

COMPTE RENDU

Pénalisation : Implémentation et Application

Souheib Ben Mabrouk

INDP2E

16 octobre 2021

Introduction

L'optimisation de fonctions vectorielles, dite multicritères, vectorielle ou multiobjectif, a été, depuis les premiers résultats intéressants publiés par Kuhn et Tucker en 1951 concernant les conditions d'optimalité des fonctions vectorielles fonctions, l'objet de recherches qui ont occupé un grand nombre d'auteurs. [1]

Depuis lors, les travaux dans cette direction ont fait des progrès remarquables sur le plan théorique et pratique. Son utilité est d'une grande importance dans les domaines de décision où le choix d'une meilleure solution est essentiel.

Les difficultés d'approche du problème sont dues à la forme implicite de l'ensemble efficace (ou faiblement efficace) et au fait que même dans le cas linéaire, **problème n'est pas convexe**. Pour trouver une solution optimale au problème, nous utilisons **la méthode de pénalité** qui, sous une autre forme, a plus récemment fourni les techniques de viscosité intensivement utilisées dans divers domaines.

Le principe de cette méthode, consiste à réduire le problème à la résolution d'une série de problèmes appelés problèmes pénalisés, dont les contraintes sont simples. Les théorèmes consacrés à la convergence de ces méthodes, montrent que sous certaines hypothèses, toute valeur d'adhérence de la séquence de solutions optimales des problèmes pénalisés, est une solution optimale du problème.

L'objectif de ce rapport est de présenter la méthode dans le cadre du cours de l'optimisation ainsi que implémenter quelques cas d'étude tout en tirant des conclusions.

Pour ce faire, j'ai choisi de travailler avec **Python 3.8** comme langage de programmation. D'autre part, j'utiliserai le système de composition de documents **LATEX** afin d'avoir un rapport bien conçu avec les formulations mathématiques nécessaires.

Table des matières

Introduction	1
1 Principe	4
1.1 Présentation	4
1.2 Modélisation Mathématique de la méthode	5
1.3 Pénalisation exacte / inexacte	6
1.4 Pénalisation extérieure	7
1.4.1 Présentation	7
1.4.2 Pénalisation quadratique	7
1.5 Pénalisation intérieure	8
1.5.1 Principe	8
1.5.2 Pénalisation logarithmique	9
2 Implémentation	10
2.1 Présentation du problème	10
2.2 Implémentation de l'algorithme en Python	11
3 Creuser d'avantage	13
3.1 Problème	13
3.2 Pénalisations non-bornées	13
3.3 Difficultés de solution des problèmes pénalisés	13
Conclusion	15

Table des figures

1.1	Exemple d'une pénalisation	5
1.2	Mise en oeuvre de la méthode de pénalisation	6
1.3	Exemple d'une pénalisation extérieure	7
1.4	Exemple d'une pénalisation quadratique	8
1.5	Exemple d'une pénalisation intérieure	8
1.6	Exemple d'une pénalisation logarithmique	9
2.1	Résultat final obtenu	12

Chapitre 1

Principe

1.1 Présentation

La pénalisation est un concept simple qui permet de transformer un problème d'optimisation avec contraintes en un problème ou en une suite de problèmes d'optimisation sans contrainte ou avec des contraintes simples. C'est un concept qui a une utilité à la fois théorique et numérique.^[2]

En analyse, l'approche par pénalisation est parfois utilisée pour étudier un problème d'optimisation dont certaines contraintes sont difficiles à prendre en compte, alors que le problème pénalisant ces contraintes difficiles a des propriétés (l'existence de solution par exemple) mieux comprises ou plus simples à mettre en évidence. Si l'on a de la chance ou si la pénalisation est bien choisie, des passages à la limite parfois délicats permettent **d'obtenir des propriétés du problème original**.

Par exemple, on peut obtenir des conditions d'optimalité d'un problème avec contraintes, à partir des conditions d'optimalité des problèmes pénalisés. D'autre part, comme nous allons le souligner ci-dessous, la pénalisation est un outil permettant d'étudier les problèmes d'optimisation avec et sans contrainte dans un même formalisme.

D'un **point de vue numérique**, cette transformation en problèmes sans contrainte (ou avec contraintes simples) permet **d'utiliser des algorithmes d'optimisation sans contrainte** (ou avec ces contraintes simples) pour obtenir la solution de problèmes dont l'ensemble admissible peut avoir une structure complexe.

Cela semble merveilleux, inespéré, de voir que l'on puisse ainsi utiliser des algorithmes qui ne cherchent qu'à minimiser une fonction pour trouver des points qui, en plus d'être optimaux, sont admissibles. Cette approche est de ce fait très souvent utilisée. Elle permet d'obtenir une solution de qualité suffisante rapidement sans avoir à entrer dans l'algorithmique sophistiquée de l'optimisation avec contraintes.

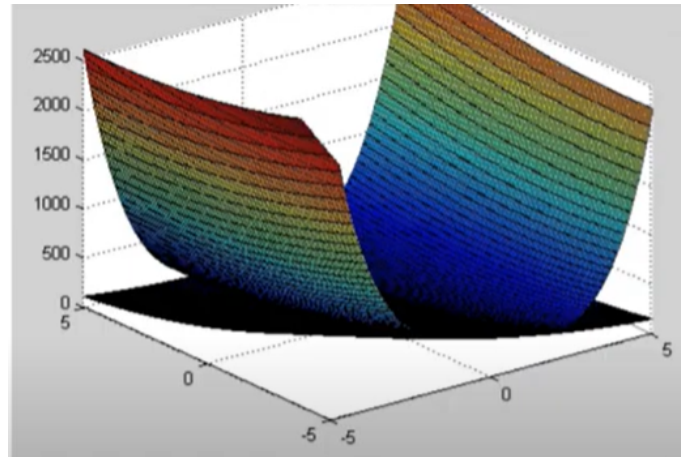


Fig. 1.1 — Exemple d'une pénalisation

1.2 Modélisation Mathématique de la méthode

Le principe de la pénalisation consiste à remplacer le problème sous contraintes :

$$(P) \quad \min_{x \in X} f(x) \quad (1.1)$$

Par un problème (ou une suite de problèmes) de la forme :

$$(P_c) \quad \min_{x \in X} f(x) + c\pi(x) \quad (1.2)$$

où $c > 0$ est appelé paramètre de pénalisation et $\pi : R^n \rightarrow R$ la fonction de pénalisation à choisir (souvent choisie à valeurs positives). Plus le paramètre c est grand, plus on accorde de l'importance à la pénalisation.

L'intérêt du problème **(P_c)** est de pouvoir être résolu par une méthode de l'optimisation sans contrainte.

La question qui se pose alors est si résoudre le problème **(P_c)** permettra de résoudre le problème **(P)**. Ou autrement dit, si les ensembles de solutions des deux problèmes coïncident. Cela dépendra bien évidemment du choix de c et π . [3]

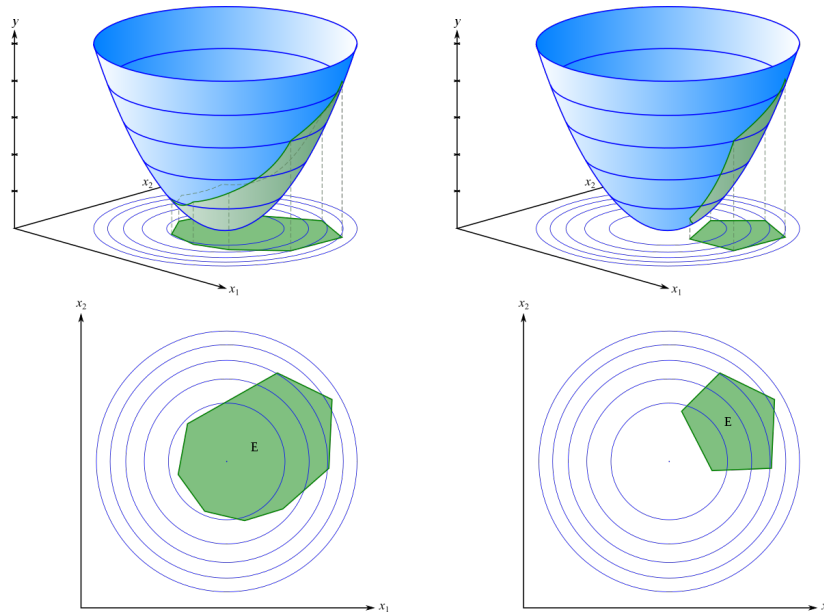


Fig. 1.2 – Mise en oeuvre de la méthode de pénalisation

1.3 Pénalisation exacte / inexacte

Une pénalisation est dite exacte si toute solution du problème (P) initial est solution du problème pénalisé (Pc), et inexacte dans le cas contraire.

Exemple :

Un exemple trivial de pénalisation exacte mais peu satisfaisant :

$$\begin{cases} \pi(x) = 0 & \text{Si } x \in X \\ \pi(x) = +\infty & \text{Sinon} \end{cases} \quad (1.3)$$

On a alors bien équivalence du problème initial et du problème pénalisé mais ce choix est très peu utilisé en pratique car les algorithmes classiques de l'optimisation ne peuvent pas être utilisés sur des fonctions valant $+\infty$ au cours des itérations. Il reste plutôt un exemple théorique.

1.4 Pénalisation extérieure

1.4.1 Présentation

Il s'agit de définir une fonction p qui pénalise la violation des contraintes. La fonction de pénalisation est nulle sur l'espace admissible, et très grande ailleurs pour éviter une solution non admissible de (Pc)

l'exemple général de pénalisation extérieure :

$$\begin{cases} \pi \text{ continue sur } \mathbb{R}^n \\ \pi(x) \geq 0 \text{ sur } \mathbb{R}^n \\ \pi(x) = 0 \Leftrightarrow x \in X \end{cases} \quad (1.4)$$

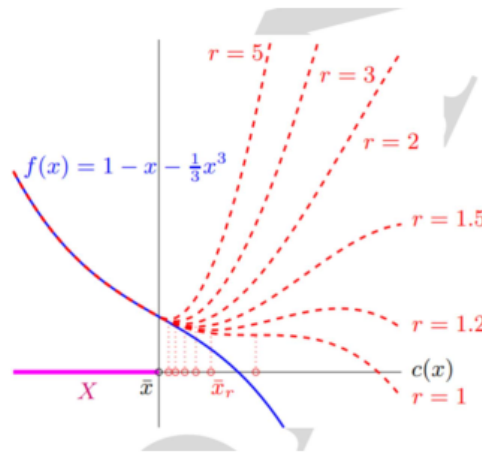


Fig. 1.3 – Exemple d'une pénalisation extérieure

1.4.2 Pénalisation quadratique

C'est un exemple de base de la pénalisation extérieure.

On prendra généralement $\pi(x) = \|x\|^2$ et $c = \frac{1}{2}$.

On note x_λ la solution de (Pc)

On résout une suite de problèmes pénalisés avec des valeurs croissantes de la pénalisation λ .

Chaque problème $k+1$ est initialisé avec la solution précédente x_k .

Problème λ_k avec pénalisation $\min f_{\lambda_k}(x) : \rightarrow \text{solution } x_k$

Il faut vérifier que la suite des solutions x_k converge vers la solution x^* du problème initial.

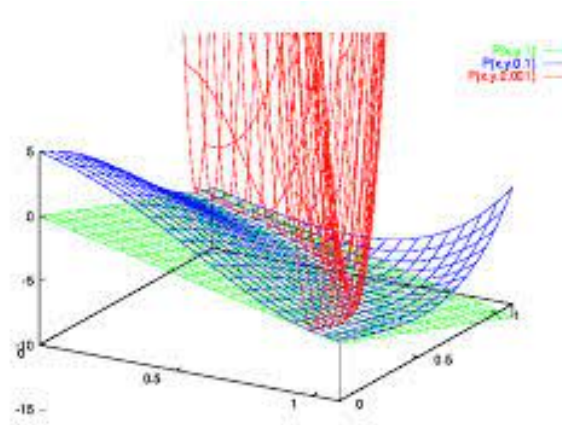


Fig. 1.4 – Exemple d'une pénalisation quadratique

1.5 Pénalisation intérieure

1.5.1 Principe

Dans certains cas, il n'est pas envisageable que les itérés n'appartiennent pas au domaine X des contraintes, en particulier quand la fonction objectif n'est pas définie en dehors de X . Dans ce cas, l'idée est d'utiliser un terme de pénalisation qui tend vers $+\infty$ lorsque x se rapproche de la frontière ∂X de X .

L'exemple général de pénalisation intérieure :

$$\begin{cases} \pi \text{ continue sur } \text{int } X \\ \pi(x) \geq 0 \text{ pour tout } x \in X \\ \pi(x) \rightarrow +\infty \text{ lorsque } x \text{ se rapproche de } \partial X \end{cases} \quad (1.5)$$

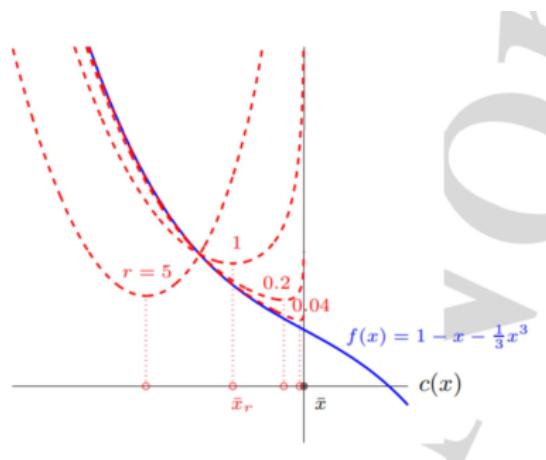


Fig. 1.5 – Exemple d'une pénalisation intérieure

1.5.2 Pénalisation logarithmique

C'est un exemple de base de la pénalisation intérieure. [4]

On prendra généralement $\pi(x) = -\log(c(x))$ et $c > 0$.

On note x_λ la solution de (P_c)

On résout une suite de problèmes pénalisés avec des valeurs décroissantes de la pénalisation $c_k > 0$.

Il faut vérifier que la suite des solutions x_k converge vers la solution x^* du problème initial.

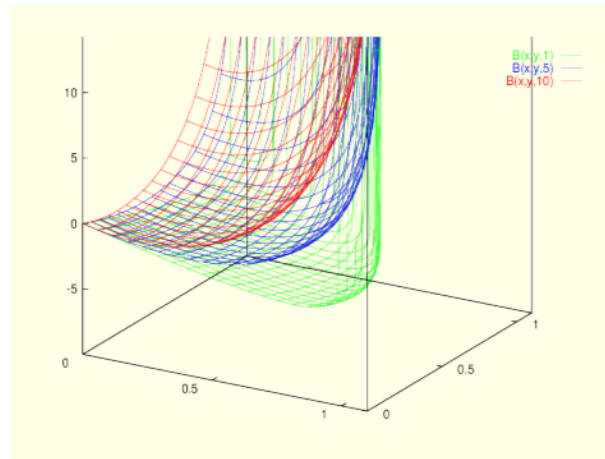


Fig. 1.6 — Exemple d'une pénalisation logarithmique

Chapitre 2

Implémentation

2.1 Présentation du problème

On considère le problème d'optimisation :

$$\left\{ \begin{array}{ll} (P_x) & : \quad \inf_{(x_1, x_2) \in R^2} \quad x_2 \\ \text{sous la contrainte} & \quad x_1^2 + x_2^2 \leq 3 \end{array} \right. \quad (2.1)$$

et on considère le problème sans contraintes :

$$(P_c) : \inf_{(x_1, x_2) \in R^2} x_2 + c_k \pi(x) \quad (2.2)$$

avec

$$\left\{ \begin{array}{l} \pi(x) = \max(f(x), 0)^2 \\ f(x) = x_1^2 + x_2^2 - 3 \end{array} \right. \quad (2.3)$$

Et on a la suite $(c_k) > 0$ et $c_k \rightarrow \infty$

On ainsi passer à un problème d'optimisation sans contrainte par le biais d'une pénalisation extérieure.

L'utilisation de **l'algorithme du gradient à pas fixe** permet de résoudre le problème d'optimisation sans contraintes.

D’ou l’algorithme correspondant :

$$\left\{ \begin{array}{l} c = 1, p = 0.01, x_1 = 1, x_2 = 0.2 \\ \text{Pour } k = 0 \dots N_{iterations} \\ x = x - p * ((0,1) + 4 * c * x * \pi(x)) \\ c = c * 10 \\ \text{fin pour} \end{array} \right. \quad (2.4)$$

2.2 Implémentation de l’algorithme en Python

Après avoir mettre en oeuvre la modélisation mathématique du problème et utiliser la pénalisation, on code le raisonnement antérieur sous forme d’un code python avec des commentaires à chaque étape pour expliquer le choix des variables et des opérations effectuées.

```
import numpy as np
from math import sqrt
def g(x):
    return x[1]
#x2
def f(x):
    return x[0]**2+x[1]**2-3
def pi(x):
    return max(f(x),0)**2
def g_c(g,pi,f,x):
    return f(x) + c*p(x)
def grad(g,pi,r,x):
    return np.array((0,1)) + 2*c*np.array((2*x[0],2*x[1]))*pi(x)
#gradient de g_c
def n_2(x):
    return sqrt(x[0]**2+x[1]**2)
#calcule la norme d'un vecteur
def minimize(g,pi,r,x):
    pas = 0.01
    while n_2(grad(g,pi,r,x))>epsilon:
        x = x - pas*grad(g,pi,r,x)
```

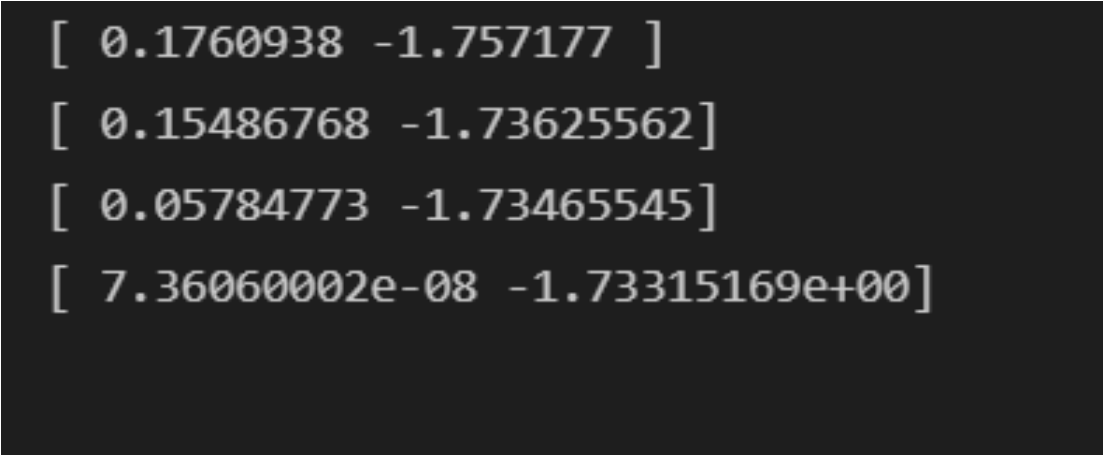
```
    return x
#update de xk

c = 1
x = np.array((0.2,1))
epsilon=0.1
Niterations = 4
#initialisations

for i in range (Niterations):
    c = 10*c
    x = minimize(g,pi,c,x)
    print(x)
    #visualisation du résultat à chaque itération
```

La solution exacte du problème est $x = (0, -\sqrt{3})$

L'algorithme fourni l'output suivant :



```
[ 0.1760938 -1.757177 ]
[ 0.15486768 -1.73625562]
[ 0.05784773 -1.73465545]
[ 7.36060002e-08 -1.73315169e+00]
```

Fig. 2.1 – Résultat final obtenu

Interprétation :

- On remarque qu'après 4 itérations, on converge numériquement vers la solution recherchée.
- Le choix de la condition initiale peut affecter le nombre d'itérations avant convergence.
- Le choix de multiplication de c par 10 à chaque itération est arbitraire. Il peut être substitué par n'importe quelle suite strictement croissante.

Chapitre 3

Creuser d'avantage

3.1 Problème

La famille d'algorithmes avec pénalisation partage beaucoup de propriétés que nous pouvons facilement remarquer. Il est souvent plus simple de présenter un résultat d'abord pour les problèmes sans contraintes et de l'étendre ensuite aux problèmes avec contraintes.[\[5\]](#)

Par contre, certaines mises-en-garde sont nécessaires. Une première, fondamentale, vient du fait que le problème de minimisation, q peut être non-borné inférieurement, et ce pour toute valeur positive de l'application \cdot . Une seconde vient du fait que sans hypothèse additionnelle, on ne peut résoudre de système d'équations par une méthode de descente.

3.2 Pénalisations non-bornées

Considérons le problème trivial suivant :

$$(P_x) : \min_{x>0} -x^4 \tag{3.1}$$

$\frac{x^2}{2c} - x^4$ n'est pas bornée inférieurement. Comme nous le verrons ci-après, rien n'assure que les minima locaux convergent vers des points réalisables, et les seuls théorèmes de convergence vraiment forts supposent la solution globale des sous-problèmes sans contrainte. L'exemple trivial montre qu'il est parfois irréaliste de supposer des solutions globales aux sous-problèmes.

3.3 Difficultés de solution des problèmes pénalisés

Le simple fait de calculer un point réalisable, qui satisfait $g(x) = 0$, constitue un problème difficile. En effet, Il s'agit de solutionner un système d'équations, et nous avons mentionné précédemment que des hypothèses sur g sont requises pour assurer que la minimisation locale de l'objectif conduit bel et bien à la solution de l'équation.

Or, la pénalité quadratique consiste en une somme pondérée de l'objectif f et du “pseudo-objectif” $\|g(x)\|^2$. Les mêmes hypothèses (monotonie de g) sont donc requises pour assurer que les points minima sont réalisables. Cette première difficulté est incontournable avec des méthodes de descente.

Remarquons que la convergence locale de telles méthodes n'est pas mise en cause, et certains auteurs affirment que si l'algorithme ne converge pas vers un point réalisable, c'est que le modèle et l'estimé initial ne sont pas adéquats.

D'où la nécessité de vérifier les conditions de convergence à chaque fois qu'on utilise les procédés à pénalisation. .

Conclusion

J'ai réussi dans ce travail à se familiariser avec les fondements mathématiques de la méthode de pénalisation et de l'explorer dans le cadre d'un exemple bien défini d'optimisation avec contrainte.

En effet, le travail de documentation que j'ai fait m'a permis de faire face à la problématique de pénalisation présentée dans le cadre du cours de l'optimisation et d'implémenter l'étude théorique en code exécutable.

De plus, dans la dernière partie du travail j'ai allé encore plus loin dans l'exploration de la méthode en montrant ses limitations la nécessité de respecter les hypothèses de convergence pour trouver des résultats fiables.

Souheib Ben Mabrouk.

Bibliographie

- [1] M. EL MAGHRI S. BOLINTINÉANU. énalisation dans l'optimisation sur l'ensemble faiblement efficient. <https://www.rairo-ro.org/articles/ro/pdf/1997/03/ro1997310302951.pdf>.
- [2] Pénalisation (optimisation). [https://fr.wikipedia.org/wiki/P%C3%A9nalisation_\(optimisation\)#P%C3%A9nalisations_ext%C3%A9rieure_et_int%C3%A9rieure](https://fr.wikipedia.org/wiki/P%C3%A9nalisation_(optimisation)#P%C3%A9nalisations_ext%C3%A9rieure_et_int%C3%A9rieure).
- [3] Algorithmes pour l'optimisation différentiable sous contrainte. <https://www.math.univ-toulouse.fr/~rondep/CoursTD/polyGMM4.pdf>.
- [4] Introduction à l'optimisation différentiable avec contraintes non linéaires. <http://www.dmi.usherb.ca/~dussault/RennesOptim/OPTChap6.pdf>.
- [5] Optimisation mathématique avec applications en imagerie. <http://www.dmi.usherb.ca/~dussault/OPT.pdf>.