \operatorname{argmin}_{\|\mathbf{s}\| \leq \delta_k} m_k(\mathbf{x}_k + \mathbf{s})$;
- Évaluer $\rho_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{x}_k) - m_k(\mathbf{x}_k + \mathbf{s}_k)}$.

Algorithme : Régions de confiance sans dérivées

- But : minimiser $f(\mathbf{x})$ sur $\mathbf{x} \in \mathbb{R}^d$;
- Évaluations de f coûteuses.

Entrées : $\mathbf{x}_0 \in \mathbb{R}^d$, $\eta \in (0, 1)$, $\delta_0 > 0$.

Pour $k = 0, 1, 2, \dots$

- Calculer un modèle $\mathbf{s} \mapsto m_k(\mathbf{x}_k + \mathbf{s})$ de f en \mathbf{x}_k ;
- Calculer $\mathbf{s}_k \approx \operatorname{argmin}_{\|\mathbf{s}\| \leq \delta_k} m_k(\mathbf{x}_k + \mathbf{s})$;
- Évaluer $\rho_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{x}_k) - m_k(\mathbf{x}_k + \mathbf{s}_k)}$.
- Si $\rho_k \geq \eta$, poser $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$ et $\delta_{k+1} \geq \delta_k$.

Algorithme : Régions de confiance sans dérivées

- But : minimiser $f(\mathbf{x})$ sur $\mathbf{x} \in \mathbb{R}^d$;
- Évaluations de f coûteuses.

Entrées : $\mathbf{x}_0 \in \mathbb{R}^d$, $\eta \in (0, 1)$, $\delta_0 > 0$.

Pour $k = 0, 1, 2, \dots$

- Calculer un modèle $\mathbf{s} \mapsto m_k(\mathbf{x}_k + \mathbf{s})$ de f en \mathbf{x}_k ;
- Calculer $\mathbf{s}_k \approx \operatorname{argmin}_{\|\mathbf{s}\| \leq \delta_k} m_k(\mathbf{x}_k + \mathbf{s})$;
- Évaluer $\rho_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{x}_k) - m_k(\mathbf{x}_k + \mathbf{s}_k)}$.
- Si $\rho_k \geq \eta$, poser $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$ et $\delta_{k+1} \geq \delta_k$.
- Sinon, poser $\mathbf{x}_{k+1} = \mathbf{x}_k$ et $\delta_{k+1} = \delta_k/2$.

Qualité du modèle

But : Approcher une fonction dérivable f par un modèle m .

- Bornes sur l'erreur d'approximation;
- Garanties locales, dans une **région de confiance**.

Qualité du modèle

But : Approcher une fonction dérivable f par un modèle m .

- Bornes sur l'erreur d'approximation;
- Garanties locales, dans une **région de confiance**.

Modèles pleinement linéaires

Le modèle m est κ -pleinement linéaire pour f en (\mathbf{x}, δ) si pour tout $\mathbf{y} \in B(\mathbf{x}, \delta)$,

$$\begin{aligned} |m(\mathbf{y}) - f(\mathbf{y})| &\leq \kappa \delta^2 \\ \|\nabla m(\mathbf{y}) - \nabla f(\mathbf{y})\| &\leq \kappa \delta. \end{aligned}$$

Construire des modèles pleinement linéaires

- \mathcal{P}_d^n : polynômes de degré n sur \mathbb{R}^d , $\dim \mathcal{P}_d^n = q + 1$;
- $\Phi = \{\phi_0(\cdot), \dots, \phi_q(\cdot)\}$: base de \mathcal{P}_d^n ;
- $\mathcal{Y} = \{\mathbf{y}^0, \dots, \mathbf{y}^p\}$: ensemble de $p + 1$ points de \mathbb{R}^d ;

Construire des modèles pleinement linéaires

- \mathcal{P}_d^n : polynômes de degré n sur \mathbb{R}^d , $\dim \mathcal{P}_d^n = q + 1$;
- $\Phi = \{\phi_0(\cdot), \dots, \phi_q(\cdot)\}$: base de \mathcal{P}_d^n ;
- $\mathcal{Y} = \{\mathbf{y}^0, \dots, \mathbf{y}^p\}$: ensemble de $p + 1$ points de \mathbb{R}^d ;
- **But** : modèle $m(\mathbf{x}) = \sum_{i=0}^q \alpha_i \phi_i(\mathbf{x})$ tels que

$$\forall j = 0, \dots, p, \quad m(\mathbf{y}^j) \approx f(\mathbf{y}^j).$$

- Reformulé comme $M(\Phi, \mathcal{Y})\alpha \approx f(\mathcal{Y})$, avec

$$M(\Phi, \mathcal{Y}) = \begin{bmatrix} \phi_0(\mathbf{y}^0) & \cdots & \phi_q(\mathbf{y}^0) \\ \vdots & \vdots & \vdots \\ \phi_0(\mathbf{y}^p) & \cdots & \phi_q(\mathbf{y}^p) \end{bmatrix}, \quad f(\mathcal{Y}) = \begin{bmatrix} f(\mathbf{y}^0) \\ \vdots \\ f(\mathbf{y}^p) \end{bmatrix}.$$

Régression polynomiale

Calculer α^* solution de

$$\text{minimiser}_{\alpha \in \mathbb{R}^{q+1}} \|M(\Phi, \mathcal{Y})\alpha - f(\mathcal{Y})\|^2.$$

et prendre $m(\mathbf{x}) = \sum_{i=0}^q \alpha_i^* \phi_i(\mathbf{x})$.

Régression polynomiale

Calculer α^* solution de

$$\text{minimiser}_{\alpha \in \mathbb{R}^{q+1}} \|M(\Phi, \mathcal{Y})\alpha - f(\mathcal{Y})\|^2.$$

et prendre $m(\mathbf{x}) = \sum_{i=0}^q \alpha_i^* \phi_i(\mathbf{x})$.

Cas classique

- Interpolation/régression linéaire avec $p = q = d$ (**technique de base d'analyse de données**);
- $\mathcal{Y} = \{\mathbf{y}^0, \mathbf{y}^1, \dots, \mathbf{y}^d\}$ sommets d'un simplexe de \mathbb{R}^d .

Qualité du modèle

But : Approcher une fonction **deux fois dérivable** f par un modèle m .

- Bornes sur l'erreur d'approximation;
- Garanties locales, dans une **région de confiance**.

Qualité du modèle

But : Approcher une fonction **deux fois dérivable** f par un modèle m .

- Bornes sur l'erreur d'approximation;
- Garanties locales, dans une **région de confiance**.

Modèles pleinement quadratiques

Le modèle m_k est κ -**pleinement quadratique** pour f sur (\mathbf{x}_k, δ_k) si pour tout $\mathbf{y} \in B(\mathbf{x}_k, \delta_k)$,

$$\begin{aligned} |m_k(\mathbf{y}) - f(\mathbf{y})| &\leq \kappa \delta_k^3 \\ \|\nabla m_k(\mathbf{y}) - \nabla f(\mathbf{y})\| &\leq \kappa \delta_k^2 \\ \|\nabla^2 m_k(\mathbf{y}) - \nabla^2 f(\mathbf{y})\| &\leq \kappa \delta_k. \end{aligned}$$

Choix classiques

Modèles m_k construits à partir de valeurs de f en $\mathcal{Y}_k = \{\mathbf{x}_k, \mathbf{y}^1, \dots, \mathbf{y}^r\}$:

- Interpolation/Régression;
- Radial basis functions (RBF, ou noyaux gaussiens).

Point clé

Bonne géométrie de $\mathcal{Y}_k \Rightarrow m_k$ pleinement quadratique sur $B(\mathbf{x}_k, \delta_k)$.

Ex) Interpolation quadratique avec $r = \mathcal{O}(n^2)$ échantillons

$$\mathcal{Y}_k = \left\{ \mathbf{x}_k, \{\mathbf{x}_k \pm \delta_k \mathbf{e}_i\}_{i=1}^n, \{\mathbf{x}_k + \delta_k \frac{\mathbf{e}_i + \mathbf{e}_j}{2}\}_{1 \leq i < j \leq n} \right\}.$$

Construire des modèles pleinement quadratiques en pratique

Choix classiques

Modèles m_k construits à partir de valeurs de f en $\mathcal{Y}_k = \{\mathbf{x}_k, \mathbf{y}^1, \dots, \mathbf{y}^r\}$:

- Interpolation/Régression;
- Radial basis functions (RBF, ou noyaux gaussiens).

Point clé

Bonne géométrie de $\mathcal{Y}_k \Rightarrow m_k$ pleinement quadratique sur $B(\mathbf{x}_k, \delta_k)$.

Ex) Interpolation quadratique avec $r = \mathcal{O}(n^2)$ échantillons

$$\mathcal{Y}_k = \left\{ \mathbf{x}_k, \{\mathbf{x}_k \pm \delta_k \mathbf{e}_i\}_{i=1}^n, \{\mathbf{x}_k + \delta_k \frac{\mathbf{e}_i + \mathbf{e}_j}{2}\}_{1 \leq i < j \leq n} \right\}.$$

Dans la pratique

- On ré-utilise autant de points que possible!
- On contrôle la géométrie et on la corrige si besoin:

Construire des modèles pleinement quadratiques en pratique

Choix classiques

Modèles m_k construits à partir de valeurs de f en $\mathcal{Y}_k = \{\mathbf{x}_k, \mathbf{y}^1, \dots, \mathbf{y}^r\}$:

- Interpolation/Régression;
- Radial basis functions (RBF, ou noyaux gaussiens).

Point clé

Bonne géométrie de $\mathcal{Y}_k \Rightarrow m_k$ pleinement quadratique sur $B(\mathbf{x}_k, \delta_k)$.

Ex) Interpolation quadratique avec $r = \mathcal{O}(n^2)$ échantillons

$$\mathcal{Y}_k = \left\{ \mathbf{x}_k, \{\mathbf{x}_k \pm \delta_k \mathbf{e}_i\}_{i=1}^n, \{\mathbf{x}_k + \delta_k \frac{\mathbf{e}_i + \mathbf{e}_j}{2}\}_{1 \leq i < j \leq n} \right\}.$$

Dans la pratique

- On ré-utilise autant de points que possible!
- On contrôle la géométrie et on la corrige si besoin:
 - Peut demander r nouvelles valeurs;

Construire des modèles pleinement quadratiques en pratique

Choix classiques

Modèles m_k construits à partir de valeurs de f en $\mathcal{Y}_k = \{\mathbf{x}_k, \mathbf{y}^1, \dots, \mathbf{y}^r\}$:

- Interpolation/Régression;
- Radial basis functions (RBF, ou noyaux gaussiens).

Point clé

Bonne géométrie de $\mathcal{Y}_k \Rightarrow m_k$ pleinement quadratique sur $B(\mathbf{x}_k, \delta_k)$.

Ex) Interpolation quadratique avec $r = \mathcal{O}(n^2)$ échantillons

$$\mathcal{Y}_k = \left\{ \mathbf{x}_k, \{\mathbf{x}_k \pm \delta_k \mathbf{e}_i\}_{i=1}^n, \{\mathbf{x}_k + \delta_k \frac{\mathbf{e}_i + \mathbf{e}_j}{2}\}_{1 \leq i < j \leq n} \right\}.$$

Dans la pratique

- On ré-utilise autant de points que possible!
- On contrôle la géométrie et on la corrige si besoin:
 - Peut demander r nouvelles valeurs;
 - En pratique, bien plus économe.

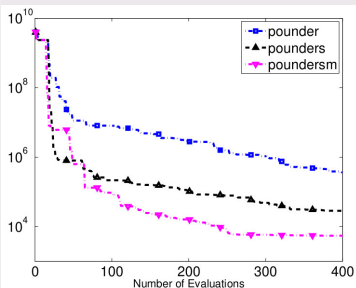
Remarque : Utiliser la structure

- Nous avons traité la fonction en “boîte noire”;
- Souvent on peut calculer les dérivées d'une partie de la fonction;
- Globalement, connaître la structure peut aider!

Remarque : Utiliser la structure

- Nous avons traité la fonction en “boîte noire”;
- Souvent on peut calculer les dérivées d'une partie de la fonction;
- Globalement, connaître la structure peut aider!

Ex) minimiser $\mathbf{x} \in \mathbb{R}^d$ $f(\mathbf{x}) = \frac{1}{2} \|r(\mathbf{x})\|^2$ $r : \mathbb{R}^d \rightarrow \mathbb{R}^n$.



- Dérivées de r
inaccessibles mais on a :

$$\nabla f(\mathbf{x}) = \mathbf{J}_r(\mathbf{x})^T r(\mathbf{x}).$$

- On construit ainsi des modèles plus précis.

Avec des modèles linéaires

- En pratique, ré-utiliser des points/prendre moins de $d + 1$ points marche;
- En théorie, il faut $\mathcal{O}(d)$ points pour être pleinement linéaires!

Avec des modèles linéaires

- En pratique, ré-utiliser des points/prendre moins de $d + 1$ points marche;
- En théorie, il faut $\mathcal{O}(d)$ points pour être pleinement linéaires!

Idée

Supposer que les modèles sont pleinement linéaires **en probabilité**.

- Processus aléatoire;
- Analysé via des arguments statistiques (martingales).

Sous-échantillonnage

$$f(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{x}), \nabla f_i \text{ disponible mais pas } \nabla f$$

- Choisir $S \subset \{1, \dots, N\}$ aléatoirement;
- Poser $m(\mathbf{x} + \mathbf{s}) = f(\mathbf{x}) + \frac{1}{|S|} \sum_{i \in S} \nabla f_i(\mathbf{x})^\top \mathbf{s}$;
- Avec $|S| = \mathcal{O}(\delta^{-2})$, le modèle est pleinement linéaire en probabilité.

Sous-échantillonnage

$$f(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{x}), \nabla f_i \text{ disponible mais pas } \nabla f$$

- Choisir $S \subset \{1, \dots, N\}$ aléatoirement;
- Poser $m(\mathbf{x} + \mathbf{x}) = f(\mathbf{x}) + \frac{1}{|S|} \sum_{i \in S} \nabla f_i(\mathbf{x})^\top \mathbf{s}$;
- Avec $|S| = \mathcal{O}(\delta^{-2})$, le modèle est pleinement linéaire en probabilité.

Modèles pleinement quadratiques en proba.

- Déterministe : $\mathcal{O}(d^2)$ évaluations;
- Si la matrice hessienne est creuse, vrai (en proba.) avec $\mathcal{O}(d(\log d)^4)$ évaluations via des techniques de **compression de signal**.

Paradigme de l'optimisation bayésienne

- ➊ À chaque itération, calculer une *distribution a posteriori* relativement aux évaluations connues;
- ➋ Trouver un nouveau point en maximisant une *fonction d'acquisition*;
- ➌ Répéter.

Paradigme de l'optimisation bayésienne

- 1 À chaque itération, calculer une *distribution a posteriori* relativement aux évaluations connues;
- 2 Trouver un nouveau point en maximisant une *fonction d'acquisition*;
- 3 Répéter.

- Populaire chez les *data scientists*:
- Les modèles sont basés sur des processus gaussiens et donc des fonctions RBF :

$$m_k(\mathbf{x}_k + \mathbf{s}) = \sum_{i=1}^{|\mathcal{Y}|} \exp(-\|\mathbf{y}_i - \mathbf{s}\|^2)$$

⇒ Ce sont des modèles pleinement linéaires !

Construire des modèles

- Utilise l'historique de l'algorithme;
- Implémentations efficaces meilleures qu'estimer les dérivées directement;
- Il faut exploiter de la structure s'il en existe.

Popularité des modèles

- Méta-modèles/*Surrogates* en calcul scientifique;
- Méta-modèles/processus gaussiens en optimisation bayésienne.

Codes de M. J. D. Powell

- En FORTRAN 77, mais toujours l'un des meilleurs codes disponibles;
- PDFO (<https://www.pdf0.net/>), interfaces Python et MATLAB;
- Récemment utilisé en IA (apprentissage adverse).

Autres codes

- POUNDERS : Moindres carrés sans dérivées (structure);
- ORBIT : Régions de confiance, modèle RBF;
- Packages Python/R pour l'optimisation de *surrogates*/ l'optimisation bayésienne.

Calcul de gradient

- Via différentiation automatique.
- Backpropagation pour les réseaux de neurones.

En l'absence de gradients

- Dérivées généralisées (type sous-gradients).
- Algorithmes généralisés.
- Différentiation automatique.

Optimisation sans dérivées/DFO

- Pas de dérivées dans l'algorithme (ou pas toutes).
- Tout dépend de l'évaluation et de son coût.
- Application : Optimisation d'hyperparamètres.

Différentiation

- A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Automatic Differentiation*. SIAM, 2008.
- Packages : JAX, Autograd.

Optimisation sans dérivées

- J. Larson, M. Menickelly and S. M. Wild, *Derivative-free optimization methods*, Acta Numerica, 2019.
- M. Feurer and F. Hutter, *Hyperparameter Optimization*, in *Automated Machine Learning*, Springer, 2019.
- P. I. Frazier, *Bayesian Optimization*, Tutorials in Operations Research, 2018.

Merci de votre attention !

`clement.royer@lamsade.dauphine.fr`

Crédits

- <https://towardsdatascience.com/>
- <https://commons.wikimedia.org/>
- A. J. Booker, J. E. Dennis Jr., P. D. Frank, D. B. Serafini and V. Torczon, *A rigorous framework for optimization of expensive functions by surrogates*, Structural Optim., 1999.
- M. Feurer and F. Hutter, *Hyperparameter Optimization*, in *Automated Machine Learning*, Springer, 2019.
- D. Gaudrie (Stellantis).
- S. M. Wild, *Beyond the Black Box in Derivative-Free and Simulation-Based Optimization*, SIAM Annual Meeting, 2016.