

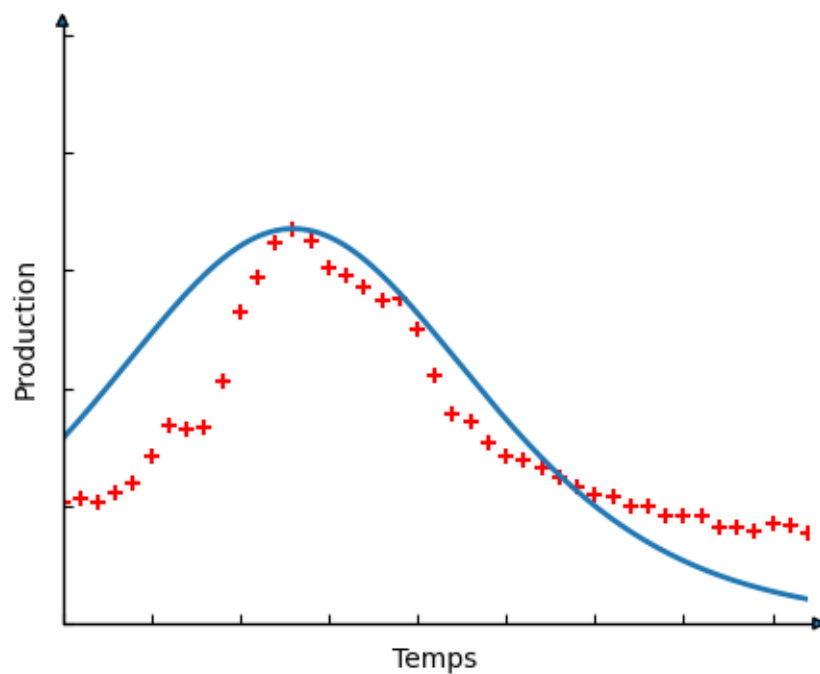
Projet : modélisation de pics de production de ressources

Yoan Thomas

Aya Ismahene Kroussa

Melvin Cerba

April 30, 2022



Contents

1	Éléments historiques	3
1.1	Modélisation des pics de production	3
1.2	Courbe de Hubbert et sigmoïde	3
2	Algorithme d'approximation	5
2.1	Caractérisation de l'algorithme	5
2.1.1	Gradient	5
2.1.2	Direction de descente	6
2.1.3	Recherche linéaire et implémentation python	7
2.1.4	Figure des isocourbes	8
2.1.5	Matrice de mise à l'échelle	9
2.2	Test de contrôle de l'algorithme et des fonctions associées	11
2.2.1	Vérification du gradient par différences finies	11
2.2.2	Test de convergence avec données bruitées et non bruitées, en commençant plus ou moins loin de la solution	12
2.3	Performance	14
2.3.1	Temps de convergence selon différent paramètres	14
3	Données réelles	15
3.1	Modélisation de la production pétrolière de la France	15
3.2	Application de l'algorithme aux données des pays de l'OCDE	15
3.2.1	Un modèle généralement pertinent...	16
3.2.2	...mais parfois inadapté	16
3.2.3	Des cas ambigus	17
3.2.4	Modélisation de la production mondiale	17
3.3	Prévisions à partir du modèle	18
3.3.1	Optimisation sur données incomplètes	18
3.3.2	Capacités prédictive du modèle sur des données de production pétrolière	19
3.4	Amélioration du modèle	20
4	Conclusion	21
4.1	Un algorithme performant	21
4.2	Un modèle imparfait	21
4.3	Conjectures	22
4.4	Remerciement	22

1 Éléments historiques

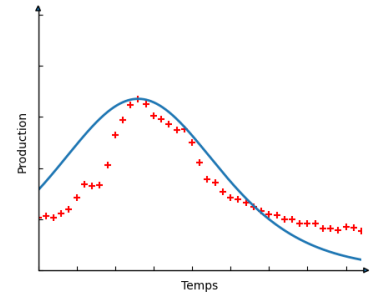
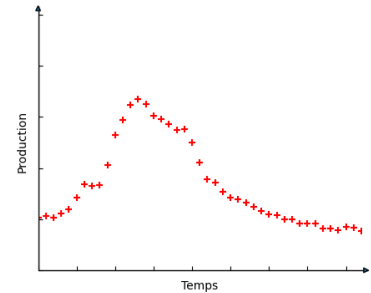
Peut-on prédire l'évolution de la production des ressources non renouvelables ? Pour tenter de répondre à cette question, nous proposons d'étudier à travers ce projet certains outils mathématiques qui permettent de modéliser la production des ressources non renouvelables. Plus spécifiquement, nous nous intéresserons à la production de pétrole.

1.1 Modélisation des pics de production

Être capable de prédire l'évolution de la production de diverses ressources a toujours été un enjeu économique majeur. Il est donc naturel que de nombreux modèles mathématiques promettant de modéliser cette évolution soient apparus. A travers ce projet, nous vous proposons d'étudier en détail l'un d'entre eux : la courbe de Hubbert [1].

En 1956, Marion King Hubbert présentait sa "courbe de Hubbert" à l'American Petroleum Institute. Son modèle, qui postule que la production croît, atteint un unique pic, puis décroît au même rythme qu'elle a augmenté en premier lieu, ne fit pas beaucoup parler de lui à l'époque. Mais lorsqu'en 1971, conformément à ses prédictions, la production pétrolière américaine atteignit son maximum et commença à décliner, ses travaux furent réexaminés avec beaucoup plus d'intérêt. Les chocs pétroliers de 1973 et 1979 semblèrent cependant définitivement invalider son modèle, qui perdit rapidement l'attention de l'industrie.

Malgré tout, l'avènement du calcul informatique et la grande disponibilité de données poussèrent des auteurs modernes à exhumer le modèle de Hubbert et à l'étendre, notamment en donnant une formule mathématique à sa courbe, permettant ainsi de calculer son intégrale. C'est sur la base de ces nouveaux travaux que nous avons construit notre projet.



1.2 Courbe de Hubbert et sigmoïde

Les auteurs qui se sont approprié la courbe de Hubbert ont notamment travaillé à lui donner une expression mathématique. Ceci leur a permis de définir son intégrale, que nous appellerons "fonction sigmoïde" tout au long de ce rapport. Pour des raisons que nous expliciterons plus tard, cette dernière s'avère plus facile à manier que la courbe de Hubbert.

:

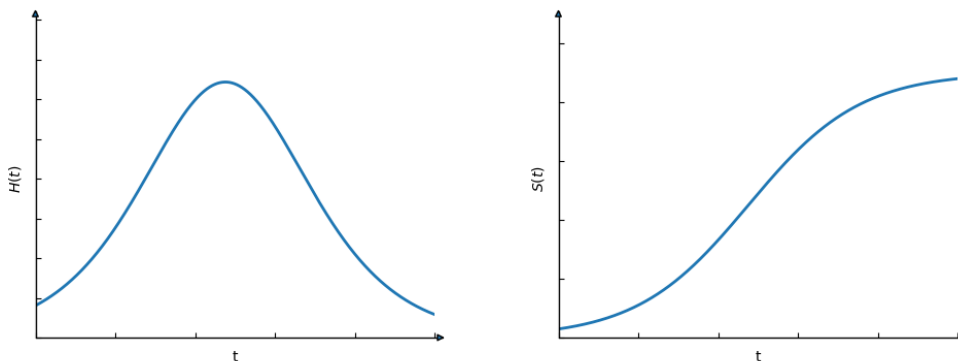
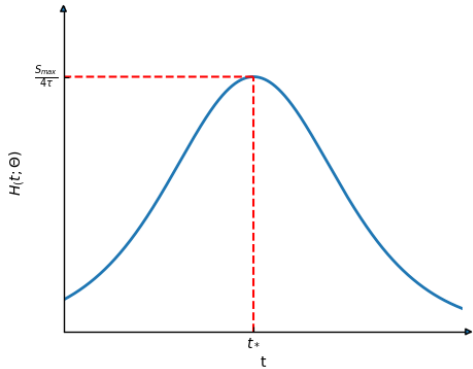


Figure 1: Une courbe de Hubbert et sa sigmoïde (son intégrale)

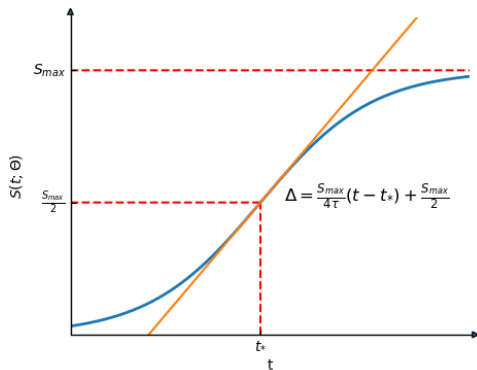
Une *courbe de Hubbert* [1] [2] est une courbe qui décrit de manière explicite l'évolution au cours du temps de la production d'une ressource donnée (par exemple, du pétrole). Cette fonction s'écrit

$$H(t; \theta) = \frac{S_{\max}}{\tau} \frac{e^{-\frac{t-t_\star}{\tau}}}{\left(1 + e^{-\frac{t-t_\star}{\tau}}\right)^2} \quad (1)$$


Dans l'expression ci-dessus, t représente la variable de temps et le triplet de paramètres $\theta := (S_{\max}, t_\star, \tau)$ permet de définir sans ambiguïté la courbe dans la famille paramétrée. Comme le montre l'illustration ci-dessus, cette courbe a la forme d'une “cloche” et les trois paramètres définissent son allure générale. Plus spécifiquement, on a

- S_{\max} contrôle l'amplitude maximale de la courbe ;
- t_\star repère la position de ce maximum sur l'axe des temps ;
- τ contrôle le caractère plus ou moins “piqué” de la cloche.

L'intégrale de cette fonction de Hubbert possède la forme explicite suivante :

$$S(t; \theta) = \frac{S_{\max}}{1 + e^{-\frac{t-t_\star}{\tau}}} \quad (2)$$


Cette fonction est largement connue sous le nom de *fonction sigmoïde*. Dans le cadre d'une interprétation de la consommation de ressources naturelles, la sigmoïde modélise l'évolution temporelle *cumulée* de ressources extraites du sol.

Rappelons notre objectif et nos hypothèses : nous voulons modéliser la production de pétrole au cours du temps, et nous supposons que celle-ci atteint un unique pic. Il nous faudra donc déterminer le temps t_\star pour lequel ce pic est atteint ainsi que la hauteur de ce pic (quantité de pétrole produit à l'instant t).

2 Algorithme d'approximation

Avant de rentrer dans le détail algorithmique, il nous faut cependant définir la fonction à minimiser, que nous appellerons également *critère*. Comme il est usuel en analyse de données, nous allons utiliser comme critère la norme usuelle des N -uplets réels (donc de \mathbb{R}^N) pour mesurer la distance qui sépare les données de la courbe issues du modèle. Plus précisément, nous additionnerons pour chaque année le carré de la différence entre la prédiction issue du modèle et les données réelles.

$$C(\theta) = \sum_{k=1}^N \delta t \times (S(t_k; \theta) - D_k)^2 \quad (3)$$

avec D_k la somme des données de production de l'instant t_1 à l'instant t_k . Comme la prédiction du modèle dépend de la valeur des paramètres $\theta := (S_{\max}, t_*, \tau)$ choisis, on comprend aisément que cette distance puisse varier en fonction de ces paramètres. L'estimation paramétrique à partir de données expérimentales consiste donc à rechercher les paramètres qui permettent de minimiser cette distance. Une optimisation "à la main" des paramètres de la sigmoïde, si elle est possible, est souvent longue et inefficace : nous allons donc nous tourner vers une méthode informatique de minimisation. Pour cela, nous choisissons une méthode locale dite à base de *direction de descente*. Ces méthodes standards, bien référencées dans la littérature [3, Chap. 3], sont brièvement décrites dans ce qui suit.

2.1 Caractérisation de l'algorithme

Dans cette partie, nous allons exposer en détails chaque partie de l'algorithme de la *descente de gradient*. Nous y avons apporté quelques modifications au cours du projet, mais il suit toujours ces principes :

Conditions à vérifier

- *Initialisation* : pour que l'algorithme "descende" vers le minimum local de la fonction, il faut l'initialiser dans un "creux" qui le contienne, c'est à dire un domaine sur lequel la fonction est *strictement convexe* vis à vis des paramètres d'intérêt.
- *Direction de descente* : par ailleurs, il faut être certain que l'algorithme sache en tout instant dans quelle direction aller (ie. quelles modifications apporter aux paramètres) pour "descendre" vers le minimum. Pour cela, nous utiliserons le gradient.

L'algorithme

1. Évalue la distance qui sépare la fonction sigmoïde des données grâce à une fonction critère.
2. Définit une direction de descente à partir du gradient de la fonction critère.
3. Modifie les paramètres initiaux selon la direction de descente un pas donné.
4. Recommence jusqu'à satisfaire des conditions prédéfinies qui assure une certaine proximité au minimum de la fonction.

Techniquement, il faut noter que le critère n'est pas strictement convexe sur la totalité de son domaine de définition. De ce point de vu, des minima locaux peuvent coexister avec des minima globaux, tous ces minima pouvant être isolés ou non. Comme le caractère "local" des algorithmes basés sur le gradient ne peut garantir la convergence vers un minimum global, nous avons supposé principalement dans ce travail que notre algorithme est systématiquement initialisé suffisamment "proche" d'un minimum isolé, ce minimum pouvant ou non être global.

2.1.1 Gradient

Afin de converger vers les paramètres optimaux, l'algorithme de la descente de gradient a besoin d'une direction de descente. Cela passe avant tout par le calcul du gradient de la fonction à minimiser, afin d'avoir une idée précise de l'évolution de cette dernière en fonction de chaque paramètre.

A partir de (3), on obtient :

$$\nabla C(\theta) = 2 \delta t \times \sum_{k=1}^N \nabla S(t_k; \theta) \times (S(t_k; \theta) - D_k) \quad (4)$$

Il reste donc à obtenir $\nabla S(\theta)$ qui se déduit facilement par le calcul

$$\nabla S(t; \theta) = \begin{bmatrix} \frac{\partial S}{\partial S_{\max}}(t, \theta) \\ \frac{\partial S}{\partial t_*}(t, \theta) \\ \frac{\partial S}{\partial \tau}(t, \theta) \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-\frac{(t-t_*)}{\tau}}} \\ \frac{-S_{\max}}{\tau} \times \frac{e^{-\frac{(t-t_*)}{\tau}}}{\left(1+e^{-\frac{(t-t_*)}{\tau}}\right)^2} \\ \frac{-S_{\max}}{\tau^2} \times (t-t_*) \times \frac{e^{-\frac{(t-t_*)}{\tau}}}{\left(1+e^{-\frac{(t-t_*)}{\tau}}\right)^2} \end{bmatrix}$$

Nous sommes donc capables de calculer le gradient de la sigmoïde pour des paramètres S_{\max} , t_* et τ donnés. Ce gradient est une pièce maîtresse de la construction d'une direction de descente pour la minimisation itérative du critère. C'est ce que nous voyons maintenant.

2.1.2 Direction de descente

La méthode de descente consiste à corriger les paramètres courants de $\theta^{(n)}$ en utilisant la "pente" locale de la fonction objective, qui est alors donnée par le gradient du critère calculé en $\theta^{(n)}$. Si le critère est suffisamment régulier (en l'occurrence, il doit être au moins continûment différentiable), il est en effet possible de construire une direction à partir du gradient et qui produise une décroissance locale stricte du critère [3, p. 36]. Partant d'une valeur initiale $\theta^{(0)}$ donnée, la mise à jour $n \rightarrow n+1$ des paramètres courants est défini par

$$\theta^{(n+1)} = \theta^{(n)} + \alpha^{(n)} d^{(n)}$$

avec $d^{(n)}$ la direction de correction courante et $\alpha^{(n)} > 0$ la longueur du déplacement le long de cette direction. Pour la classe des algorithmes de descente, les directions sont dites "gradient reliées" et elles s'écrivent

$$d^{(n)} := -B^{(n)} \nabla C(\theta^{(n)})$$

avec $B^{(n)}$ une matrice symétrique dont les valeurs propres sont positives et uniformément bornées au dessus de zéro, quel que soit $n \in \mathbb{N}$. De ce point de vu, $B^{(n)}$ pris identique à l'identité est un choix admissible, et conduit à l'algorithme dit de *plus profonde descente*.

$$d^{(n)} := -\nabla C(\theta^{(n)})$$

Nous verrons que ce choix peut s'avérer très inefficace en pratique, pour des raisons que nous expliquerons. On notera également que les conditions sur $B^{(n)}$ permettent à la direction courante de vérifier

$$\langle d^{(n)}, \nabla C(\theta^{(n)}) \rangle = -\langle B^{(n)} \nabla C(\theta^{(n)}), \nabla C(\theta^{(n)}) \rangle < 0$$

ce qui indique qu'elle fait un angle obtus (> 90 degrés) avec la direction du gradient, qui lui indique la direction de plus forte croissance locale. Cette condition assure donc que $d^{(n)}$ permet de faire décroître *localement* le critère.

Voici ce à quoi ressemble notre algorithme d'optimisation pour l'instant :

Algorithm 1: Algorithme de descente de gradient

Data: $D, \theta^{(0)}, \epsilon, N_{\text{iter}}, \delta t$
Result: $\theta^{(\text{final})}$
initialization;
while $\|\nabla C_D(t; \theta^{(n)})\| > \epsilon$ **and** $n < N_{\text{iter}}$ **do**
 $d^{(n)} = -I_3 \nabla C_D(t; \theta^{(n)})$;
 $\theta^{(n)} = \theta^{(n-1)} + \delta t * d^{(n)}$;
 $n = n + 1$;
end

Pour obtenir un algorithme globalement convergent, il reste enfin à "régler" la longueur $\alpha^{(n)}$ du déplacement dans la direction courante. Cette étape porte le nom de *recherche linéaire*.

2.1.3 Recherche linéaire et implémentation python

Un point délicat que l'on doit gérer en pratique est que la direction $\mathbf{d}^{(n)}$, bien que décroissante localement, n'est pas forcément bien "mise à l'échelle". En d'autres termes, si on examine la fonction

$$q(\alpha) := C(\boldsymbol{\theta}^{(n)} + \alpha \times \mathbf{d}^{(n)})$$

le choix $\alpha = 1$ peut très bien produire faire remonter le critère au lieu de le faire décroître. De manière générale, on est donc amené à utiliser une règle de *sélection de pas* qui garantisse une décroissance stricte à chaque itération et qui évite de brider la vitesse de convergence. La *règle d'Armijo* est probablement la règle de sélection de pas qui est la plus répandue, car très simple à programmer. Cette règle utilise un paramètre $0 < m < 1$ fixé initialement et elle recherche une valeur α qui vérifie la condition :

$$q(\alpha) \leq q(0) + m\alpha q'(0).$$

La recherche d'un $\alpha > 0$ qui vérifie cette condition utilise une technique dite de "backtracking" [3, p.41] :

- (i) on fixe $\alpha = \bar{\alpha}$, avec $\bar{\alpha}$ une valeur fixée initialement,
- (ii) on test l'inégalité ci-dessus pour voir si la condition est vérifiée,
- (iii) le pas est accepté si l'inégalité est vraie
- (iv) dans le cas contraire, on réduit la taille du pas testé en posant $\alpha = \alpha \times \beta$ avec $0 < \beta < 1$, puis on reprend les opérations à partir de l'étape (ii).

L'implémentation en python de l'algorithme de descente complet est simple. En partant de $\boldsymbol{\theta}^{(0)} = [S_{\max}^{(0)}, t_*^{(0)}, \tau^{(0)}]$, les mises à jour successives réduisent progressivement le critère. Un test d'arrêt permet d'arrêter les itérations si la norme du gradient descend en dessous d'un certain seuil fixé au préalable ; à défaut, si ce critère d'arrêt n'est pas atteint, l'algorithme est stoppé par un nombre maximum d'itérations que nous avons fixé à 1000.

Voici ce à quoi ressemble notre algorithme maintenant qu'il implémente le critère d'Armijo :

Algorithm 2: Algorithme de descente de gradient avec critère d'Armijo

Data: $D, \boldsymbol{\theta}^{(0)}, \epsilon, N_{\text{iter}}, \bar{\alpha}, \beta, m$
Result: $\boldsymbol{\theta}^{(\text{final})}$
initialization;
while $\|\nabla C_D(t; \boldsymbol{\theta}^{(n)})\| > \epsilon * \|\nabla C_D(t; \boldsymbol{\theta}^{(0)})\|$ **and** $n < N_{\text{iter}}$ **do**
 $\mathbf{d}^{(n)} = -I_3 \nabla C_D(t; \boldsymbol{\theta}^{(n)})$;
 $\alpha^{(n)} = \bar{\alpha}$;
 $\boldsymbol{\theta}^{(n)} = \boldsymbol{\theta}^{(n-1)} + \alpha^{(n)} * \mathbf{d}^{(n)}$;
 $\text{deltaF}^{(n)} = \nabla C_D(t; \boldsymbol{\theta}^{(n-1)}) - \nabla C_D(t; \boldsymbol{\theta}^{(n)})$;
 while $\text{deltaF}^{(n)} \cdot \alpha^{(n)} * m * \langle \nabla C_D(t; \boldsymbol{\theta}^{(n)}), \mathbf{d}^{(n)} \rangle$ **do**
 $\alpha^{(n)} = m * \alpha^{(n)}$;
 $\boldsymbol{\theta}^{(n+1)} = \boldsymbol{\theta}^{(n)} + \alpha^{(n)} * \mathbf{d}^{(n)}$;
 end
 $n = n + 1$;
end

Pour tester cet algorithme, nous avons généré des données à partir d'une courbe sigmoïde de paramètres $\boldsymbol{\theta}_{\text{gen}}$, et initialisé $\boldsymbol{\theta}^{(0)}$ à $0,8 * \boldsymbol{\theta}_{\text{gen}}$. Voici ce que l'on obtient lorsqu'on lance notre algorithme d'optimisation avec ces paramètres :

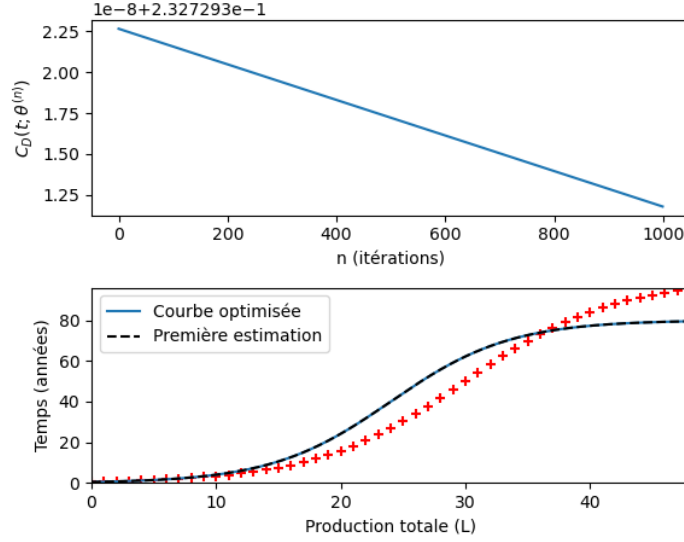


Figure 2: Résultat de l'optimisation de la sigmoïde par descente de gradient avec critère de Armijo (sur des données générées à partir du modèle et des paramètres de départ égaux à 80% des paramètres de la courbe optimale) : la courbe obtenue et la courbe de départ sont quasiment superposées.

De fait, lorsque la matrice identité est utilisée pour $\mathbf{B}^{(n)}$ (c.à.d. lorsque notre algorithme correspond à la technique dite de la plus profonde descente), nous remarquons que notre algorithme converge très lentement et que le nombre maximum d'itération est systématiquement atteint. Cet effet est le signe que nous avons un problème de mise à l'échelle. L'algorithme qui utilise uniquement la direction du gradient comme direction de descente est lent et il vaut mieux l'éviter, d'autant plus qu'il existe d'autres directions simples à calculer et conduisant à des algorithmes beaucoup plus efficaces.

2.1.4 Figure des isocourbes

Pour comprendre la raison qui fait que notre algorithme ne converge pas, nous allons fixer individuellement chaque paramètre, faire varier les deux autres, et tracer les *isocourbes* de la fonction critère. Une isocourbe est constituée de tous les points en lesquels la fonction critère est égale à une même constante.

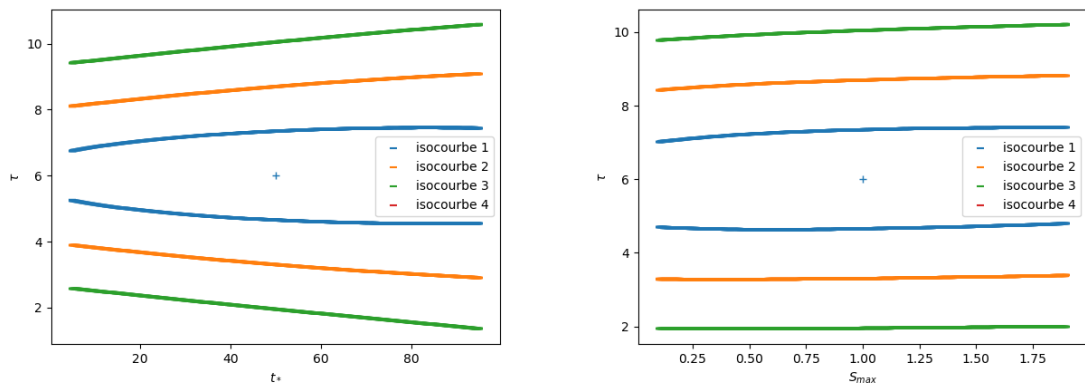


Figure 3: Quatre isocourbes de $C_D(S(t; \theta))$ avec respectivement S_{max} et t_* .

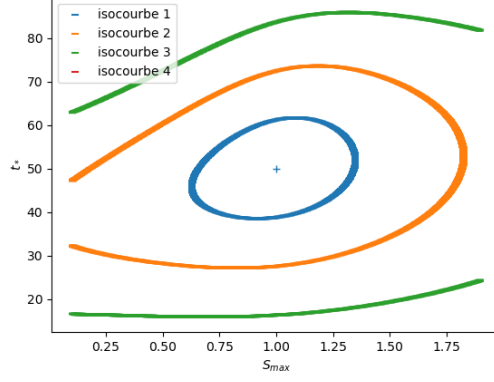


Figure 4: Quatre isocourbes de $C_D(S(t; \theta))$ avec τ fixé.

On constate que les trois paramètres n'ont pas du tout la même "échelle", c'est à dire qu'une même variation aura un effet important, faible ou quasiment nul selon si elle est apportée respectivement à τ , t_* ou S_{\max} . Visuellement, il faut imaginer que le gradient de chaque point d'un isocourbe est perpendiculaire à celle-ci : on comprend alors que notre direction de descente (l'opposé du gradient) ne pointe pas vers le centre. Ceci induit la nécessité d'une "mise à l'échelle" de la direction de descente, grâce à laquelle notre algorithme suivra la direction de descente la plus directe.

2.1.5 Matrice de mise à l'échelle

On suppose ici que la fonction *critère* à minimiser est deux fois différentiable en θ , et on désigne par $\nabla^2 C(t; \theta)$ son hessien en θ .

La direction est définie en un point θ en lequel le hessien de la fonction *critère* est inversible par :

$$d^{(n)} = B^{(n)} \nabla C(t; \theta^{(n)}) \quad (5)$$

Avec :

- $B^{(n)} = -\nabla^2 C(t; \theta^{(n)})^{-1}$ la matrice de mise à l'échelle
- $\theta^{(n+1)} = \theta^{(n)} + \alpha^{(n)} d^{(n)}$, $\alpha^{(n)}$ le pas déterminé par le critère d'Armijo.

$d^{(n)}$ constitue une direction de descente si:

1. $\nabla C(t; \theta) \neq 0$,
2. $B^{(n)} = -\nabla^2 C(t; \theta)^{-1}$ est définie positive.

En effet,

$$\nabla C(t; \theta) \cdot d = \langle \nabla C(t; \theta), d \rangle = -\langle \nabla C(t; \theta), \nabla^2 C(t; \theta)^{-1} \nabla C(t; \theta) \rangle \leq -\lambda_{\max}^{-1} \|\nabla C(t; \theta)\|^2 < 0 \quad (6)$$

où λ_{\max} désigne la plus grande valeur propre de $\nabla^2 C(t; \theta)$.

Calculer le hessien du critère est coûteux, on choisit alors cette approximation du hessien [3, Chap. 10.2], avec $J(t; \theta)$ le jacobien du critère :

$$\nabla^2 C(t; \theta) \approx J(t; \theta)^T J(t; \theta)$$

Voici donc l'algorithme final, implémentant le critère d'Armijo et la mise à l'échelle de la direction de descente :

Algorithm 3: Algorithme de descente de gradient avec mise à l'échelle et critère d'Armijo

Data: $D, \theta^{(0)}, \epsilon, N_{\text{iter}}, \bar{\alpha}, \beta, m$
Result: $\theta^{(\text{final})}$
initialization;
while $\|\nabla C_D(t; \theta^{(n)})\| > \epsilon * \|\nabla C_D(t; \theta^{(0)})\|$ **and** $n < N_{\text{iter}}$ **do**
 $d^{(n)} = -B^{(n)} \nabla C_D(t; \theta^{(n)})$;
 $\alpha^{(n)} = \bar{\alpha}$;
 $\theta^{(n)} = \theta^{(n-1)} + \alpha^{(n)} * d^{(n)}$;
 $\text{delta}F^{(n)} = \nabla C_D(t; \theta^{(n-1)}) - \nabla C_D(t; \theta^{(n)})$;
 while $\text{delta}F^{(n)} \cdot \alpha^{(n)} > m * \langle \nabla C_D(t; \theta^{(n)}), d^{(n)} \rangle$ **do**
 $\alpha^{(n)} = m * \alpha^{(n)}$;
 $\theta^{(n+1)} = \theta^{(n)} + \alpha^{(n)} * d^{(n)}$;
 end
 $n = n + 1$;
end

Procédons maintenant au même test que sur la versions antérieure de l'algorithme : lançons l'optimisation sur des données parfaites et avec des arguments initiaux égaux à 80% des arguments de génération.

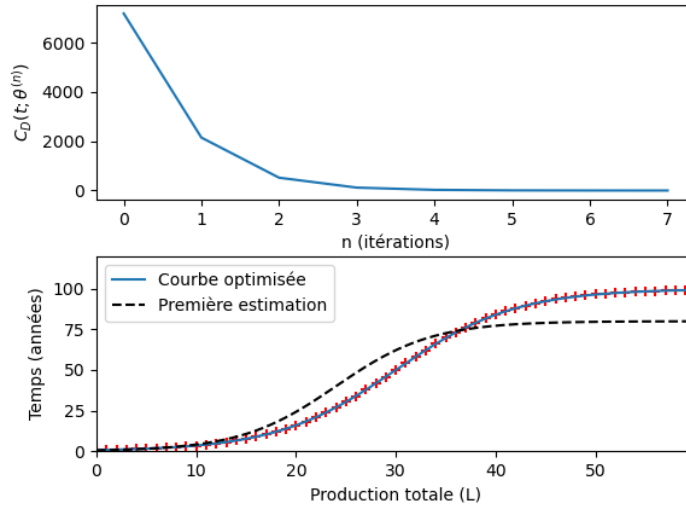


Figure 5: Résultat de l'optimisation de la sigmoïde par descente de gradient avec critère de Armijo et mise à l'échelle de la direction de descente (sur des données générées à partir du modèle et des paramètres de départ égaux à 80% des paramètres de courbe optimale)

Le résultat est sans équivoque, puisqu'on parvient à une courbe dont le critère est quasiment nul en 7 itérations seulement. C'est donc cet algorithme que nous utiliserons pour la suite de ce projet.

2.2 Test de contrôle de l'algorithme et des fonctions associées

Avant de lancer l'algorithme sur des données réelles, on vérifie que nos fonctions soient justes et que l'algorithme converge pour des données simulées.

2.2.1 Vérification du gradient par différences finies

En analyse numérique, la *méthode des différences finies* [4] est une technique courante de recherche de solutions approchées d'équations aux dérivées partielles qui consiste à résoudre un système de relations (schéma numérique) liant les valeurs des fonctions inconnues en certains points suffisamment proches les uns des autres.

Cette méthode apparaît comme étant la plus simple à mettre en œuvre car elle procède en deux étapes : d'une part la discrétisation par *différences finies* des opérateurs de dérivation/différentiation, d'autre part la convergence du schéma numérique ainsi obtenu lorsque la distance entre les points diminue.

Définition : Soit $N \in \mathbb{N}^*$, on appelle *discrétisation régulière* de $[a, b]$ à N pas ou $N + 1$ l'ensemble des points $a + nh$, $n \in [0, N]$ où le pas h est donné par $h = (b - a)/N$.

Soient $\{t^n\}_{n \in [0, N]}$ une discrétisation régulière de $[a, b]$ et $(Dy)_n$ une approximation de $y'(t^n)$.

On appelle :

- *Différence finie progressive* l'approximation :

$$(Dy)_n^P = \frac{y(t^{n+1}) - y(t^n)}{h}, \quad \forall n \in [0, N - 1] \quad (7)$$

- *Différence finie rétrograde* l'approximation :

$$(Dy)_n^R = \frac{y(t^n) - y(t^{n-1})}{h}, \quad \forall n \in [1, N] \quad (8)$$

- *Différence finie centrée* l'approximation :

$$(Dy)_n^C = \frac{y(t^{n+1}) - y(t^{n-1})}{2h}, \quad \forall n \in [1, N - 1] \quad (9)$$

Et dans notre cas on vas travailler avec l'approximation *progressive* c'est à dire $y'(t) = \lim_{h \rightarrow 0} \frac{y(t+h) - y(t)}{h}$. On appliquant cette formule à notre fonction sigmoïde, on aura :

$$\begin{aligned} \frac{\partial S(t, \theta)}{\partial S_{\max}} &= \frac{S(t; S_{\max} + \text{delta}, t_s, \tau) - S(t; S_{\max}, t_s, \tau)}{\text{delta}} \\ \frac{\partial S(t, \theta)}{\partial t_s} &= \frac{S(t; S_{\max}, t_s + \text{delta}, \tau) - S(t; S_{\max}, t_s, \tau)}{\text{delta}} \\ \frac{\partial S(t, \theta)}{\partial S_{\max}} &= \frac{S(t; S_{\max}, t_s, \tau + \text{delta}) - S(t; S_{\max}, t_s, \tau)}{\text{delta}} \end{aligned}$$

Avec $\text{delta} = (t_f - t_i)/N$ qui représente le pas de notre approximation.

Pour vérifier si notre calcul du gradient est bon, on fait un simple calcul qui est la différence en valeur absolue du gradient calculé dans la partie précédente et du gradient trouvé par la méthode des différences finies.

C'est à dire :

$$\text{diff} = |\nabla S(t, \theta) - (\frac{\partial S(t, \theta)}{\partial S_{\max}}, \frac{\partial S(t, \theta)}{\partial t_s}, \frac{\partial S(t, \theta)}{\partial S_{\max}})|$$

On trouve bien que diff est un petit nombre qui converge vers 0.

Alors, on a bien fait la vérification de notre gradient par la méthode de différences finies de la fonction *sigmoïde*.

Et de la même manière on vérifie aussi le gradient de la fonction *Critère*.

2.2.2 Test de convergence avec données bruitées et non bruitées, en commençant plus ou moins loin de la solution

Tentons désormais de définir le cadre au sein duquel notre algorithme fonctionne. Pour ce faire, nous allons tout d'abord devoir générer des données "parfaites", c'est à dire qui correspondent exactement à une courbe sigmoïde, dont nous aurons choisi les paramètres S_{\max}^{init} , t_*^{init} et τ^{init} .

Ensuite, nous allons tester les performances de notre algorithme en jouant sur deux paramètres :

- La *distance* à la solution des paramètres d'initialisation. Comme nous générons les données à partir d'une courbe sigmoïde que nous définissons, nous avons connaissance des paramètres optimaux Θ_{opti} . Nous pouvons donc définir les paramètres d'initialisation Θ_{init} de l'algorithme en fonction de ceux-ci, et donc décider de la "distance" qui sépare l'état initial de l'état optimal.

$$\Theta^{(\text{init})} = (1 + a) * \Theta^{(\text{opti})} \quad (10)$$

Nous ferons donc varier le paramètre a .

- Le *bruit*. Nous allons "salir" nos données en leur ajoutant un bruit gaussien, c'est à dire une perturbation égale à une variable aléatoire qui suit la loi de probabilités normale centrée en 0 :

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \quad (11)$$

Nous jouerons ici sur le paramètre σ .

Cherchons donc la distance "limite" : celle à partir de laquelle l'algorithme ne converge plus, même avec des données non-bruitées.

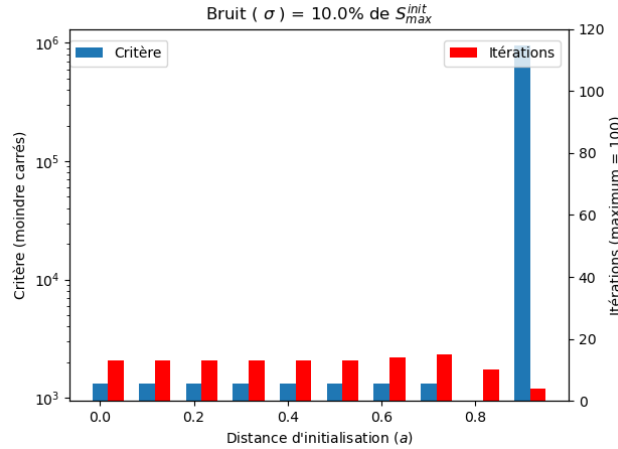


Figure 6: Nombre d'itérations et critère obtenu par l'algorithme d'optimisation pour différentes valeurs de a et $\sigma = 0.1 * S_{\max}^{(\text{init})}$.

On remarque donc qu'avec des données faiblement bruitées, l'algorithme est très performant, même lorsqu'il est initialisé assez loin de la solution.

Cherchons maintenant le niveau de bruit "limite" : la valeur de σ au delà de laquelle notre algorithme ne converge plus, même avec des paramètres initiaux très proches de la solution ($\Theta_{\text{init}} = 1.1 * \theta_{\text{opti}}$). Nous fixerons les valeurs de σ en fonction de S_{\max}^{init} .

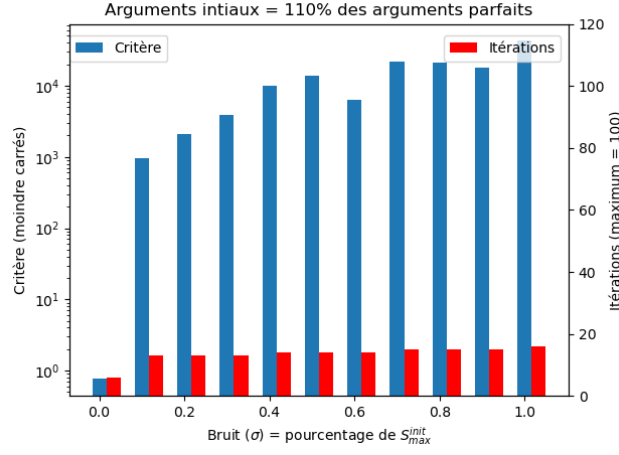


Figure 7: Nombre d'itérations et critère obtenu par l'algorithme d'optimisation pour différentes valeurs de σ (définies en pourcentage de $S_{\max}^{(init)}$) avec $a = 0.1$.

On constate qu'avec des paramètres initiaux très proches de la solution, l'algorithme a des résultats inconstants à partir d'un bruit égal à $0.4 * \theta_{opti}$ (on attire votre attention sur le fait que l'échelle est logarithmique).

2.3 Performance

Testons maintenant les performances de notre algorithme de manière plus rigoureuse.

2.3.1 Temps de convergence selon différents paramètres

Pour chaque valeurs de σ et de a , nous allons lancer l'algorithme d'optimisation 100 fois et calculer les moyennes du temps d'exécution et de la valeur du critère final (l'adéquation au données du modèle obtenu).

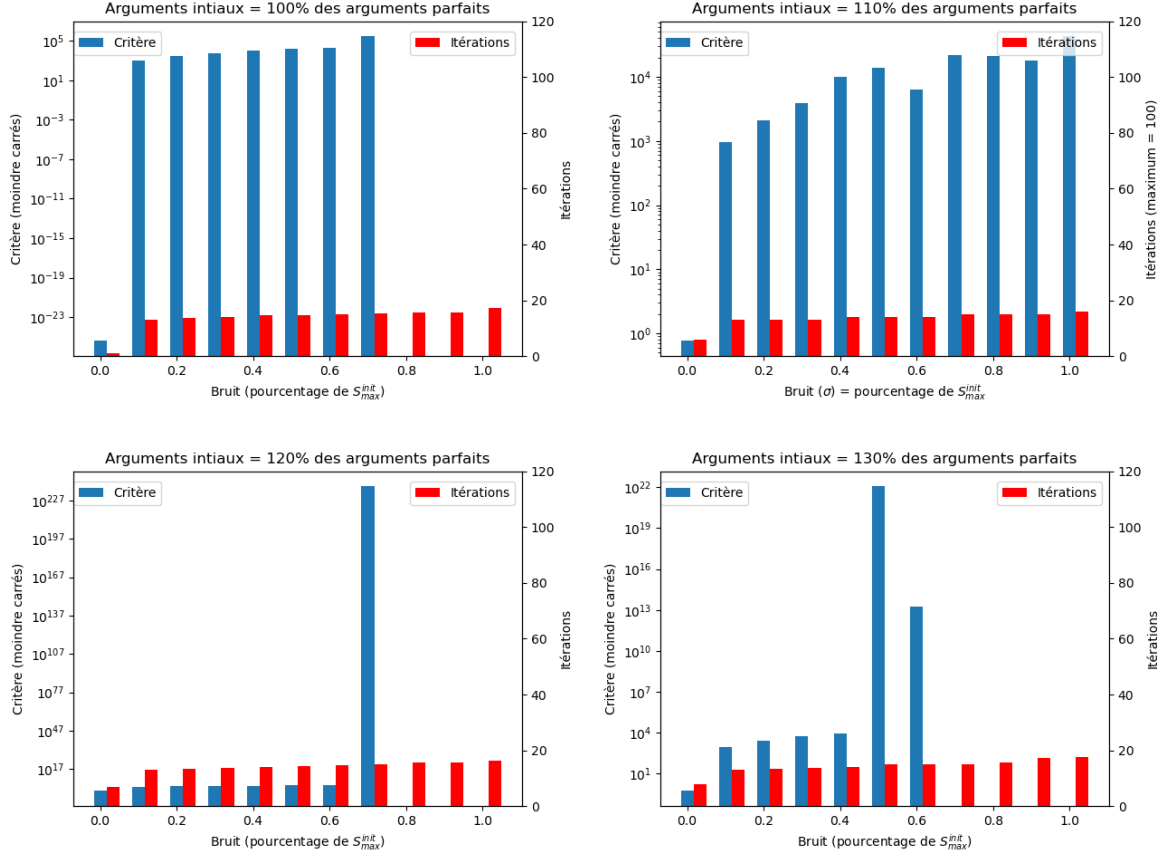


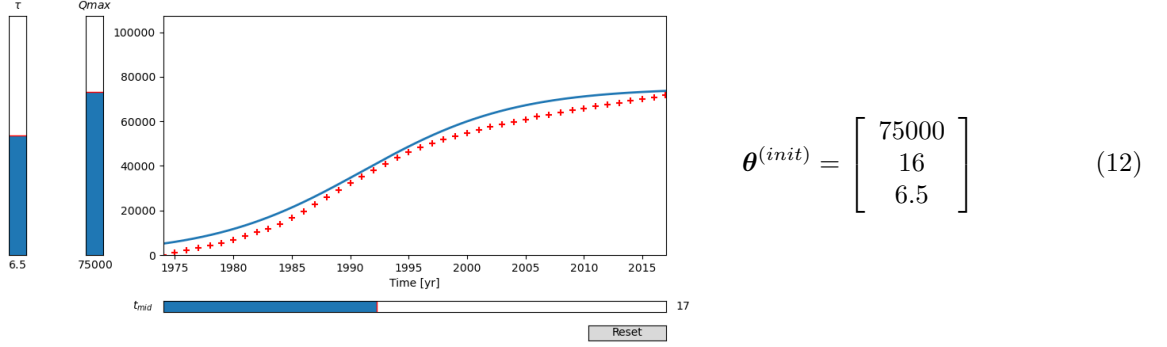
Figure 8: Temps d'exécution et critère final moyens selon différents niveau de bruitage des données initiales pour des arguments initiaux égaux à 100%, 110%, 120% et 130% des arguments de génération. (un critère nul signifie que l'algorithme n'a pas abouti)

3 Données réelles

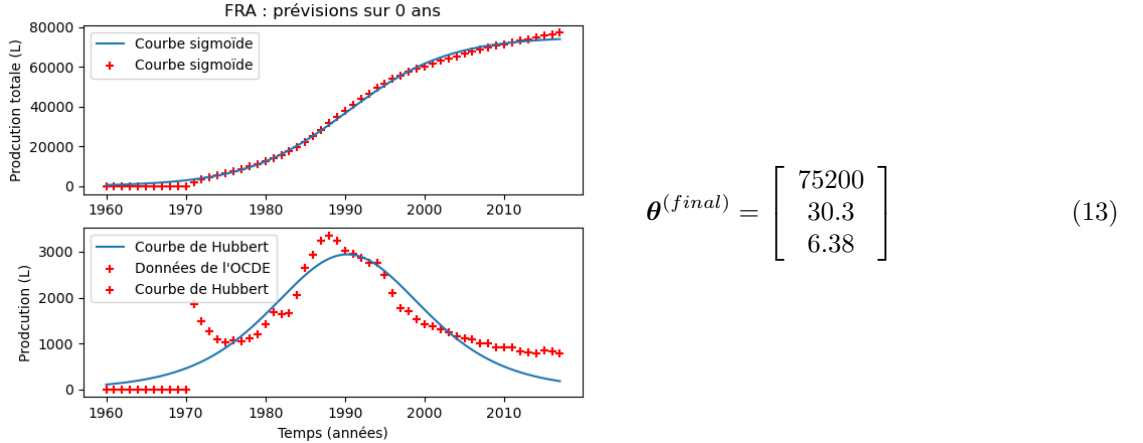
Maintenant que nous savons que notre algorithme fonctionne, voyons si le modèle qu'il génère est pertinent lorsqu'il s'agit de représenter des données réelles. Pour cela, nous utiliserons les données de production pétrolière que l'OCDE met à disposition de chacun sur son site web.

3.1 Modélisation de la production pétrolière de la France

Essayons par exemple d'appliquer notre algorithme aux données de production pétrolière de la France. En jouant manuellement sur les paramètres, on trouve une première approximation de la sigmoïde optimale :



Nous allons donc lancer l'algorithme avec ces paramètres initiaux. Voici le modèle que l'on obtient :



Le résultat est probant : il colle mieux aux données que notre première estimation, et reconnaît avec une marge d'erreur assez faible le pic de production ainsi que la quantité totale de pétrole produite. A priori, l'algorithme est assez efficace et ses résultats sont pertinents. Mais d'une part nous avons dû l'initialiser manuellement, et d'autre part, la production pétrolière de certains pays ne suit pas aussi clairement une courbe en cloche que celle de la France.

3.2 Application de l'algorithme aux données des pays de l'OCDE

Essayons maintenant d'automatiser notre démarche pour la généraliser à l'ensemble des pays dont les données de production pétrolière sont présentes sur le site de l'OCDE.

Dans le cas de la France, nous avons approximé manuellement les des paramètres initiaux. Ici, afin d'automatiser le processus, nous allons définir Θ^{init} comme suit :

$$\Theta^{(init)} = \begin{bmatrix} S_{\max}^{(init)} = \max \{D_n\} \\ t_*^{(init)} = \frac{N}{2} \\ \tau^{(init)} = 6,5 \end{bmatrix} \quad (14)$$

Avec D_n la valeur des données de production cumulée à l'instant t_n et N le nombre de données.

3.2.1 Un modèle généralement pertinent...

La plupart des cas sont semblables à celui de la Norvège : la courbe obtenue colle relativement bien aux données, la production totale et l'année du pic sont estimées avec une marge d'erreur assez faible.

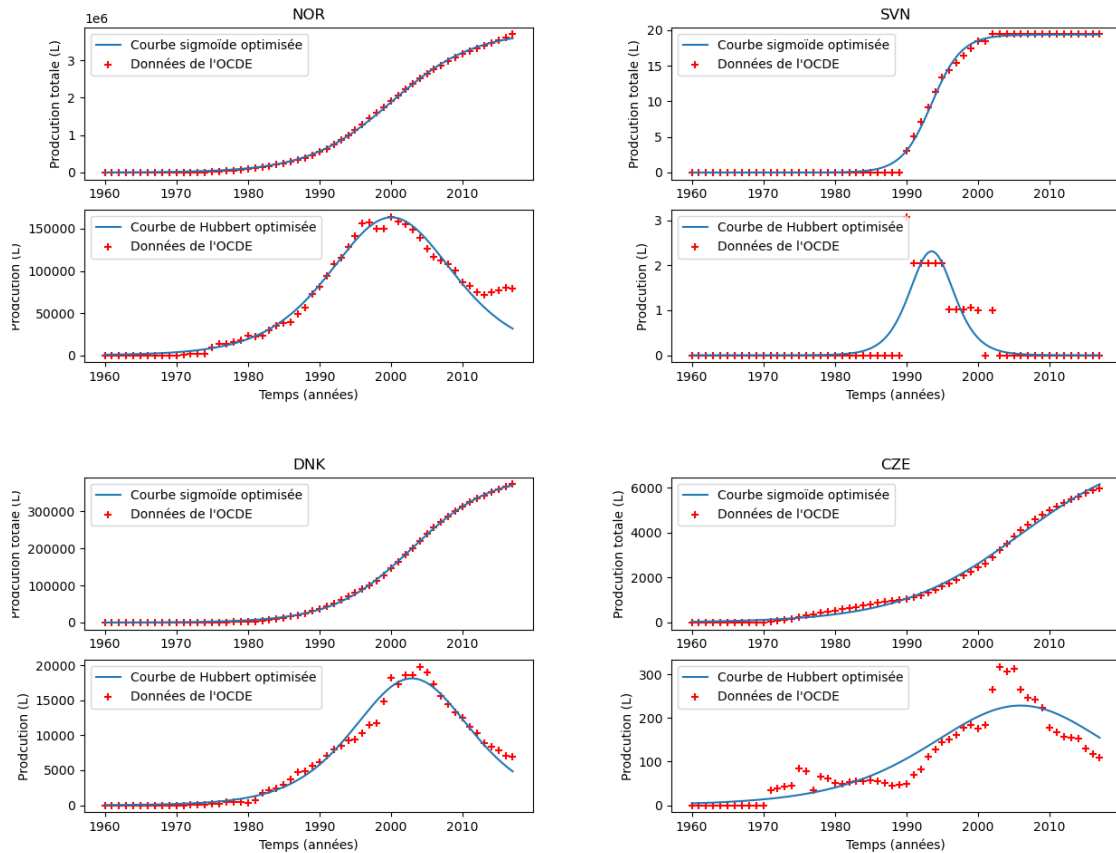


Figure 9: Productions pétrolières de la Norvège, de la Slovaquie, du Danemark et de la République tchèque.

3.2.2 ...mais parfois inadapté

Dans des pays comme le Mexique, où la production est relativement constante, le modèle s'avère sans surprise complètement inadapté.

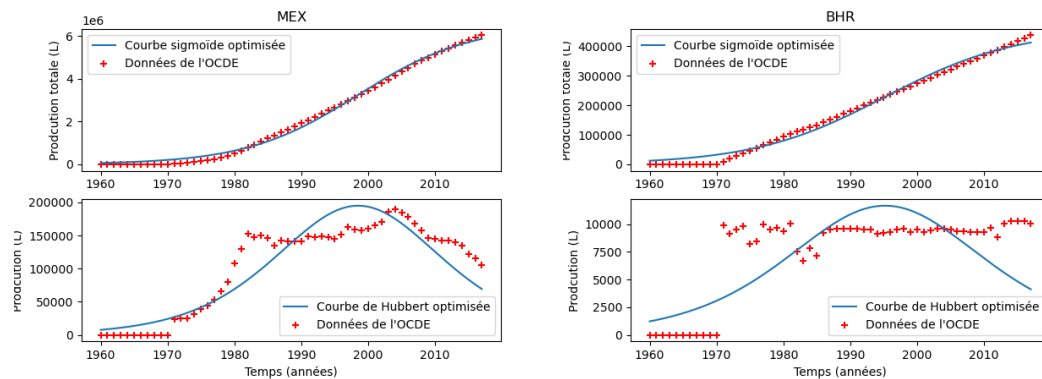


Figure 10: Productions pétrolières du Mexique et de Bahreïn.

3.2.3 Des cas ambigus

Pour certains pays, comme l'Albanie ou l'Australie, le modèle colle vaguement aux données, mais il apparaît assez clairement qu'il est limité par ses hypothèses : il y a eu dans ces pays plus d'un pic de production. Néanmoins, si l'on regarde chaque pic indépendamment, il ne semble pas déraisonnable de penser qu'il pourrait être modélisé par une courbe de Hubbert. Pourrait-on modifier notre modèle pour lui permettre de représenter des cas de ce genre ? Nous étudierons la question par la suite.

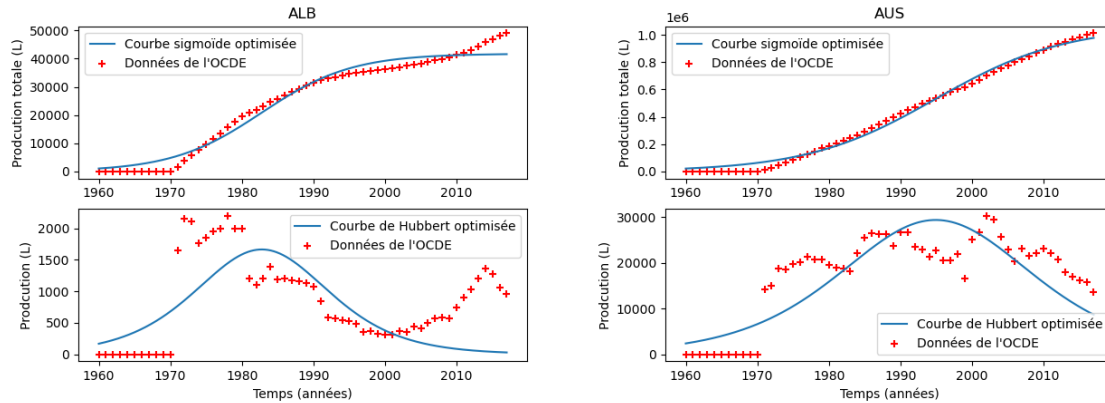


Figure 11: Production pétrolière de l'Albanie et de l'Australie.

3.2.4 Modélisation de la production mondiale

Qu'en est-il de la production cumulée de tous les pays de l'OCDE ? La courbe de Hubbert permet-elle d'estimer la quantité de pétrole que l'humanité entière pourrait exploiter ? Il apparaît que non : la production mondiale ne semble pas vérifier l'hypothèse du pic unique et symétrique.

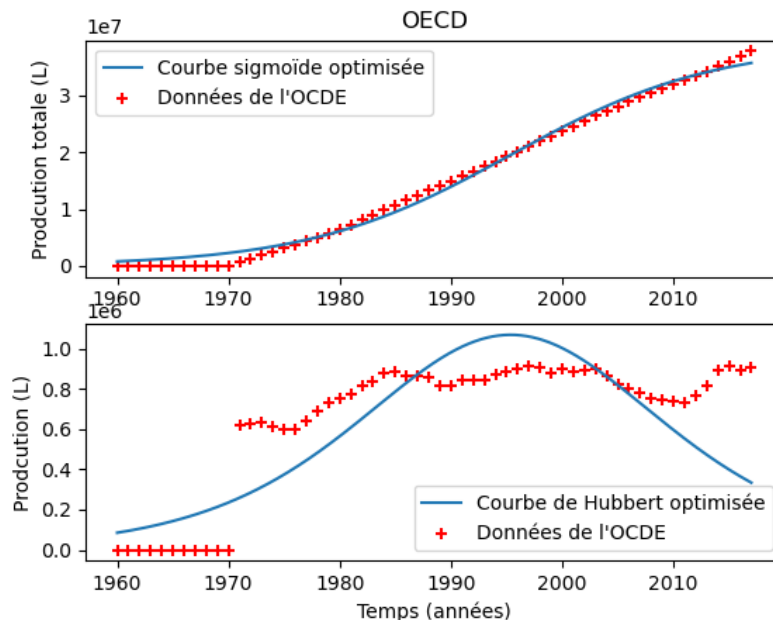


Figure 12: Production pétrolière cumulée des 55 pays de l'OCDE.

3.3 Prévisions à partir du modèle

Nous avons constaté que, lorsque certaines hypothèses sont vérifiées, le modèle de Hubbert colle relativement bien aux données réelles. Cependant, modéliser des données à posteriori n'a que peu d'intérêt. Notre algorithme est-il capable de plus ? Le modèle obtenu permet-il de prédire l'évolution de la production dans une fenêtre qui dépasse celle des données sur lesquelles il a été optimisé ?

3.3.1 Optimisation sur données incomplètes

Pour évaluer la capacité de notre algorithme à prévoir l'évolution de la production, nous allons le lancer sur des versions écourtées des données, et comparer les prévisions du modèle obtenu aux données réelles. Voici par exemple ce que cela donne avec des données générées avec et sans bruit :

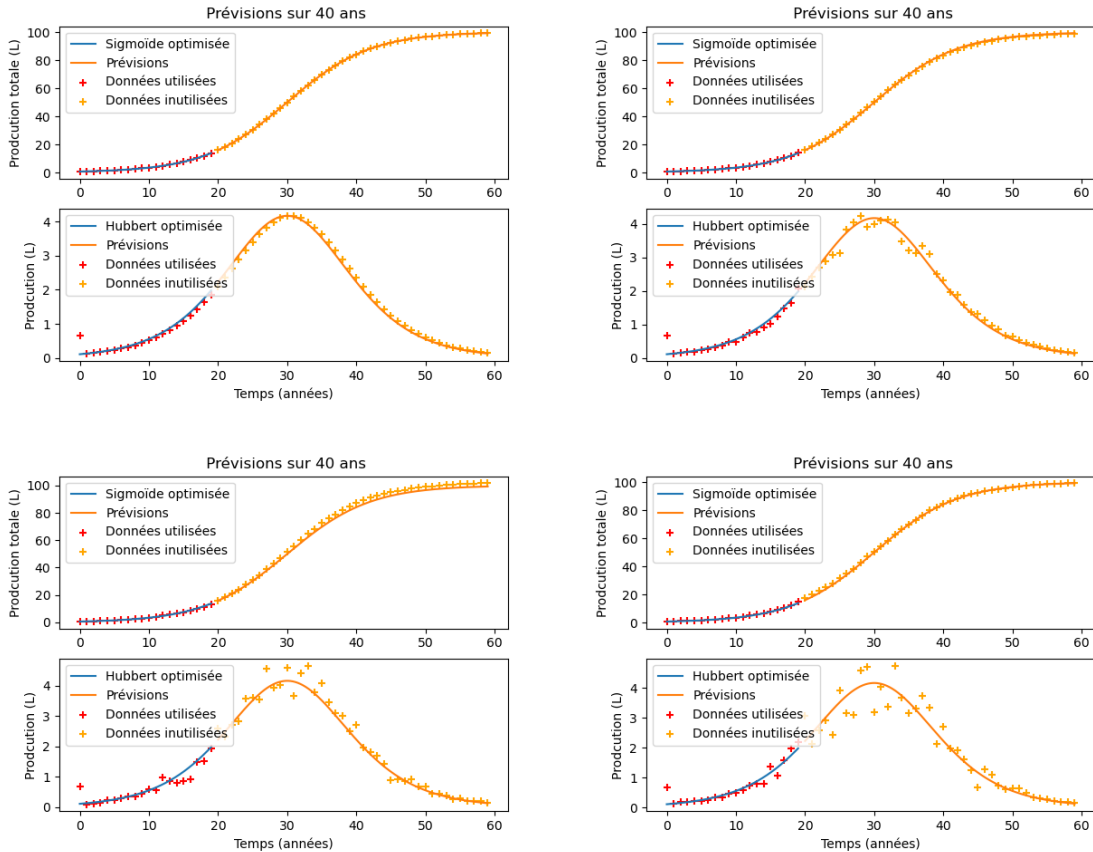


Figure 13: Prévisions sur 40 ans sur des données générées avec et sans bruit.

Les résultats semblent probants : avec seulement un tiers des données, notre algorithme est capable de prédire assez fidèlement l'allure de la courbe. Bien sûr, il s'agit ici de données générées à partir du modèle, alors ces résultats ne permettent pas de tirer de conclusions fiables sur les capacités prédictives de notre modèle sur des données réelles.

3.3.2 Capacités prédictive du modèle sur des données de production pétrolière

Pour estimer les capacités prédictives de notre algorithme sur des données réelles, appliquons la même méthode aux données de production pétrolière des pays de l'OCDE : voyons si des modèles obtenus à partir de données écourtées collent à la réalité. (Nous nous limiterons bien entendu aux pays pour lesquels nous obtenions des résultats cohérents lorsque nous lançons l'algorithme sur les données complètes.)

Dans le cas de certains pays, on constate des résultats encourageants. On remarque cependant que l'algorithme ne converge pas en deçà d'une certaine quantité de données. Un bon exemple de ce type de résultat est le Danemark :

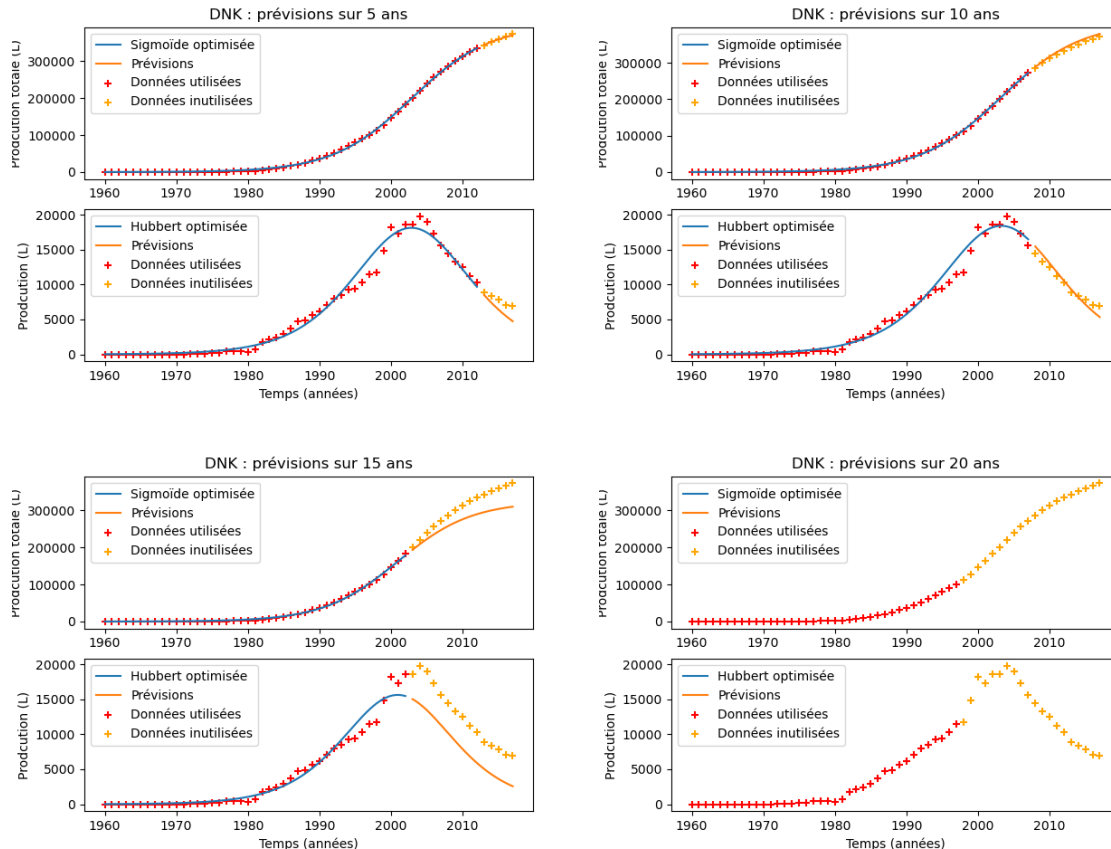


Figure 14: Prévisions de la production pétrolière du Danemark sur 5, 10, 15 et 20 ans. (ici, l'algorithme n'a pas été capable de produire un modèle en se basant uniquement sur des données antérieures au pic de production)

Lorsque les données sont un peu plus éloignées des hypothèses du modèle, la prédiction est quasiment impossible. En témoignent les cas de la Slovénie ou de la République Tchèque, par exemple :

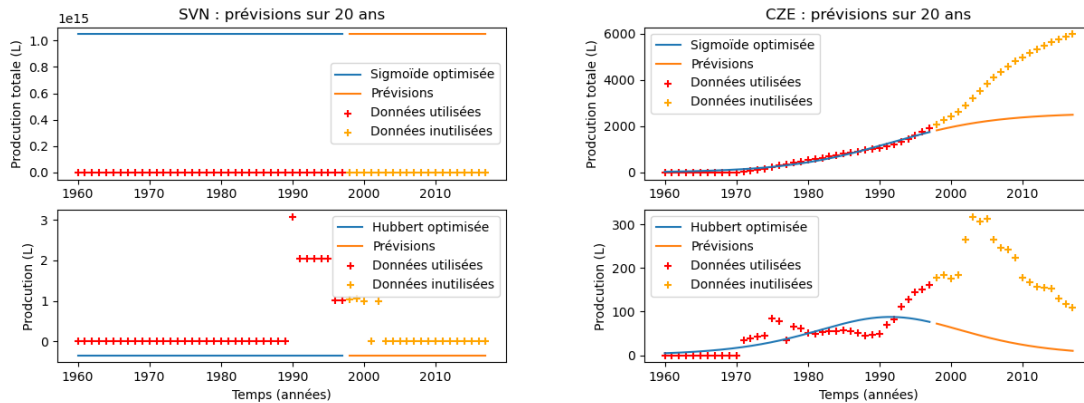


Figure 15: Modèle prévoyant la production pétrolière de la Slovénie et de la République Tchèque sur 20 ans.

3.4 Amélioration du modèle

Le modèle de Hubbert nous donne parfois des résultats satisfaisants : lorsqu'un pays ne connaît qu'un seul pic de production pétrolière, la courbe sigmoïde optimisée colle généralement bien aux données. Mais cette hypothèse est trop contraignante. Nous avons donc réfléchi à une façon de l'étendre, de sorte à ce que le modèle de Hubbert soit applicable à des cas plus variés.

Les cas ambigus

Rappelons que lorsque nous modélisons la production pétrolière des pays de l'OCDE, nous avons identifié des cas "ambigus" : dans certains pays, plusieurs pics de production se succédaient au cours des années.

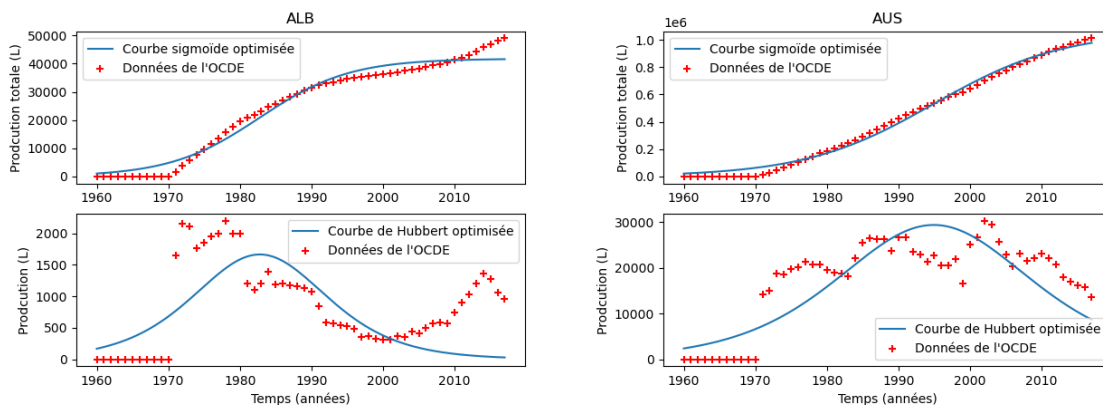


Figure 16: Les productions pétrolières de l'Albanie et de l'Australie ont atteint respectivement 2 et 4 pics au cours des 60 dernières années.

Vers un modèle à n sigmoïdes

Notre idée consiste alors à ne pas limiter notre modèle à un pic : pourquoi ne pas cumuler plusieurs sigmoïde ? Pourrait-on ainsi décomposer la production pétrolière d'un pays en différentes courbes de Hubbert successives ?

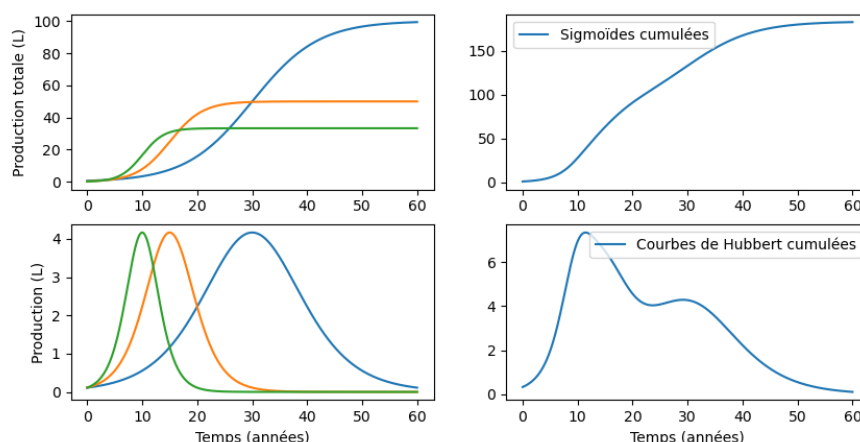


Figure 17: Voici ce à quoi pourrait ressembler un modèle à trois sigmoïdes.

Nous n'avons cependant pas eu le temps d'approfondir cette idée.

4 Conclusion

Nous vous proposons dans cette partie de récapituler, de manière très concise, les conclusions auxquelles nous a mené la réalisation de ce projet.

4.1 Un algorithme performant

La partie la plus concluante de ce projet est sans aucun doute l'algorithme d'optimisation auquel nous sommes parvenus à force d'améliorations successives. La descente de gradient basique, que nous utilisions initialement, semblait ne pas converger vers la solution tant elle était lente. Mais l'implémentation de la règle d'Armijo et, surtout, de la matrice de mise à l'échelle nous ont permis d'arriver à un algorithme qui non seulement converge, mais le fait dans la majorité des cas en une dizaine d'itérations seulement.

Cet algorithme a donc été l'origine d'une découverte intéressante et d'une réussite incontestable. L'objet de son optimisation, le modèle de Hubbert, a donné des résultats plus nuancés, en revanche.

4.2 Un modèle imparfait

Comme nous l'avons vu, la pertinence du modèle de Hubbert est inconstante. Si dans la plupart des cas il colle assez bien à la réalité, les exemples qui le tiennent en échec ne manquent pas. Cela vient du fait qu'il repose sur les hypothèses assez contraignantes que nous avons définies dans l'introduction :

- La production de ressources premières croît, atteint un unique pic, puis décroît.
- Son évolution est symétrique par rapport à cet unique pic

L'expérience nous montre que ces hypothèses sont rarement totalement vérifiées : certains pays, comme l'Albanie ou l'Australie, découvrent un nouveau gisement après avoir commencé, et potentiellement terminé, d'en exploiter un premier. D'autres, comme le Mexique, n'entrent pas dans le schéma industriel classique : plutôt que de suivre une courbe en cloche, la production atteint un seuil à partir duquel elle stagne.

Ceci fait que le modèle de Hubbert n'est pas toujours pertinent pour décrire la production pétrolière, et n'est en aucun cas un outil prédiction fiable sur le long terme.

4.3 Conjectures

Selon nos observations, il est possible qu'un modèle étendu donne de meilleurs résultats : dans des pays comme l'Albanie ou l'Australie, qui ont découvert plusieurs gisements de pétrole au fil des années, nous avons formulé l'hypothèse que la production pétrolière pourrait être décomposée en plusieurs courbes de Hubbert différentes.

Nous n'avons cependant pas eu le temps d'expérimenter cette hypothèse.

4.4 Remerciement

Nous aimerions remercier nos deux tuteurs, Marc Allain et Julien Lefevre, pour leur aide et leur bienveillance, en dépit de nos diverses difficultés.

References

- [1] M. K. Hubbert, “Nuclear energy and fossil fuel,” *Drilling and production practice*, 1956.
- [2] “Pic de hubbert, wikipedia.” https://fr.wikipedia.org/wiki/Pic_de_Hubbert. Accessed : 2022 – 04 – 30.
- [3] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, 2e ed., 2006.
- [4] G. Allaire, *Analyse numérique et optimisation*. ECOLE POLYTECHNIQUE, french ed., 2012.