

Algorithmes d'optimisation numérique

Une introduction

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Introduction – Pourquoi des algorithmes d'optimisation ?

- ▶ En maximum de vraisemblance, on cherche $\hat{\theta}_N$ tel que :

$$\hat{\theta}_N = \arg \max_{\theta \in \Theta} \ell(y|X, \theta)$$

où $\ell(\cdot)$ est la log-vraisemblance.

- ▶ En général, la condition du premier ordre :

$$s(\hat{\theta}_N) = \left. \frac{\partial \ell}{\partial \theta} \right|_{\hat{\theta}_N} = 0$$

n'admet pas de solution analytique (contrairement aux MCO).

- ▶ \Rightarrow Nécessité de méthodes **numériques itératives**.

Introduction – Notations

▶ $\ell(\theta)$: log-vraisemblance (fonction à maximiser)

▶ $s(\theta) = \frac{\partial \ell(\theta)}{\partial \theta}$: **score** (gradient, vecteur $p \times 1$)

▶ $H(\theta) = \frac{\partial^2 \ell(\theta)}{\partial \theta \partial \theta'}$: **hessien** (matrice $p \times p$)

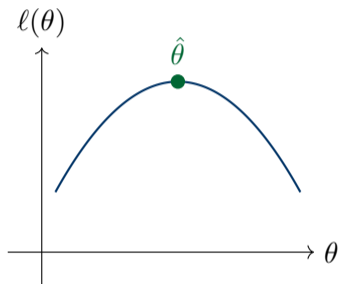
▶ Contributions individuelles :

$$\ell(\theta) = \sum_{i=1}^N \ln f(y_i | X_i, \theta), \quad s(\theta) = \sum_{i=1}^N \frac{\partial \ln f_i}{\partial \theta}, \quad H(\theta) = \sum_{i=1}^N \frac{\partial^2 \ln f_i}{\partial \theta \partial \theta'}$$

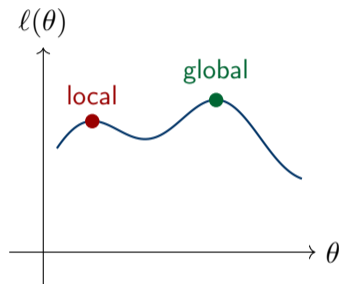
Introduction – Concavité et unicité

- ▶ $\ell(\theta)$ est **concave** si $H(\theta) \prec 0$ (définie négative) pour tout $\theta \in \Theta$.
- ▶ Conséquences de la concavité :
 - ▶ Le maximum est unique
 - ▶ La CPO $s(\hat{\theta}_N) = 0$ est suffisante
 - ▶ Tout algorithme croissant converge vers ce maximum
- ▶ Modèles à log-vraisemblance concave : **Logit**, **Probit**, **Poisson**, modèle linéaire gaussien.

Introduction – Concavité (illustration)



Concave
(maximum unique)



Non concave
(maxima multiples)

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Algorithme – Les quatre composantes

1. **Valeur initiale** $\theta_{(0)}$
2. **Règle d'itération** : $\theta_{(p+1)} = \theta_{(p)} + M(\theta_{(p)})$
3. **Critère d'arrêt** : quand s'arrêter ?
4. **Vérification** : a-t-on bien un maximum ?



Algorithme – Choix de la valeur initiale

- ▶ **Cas d'une fonction concave :**
 - ▶ Tout point de départ mène au maximum (avec un algorithme croissant)
 - ▶ On peut prendre $\theta_{(0)} = 0$ ou une valeur raisonnable
- ▶ **Cas d'une fonction non concave :**
 - ▶ Le choix de $\theta_{(0)}$ est crucial
 - ▶ Utiliser un estimateur convergent comme point de départ
 - ▶ Exemple : estimateur en deux étapes (plus simple mais inefficace)
- ▶ Un bon point de départ accélère la convergence, même avec une fonction concave.

Algorithme – Règle d'itération

- ▶ Forme générale :

$$\theta_{(p+1)} = \theta_{(p)} + M(\theta_{(p)})$$

où $M(\theta_{(p)})$ est le **pas de l'itération**.

- ▶ Propriétés souhaitables :

- ▶ Le pas ne dépend que de $\theta_{(p)}$ (et des données)

- ▶ À l'optimum : $M(\hat{\theta}_N) = 0$ (stationnarité)

- ▶ L'algorithme est **croissant** : $\ell(\theta_{(p+1)}) \geq \ell(\theta_{(p)})$

- ▶ Les différents algorithmes diffèrent par leur définition de $M(\cdot)$.

Algorithme – Critères d'arrêt

1. **Variation de l'objectif** : $|\ell(\theta_{(p+1)}) - \ell(\theta_{(p)})| < \varepsilon_1$
2. **Norme du gradient** : $\|s(\theta_{(p)})\| < \varepsilon_2$
3. **Variation des paramètres** : $\|\theta_{(p+1)} - \theta_{(p)}\| < \varepsilon_3$
4. **Élasticité** (insensible aux unités) : $\left| \frac{\theta_{(p)}}{\ell(\theta_{(p)})} \cdot s(\theta_{(p)}) \right| < \varepsilon_4$

À l'optimum, tous ces critères sont équivalents (tous nuls).

Algorithme – Vérification du maximum

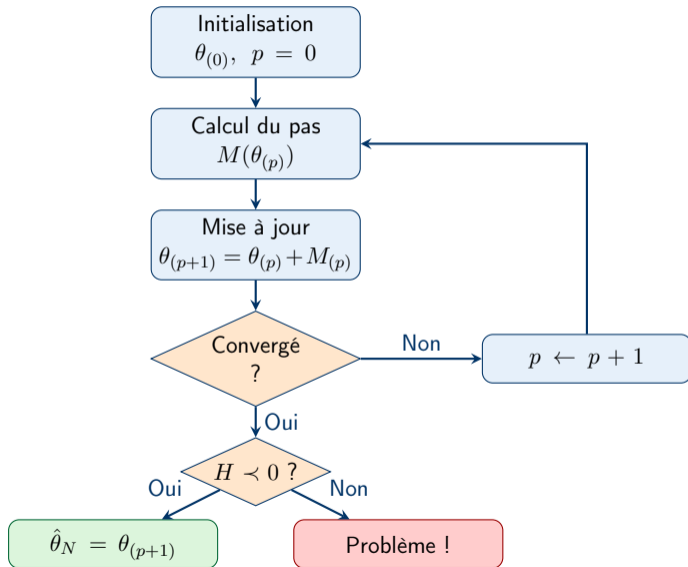
- ▶ Pour un **maximum local**, le hessien doit être défini négatif :

$$H(\hat{\theta}_N) = \frac{\partial^2 \ell}{\partial \theta \partial \theta'} \Big|_{\hat{\theta}_N} \prec 0$$

- ▶ Importance pratique :

- ▶ Si $H(\hat{\theta}_N) \not\prec 0$: on n'est pas à un maximum !
- ▶ L'inverse du hessien estime la variance : $\widehat{\text{Var}}(\hat{\theta}_N) \approx -H(\hat{\theta}_N)^{-1}$
- ▶ Si H n'est pas définie négative \Rightarrow variance non définie positive \Rightarrow inutilisable

Algorithme – Schéma complet



Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Gradient – Principe général

- ▶ Un algorithme de gradient suit la règle d'itération :

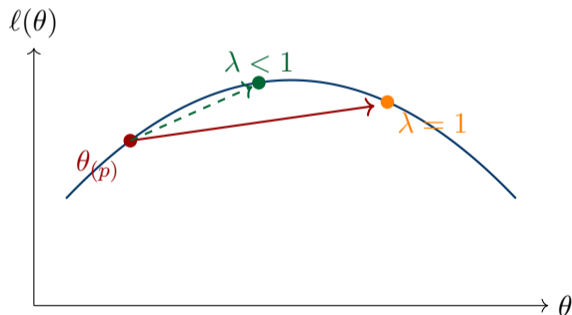
$$\theta_{(p+1)} = \theta_{(p)} + \lambda W_{(p)} s(\theta_{(p)})$$

où :

- ▶ $s(\theta_{(p)})$: gradient (score) au point courant
 - ▶ $W_{(p)}$: matrice de pondération ($p \times p$, symétrique)
 - ▶ $\lambda \in (0, 1]$: paramètre de pas (*step size*)
-
- ▶ À l'optimum : $s(\hat{\theta}_N) = 0$ donc le pas est nul et l'algorithme s'arrête.

Gradient – Le paramètre de pas λ

- ▶ Contrôle l'amplitude du déplacement. Par défaut : $\lambda = 1$.
- ▶ Si $\ell(\theta_{(p+1)}) < \ell(\theta_{(p)})$: réduire λ (*line search*).



Gradient – Propriété de croissance

Théorème

Si $W_{(p)}$ est symétrique définie positive, alors pour λ suffisamment petit :

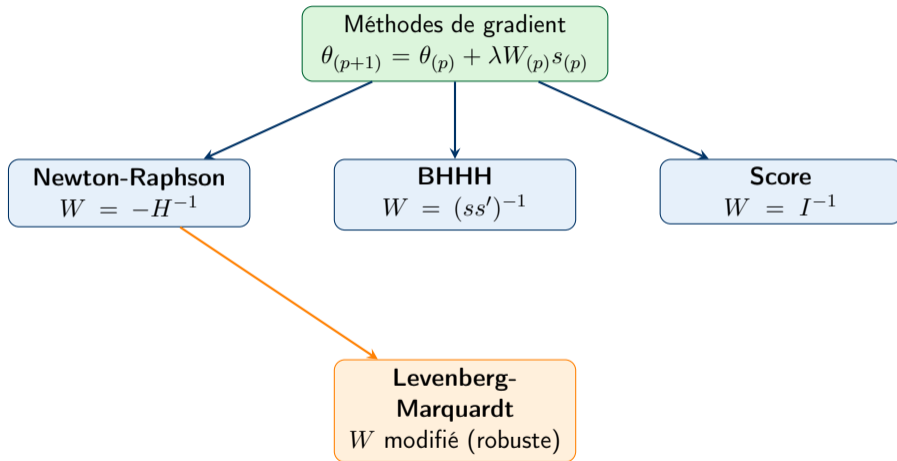
$$\ell(\theta_{(p+1)}) \geq \ell(\theta_{(p)})$$

- ▶ Idée de la preuve. Développement de Taylor :

$$\ell(\theta_{(p+1)}) \approx \ell(\theta_{(p)}) + \lambda \underbrace{s(\theta_{(p)})' W_{(p)} s(\theta_{(p)})}_{>0 \text{ si } W_{(p)} \succ 0}$$

- ▶ Pour une fonction concave, un algorithme croissant converge vers le maximum global.

Gradient – Vue d'ensemble



Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

Méthodes d'optimisation globale

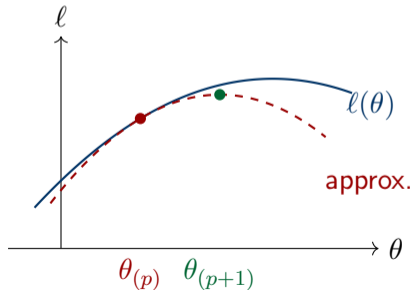
Méthodologie de programmation

Récapitulatif

Newton-Raphson – Principe

- ▶ **Idée** : Approximer $\ell(\theta)$ par une fonction quadratique au voisinage de $\theta_{(p)}$, puis maximiser cette approximation.
- ▶ Développement de Taylor (ordre 2) :

$$\ell(\theta) \approx \ell(\theta_{(p)}) + s(\theta_{(p)})'(\theta - \theta_{(p)}) + \frac{1}{2}(\theta - \theta_{(p)})'H(\theta_{(p)})(\theta - \theta_{(p)})$$



Newton-Raphson – Dérivation

- ▶ Maximisation de l'approximation quadratique (CPO) :

$$s(\theta_{(p)}) + H(\theta_{(p)})(\theta - \theta_{(p)}) = 0$$

- ▶ Solution :

$$\theta - \theta_{(p)} = -H(\theta_{(p)})^{-1}s(\theta_{(p)})$$

- ▶ Règle d'itération :

$$\theta_{(p+1)} = \theta_{(p)} - \lambda H(\theta_{(p)})^{-1}s(\theta_{(p)})$$

Ici : $W_{(p)} = -H(\theta_{(p)})^{-1}$

Newton-Raphson – Propriétés

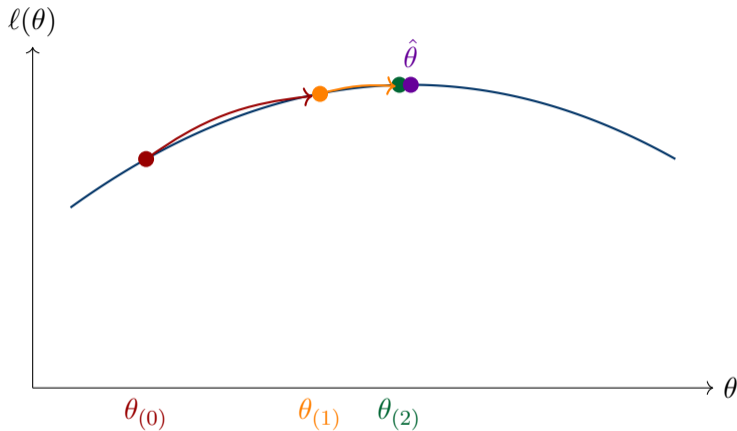
▶ **Avantages :**

- ▶ Convergence rapide (quadratique près de l'optimum)
- ▶ Peu d'itérations en général
- ▶ Vérifie la CSO (on calcule H à chaque itération)
- ▶ Fournit la variance : $\widehat{\text{Var}}(\hat{\theta}_N) \approx -H(\hat{\theta}_N)^{-1}$

▶ **Inconvénients :**

- ▶ Nécessite le calcul des dérivées secondes
- ▶ Coûteux en temps si dérivées numériques
- ▶ Peut diverger si H n'est pas définie négative
- ▶ Sensible au point de départ si fonction non concave

Newton-Raphson – Convergence



Newton-Raphson converge souvent en **3 à 5 itérations** pour des problèmes bien conditionnés.

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

BHHH – Motivation

- ▶ Le calcul du hessien $H(\theta)$ peut être complexe analytiquement ou coûteux numériquement.
- ▶ **Idée** : Utiliser l'égalité de la matrice d'information :

$$\mathbb{E} \left[-\frac{\partial^2 \ln f}{\partial \theta \partial \theta'} \right] = \mathbb{E} \left[\frac{\partial \ln f}{\partial \theta} \frac{\partial \ln f}{\partial \theta'} \right]$$

- ▶ \Rightarrow Approximer $-H(\theta)$ par les **produits croisés du score**.

BHHH – Approximation du hessien

- ▶ On remplace :

$$-H(\theta) = -\sum_{i=1}^N \frac{\partial^2 \ln f_i}{\partial \theta \partial \theta'}$$

par :

$$\sum_{i=1}^N \frac{\partial \ln f_i}{\partial \theta} \frac{\partial \ln f_i}{\partial \theta'} = \sum_{i=1}^N s_i s_i'$$

où $s_i = \frac{\partial \ln f_i}{\partial \theta}$ est le score individuel.

- ▶ Cette matrice est automatiquement semi-définie positive (somme de matrices $s_i s_i' \succeq 0$).

BHHH – Règle d'itération

- ▶ Règle d'itération :

$$\theta_{(p+1)} = \theta_{(p)} + \lambda \left(\sum_{i=1}^N s_i(\theta_{(p)}) s_i(\theta_{(p)})' \right)^{-1} s(\theta_{(p)})$$

où $s_i(\theta) = \frac{\partial \ln f(y_i|X_i,\theta)}{\partial \theta}$ et $s(\theta) = \sum_i s_i(\theta)$.

- ▶ En notation compacte, soit S la matrice $N \times p$ des scores individuels (ligne i : s_i') :

$$W_{(p)} = (S' S)^{-1}$$

BHHH – Propriétés

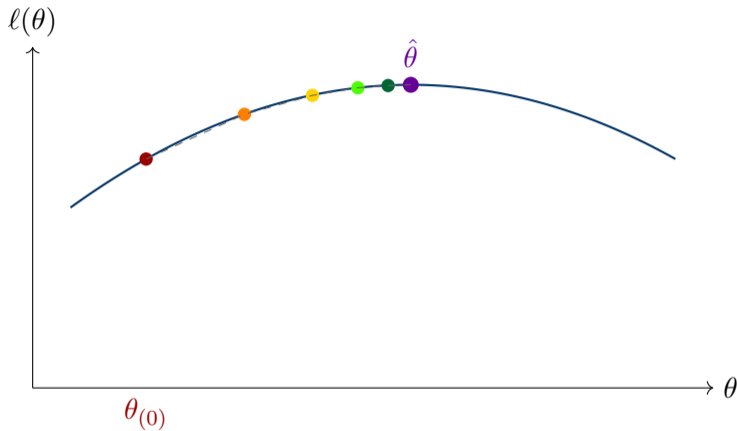
▶ **Avantages :**

- ▶ Ne nécessite que les dérivées premières
- ▶ Facile à programmer
- ▶ $W_{(p)} = (S'S)^{-1}$ est toujours définie positive
- ▶ Algorithme toujours croissant

▶ **Inconvénients :**

- ▶ Convergence plus lente que Newton-Raphson
- ▶ Plus d'itérations nécessaires
- ▶ Ne vérifie pas directement que $H \prec 0$
- ▶ L'approximation peut être mauvaise loin de l'optimum

BHHH – Convergence



BHHH fait des **pas plus petits** que Newton-Raphson \Rightarrow plus d'itérations mais plus stable.

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Score – Principe

- ▶ **Idée** : Raffinement de BHHH. Remplacer les produits croisés du score par leur espérance mathématique (information de Fisher).
- ▶ Approximation :

$$-H(\theta) \approx \sum_{i=1}^N \mathbb{E}_y \left[\frac{\partial \ln f_i}{\partial \theta} \frac{\partial \ln f_i}{\partial \theta'} \right] = N \cdot I_1(\theta)$$

où $I_1(\theta)$ est l'information de Fisher pour une observation.

Score – Règle d'itération

- ▶ Règle d'itération :

$$\theta_{(p+1)} = \theta_{(p)} + \lambda \left(\sum_{i=1}^N \mathbb{E}_y [s_i s_i'] \Big|_{\theta_{(p)}} \right)^{-1} s(\theta_{(p)})$$

- ▶ Simplification :

$$W_{(p)} = I_N(\theta_{(p)})^{-1} = \frac{1}{N} I_1(\theta_{(p)})^{-1}$$

- ▶ L'algorithme du Score utilise l'information de Fisher **théorique** plutôt qu'empirique.

Score – Équivalence avec Newton-Raphson

Théorème

Si les dérivées secondes de $\ln f$ ne dépendent pas de y , alors :

$$\mathbb{E}_y \left[-\frac{\partial^2 \ln f}{\partial \theta \partial \theta'} \right] = -\frac{\partial^2 \ln f}{\partial \theta \partial \theta'}$$

et l'algorithme du Score est **identique** à Newton-Raphson.

- ▶ Modèles où Score = Newton-Raphson :
 - ▶ **Logit** : H ne dépend que de X et θ
 - ▶ **Poisson** : idem
 - ▶ Modèle linéaire gaussien

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

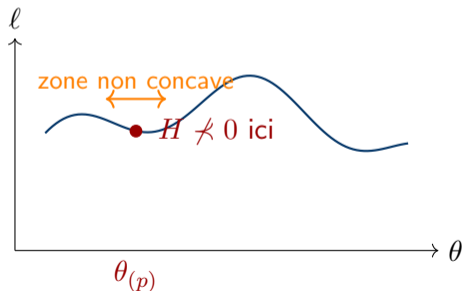
Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Levenberg-Marquardt – Problème de départ

- ▶ Si $H(\theta_{(p)})$ n'est pas définie négative au point $\theta_{(p)}$:
 - ▶ Newton-Raphson n'est plus garanti croissant
 - ▶ L'algorithme peut diverger
 - ▶ BHHH peut avoir une valeur propre nulle \Rightarrow non inversible



Levenberg-Marquardt – Idée

- ▶ **Solution** : Modifier le hessien pour le rendre défini négatif, tout en restant aussi proche que possible de l'original.
- ▶ On remplace $H(\theta_{(p)})$ par :

$$\tilde{H}_{(p)} = H(\theta_{(p)}) - (1 + \alpha)\mu_H I_k$$

où :

- ▶ μ_H : plus grande valeur propre de $H(\theta_{(p)})$ (supposée > 0)
- ▶ $\alpha > 0$: paramètre choisi par l'utilisateur (petit)
- ▶ I_k : matrice identité de dimension k (taille de θ)

Levenberg-Marquardt – Justification

- ▶ Si $\lambda_1, \dots, \lambda_k$ sont les valeurs propres de H , alors les valeurs propres de \tilde{H} sont :

$$\tilde{\lambda}_j = \lambda_j - (1 + \alpha)\mu_H$$

- ▶ En particulier, pour $\lambda_j = \mu_H$ (la plus grande) :

$$\tilde{\lambda}_j = \mu_H - (1 + \alpha)\mu_H = -\alpha\mu_H < 0$$

- ▶ Toutes les valeurs propres de \tilde{H} sont strictement négatives $\Rightarrow \tilde{H} \prec 0$.

Levenberg-Marquardt – Règle d'itération

- ▶ Règle complète :

$$\theta_{(p+1)} = \begin{cases} \theta_{(p)} - \lambda (H(\theta_{(p)}) - (1 + \alpha)\mu_H I_k)^{-1} s(\theta_{(p)}) & \text{si } \mu_H \geq 0 \\ \theta_{(p)} - \lambda H(\theta_{(p)})^{-1} s(\theta_{(p)}) & \text{si } \mu_H < 0 \end{cases}$$

- ▶ Si $\mu_H < 0$: $H \prec 0$ donc on utilise Newton-Raphson standard.
- ▶ Si $\mu_H \geq 0$: on modifie H pour le rendre défini négatif.

Levenberg-Marquardt – Choix de α

- ▶ Commencer avec une petite valeur : $\alpha = 0.01$ ou 0.1 .
- ▶ Observer l'évolution de μ_H au fil des itérations :
 - ▶ Si μ_H croît : α trop élevé
 - ▶ Si μ_H tend vers 0 : α trop petit
 - ▶ Valeur correcte : comportement intermédiaire
- ▶ L'algorithme peut être sensible à de petites variations de α (ex. : $\alpha = 0.1$ fonctionne, $\alpha = 0.2$ diverge).
- ▶ Une fois α trouvé pour un échantillon, on le garde généralement constant.

Levenberg-Marquardt – Applications

- ▶ **Modèles à objectif non concave :**
 - ▶ Tobit généralisé (modèle de Heckman)
 - ▶ Moindres carrés non linéaires
 - ▶ Modèles à équations simultanées
 - ▶ Certains modèles de mélange
- ▶ **Avantages :**
 - ▶ Robuste : fonctionne même si H n'est pas définie négative
 - ▶ Retrouve Newton-Raphson dans les zones concaves
 - ▶ Permet de traverser les régions non concaves

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Comparaison – Tableau récapitulatif

Algorithme	Matrice W	Dérivées	Vitesse	Robustesse
Newton-Raphson	$-H^{-1}$	2nd ordre	***	*
BHHH	$(SS')^{-1}$	1er ordre	*	**
Score	I^{-1}	$\mathbb{E}[ss']$	**	**
Levenberg-Marquardt	$-\tilde{H}^{-1}$	2nd ordre	**	***

* = faible, ** = moyen, *** = élevé

Comparaison – Guide de choix

- ▶ **Fonction objectif concave (cas usuel) :**
 1. Newton-Raphson si dérivées analytiques disponibles
 2. BHHH pour le prototypage ou si dérivées 2nd ordre complexes
 3. Score si H ne dépend pas de y (équivalent à N-R)
- ▶ **Fonction objectif non concave :**
 1. Levenberg-Marquardt : choix par défaut
 2. Commencer par un bon point initial (estimateur convergent)
 3. Ajuster α si nécessaire

Comparaison – Stratégie pratique

1. Phase de développement :

- ▶ Utiliser BHHH (plus simple à coder)
- ▶ Vérifier numériquement les dérivées premières

2. Phase de production :

- ▶ Passer à Newton-Raphson pour la vitesse
- ▶ Vérifier numériquement les dérivées secondes

3. En cas de problème :

- ▶ Utiliser Levenberg-Marquardt
- ▶ Vérifier le point de départ
- ▶ Ajuster α et λ

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

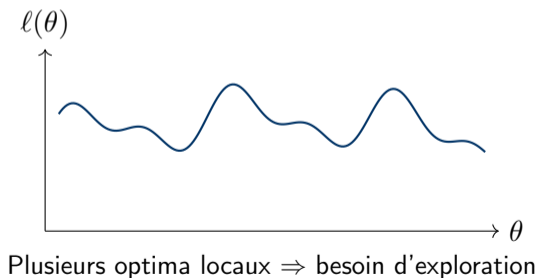
Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Optimisation globale – Motivation

- ▶ Les méthodes de gradient convergent vers un **optimum local** qui dépend du point de départ.
- ▶ Pour une fonction non concave, rien ne garantit que l'optimum local trouvé est le **maximum global**.
- ▶ Les méthodes d'optimisation globale explorent l'espace des paramètres de manière plus large, sans dépendre d'un seul point de départ.



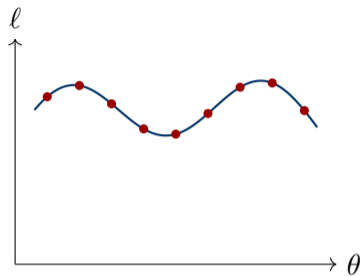
Recherche par grille (*grid search*)

- ▶ **Principe** : évaluer $\ell(\theta)$ en chaque point d'une grille régulière et retenir le meilleur :

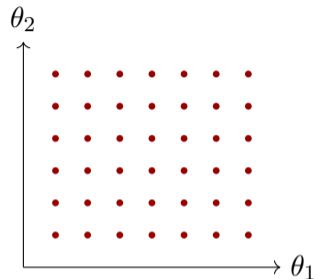
$$\hat{\theta} = \arg \max_{\theta \in \mathcal{G}} \ell(\theta), \quad \mathcal{G} = \{\theta_1, \theta_2, \dots, \theta_M\}$$

- ▶ **Construction de la grille** (en dimension k) : si chaque composante prend m valeurs, alors $M = m^k$ points.
- ▶ **Avantages** :
 - ▶ Extrêmement simple à implémenter
 - ▶ Garantit de trouver le maximum global sur \mathcal{G}
 - ▶ Ne nécessite aucune dérivée
- ▶ **Inconvénient majeur** : malédiction de la dimension (m^k croît exponentiellement).

Recherche par grille – Illustration



$$k = 1, m = 9, M = 9$$



$$k = 2, m = 7, M = 49$$

Dimension k	1	2	5	10	20
Points ($m = 10$)	10	100	10^5	10^{10}	10^{20}

Recherche aléatoire (*random search*)

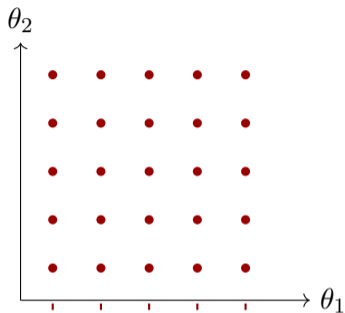
- ▶ **Principe** : tirer M points au hasard dans Θ et retenir le meilleur :

$$\hat{\theta} = \arg \max_{j=1, \dots, M} \ell(\theta_j), \quad \theta_j \stackrel{i.i.d.}{\sim} \pi(\theta)$$

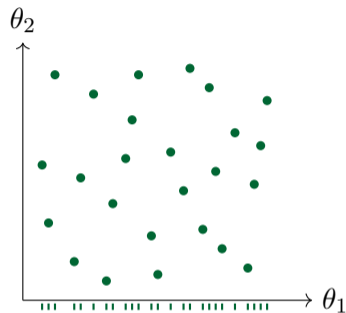
où π est une distribution sur l'espace des paramètres (uniforme, normale, etc.).

- ▶ **Avantages par rapport à la grille** :
 - ▶ Pas de structure rigide \Rightarrow explore mieux en grande dimension
 - ▶ Coût fixe en M (indépendant de k)
 - ▶ Si une seule composante de θ influence ℓ , la recherche aléatoire explore M valeurs de cette composante, contre $m = M^{1/k}$ pour la grille
- ▶ **Inconvénient** : pas de garantie de couverture uniforme.

Recherche aléatoire – Grille vs aléatoire



Grille : 5 valeurs par axe



Aléatoire : 25 valeurs par axe

En projetant sur θ_1 , la recherche aléatoire explore **25 valeurs distinctes** contre 5 pour la grille (avec le même budget de 25 évaluations).

Recherche aléatoire – Séquences quasi-Monte Carlo

- ▶ **Problème** : un tirage purement aléatoire peut laisser des « trous » dans l'espace de recherche.
- ▶ **Séquences à faible discrépance** (*quasi-Monte Carlo*) : suites **déterministes** qui remplissent l'hypercube $[0, 1]^k$ de manière plus uniforme qu'un tirage aléatoire.
- ▶ **Avantage** : pour un budget de M évaluations, la couverture de l'espace est garantie. L'erreur d'approximation décroît en $O((\log M)^k/M)$ au lieu de $O(1/\sqrt{M})$.
- ▶ **Exemples classiques** : suite de Halton, suite de Sobol', hypercube latin.

Séquences quasi-Monte Carlo – Suite de Halton

- ▶ **Idée** : écrire l'entier n en base b , puis **renverser ses chiffres** après la virgule.

- ▶ **Exemple en base $b = 2$** :

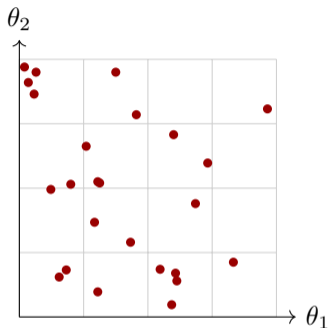
n	1	2	3	4	5	6	7	8
Écriture binaire	1	10	11	100	101	110	111	1000
Miroir	0,1	0,01	0,11	0,001	0,101	0,011	0,111	0,0001
$\phi_2(n)$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{8}$	$\frac{5}{8}$	$\frac{3}{8}$	$\frac{7}{8}$	$\frac{1}{16}$

- ▶ En dimension k : on utilise une base première **différente** par dimension ($b_1 = 2$, $b_2 = 3$, $b_3 = 5$, ...). Le point n est $(\phi_2(n), \phi_3(n), \phi_5(n), \dots)$.

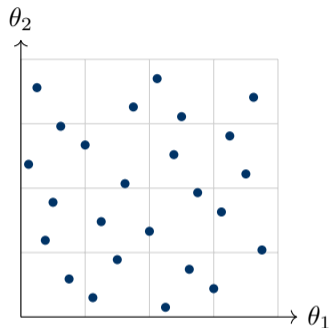
Séquences quasi-Monte Carlo – Suite de Sobol' et hypercube latin

- ▶ **Suite de Sobol'** : construit des suites récurrentes en **base 2** à l'aide d'opérations binaires (XOR). Chaque dimension utilise un polynôme générateur différent sur le corps $\mathbb{F}_2 = \{0, 1\}$, ce qui garantit l'indépendance entre dimensions.
- ▶ Les polynômes générateurs sont choisis **irréductibles** sur \mathbb{F}_2 (analogue des nombres premiers pour les polynômes). Des tables de ces polynômes sont disponibles dans les bibliothèques standard.
- ▶ La suite de Sobol' est la plus utilisée en pratique (notamment en finance et en optimisation).
- ▶ **Hypercube latin** (*Latin Hypercube Sampling*, LHS) : on découpe $[0, 1]$ en M strates de taille $1/M$ dans chaque dimension, et on place **exactement un point par strate**. Les strates sont appariées par une permutation aléatoire.

Séquences quasi-Monte Carlo – Illustration



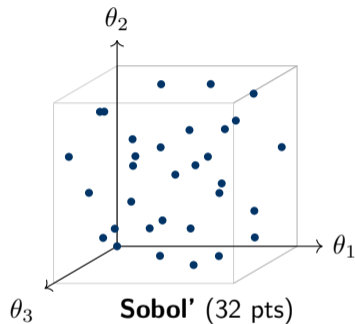
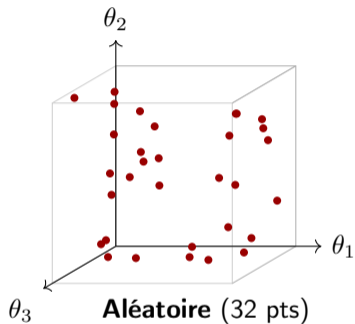
Aléatoire



Halton (bases 2, 3)

Avec 25 points, la suite de Halton couvre l'espace de manière beaucoup plus **régulière** que le tirage aléatoire, qui laisse des zones vides et des amas.

Séquences quasi-Monte Carlo – Sobol' en 3D



En dimension 3, la suite de Sobol' remplit le cube $[0, 1]^3$ de manière **quasi uniforme**, tandis que le tirage aléatoire produit des amas et des vides.

Recuit simulé (*simulated annealing*) – Principe

- ▶ **Analogie physique** : en métallurgie, un refroidissement lent permet aux atomes de trouver une configuration d'énergie minimale (cristal). Un refroidissement rapide fige les défauts (verre).
- ▶ **Idée** : accepter parfois des solutions **moins bonnes** pour échapper aux optima locaux. La probabilité d'accepter une dégradation **décroît** au fil du temps.
- ▶ **Paramètre clé** : la **température** T , qui contrôle l'exploration :
 - ▶ T élevé \Rightarrow forte exploration (on accepte souvent des dégradations)
 - ▶ T faible \Rightarrow exploitation locale (comportement glouton)
- ▶ T décroît selon un **schéma de refroidissement** : $T_0 > T_1 > T_2 > \dots > 0$

Recuit simulé – Algorithme

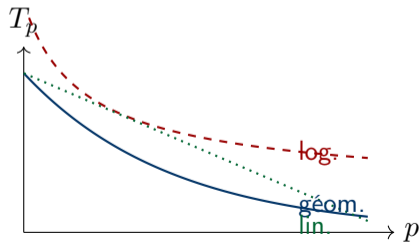
- ▶ À chaque itération p :
 1. Tirer un candidat $\theta' = \theta_{(p)} + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$
 2. Calculer $\Delta = \ell(\theta') - \ell(\theta_{(p)})$
 3. **Règle d'acceptation de Metropolis :**

$$\theta_{(p+1)} = \begin{cases} \theta' & \text{si } \Delta \geq 0 \text{ (amélioration : toujours acceptée)} \\ \theta' & \text{avec probabilité } e^{\Delta/T_p} \text{ si } \Delta < 0 \\ \theta_{(p)} & \text{sinon} \end{cases}$$

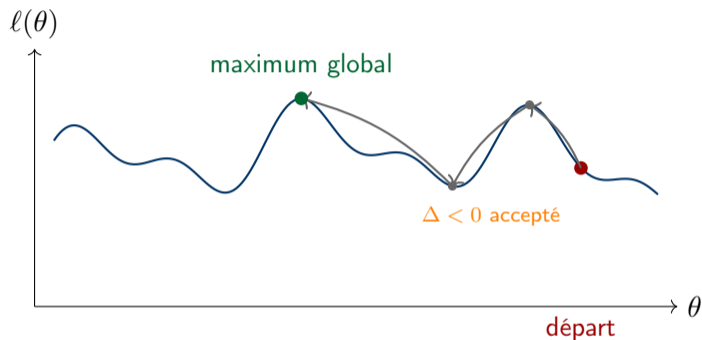
- ▶ La probabilité d'accepter une dégradation $\Delta < 0$ est e^{Δ/T_p} :
 - ▶ T_p grand : $e^{\Delta/T_p} \approx 1$ (on accepte presque tout)
 - ▶ T_p petit : $e^{\Delta/T_p} \approx 0$ (on n'accepte que les améliorations)

Recuit simulé – Schémas de refroidissement

- ▶ Schémas classiques :
 - ▶ **Géométrique** : $T_p = \alpha^p T_0$, $\alpha \in (0.9, 0.99)$
 - ▶ **Logarithmique** : $T_p = \frac{T_0}{\ln(1+p)}$ (convergence théorique garantie)
 - ▶ **Linéaire** : $T_p = T_0 \left(1 - \frac{p}{P}\right)$, $P =$ nombre total d'itérations
- ▶ En pratique, le schéma géométrique avec $\alpha \approx 0.95$ est le plus utilisé.



Recuit simulé – Illustration



L'acceptation occasionnelle de dégradations permet de **franchir les vallées** entre optima locaux et d'atteindre le maximum global.

Essaim de particules (*particle swarm optimization*)

- ▶ **Analogie** : comportement collectif d'un banc de poissons ou d'une volée d'oiseaux. Chaque individu ajuste sa trajectoire en fonction de sa propre expérience et de celle du groupe.
- ▶ **Population** : N_p particules, chacune ayant une position θ_i et une vitesse v_i .
- ▶ **Mémoires** :
 - ▶ θ_i^* : meilleure position visitée par la particule i (*personal best*)
 - ▶ θ_g^* : meilleure position trouvée par l'essaim entier (*global best*)
- ▶ **Initialisation** : les positions θ_i sont tirées aléatoirement dans l'espace de recherche. Les vitesses initiales v_i sont tirées dans $\left[-\frac{\theta_{\max} - \theta_{\min}}{2}, \frac{\theta_{\max} - \theta_{\min}}{2} \right]$ (ou mises à zéro).

Essaim de particules – Mise à jour

- ▶ **Vitesse** : la vitesse v_i détermine la direction et l'amplitude du déplacement de chaque particule. Elle combine trois composantes :
 - ▶ **Inertie** (ωv_i) : tendance à continuer dans la même direction
 - ▶ **Cognitive** ($c_1 r_1 (\theta_i^* - \theta_i)$) : attraction vers le meilleur point personnel
 - ▶ **Sociale** ($c_2 r_2 (\theta_g^* - \theta_i)$) : attraction vers le meilleur point du groupe
- ▶ **Règle de mise à jour** :

$$v_i \leftarrow \omega v_i + c_1 r_1 (\theta_i^* - \theta_i) + c_2 r_2 (\theta_g^* - \theta_i)$$

$$\theta_i \leftarrow \theta_i + v_i$$

où $r_1, r_2 \sim \mathcal{U}(0, 1)$ sont retirés à chaque itération et pour chaque particule.

Essaim de particules – Paramètres et propriétés

▶ Paramètres :

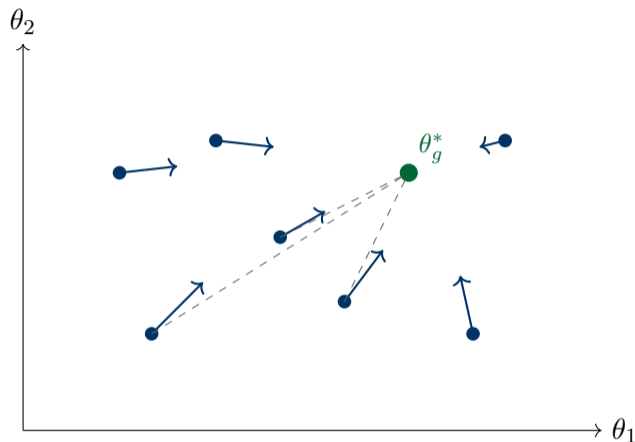
- ▶ $\omega \in (0, 1)$: inertie (mémoire de la direction précédente)
- ▶ c_1 : attraction vers le *personal best* (composante cognitive)
- ▶ c_2 : attraction vers le *global best* (composante sociale)
- ▶ $r_1, r_2 \sim \mathcal{U}(0, 1)$: aléa à chaque itération

▶ Valeurs classiques : $\omega = 0.7$, $c_1 = c_2 = 1.5$, $N_p = 20\text{--}50$.

▶ Avantages :

- ▶ Ne nécessite aucune dérivée
- ▶ Parallélisable (évaluations indépendantes)
- ▶ Efficace en dimension modérée

Essaim de particules – Illustration



Chaque particule (●) est attirée vers le meilleur global (●) tout en gardant une part d'inertie et d'exploration individuelle.

Méthodes globales – Récapitulatif

Méthode	Dérivées	Scalabilité	Garantie globale
Grille	Non	Très faible	Sur \mathcal{G}
Aléatoire	Non	Faible	Asymptotique
Recuit simulé	Non	Moyenne	Théorique (log.)
Essaim de particules	Non	Moyenne	Non

- ▶ Aucune de ces méthodes ne nécessite de dérivées.
- ▶ En pratique, on les utilise pour **trouver un bon point de départ** $\theta_{(0)}$, puis on affine avec une méthode de gradient (Newton-Raphson, BHHH, etc.).
- ▶ Le recuit simulé est le plus utilisé en économétrie pour les fonctions non concaves.

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

Comparaison et recommandations

Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Méthodologie – Vérifier la fonction objectif

▶ Méthodes :

- ▶ Calcul analytique : dériver $\ell(\theta)$ à la main
- ▶ Vérification croisée : comparer avec les références (articles, manuels)
- ▶ Cas particuliers : tester sur des exemples simples à solution connue

▶ Erreurs courantes :

- ▶ Oubli du signe moins devant certains termes
- ▶ Erreurs dans les constantes (facteur $\frac{1}{2}$, $\frac{N}{2}$, etc.)
- ▶ Confusion entre $\ln f$ et f

Méthodologie – Vérifier le gradient

- ▶ Comparer le gradient analytique $s(\theta)$ avec l'approximation numérique :

$$s_j^{\text{num}}(\theta) \approx \frac{\ell(\theta + he_j) - \ell(\theta - he_j)}{2h}$$

où e_j est le j -ème vecteur de base et $h \approx 10^{-5}$.

- ▶ Critère de vérification :

$$\frac{|s_j^{\text{ana}} - s_j^{\text{num}}|}{|s_j^{\text{ana}}| + |s_j^{\text{num}}| + \varepsilon} < \text{tol}$$

avec $\text{tol} \approx 10^{-4}$ à 10^{-6} .

- ▶ Cette étape permet de détecter les erreurs dans ℓ et dans s .

Méthodologie – Vérifier le hessien

- ▶ Calculer le hessien numériquement à **partir du gradient analytique** :

$$H_{jk}^{\text{num}}(\theta) \approx \frac{s_j(\theta + he_k) - s_j(\theta - he_k)}{2h}$$

- ▶ Calculer H numériquement à partir de ℓ (différences finies d'ordre 2) cumule les erreurs d'approximation \Rightarrow utiliser le gradient analytique vérifié comme base.
- ▶ On peut conserver les dérivées numériques si le calcul analytique est trop complexe.

Méthodologie – Paramétrer le programme

- ▶ **Objectif** : Éviter toute intervention manuelle après vérification \Rightarrow source d'erreurs.
- ▶ **Bonnes pratiques** :
 - ▶ Utiliser des paramètres pour les tolérances, λ , α
 - ▶ Créer des fonctions réutilisables (log-vraisemblance, gradient, hessien)
 - ▶ Documenter le code
 - ▶ Prévoir des messages d'erreur informatifs
 - ▶ Sauvegarder les résultats intermédiaires

Méthodologie – Checklist

Élément	Vérification
Log-vraisemblance ℓ	<input type="checkbox"/> Formule correcte <input type="checkbox"/> Cas particuliers testés
Gradient s	<input type="checkbox"/> Dérivation analytique <input type="checkbox"/> Vérification numérique
Hessien H	<input type="checkbox"/> Dérivation analytique <input type="checkbox"/> Vérification numérique <input type="checkbox"/> Définie négative à l'optimum
Convergence	<input type="checkbox"/> Critères d'arrêt atteints <input type="checkbox"/> Résultats stables (différents $\theta_{(0)}$)

Plan

Introduction et motivation

Structure générale d'un algorithme

Les méthodes de gradient

Algorithme de Newton-Raphson

Algorithme BHHH

Algorithme du Score

Algorithme de Levenberg-Marquardt

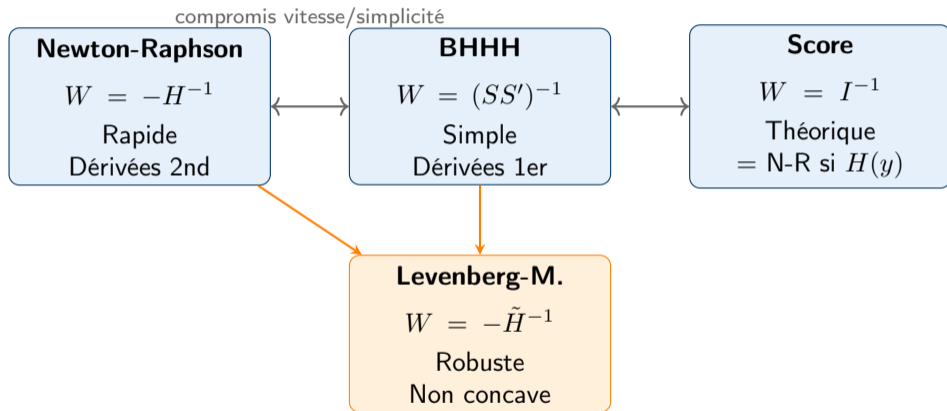
Comparaison et recommandations

Méthodes d'optimisation globale

Méthodologie de programmation

Récapitulatif

Récapitulatif – Synthèse



Récapitulatif – Formules essentielles

- Forme générale :

$$\theta_{(p+1)} = \theta_{(p)} + \lambda W_{(p)} s(\theta_{(p)})$$

Algorithme	Matrice $W_{(p)}$
Newton-Raphson	$-H(\theta_{(p)})^{-1}$
BHHH	$(\sum_i s_i s_i')^{-1}$
Score	$(\sum_i \mathbb{E}[s_i s_i'])^{-1}$
Levenberg-Marquardt	$-(H - (1 + \alpha)\mu_H I)^{-1}$

Récapitulatif – Points clés

1. **Concavité** : si ℓ est concave, tout algorithme croissant converge.
2. **Newton-Raphson** : rapide mais nécessite H .
3. **BHHH** : simple, utilise seulement le gradient.
4. **Levenberg-Marquardt** : robuste pour les cas non concaves.
5. **Vérification** : toujours contrôler numériquement les dérivées.
6. **CSO** : vérifier que $H(\hat{\theta}_N) \prec 0$ à la fin.

Références

- ▶ Gouriéroux, C. et Monfort, A. (1989). *Statistique et Modèles Économétriques*, Economica, ch. XIII
- ▶ Davidson, R. et MacKinnon, J. (2004). *Econometric Theory and Methods*, Oxford
- ▶ Greene, W. (2018). *Econometric Analysis*, 8th ed., Pearson
- ▶ Berndt, E., Hall, B., Hall, R. et Hausman, J. (1974). "Estimation and Inference in Nonlinear Structural Models", *Annals of Economic and Social Measurement*, 3(4), 653-665.