



14 janvier 2023

Awa Khouna, Aurélien Stumpf Mascles, Styve Ngamou



TABLE DES MATIÈRES

1	Introduction	3
2	Résultats généraux des méthodes d'optimisation	4
2.1	Formulation générale du problème d'optimisation	4
2.2	Méthode d'ordre 0	4
2.3	Méthode d'ordre 1	4
2.3.1	Démonstration du théorème 1.4	5
2.4	Exemples	6
3	Résultats des méthodes d'optimisation sous contraintes	8
3.1	Algorithmes de gradient à pas fixe avec projection	8
3.1.1	Démonstration du théorème 2.1	8
3.2	Algorithmes de calcul de projection	9
3.2.1	L'algorithme de projections alternées	9
3.2.2	L'algorithme de projection euclidienne sur le simplexe	9
3.3	Application numérique	11
4	Descente miroir déterministe	12
4.1	Le point de vue proximal	12
4.2	Divergence de Bregman	12
4.3	Changement de la distance	13
4.4	Le point de vue dual	13
4.4.1	Normes et leurs duals	13
4.4.2	Définir les fonctions miroirs	14
4.4.3	L'algorithme	14
4.5	L'analyse	14
4.6	Exemples	15
5	Algorithmes stochastiques	18
5.1	Schéma stochastique	18
5.2	Vitesse de convergence de la descente de gradient stochastique pour une fonction convexe	19
5.3	Vitesse de convergence de la descente de gradient stochastique pour une fonction strictement convexe	20
6	Descente Miroir Stochastique	21
6.1	Introduction	21
6.2	La méthode de descente miroir stochastique	21
6.2.1	L'algorithme	22
6.2.2	Regret pour la méthode de descente miroir stochastique	22

6.3	Démonstration du théorème	23
7	Application de la descente mirroir à l'optimisation d'un portefeuille	26
7.1	Introduction du problème	26
7.2	Formalisation du problème pour la descente mirroir sans contrainte de risque . .	26
7.3	Ajout d'une contrainte de risque	28
7.4	Résultats expérimentaux	30
7.4.1	Optimisation de la valeur du portefeuille dans le cas sans contraintes . .	31
7.4.2	Optimisation de la valeur du portefeuille dans le cas avec contraintes de risque	33
8	Remerciements	35

1 INTRODUCTION

Certains problèmes d'optimisation sous contraintes sont difficiles à résoudre car la projection euclidienne sur l'espace des contraintes est complexe et coûteuse à calculer. C'est notamment le cas de la projection sur le simplexe (ensemble des mesures de probabilités à n poids). Nesterov a ainsi eu l'idée d'adapter la mesure utilisée dans la projection afin de simplifier le calcul. La méthode de descente miroir est une généralisation de la descente de gradient classique appliquée à une classe particulière de distances appelées distances de Bregman.

Ce rapport a pour objet l'étude théorique de la descente miroir déterministe et stochastique ainsi que leur implémentation numérique sur des cas tests concrets. Il aboutit à la résolution d'un problème d'optimisation de portefeuille d'actions dans le cas libre et dans le cas constraint.

La première partie du rapport est consacrée aux méthodes déterministes. Les deux premières sections seront consacrées aux méthodes de descente de gradient classiques avec et sans contraintes. La troisième présente la descente miroir déterministe comme la généralisation du cas de la descente de gradient euclidienne aux métriques associées aux distances de Bregman.

La deuxième partie est consacrée aux problèmes stochastiques. La quatrième section formalise la notion d'algorithme stochastique, et permet de prolonger la descente miroir classique au cas stochastique dans la cinquième section.

Enfin, la dernière partie est consacrée à la formalisation et la résolution du problème d'optimisation d'un portefeuille d'actions grâce aux outils développées tout au long du rapport. Nous nous intéresserons d'abord au cas sans contrainte puis au cas où le risque de perte à 5% est majoré. Nous montrerons en particulier que la diversification du portefeuille est nécessaire afin de se prémunir contre les risques de perte.

On pourra trouver en annexe le code utilisé afin de générer les simulations numériques.

2

RÉSULTATS GÉNÉRAUX DES MÉTHODES D'OPTIMISATION

2.1 FORMULATION GÉNÉRALE DU PROBLÈME D'OPTIMISATION

On considère $x \in \mathbb{R}^p$ et $f : \mathbb{R}^p \rightarrow \mathbb{R}$ et on veut résoudre le problème suivant :

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

où S est un ensemble de contraintes.

2.2 MÉTHODE D'ORDRE 0

On considère dans cette partie que f est L -Lipschitz et que S est bornée.

Théorème 2.1. *Une ϵ solution de (P) peut être résolue en $\left\{\frac{L}{2\epsilon} + 2\right\}^n$*

Pour cela, il suffit de considérer un maillage de points dont on contrôle la distance entre les points adjacents et que l'on parcourt de façon exhaustive.

2.3 MÉTHODE D'ORDRE 1

La méthode de descente de gradient est l'algorithme suivant :

Algorithme Schéma de la descente de gradient

Input : f et $(\gamma_k)_{k \in \mathbb{N}}$ suite de pas

Initialisation : Choisir $x_0 \in \mathbb{R}^n$

Itérer :

$$\forall k \in \mathbb{N}, \quad x_{k+1} = x_k - \gamma_k \nabla f(x_k)$$

Output : x_k vérifiant une certaine condition

On a les résultats de convergence suivants :

Théorème 2.2. *Si f est une fonction convexe C^1 dont le gradient est L -Lipschitz et possédant un minimum x^* , alors $\lim_{k \rightarrow +\infty} x_k = x^*$*

Théorème 2.3. Si f est une fonction convexe C^1 dont le gradient est L -Lipschitzien, alors la méthode de gradient de pas $\eta = L^{-1}$ satisfait :

$$f(x_{n+1}) - f(x^*) \leq \frac{2L\|x_0 - x^*\|^2}{n}$$

Théorème 2.4. Si f est une fonction α -convexe C^1 dont le gradient est L -Lipschitzien, alors la méthode de gradient de pas $\eta = \frac{2}{L+\alpha}$ satisfait :

$$f(x_{n+1}) - f(x^*) \leq \frac{L}{2} \exp\left(-\frac{4n}{\kappa+1}\right) \|x_1 - x^*\|^2$$

où $\kappa = \frac{L}{\alpha}$

2.3.1 • DÉMONSTRATION DU THÉORÈME 1.4

Lemme 2.5. $f \in SC(\alpha) \iff \forall (x, y) \in \mathbb{R}^d, f(x) \leq f(y) + \langle \nabla f(x), x - y \rangle - \frac{\alpha}{2} \|y - x\|^2$

Démonstration.

$$\begin{aligned} f \in SC(\alpha) &\iff f - \frac{\alpha}{2} \|\cdot\|^2 \text{ est convexe} \\ &\iff \forall (x, y) \in \mathbb{R}^d, f(y) - \frac{\alpha}{2} \|y\|^2 \geq f(x) - \frac{\alpha}{2} \|x\|^2 + \langle \nabla f(x) - \alpha x, y - x \rangle \\ &\iff \forall (x, y) \in \mathbb{R}^d, f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\alpha}{2} \|y - x\|^2 \end{aligned}$$

□

Proposition 2.6. Si $f \in SC(\alpha)$, alors

$$\forall (x, y) \in \mathbb{R}^d, \langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \alpha \|y - x\|^2$$

On peut améliorer cette borne inférieure, grâce au lemme suivant :

Lemme 2.7. Si $f \in SC(\alpha)$ et le gradient de f est L -Lipschitzien, alors

$$\forall (x, y) \in \mathbb{R}^d, \langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \frac{\alpha L}{\alpha + L} \|y - x\|^2 + \frac{1}{\alpha + L} \|\nabla f(y) - \nabla f(x)\|^2$$

On peut maintenant démontrer le théorème

Démonstration. Notons x^* le minimum de f .

Comme f est α -convexe et que son gradient est L -Lipschitzien, elle vérifie l'inégalité :

$$f(x) - f(x^*) \leq \langle \nabla f(x^*), x - x^* \rangle + \frac{L}{2} \|x - x^*\|^2 = \frac{L}{2} \|x - x^*\|^2$$

car $\nabla f(x^*) = 0$

On va maintenant appliquer une formule de récurrence au terme $\|x_{t+1} - x_t\|^2$.

$$\begin{aligned}
 \|x_{t+1} - x^*\|^2 &= \|x_t - x^* - \gamma \nabla f(x_t)\|^2 \\
 &= \|x_t - x^*\|^2 - 2\gamma \langle \nabla f(x_t), x_t - x^* \rangle + \gamma^2 \|\nabla f(x_t)\|^2 \\
 &\leq \|x_t - x^*\|^2 - 2\gamma \left(\frac{\alpha L}{\alpha + L} \|x_t - x^*\|^2 + \frac{1}{\alpha + L} \|\nabla f(x_t)\|^2 \right) + \gamma^2 \|\nabla f(x_t)\|^2 \\
 &\leq \left(1 - \frac{2\gamma L \alpha}{L + \alpha} \right) \|x_t - x^*\|^2 + \|\nabla f(x_t)\|^2 \left(\gamma^2 - \frac{2\gamma}{L + \alpha} \right)
 \end{aligned}$$

Prendre $\gamma = \frac{2}{L+\alpha}$ permet de supprimer le terme à droite et on obtient :

$$\|x_{t+1} - x^*\|^2 \leq \left(1 - \frac{2}{\kappa + 1} \right)^2 \|x_t - x^*\|^2 \leq \exp\left(-\frac{4}{\kappa + 1}\right) \|x_t - x^*\|^2$$

car $1 - x \leq \exp(-x)$, ce qui conclut la preuve. □

2.4 EXEMPLES

On considère la fonction suivante : $\forall (x, y) \in \mathbb{R}^2, \quad f(x, y) = \sin(x^2 + y^2)$ qui est localement strictement convexe autour de 0.

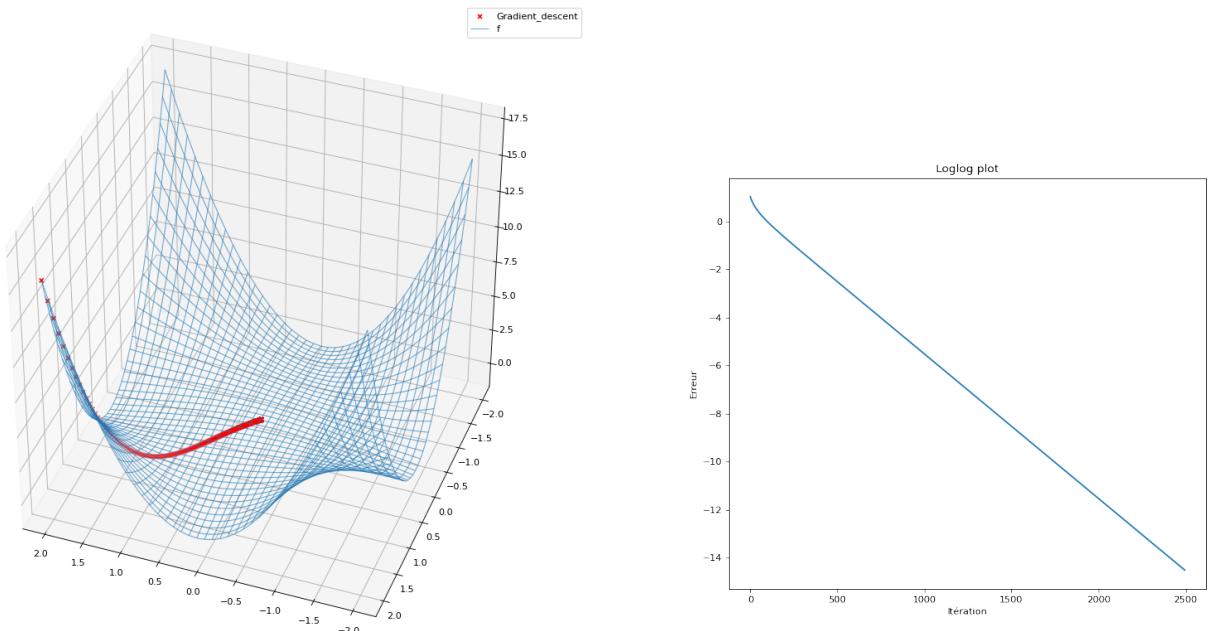


FIGURE 1 – Visualisation de la descente de gradient.

FIGURE 2 – Graphique du log de l'erreur en fonction du nombre d'itérations

2. RÉSULTATS GÉNÉRAUX DES MÉTHODES D'OPTIMISATION



Le graphique logarithmique en l'erreur montre une vitesse de convergence de l'algorithme qui est bien exponentielle.

3

RÉSULTATS DES MÉTHODES D'OPTIMISATION SOUS CONTRAINTES

3.1 ALGORITHMES DE GRADIENT À PAS FIXE AVEC PROJECTION

On se limite dans cette section au cas où K est un convexe fermé non vide de \mathbb{R}^n . On note P_K la projection sur l'ensemble convexe K .

On considère l'algorithme de descente de gradient à pas fixe avec projection :

Algorithme Schéma de la descente de gradient

Input : f et $(\gamma_k)_{k \in \mathbb{N}}$ suite de pas

Initialisation : Choisir $x_0 \in \mathbb{R}^n$

Itérer :

$$\forall k \in \mathbb{N}, \quad x_{k+1} = P_K(x_k - \gamma_k \nabla f(x_k))$$

Output : x_k vérifiant une certaine condition

Théorème 3.1. *On suppose que f est une fonction α -convexe C^1 dont le gradient est L -Lipschitzien. Alors la méthode de gradient de pas $\eta = \frac{2}{L+\alpha}$ satisfait également :*

$$f(x_{n+1}) - f(x^*) \leq \frac{L}{2} \exp\left(-\frac{4n}{\kappa + 1}\right) \|x_1 - x^*\|^2$$

où $\kappa = \frac{L}{\alpha}$

3.1.1 • DÉMONSTRATION DU THÉORÈME 2.1

Toutes les inégalités préliminaires démontrées dans la partie 1.3.1 restent vraies.

Par ailleurs, le calcul suivant :

Soient $y \in \mathbb{R}^n$ et $x \in K$,

$$\begin{aligned} \|y - x\|^2 &= \|y - P_K(y) + P_K(y) - x\|^2 \\ &= \|y - P_K(y)\|^2 + \|P_K(y) - x\|^2 + \langle y - P_K(y), P_K(y) - x \rangle \\ &\geq \|P_K(y) - x\|^2 \end{aligned}$$

car $\langle y - P_K(y), P_K(y) - x \rangle \geq 0$ d'après les propriétés de projection sur un convexe fermé.

Ainsi,

$$\|x_{k+1} - x^*\|^2 = \|P_K(x_k - \gamma_k \nabla f(x_k)) - x^*\|^2 \leq \|x_k - \gamma_k \nabla f(x_k) - x^*\|^2$$

On peut donc mener les mêmes calculs que dans le cas de l'algorithme sans projection ce qui mène au même résultat de vitesse de convergence.

3.2 ALGORITHMES DE CALCUL DE PROJECTION

3.2.1 • L'ALGORITHME DE PROJECTIONS ALTERNÉES

L'algorithme de projections alternées est un algorithme très simple pour calculer la projection convexe d'un point sur une intersection de convexe.

La méthode des projections alternées est l'algorithme suivant :

Algorithme Méthode des projections alternées

Input : C et D deux ensembles convexes, P_D et P_C leurs projections associées, x_0 le point dont on veut calculer la projection

Itérer :

$$y_k = P_D(x_k) \quad x_{k+1} = P_C(y_k)$$

Output : $x_* \in C \cap D$

3.2.2 • L'ALGORITHME DE PROJECTION EUCLIDIENNE SUR LE SIMPLEXE

On veut résoudre le problème suivant :

$$\begin{aligned} \min_{x \in \mathbb{R}^D} \quad & \frac{1}{2} \|x - y\|^2 \\ \text{tel que} \quad & x^T \mathbf{1} = 1 \\ & x \geq 0 \end{aligned}$$

L'algorithme suivant trouve une solution au problème :

Algorithme Projection euclidienne sur le simplexe

Input : $y \in \mathbb{R}^D$

Trier y en u : $u_1 \geq \dots \geq u_D$

Trouver $\rho = \max\{1 \leq j \leq D : u_j + \frac{1}{j}(1 - \sum_{i=1}^j u_i) > 0\}$

Définir $\lambda = \frac{1}{\rho}(1 - \sum_{i=1}^{\rho} u_i)$

Output : x tel que $x_i = \max\{y_i + \lambda, 0\}, i = 1, \dots, D$

Démonstration. On va appliquer les conditions de Kuhn et Tucker au lagrangien du problème :

$$\mathcal{L}(x, \lambda, \beta) = \frac{1}{2} \|x - y\|^2 - \lambda(x^T \mathbf{1} - 1) - \beta^T x$$

3. RÉSULTATS DES MÉTHODES D'OPTIMISATION SOUS CONTRAINTES

où λ et β sont les multiplicateurs de lagrange du problème. Comme la fonction à minimiser et les contraintes sont convexes, le lagrangien possède une solution optimale qui vérifie :

$$\begin{aligned} x_i - y_i - \lambda - \beta_i &= 0, \quad i = 1, \dots, D \\ x_i &\geq 0, \quad i = 1, \dots, D \\ \beta_i x_i &= 0, \quad i = 1, \dots, D \end{aligned}$$

Si $x_i > 0$, alors $\beta_i = 0$ et $x_i = y_i + \lambda > 0$.

Si $x_i = 0$, alors $\beta_i \geq 0$, et $x_i = y_i + \lambda + \beta_i = 0$ donc $y_i + \lambda = -\beta_i \leq 0$.

On suppose que y est triée par ordre décroissant, ce qui implique que x l'est aussi. On note ρ le dernier indice tel que $x_\rho > 0$ et donc $x_{\rho+1} = \dots = x_D = 0$.

La dernière condition donne :

$$1 = \sum_{i=1}^D x_i = \sum_{i=1}^\rho x_i = \sum_{i=1}^\rho (y_i + \lambda)$$

Ce qui donne $\lambda = \frac{1}{\rho}(1 - \sum_{i=1}^D y_i)$. Ainsi ρ est la clé du problème. Une fois ρ connu, on peut calculer λ et donc les composantes de x .

Exprimons maintenant ρ en fonction de y . Montrons que :

$$\rho = \max\{1 \leq j \leq D, y_j + \frac{1}{j}(1 - \sum_{i=1}^j y_i) > 0\}$$

On va montrer que le test $y_j + \frac{1}{j}(1 - \sum_{i=1}^j y_i) > 0$ va être vrai jusqu'à $j = \rho$.

- Pour $j = \rho$, on a $y_\rho + \frac{1}{\rho}(1 - \sum_{i=1}^\rho y_i) = y_\rho + \lambda = x_\rho > 0$
- Pour $j < \rho$,

$$\begin{aligned} y_j + \frac{1}{j}(1 - \sum_{i=1}^j y_i) &= \frac{1}{j}(jy_j + 1 - \sum_{i=1}^j y_i) = \frac{1}{j}(jy_j + \sum_{i=j+1}^\rho y_i + 1 - \sum_{i=1}^j y_i) \\ &= \frac{1}{j}(jy_j + \sum_{i=j+1}^\rho y_i + \rho\lambda) = \frac{1}{\rho}(j(y_j + \lambda) + \sum_{i=j+1}^\rho y_i) > 0 \end{aligned}$$

car $y_i + \lambda > 0$ pour $i = j, \dots, \rho$

- Pour $j > \rho$, on a

$$\begin{aligned} y_j + \frac{1}{j}(1 - \sum_{i=1}^j y_i) &= \frac{1}{j}(jy_j + 1 - \sum_{i=1}^j y_i) \\ &= \frac{1}{j}(jy_j + 1 - \sum_{i=1}^\rho y_i - \sum_{i=\rho+1}^j y_i) \\ &= \frac{1}{j}(jy_j + \rho\lambda - \sum_{i=\rho+1}^j y_i) = \frac{1}{j}(\rho(y_j + \lambda) + \sum_{i=\rho+1}^j (y_j - y_i)) < 0 \end{aligned}$$

car y est triée par ordre décroissant et $y_j + \lambda \leq 0$ pour $j > \rho$.

□

3.3 APPLICATION NUMÉRIQUE

On définit la fonction test :

$$\forall (x, y) \in \mathbb{R}^2, \quad f(x, y) = (1 - x)^2 + (y - x)^2$$

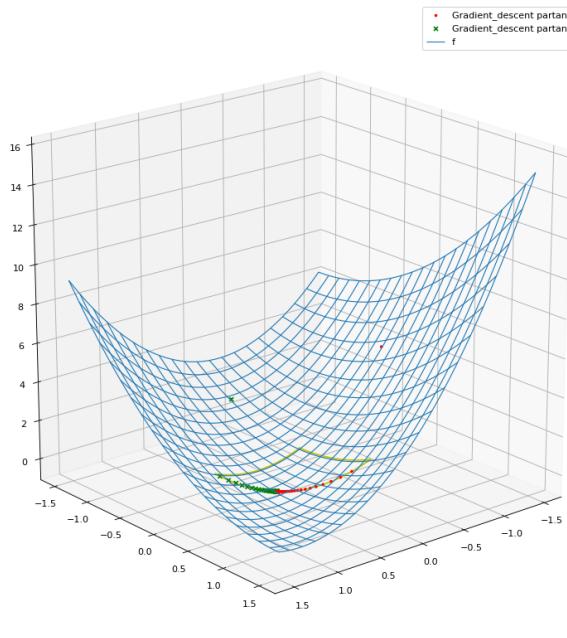


FIGURE 3 – Visualisation de la descente de gradient avec projection sur le simplexe.

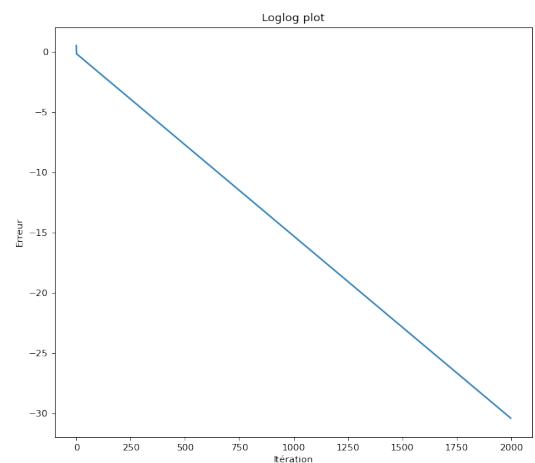


FIGURE 4 – Graphique du log de l'erreur en fonction du nombre d'itérations

On retrouve à nouveau la vitesse de convergence exponentielle.

4

DESCENTE MIRROIR DÉTERMINISTE

La méthode de descente de gradient avec projection sur un convexe est une méthode très générale qui peut être difficile à implémenter dans certains problèmes. Par exemple, dans le cas de la minimisation sur le simplexe, il n'existe pas de formule pour la projection sur le simplexe, ce qui rend l'implémentation de l'algorithme plus difficile.

Cela suggère de se poser la question suivante : "Peut-on améliorer la descente de gradient de façon à ce qu'elle s'adapte à la géométrie du problème ?"

4.1 LE POINT DE VUE PROXIMAL

La méthode proximale est l'algorithme suivant :

Algorithme Schéma de la descente de gradient proximale

Input : f et $(\gamma_k)_{k \in \mathbb{N}}$ suite de pas

Initialisation : Choisir $x_0 \in \mathbb{R}^n$

Itérer :

$$\forall k \in \mathbb{N}, \quad x_{k+1} = \operatorname{argmin}_x \left\{ \gamma_k \langle \nabla f(x_k), x \rangle + \frac{1}{2} \|x - x_k\|^2 \right\}$$

Output : x_k vérifiant une certaine condition

En annulant le gradient de cette fonction, on obtient exactement la descente de gradient de pas γ_k . Néanmoins, cela permet de voir la descente de gradient d'un autre point de vue, celui de la minimisation de l'approximation linéaire de f en x_k , c'est-à-dire $f(x_k) + \langle \nabla f(x_k), x \rangle$, en ayant rajouté un terme de pénalisation $\frac{1}{2} \|x - x_k\|^2$ qui matérialise l'éloignement à l'approximation. Dans le cas d'un problème sous contrainte, il suffit de considérer la résolution du problème :

$$\operatorname{argmin}_{x \in K} \left\{ \gamma_k \langle \nabla f(x_k), x \rangle + \frac{1}{2} \|x - x_k\|^2 \right\}$$

4.2 DIVERGENCE DE BREGMAN

On se donne une fonction strictement convexe h , et on va définir une distance basée sur l'éloignement de cette fonction par rapport à son approximation linéaire.

Définition 4.1. La divergence de Bregman de x à y selon la fonction h est :

$$D_h(y \| x) = h(y) - h(x) - \langle \nabla h(x), y - x \rangle$$

Exemple 4.2. Pour la fonction $h(x) = \frac{1}{2}\|x\|^2$, la divergence de Bregman associée est la distance euclidienne.

On peut définir de nombreuses divergences de Bregman intéressantes pour résoudre certains problèmes.

4.3 CHANGEMENT DE LA DISTANCE

Comme $\frac{1}{2}\|x - y\|^2$ est une distance de Bregman, on peut donc considérer l'algorithme proximal avec d'autres distances de Bregman. Ainsi le problème devient :

$$x_{k+1} = \operatorname{argmin}_x \{\gamma_k \langle \nabla f(x_k), x \rangle + D_h(x \| x_k)\}$$

En annulant le gradient, on obtient :

$$\begin{aligned} \gamma_k \nabla f(x_k) + \nabla h(x_{k+1}) - \nabla h(x_k) &= 0 \\ \nabla h(x_{k+1}) &= \nabla h(x_k) - \gamma_k \nabla f(x_k) \\ x_{k+1} &= \nabla h^{-1}(\nabla h(x_k) - \gamma_k \nabla f(x_k)) \end{aligned}$$

4.4 LE POINT DE VUE DUAL

Le principe de l'algorithme est le suivant :

- On envoie le point x_k sur un point θ_k dans l'espace dual grâce à une fonction "mirroir" bijective.
- On effectue un pas de gradient

$$\theta_{k+1} = \theta_k - \gamma_k \nabla f(x_k)$$

- On renvoie θ_{k+1} sur x'_{k+1} dans l'espace primal
- Dans le cas contraint, on projète x'_{k+1} en x_{k+1} sur K .

Le choix de la fonction miroir dépend de la fonction que l'on veut minimiser et de l'ensemble K de contrainte.

4.4.1 • NORMES ET LEURS DUALS

Définition 4.3. Norme duale Soit $\|\cdot\|$ une norme. La norme duale de $\|\cdot\|$ est une fonction $\|\cdot\|_*$ définie par :

$$\|y\|_* = \sup\{ \langle x, y \rangle : \|x\| \leq 1 \}$$

Théorème 4.4. (Cauchy-Schwarz) $\forall x, y \in \mathbb{R}^n, \langle x, y \rangle \leq \|x\| \|y\|_*$

4.4.2 • DÉFINIR LES FONCTIONS MIRROIRS

Pour définir, une fonction miroir, on fixe tout d'abord une norme $\|\cdot\|$, and on choisit une fonction convexe différentiable $h : \mathbb{R}^n \rightarrow \mathbb{R}$ qui est α -convexe par rapport à la norme $\|\cdot\|$.

Une telle fonction doit satisfaire :

$$h(y) \geq h(x) + \langle \nabla h(x), y - x \rangle + \frac{\alpha}{2} \|y - x\|^2$$

On définit alors la fonction miroir par :

$$\nabla(h) : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

Comme h est différentiable et fortement convexe, d'après le théorème d'inversion globale, h est inversible et son inverse est différentiable.

Ainsi on peut définir les fonctions miroir par :

$$\theta_k = \nabla h(x_k) \quad \text{et} \quad x'_{k+1} = (\nabla h)^{-1}(\theta_{k+1})$$

4.4.3 • L'ALGORITHME

Supposons que l'on doive minimiser une fonction convexe f sur un ensemble convexe $K \subseteq \mathbb{R}^n$. On fixe une norme $\|\cdot\|$ sur \mathbb{R}^n et on choisit une fonction génératrice de distance $h : \mathbb{R}^n \rightarrow \mathbb{R}$. L'algorithme est alors le suivant :

- On passe dans l'espace dual : $\theta_k = \nabla h(x_k)$
- On effectue un pas de gradient dans l'espace dual : $\theta_{k+1} = \theta_k - \gamma_k \nabla f(x_k)$
- On revient dans l'espace primal : $x'_{k+1} = (\nabla h)^{-1}(\theta_{k+1})$
- Dans le cas contraint, on projète $x_{k+1} = \min_{x \in K} D_h(x \| x'_{k+1})$ sur K avec la distance associée à la divergence de Bregman.

4.5 L'ANALYSE

Théorème 4.5. (*Borne supérieure de la descente miroir*)

Soit $\|\cdot\|$ une norme de \mathbb{R}^n , h une fonction α -convexe relativement à la norme, et f convexes, différentiable et telle que $\|\nabla f\|_* \leq G$. L'algorithme de descente miroir initialisé au point x_0 et de pas constant η , produisant les points x_1, \dots, x_N à chaque itération,

$$\forall x^* \in \mathbb{R}^n, \sum_{k=1}^N f(x_k) \leq N f(x^*) + \underbrace{\frac{D_h(x^* \| x_1)}{\eta} + \frac{\eta \sum_{k=1}^N \|\nabla f(x_k)\|_*^2}{2\alpha}}_{\text{regret}}$$

Démonstration. Posons $\Phi_k = \frac{D_h(x^* \| x_k)}{\eta}$.

$$\begin{aligned}
 \Phi_{k+1} - \Phi_k &= \frac{1}{\eta} (D_h(x^* \| x_{k+1}) - D_h(x^* \| x_k)) \\
 &= \frac{1}{\eta} (h(x^*) - h(x_{k+1}) - \underbrace{\langle \nabla h(x_{k+1}), x^* - x_{k+1} \rangle}_{\theta_{k+1}} - h(x^*) + h(x_k) + \underbrace{\langle \nabla h(x_k), x^* - x_k \rangle}_{\theta_k}) \\
 &= \frac{1}{\eta} (h(x_k) - h(x_{k+1}) - \langle \theta_k - \eta \nabla f(x_k), x^* - x_{k+1} \rangle + \langle \theta_k, x^* - x_k \rangle) \\
 &= \frac{1}{\eta} (h(x_k) - h(x_{k+1}) - \langle \theta_k, x_k - x_{k+1} \rangle + \eta \langle \nabla f(x_k), x^* - x_{k+1} \rangle) \\
 &\leq \frac{1}{\eta} \left(-\frac{\alpha}{2} \|x_{k+1} - x_k\|^2 + \eta \langle \nabla f(x_k), x^* - x_{k+1} \rangle \right)
 \end{aligned}$$

Par α -convexité de h .

Ainsi,

$$\begin{aligned}
 f(x_k) - f(x^*) + \Phi_{k+1} - \Phi_k &\leq f(x_k) - f(x^*) - \frac{\alpha}{2\eta} \|x_{k+1} - x_k\|^2 + \langle \nabla f(x_k), x^* - x_{k+1} \rangle \\
 &\leq \underbrace{f(x_k) - f(x^*) + \langle \nabla f(x_k), x^* - x_k \rangle}_{\leq 0 \text{ par convexité de } f} - \frac{\alpha}{2\eta} \|x_{k+1} - x_k\|^2 + \langle \nabla f(x_k), x^* - x_{k+1} \rangle \\
 &\leq -\frac{\alpha}{2\eta} \|x_{k+1} - x_k\|^2 + \|\nabla f(x_k)\|_*^2 \|x_k - x_{k+1}\| \\
 &\leq -\frac{\alpha}{2\eta} \|x_{k+1} - x_k\|^2 + \frac{1}{2} \left(\frac{\eta}{\alpha} \|\nabla f(x_k)\|_*^2 + \frac{\alpha}{\eta} \|x_k - x_{k+1}\|^2 \right) \quad \text{par l'inégalité de Young} \\
 &= \frac{\eta}{2\alpha} \|\nabla f(x_k)\|_*^2
 \end{aligned}$$

On obtient le résultat en sommant sur k . \square

4.6 EXEMPLES

On va appliquer la méthode de descente miroir à la minimisation de la fonction test f de la partie précédente sur le simplexe.

On introduit la fonction strictement convexe suivante :

$$\forall u \in \Delta_m, \quad \phi(u) = \sum_{i=1}^m u_i \log u_i - u_i$$

On définit la divergence de Bregman associée $D_\phi(u, v)$ définie par :

$$\forall (u, v) \in \Delta_m^2, \quad D_\phi(u, v) = \phi(u) - \phi(v) - \langle \nabla \phi(v), u - v \rangle$$

4. DESCENTE MIRROIR DÉTERMINISTE

On montrera dans la partie 6 que la projection sur le simplexe pour cette divergence de Bregman s'écrit simplement :

$$x = \frac{y}{\|y\|_1}$$

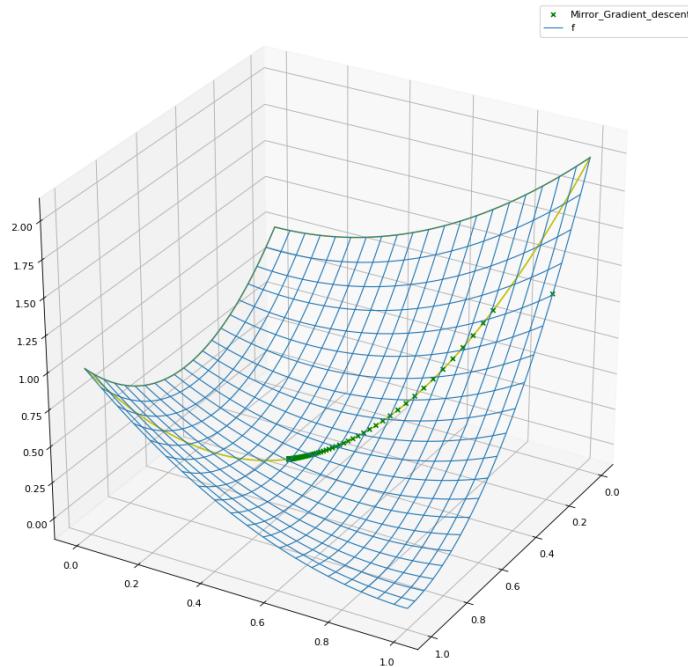


FIGURE 5 – Visualisation de la descente de gradient miroir avec projection sur le simplexe.

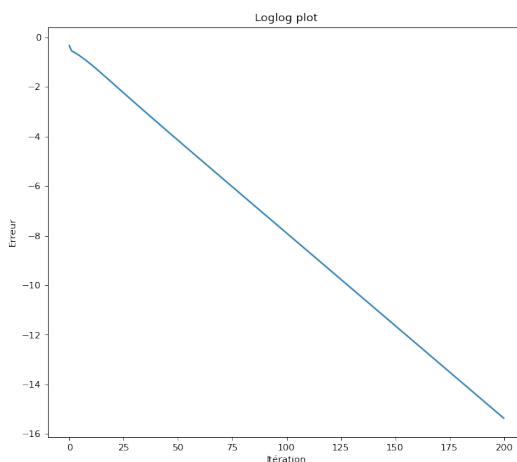


FIGURE 6 – Graphique du log de l'erreur en fonction du nombre d'itérations

On peut à nouveau observer ici une convergence exponentielle, ce qui est bien meilleur que le résultat donné par la borne théorique.

4. DESCENTE MIRROIR DÉTERMINISTE

La comparaison des vitesses de convergence par la méthode classique et par la méthode miroir donne :

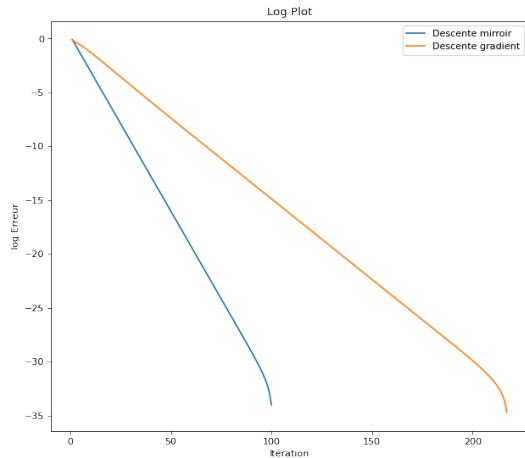


FIGURE 7 – Comparaison des vitesses de convergence

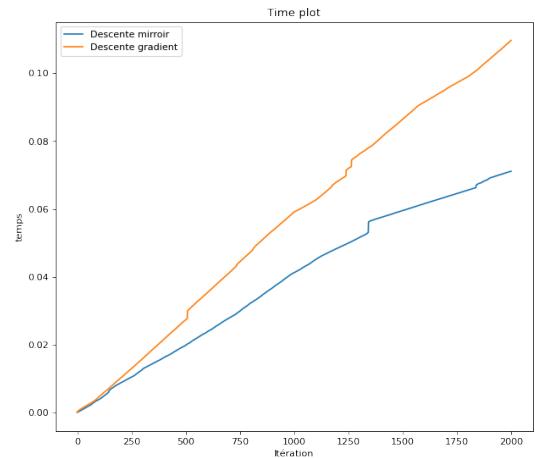


FIGURE 8 – Comparaison des temps d'exécution

On peut observer que la vitesse de convergence par la méthode miroir est moins grande que la vitesse de convergence par la méthode classique. En revanche, l'exécution de la méthode miroir est plus rapide que celui de la méthode classique. Cela montre bien l'intérêt de la méthode miroir, qui est d'augmenter la rapidité de la projection sur le convexe des contraintes.

5

ALGORITHMES STOCHASTIQUES

Notation 5.1. (*Filtration*) La filtration canonique associée à un vecteur aléatoire (X_1, \dots, X_n) est noté $F_n^X = \sigma(X_1, \dots, X_n)$.

Définition 5.2. (Algorithme stochastique) Un algorithme stochastique est défini par :

$$X_{n+1} = X_n - \gamma_{n+1} h(X_n) + \gamma_{n+1} \Delta M_{n+1}$$

où ΔM_{n+1} est une F_n^X -martingale, h une fonction arbitraire et (γ_n) est une suite de pas positifs telle que :

$$\gamma_n \xrightarrow{n \rightarrow \infty} 0 \quad \text{et} \quad \sum_{n \geq 1} \gamma_n = +\infty$$

Le but de cette partie est de décrire le comportement de ces algorithmes lorsque n tend vers l'infini.

5.1 SCHÉMA STOCHASTIQUE

On suppose dans cette section que h est le gradient d'une fonction U que l'on veut minimiser. Sans perte de généralité on peut supposer que le minimiseur est atteint en 0.

Intractabilité du gradient : Imaginons que U soit donné par :

$$\forall x \in \mathbb{R}^d, \quad U(x) = \int_E \mathcal{U}(x, y) \mu(dy)$$

où μ est une mesure de probabilité sur E .

Supposons que la fonction h est donnée par :

$$h(x) = \nabla U(x) = \int_E \partial_x \mathcal{U}(x, y) \mu(dy)$$

Le gradient de U n'est donc pas calculable explicitement dans le cas général.

Dans ce cas, nous allons considérer l'algorithme :

$$X_{n+1} = X_n - \gamma_{n+1} \partial_x \mathcal{U}(X_n, Y_{n+1})$$

où $(Y_n)_{n \geq 1}$ est une suite de variables aléatoires distribués selon μ .

Réécrivons,

$$X_{n+1} = X_n - \gamma_{n+1} h(X_n) + \gamma_{n+1} \Delta M_{n+1}$$

avec

$$\Delta M_{n+1} = h(X_n) - \partial_x \mathcal{U}(X_n, Y_{n+1})$$

Alors, (ΔM_n) bien une martingale car :

$$\mathbb{E}[\Delta M_{n+1}|F_n] = \int_E \mathcal{U}(X_n, y)\mu(dy) - \mathbb{E}[\partial_x \mathcal{U}(X_n, Y_{n+1})|F_n] = 0$$

On retrouve donc bien la forme standard d'un algorithme stochastique, qui sera appellée **descente de gradient stochastique**.

5.2 VITESSE DE CONVERGENCE DE LA DESCENTE DE GRADIENT STOCHASTIQUE POUR UNE FONCTION CONVEXE

Nous allons maintenant nous intéresser à la convergence de l'algorithme dans le cas où la fonction à minimiser est convexe. Nous supposons également que le problème de minimisation est localisé sur la sphère euclidienne de \mathbb{R}^d de rayon D .

L'algorithme de descente de gradient stochastique peut être redéfini par :

$$\theta_n = \Pi_C(\theta_{n-1} - \gamma_n \nabla f(\theta_{n-1}) + \gamma_n \Delta M_n)$$

où Π_C est la projection euclidienne sur la sphère.

Théorème 5.3. *Supposons que*

$$\|\Delta M_n + \nabla f(\theta_{n-1})\|_\infty \leq B$$

Alors le choix $\gamma_n = \frac{2D}{B\sqrt{n}}$ donne

$$\mathbb{E}[f(\hat{\theta}_n)] - f(\theta^*) \leq \frac{2BD}{\sqrt{n}}$$

où

$$\hat{\theta}_n = \frac{1}{n} \sum_{k=0}^{n-1} \theta_k$$

Démonstration. Cette preuve fonctionne de façon similaire au cas déterministe.

Notons

$$\nabla f(\theta_{n-1}) - \Delta M_n = f'_n(\theta_{n-1})$$

où f'_n est une variable aléatoire centrée autour de $\nabla f(\theta_{n-1})$.

$$\begin{aligned} \|\theta_n - \theta^*\|^2 &= \|\Pi_C(\theta_{n-1} - \gamma_n f'_n(\theta_{n-1}) - \theta^*)\|^2 \\ &\leq \|\theta_{n-1} - \gamma_n f'_n(\theta_{n-1}) - \theta^*\|^2 \\ &\leq \|\theta_{n-1} - \theta^*\|^2 + B^2 \gamma_n^2 - 2\gamma_n \langle \theta_{n-1} - \theta^*, f'_n(\theta_{n-1}) \rangle \end{aligned}$$

On prend l'espérance conditionnellement à F_{n-1} et on en déduit :

$$\mathbb{E}[\|\theta_n - \theta^*\|^2 | F_{n-1}] \leq \|\theta_{n-1} - \theta^*\|^2 + B^2 \gamma_n^2 - 2\gamma_n \langle \theta_{n-1} - \theta^*, \nabla f_n(\theta_{n-1}) \rangle$$

Par ailleurs, la convexité de f donne

$$f(x) + \langle \nabla f(x), y - x \rangle \leq f(y)$$

Ce qui donne,

$$\mathbb{E}[\|\theta_n - \theta^*\|^2 | F_{n-1}] \leq \|\theta_{n-1} - \theta^*\|^2 + B^2\gamma_n^2 - 2\gamma_n[f(\theta_{n-1}) - f(\theta^*)]$$

Et donc, en prenant l'espérance totale des deux côtés,

$$\mathbb{E}[\|\theta_n - \theta^*\|^2] \leq \|\theta_{n-1} - \theta^*\|^2 + B^2\gamma_n^2 - 2\gamma_n\mathbb{E}[f(\theta_{n-1}) - f(\theta^*)]$$

Ce qui se réécrit,

$$\mathbb{E}[f(\theta_{n-1}) - f(\theta^*)] - f(\theta^*) \leq \frac{B^2\gamma_n}{2} + \frac{1}{2\gamma_n}[\mathbb{E}[\|\theta_{n-1} - \theta^*\|^2] - \mathbb{E}[\|\theta_n - \theta^*\|^2]]$$

Puis, en sommant,

$$\sum_{k=1}^n \mathbb{E}[f(\theta_{k-1})] - nf(\theta^*) \leq \sum_{k=1}^n \frac{B^2\gamma_k}{2} + \sum_{k=1}^n \frac{1}{2\gamma_k} \{\mathbb{E}[\|\theta_{k-1} - \theta^*\|^2] - \mathbb{E}[\|\theta_k - \theta^*\|^2]\}$$

La majoration $\mathbb{E}[\|\theta_{k-1} - \theta^*\|^2] \leq 4D^2$ donne alors

$$\sum_{k=1}^n \mathbb{E}[f(\theta_{k-1})] - nf(\theta^*) \leq \sum_{k=1}^n \frac{B^2\gamma_k}{2} + \frac{4D^2}{2\gamma_n}n \leq 2DB\sqrt{n} \quad \text{quand} \quad \gamma_n = \frac{2D}{B\sqrt{n}}$$

et donc finalement,

$$\mathbb{E}[f(\hat{\theta}_n)] - f(\theta^*) \leq \frac{2BD}{\sqrt{n}}$$

□

5.3 VITESSE DE CONVERGENCE DE LA DESCENTE DE GRADIENT STOCHASTIQUE POUR UNE FONCTION STRICTEMENT CONVEXE

On suppose maintenant f μ -fortement convexe.

Théorème 5.4. Si $\gamma_n = \frac{2}{\mu(n+1)}$, on a :

$$\mathbb{E}[f\left(\frac{1}{n} \sum_{k=1}^n \theta_{k-1}\right)] - f(\theta^*) \leq \frac{B^2 \log(n)}{2n\mu}$$

6 DESCENTE MIRROIR STOCHASTIQUE

6.1 INTRODUCTION

La descente miroir stochastique est une généralisation de la descente miroir déterministe pour résoudre le problème suivant :

$$Opt = \min_{u \in X} \{f(u) = \mathbb{E}[F(u, \xi)]\}$$

où $X \subset \mathbb{R}^n$ est un convexe fermé et borné, ξ un vecteur aléatoire de support $\Xi \in \mathbb{R}^d$, $F(\cdot, \xi)$ convexe.

On suppose de plus que

$$\mathbb{E}[F(x, \xi)] = \int_{\Xi} F(x, \xi) dP(\xi) dx$$

est bien définie.

La difficulté principale de ce problème est que la fonction f est implicite.

Historiquement, deux approches sont utilisées pour résoudre ce problème :

- La méthode SAA (Sample Average Approximation) qui consiste à générer ξ_1, \dots, ξ_N , N réalisations indépendantes de la variable aléatoire ξ et d'approximer la fonction f par son approximation :

$$\hat{f}_N(x) = N^{-1} \sum_{t=1}^N F(x, \xi_t)$$

- La méthode SA (Stochastic Approximation) que nous allons développer dans cette partie.

6.2 LA MÉTHODE DE DESCENTE MIRROIR STOCHASTIQUE

On suppose que l'espace \mathbb{R}^n est muni d'une norme $\|\cdot\|$.

On suppose qu'il existe une fonction génératrice de distance h , c'est-à-dire qu'elle est continûment différentiable α -convexe sur X , i.e

$$\forall x, x' \in X, \langle x' - x, \nabla h(x') - \nabla h(x) \rangle \geq \alpha \|x' - x\|^2$$

On redéfinit la distance de Bregman associée :

$$D_h(u, v) = h(u) - h(v) - \langle \nabla h(v), u - v \rangle$$

et la prox-mapping :

$$P_x(y) = \operatorname{argmin}_{z \in X} \{ \langle y, z - x \rangle + D_h(x, z) \}$$

ainsi que la constante

$$D_{h,X} = \sqrt{\max_{x \in X} h(x) - \min_{x \in X} h(x)}$$

Si x_1 est le minimiseur de h sur X , qui existe car X est compacte et h est continue sur X , on a donc $\forall x \in X, \langle x - x_1, \nabla h(x_1) \rangle \geq 0$.

On en déduit que :

$$\forall x \in X, \frac{1}{2}\|x - x_1\|^2 \leq D(x_1, X) \leq h(x) - h(x_1) \leq D_{h,X}^2$$

Ce qui donne :

$$\|x - x_1\| \leq \sqrt{\frac{2}{\alpha}} D_{h,X} := \Lambda_{h,X}$$

On suppose également que f est différentiable de gradient g et que l'on a à disposition une fonction G telle que $g(x) = \mathbb{E}[G(x, \xi)]$

(A) On suppose enfin que :

$$\text{Var}(F(x, \xi)) = \mathbb{E}[(f(x) - F(x, \xi))^2] \leq Q^2$$

$$\mathbb{E}[\|G(x, \xi)\|_*^2] \leq M_*^2$$

6.2.1 • L'ALGORITHME

L'algorithme de descente de gradient miroir stochastique est alors :

Algorithme Schéma de la descente de gradient proximale

Input : G et $(\gamma_t)_{t \in \mathbb{N}}$ suite de pas

Initialisation : On initialise à partir du point x_1

Itérer :

$$x_{t+1} = P_{x_t}(\gamma_t G(x_t, \xi_t))$$

Output : x_k vérifiant une certaine condition

6.2.2 • REGRET POUR LA MÉTHODE DE DESCENTE MIRROIR STOCHASTIQUE

On définit :

$$\nu_t = \frac{\gamma_t}{\sum_{i=1}^N \gamma_i} \quad \text{et} \quad \tilde{x}_N = \sum_{t=1}^N \nu_t x_t$$

Ainsi \tilde{x}_N est l'approximation moyenne générée par l'algorithme au bout de N pas.

Théorème 6.1. *On a alors la borne supérieure suivante :*

$$\mathbb{E}[f(\tilde{x}_N) - Opt] \leq \frac{D_{h,X}^2 + (2\alpha)^{-1}M^2 \sum_{t=1}^N \gamma_t^2}{\sum_{t=1}^N \gamma_t}$$

Supposons que le nombre N d'itérations est fixé.

Alors, en choisissant les pas de la façon suivante :

$$\gamma_t = \gamma = \frac{\theta\sqrt{2\alpha}D_{h,X}}{M\sqrt{N}}, \quad t = 1, \dots, N$$

avec une constante $\theta > 0$,

On a alors,

$$\mathbb{E}[f(\tilde{x}_N) - Opt] \leq \sqrt{2} \max(\theta, \theta^{-1}) \Lambda_{h,X} MN^{-1/2}$$

Par l'inégalité de Markov, on obtient donc :

$$\mathbb{P}[f(\tilde{x}_N) - Opt > \epsilon] \leq \frac{\sqrt{2} \max(\theta, \theta^{-1}) D_{h,X} M}{\epsilon \sqrt{\alpha N}}$$

6.3 DÉMONSTRATION DU THÉORÈME

On définit les fonctions :

$$\begin{aligned} f^N(x) &= \sum_{t=1}^N \nu_t \left[f(x_t) + g(x_t)^T (x - x_t) \right] \\ \hat{f}^N(x) &= \sum_{t=1}^N \nu_t \left[F(x_t, \xi_t) + G(x_t, \xi_t)^T (x - x_t) \right] \\ f_*^N &= \min_{x \in X} f^N(x) \quad \text{et} \quad f^{*N} = \sum_{t=1}^N \nu_t f(x_t) \end{aligned}$$

Comme $\sum_{t=1}^N \nu_t = 1$, il suit par la convexité de f que la fonction f^N sous-estime f partout sur X et donc $f_*^N \leq Opt$.

Par ailleurs, comme $\tilde{x}_N \in X$, on a $Opt \leq f(\tilde{x}_N)$ et par convexité de f , que $f(\tilde{x}_N) \leq f^{*N}$. Ainsi pour toute réalisation i.i.d de ξ_1, \dots, ξ_N , on a :

$$f_*^N \leq Opt \leq f(\tilde{x}_N) \leq f^{*N}$$

En passant à l'espérance,

$$\mathbb{E}[f_*^N] \leq Opt \leq \mathbb{E}[f^{*N}]$$

Nous allons démontrer le théorème suivant :

Théorème 6.2.

$$\mathbb{E}[f^{*N} - f_*^N] \leq \frac{D_{h,X}^2 + (\frac{5}{2}\alpha)^{-1}M^2 \sum_{t=1}^N \gamma_t^2}{\sum_{t=1}^N \gamma_t}$$

Nous aurons besoin du lemme suivant :

Lemme 6.3. Si $(\zeta_t) \in \mathbb{R}^n$, $v_1 \in X$ et $v_{t+1} = P_{v_t}(\zeta_t)$, $t = 1, \dots, N$, alors

$$\sum_{t=1}^N \zeta_t^T (v_t - u) \leq D(v_1, u) + (2\alpha)^{-1} \sum_{t=1}^N \|\zeta_t\|_*^2, \quad \forall u \in X$$

On définit $\delta_t = F(x_t, \xi_t) - f(x_t)$ et $\Delta_t = G(x_t, \xi_t) - g(x_t)$.

Démonstration. Si dans le lemme 1, on prend $v_1 = x_1$ et $\zeta_t = \gamma_t G(x_t, \xi_t)$, alors les itérées v_t correspondent aux itérées x_t par définition. On a donc d'après le lemme, comme $D(v_1, u) \leq D_{h,X}^2$,

$$\sum_{t=1}^N \gamma_t (x_t - u)^T G(x_t, \xi_t) \leq D_{h,X}^2 + (2\alpha)^{-1} \sum_{t=1}^N \gamma_t^2 \|G(x_t, \xi_t)\|_*^2, \quad \forall u \in X$$

On en déduit que pour tout $u \in X$,

$$\sum_{t=1}^N \nu_t \left[-f(x_t) + (x_t - u)^T g(x_t) \right] + \sum_{t=1}^N \nu_t f(x_t) \leq \frac{D_{h,X}^2 + (2\alpha)^{-1} \sum_{t=1}^N \gamma_t^2 \|G(x_t, \xi_t)\|_*^2}{\sum_{t=1}^N \gamma_t} + \sum_{t=1}^N \nu_t \Delta_t^T (x_t - u)$$

Comme $f^{*N} - f_*^N = \sum_{t=1}^N \nu_t f(x_t) + \max_{u \in X} \sum_{t=1}^N \nu_t \left[-f(x_t) + (x_t - u)^T g(x_t) \right]$,
On en déduit que

$$f^{*N} - f_*^N \leq \frac{D_{h,X}^2 + (2\alpha)^{-1} \sum_{t=1}^N \gamma_t^2 \|G(x_t, \xi_t)\|_*^2}{\sum_{t=1}^N \gamma_t} + \max_{u \in X} \nu_t \Delta_t^T (x_t - u)$$

Il reste donc à estimer le second terme de la partie droite de l'inégalité. Prenons

$$u_1 = v_1 = x_1; u_{t+1} = P_{u_t}(-\gamma_t \Delta_t); v_{t+1} = P_{v_t}(\gamma_t \Delta_t)$$

On observe que Δ_t est une fonction déterministe de ξ_t , et donc u_t et v_t sont des fonctions déterministes de ξ^{t_1} . En utilisant le lemme 1, on obtient donc :

$$\sum_{t=1}^N \gamma_t \Delta_t^T (v_t - u) \leq D_{h,X}^2 + (2\alpha)^{-1} \sum_{t=1}^N \gamma_t^2 \|\Delta_t\|_*^2, \quad \forall u \in X$$

Par ailleurs,

$$\Delta_t^T (v_t - u) = \Delta_t^T (x_t - u) + \Delta_t^T (v_t - x_t)$$

et donc,

$$\max_{u \in X} \sum_{t=1}^N \nu_t \Delta_t^T (x_t - u) \leq \sum_{t=1}^N \nu_t \Delta_t^T (x_t - v_t) + \frac{D_{h,X}^2 + (2\alpha)^{-1} \sum_{t=1}^N \gamma_t^2 \|G(x_t, \xi_t)\|_*^2}{\sum_{t=1}^N \gamma_t}$$

Par un raisonnement similaire appliqué à $-\Delta_t$, on obtient

$$\max_{u \in X} \left[- \sum_{t=1}^N \nu_t \Delta_t^T (x_t - u) \right] \leq \left[- \sum_{t=1}^N \nu_t \Delta_t^T (x_t - u_t) \right] + \frac{D_{h,X}^2 + (2\alpha)^{-1} \sum_{t=1}^N \gamma_t^2 \|\Delta_t\|_*^2}{\sum_{t=1}^N \gamma_t}$$

De plus, $\mathbb{E}_{t-1}[\Delta_t] = 0$ et u_t , v_t et x_t sont des fonctions de ξ_{t-1} et donc

$$\mathbb{E}[(x_t - v_t)^T \Delta_T] = \mathbb{E}[(x_t - u_t)^T \Delta_T] = 0$$

Enfin, nous avons aussi $\mathbb{E}[\|\Delta_T\|_*^2] \leq 4M_*^2$ par hypothèse, ce qui donne en passant à l'espérance dans l'inégalité,

$$\mathbb{E} \left[\max_{u \in X} \sum_{t=1}^N \nu_t \Delta_t^T (x_t - u) \right] \leq \frac{D_{h,X}^2 + (2\alpha)^{-1} \|M\|_*^2 \sum_{t=1}^N \gamma_t^2}{\sum_{t=1}^N \gamma_t}$$

Ce qui conclut la preuve. \square

7

APPLICATION DE LA DESCENTE MIRROIR À L'OPTIMISATION D'UN PORTEFEUILLE

7.1 INTRODUCTION DU PROBLÈME

On s'intéresse au problème d'optimisation du profit d'un portefeuille d'actions.

Ce portefeuille contient m actions dont les valeurs à un temps T sont représentées par un vecteur aléatoire $Z = (Z_0, \dots, Z_n)$. Nous supposons pour simplifier que les valeurs des actions sont normalisées à 1 au temps $T = 0$.

Une stratégie d'investissement correspond à l'allocation d'un capital initial, modélisé par un m-uplet de poids positifs (u_0, \dots, u_n) dont la somme est égale à 1. Ce m-uplet appartient au simplexe de dimensions m , noté Δ_m et défini par :

$$\Delta_m = \left\{ u \in \mathbb{R}_+^m : \sum_{i=1}^m u_i = 1 \right\}$$

Nous cherchons à trouver l'investissement optimal afin de maximiser le profit moyen au bout d'un temps T .

Plus formellement, nous sommes intéressé par le problème d'optimisation sous contraintes de la moyenne de la variable aléatoire :

$$\langle Z, u \rangle = \sum_{i=1}^m u_i Z_i$$

dans le simplexe Δ_m .

7.2 FORMALISATION DU PROBLÈME POUR LA DESCENTE MIRROIR SANS CONTRAINTE DE RISQUE

Nous cherchons tout d'abord à trouver l'investissement optimal afin de maximiser le profit moyen au bout d'un temps T sans contraintes.

On considère donc le problème stochastique suivant :

$$Opt = \min_{u \in \Delta_m} \{f(u) = \mathbb{E}[F(u, Z)]\}$$

où $F(u, Z) = \langle Z, u \rangle$.

7. APPLICATION DE LA DESCENTE MIRROIR À L'OPTIMISATION D'UN PORTEFEUILLE

On introduit la fonction strictement convexe suivante :

$$\forall u \in \Delta_m, \quad \phi(u) = \sum_{i=1}^m u_i \log u_i$$

On définit la divergence de Bregman associée $D_\phi(u, v)$ définie par :

$$\forall (u, v) \in \Delta_m^2, D_\phi(u, v) = \phi(u) - \phi(v) - \langle \nabla \phi(v), u - v \rangle$$

Lemme 7.1.

$$\forall (u, v) \in \Delta_m^2, D_\phi(u, v) = \sum_{i=1}^m u_i \log \left(\frac{u_i}{v_i} \right) - \sum_{i=1}^m u_i + \sum_{i=1}^m v_i$$

Démonstration. Tout d'abord, on calcule $\nabla \phi(x) = (1 + \log u_i)_{1 \leq i \leq m}$

$$\begin{aligned} D_\phi(u, v) &= \sum_{i=1}^m u_i \log(u_i) - \sum_{i=1}^m v_i \log(v_i) - \sum_{i=1}^m (\log(v_i) + 1)(u_i - v_i) \\ &= \sum_{i=1}^m u_i \log(u_i) - \sum_{i=1}^m u_i \log(v_i) - \sum_{i=1}^m u_i - \sum_{i=1}^m v_i \\ &= \sum_{i=1}^m u_i \log \left(\frac{u_i}{v_i} \right) - \sum_{i=1}^m u_i - \sum_{i=1}^m v_i \end{aligned}$$

□

Théorème 7.2. Fixons $y \in \mathbb{R}^m$.

Alors $D_\phi(x, y)$ est minimisée pour $x \in \Delta_m$ au point

$$x = \frac{y}{\|y\|_1}$$

Démonstration. On veut trouver x vérifiant :

$$x = \arg \min_{u \in \Delta_m} \left\{ \sum_{i=1}^m u_i \log \left(\frac{u_i}{y_i} \right) - \sum_{i=1}^m u_i + \sum_{i=1}^m y_i \right\}$$

On pose le lagrangien :

$$\mathcal{L}(u, \lambda) = \sum_{i=1}^m u_i \left(\log \left(\frac{u_i}{y_i} \right) - 1 \right) + \sum_{i=1}^m y_i - \lambda \left(\sum_{i=1}^m u_i - 1 \right)$$

Les conditions d'optimalité $\frac{\partial \mathcal{L}}{\partial u_i}$ donnent :

$$u_i = y_i e^{\lambda-1} \quad \lambda = \frac{1}{\log \sum_{i=1}^m y_i} + 1$$

Ce qui donne le résultat en substituant λ . □

7. APPLICATION DE LA DESCENTE MIRROIR À L'OPTIMISATION D'UN PORTEFEUILLE

L'ensemble de ces résultats donne l'algorithme suivant :

Algorithme

Input : y^1 et $(\gamma_t)_{t \in \mathbb{N}}$ suite de pas

Initialisation : On initialise à partir du point $x^1 = \frac{y^1}{\|y^1\|_1}$

Itérer :

$$y_i^{t+1} = y_i^t e^{-\gamma_t \nabla_i F(x^t, \xi_t)}$$

$$x^{t+1} = \frac{y^{t+1}}{\|y^{t+1}\|_1}$$

Output : x^t vérifiant une certaine condition

7.3 AJOUT D'UNE CONTRAINTE DE RISQUE

On ajoute maintenant une contrainte de mesure de risque, qui quantifie la moyenne de la perte sachant que la perte a eu lieu. Pour un risque α , on définit le $1 - \alpha$ -quantile statistique :

$$\text{V@R}_\alpha(u) = \sup\{q \in \mathbb{R} : \mathbb{P}(\langle Z, u \rangle \leq q) \leq \alpha\}$$

La contrainte CV@R_α est définie par :

$$\text{CV@R}_\alpha(u) = \inf_{\tau \in \mathbb{R}} \{\tau + (1 - \alpha)^{-1} \mathbb{E}[\langle Z, u \rangle - \tau]^+\}$$

On peut montrer que $\text{CV@R}_\alpha(u)$ peut s'écrire comme l'espérance de la perte sachant que la perte à α a eu lieu :

$$\text{CV@R}_\alpha(u) = \mathbb{E}[\mathbb{E}[\langle Z, u \rangle] - \langle Z, u \rangle | \langle Z, u \rangle \leq \text{V@R}_\alpha(u)]$$

Nous pouvons donc réécrire le problème d'optimisation pour une perte fixée M ,

$$\begin{aligned} P_M &= \arg \max_{u \in \Delta_m} \left\{ \sum_{i=1}^m u_i \mathbb{E}[Z_i] : \text{CV@R}_\alpha(u) \leq M \right\} \\ &= \arg \min_{u \in \Delta_m} \left\{ - \sum_{i=1}^m u_i \mathbb{E}[Z_i] : \text{CV@R}_\alpha(u) \leq M \right\} \end{aligned}$$

On définit également le problème pour λ fixé :

$$Q_\lambda = \arg \min_{u \in \Delta_m} \{-\mathbb{E}[\langle Z, u \rangle] + \lambda \text{CV@R}_\alpha(u)\}$$

Théorème 7.3. *Les problèmes $(P_M)_M$ et $(Q_\lambda)_\lambda$ sont équivalents. C'est-à-dire que pour $M > 0$, si u_M^* est une solution de P_M , il existe $\lambda_M^* > 0$ tel que :*

$$u_M^* = \arg \min_{u \in \Delta_m} \{-\mathbb{E}[\langle Z, u \rangle] + \lambda_M^* \text{CV@R}_\alpha(u)\}$$

7. APPLICATION DE LA DESCENTE MIRROIR À L'OPTIMISATION D'UN PORTEFEUILLE



Démonstration. On pose $J(u) = -\sum_{i=1}^m u_i \mathbb{E}[Z_i]$, qui est une fonction convexe continue. P_M est un problème d'optimisation convexe défini avec la contrainte $CV@R_\alpha \leq M$. On définit

$$g_M(u) = CV@R_\alpha(u) - M$$

. Montrons que g_M est une fonction convexe.

En effet $\forall \lambda \in [0, 1], \forall (u, v) \in \Delta_m$,

$$\begin{aligned} CV@R_\alpha(\lambda u + (1 - \lambda)v)) &\leq CV@R_\alpha(\lambda u) + CV@R_\alpha((1 - \lambda)v) \\ &= \lambda CV@R_\alpha(u) + (1 - \lambda)CV@R_\alpha(v) \end{aligned}$$

On définit alors le lagrangien :

$$L_M(u, \lambda) = J(u) + \lambda g_M(u)$$

Comme J est finie sur Δ_m , elle possède un minimum et donc les conditions de Kuhn et Tucker sont vérifiées :

$$\begin{aligned} \exists \lambda_M^* \geq 0, u_M^* &= \arg \min_{u \in \Delta_m} L(u, \lambda_M^*) \\ \lambda_M^* g_M(u_M^*) &= 0 \end{aligned}$$

ce qui fixe λ_M^* et montre par conséquent qu'il est équivalent de résoudre le problème P_M que le problème $\operatorname{argmin}_{u \in \Delta_m} \{J(u) + \lambda_M^* g_M(u)\}$ qui est donc équivalent à $\operatorname{argmin}_{u \in \Delta_m} \{J(u) + \lambda_M^* CV@R_\alpha(u)\}$ \square

On peut donc réécrire le problème comme la minimisation de la fonction :

$$p_\lambda(u, \theta) = -\mathbb{E}[\langle Z, u \rangle] + \lambda \left(\theta + \frac{1}{1-\alpha} \mathbb{E}[\langle Z, u \rangle - \theta]^+ \right)$$

C'est-à-dire :

$$Q_\lambda = \operatorname{argmin}_{(u, \theta) \in \Delta_m \times \mathbb{R}} \{p_\lambda(u, \theta)\}$$

Ce travail théorique permet alors d'élaborer l'algorithme suivant :

Algorithme

Input : Suite de pas $(\nu_k)_{k \in \mathbb{N}}$, $U_0 \in \mathbb{R}$, $\theta_0 \in \mathbb{R}$, $\alpha \in [0, 1]$

Initialisation : On initialise à partir du point $X_0 = (U_0, \theta_0)$

For k = 0 to N do :

- Simuler Z^{k+1}
- Calculer un pas de gradient stochastique :

$$\begin{aligned} g_{k+1,1} &= -Z^{k+1} + \frac{\lambda}{1-\alpha} Z^{k+1} \mathbb{1}_{(Z^{k+1}, U_k) \geq \theta_k} \\ g_{k+1,2} &= \lambda \left[1 - \frac{1}{1-\alpha} \mathbb{1}_{(Z^{k+1}, U_k) \geq \theta_k} \right] \end{aligned}$$

- Calculer la projection de X_{k+1} sur $\Delta_m \cdot \mathbb{R}$ d'après la divergence de Bregman :

$$\begin{aligned} U^{k+1} &= \frac{U_k e^{-\nu_{k+1} g_{k+1,1}}}{\|U_k e^{-\nu_{k+1} g_{k+1,1}}\|} \\ \theta^{k+1} &= \theta^k - \nu_{k+1} g_{k+1,2} \end{aligned}$$

$$X^{k+1} = (U_{k+1}, \theta^{k+1})$$

Output : X^N

7.4 RÉSULTATS EXPÉRIMENTAUX

On modélise les actions par des processus stochastiques :

$$\forall i \in [1, \dots, m], \quad S_t^i = S_0^i e^{\mu_i t + \sigma_i B_i(t)} - S_0^i$$

où les $(B_i(t))_{1 \leq i \leq m}$ sont des mouvements browniens indépendants.

Dans notre expérience, nous avons pris des actions de paramètres :

$$\mu = [0.1, 0.12, 0.15, 0.18, 0.2]$$

$$\sigma = [0, 0.12, 0.14, 0.17, 0.2]$$

Une simulation des courbes données par ces actions est donnée ci-dessous :

7. APPLICATION DE LA DESCENTE MIRROIR À L'OPTIMISATION D'UN PORTEFEUILLE

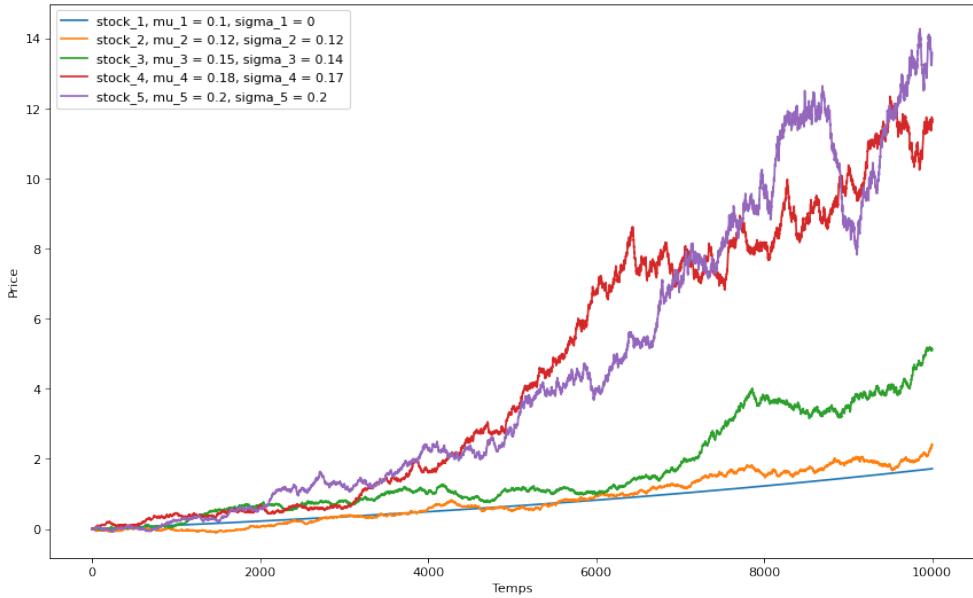


FIGURE 9 – Simulation de la valeur des actions au cours du temps

7.4.1 • OPTIMISATION DE LA VALEUR DU PORTEFEUILLE DANS LE CAS SANS CONTRAINTES

L'algorithme de la partie 7.2 donne la simulation suivante :

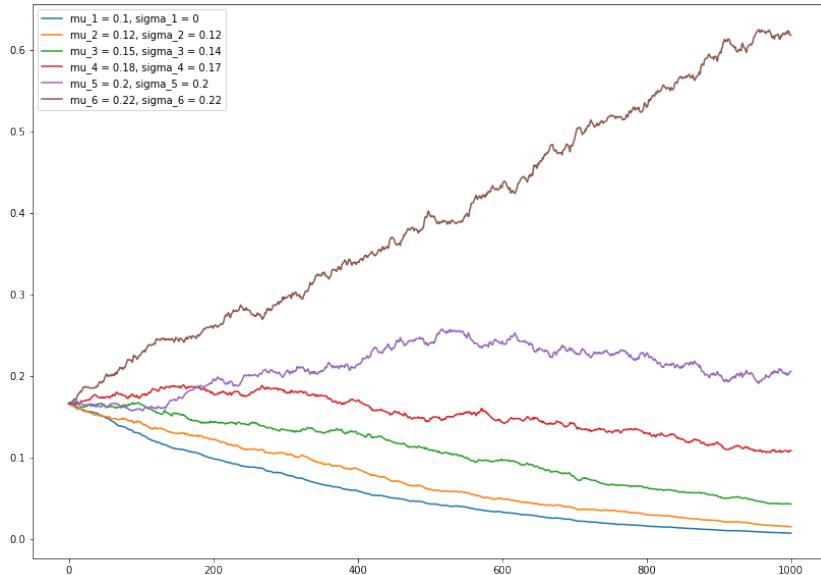


FIGURE 10 – Simulation de la répartition du portefeuille suivant le nombre d'itérations

et une valeur du portefeuille :

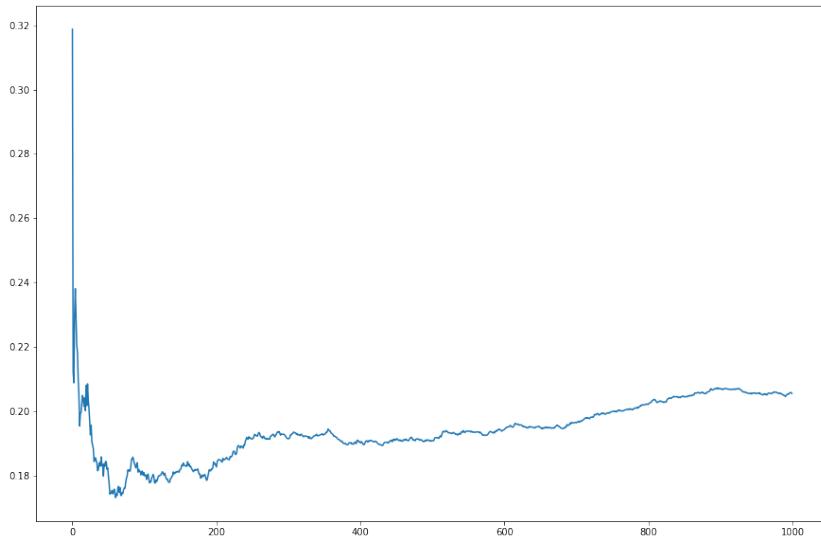


FIGURE 11 – Simulation de la valeur moyenne du portefeuille suivant le nombre d'itérations

On observe que le portefeuille est répartie de façon maximal sur l'actif qui rapporte le plus. Ceci est d'autant plus évident lorsqu'on considère un portefeuille de paramètres :

$$\mu = [3, 1, 1, 1, 1]$$

$$\sigma = [1, 1, 1, 1, 1]$$

qui donne un poids de 1 sur l'actif manifestement le plus rentable :



FIGURE 12 – Simulation de la répartition du portefeuille au cours des itérations

7.4.2 • OPTIMISATION DE LA VALEUR DU PORTEFEUILLE DANS LE CAS AVEC CONTRAINTES DE RISQUE

Néanmoins l'histoire est toute autre lorsqu'on considère une contrainte de risque comme dans la partie 7.3. On considère ici un risque à 5% donc $\alpha = 0.05$ et une valeur $\lambda = 0.9$ associée à une perte.

En effet, dans ce cas, l'algorithme de la section 7.3 donne :

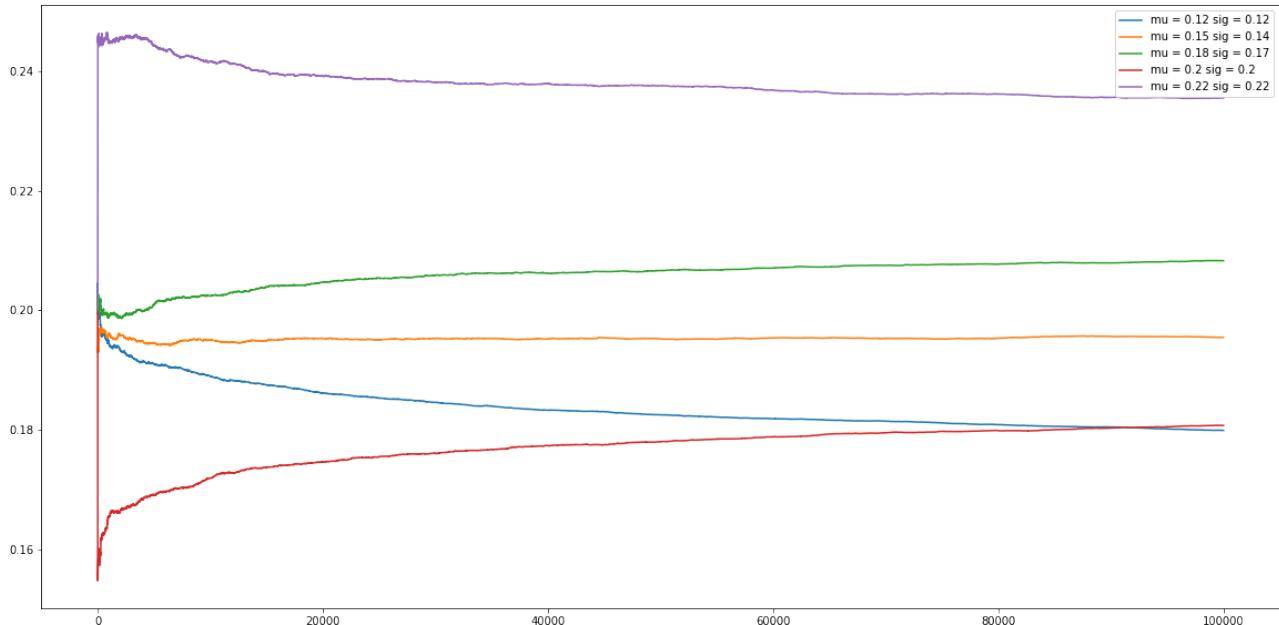


FIGURE 13 – Simulation de la répartition du portefeuille au cours des itérations

On voit ici que la répartition du portefeuille tend vers des poids comparables sur tous les actifs, ce qui justifie la pratique courante de diversifier son portefeuille pour diminuer le risque de perte.

On obtient alors les valeurs de risque et de portefeuille moyennes suivantes (voir page suivante) :

7. APPLICATION DE LA DESCENTE MIRROIR À L'OPTIMISATION D'UN PORTEFEUILLE

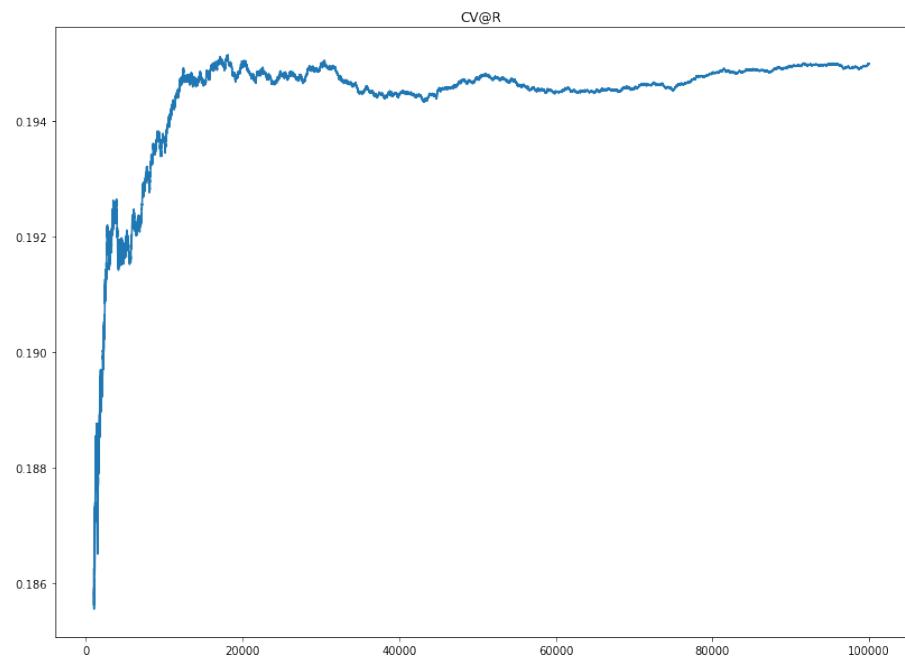


FIGURE 14 – CV@R

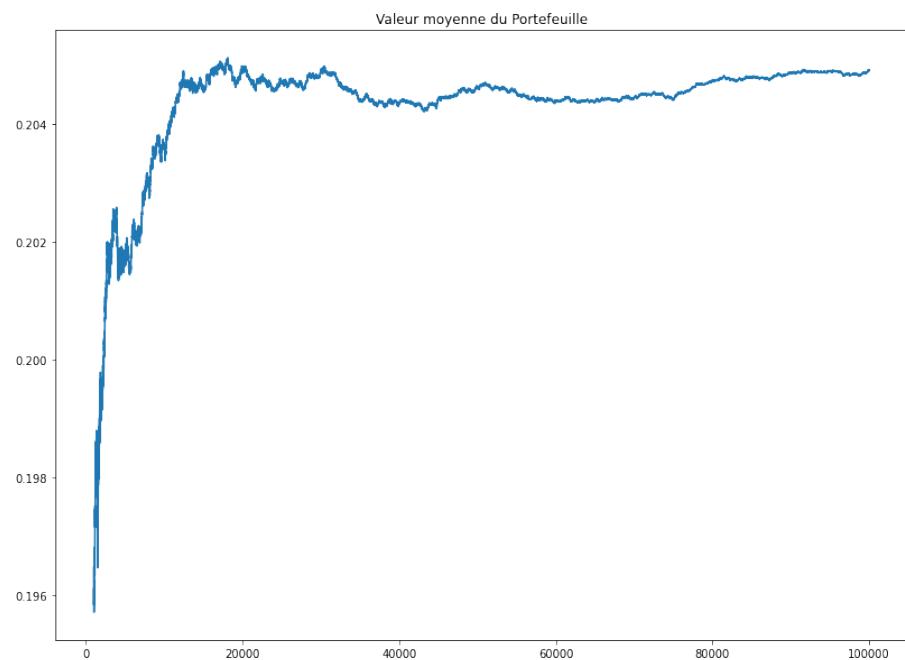


FIGURE 15 – Valeur moyenne du portefeuille

8

REMERCIEMENTS

Sébastien Gadat a supervisé l'avancée du projet tout au long de la période. Nous le remercions chaleureusement pour son aide précieuse tant sur la conduite générale du projet, que sur les aspects théoriques et techniques.

Nous avons pu découvrir une généralisation des méthodes classiques enseignées l'année dernière ainsi que l'intérêt d'une telle approche sur la résolution de problèmes concrets. Ce projet aura sans aucun doute été bénéfique pour nous, ne serait-ce que comme une première confrontation à la recherche en mathématiques appliquées.

RÉFÉRENCES

- [1] Sébastien Gadat. *Master Eco Stat Magistère UT1 - Optimisation for Big Data*. 2017.
- [2] Alexander Shapiro Guanghui Lan, Arkadi Nemirovski. *Validation analysis of mirror descent stochastic approximation method*. Springer, 2011.
- [3] Anupam Gupta. *Advanced Algorithms, Fall 2020*.
- [4] Lorick Huang Manon Costa, Sébastien Gadat. *Portfolio optimization under CV@R constraint with stochastic mirror descent*. 2022.

Validation analysis of mirror descent stochastic approximation method

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import time
from scipy.stats import wasserstein_distance
```

Preparation du jeu de données

Avant de lancer la cellule suivante, il faut télécharger le fichier [preprocessed_CAC40.csv](#) (<https://drive.google.com/file/d/1CSZYxJRCnYnspffZ9Hv8FyzvXBdPf1zM/view?usp=sharing>) \ Ensuite, uploader le fichier sur colab dans *fichiers*.

```
In [2]: df = pd.read_csv('preprocessed_CAC40.csv')
df = df.drop(columns=['Unnamed: 0'])
df.head(5)
```

Out[2]:

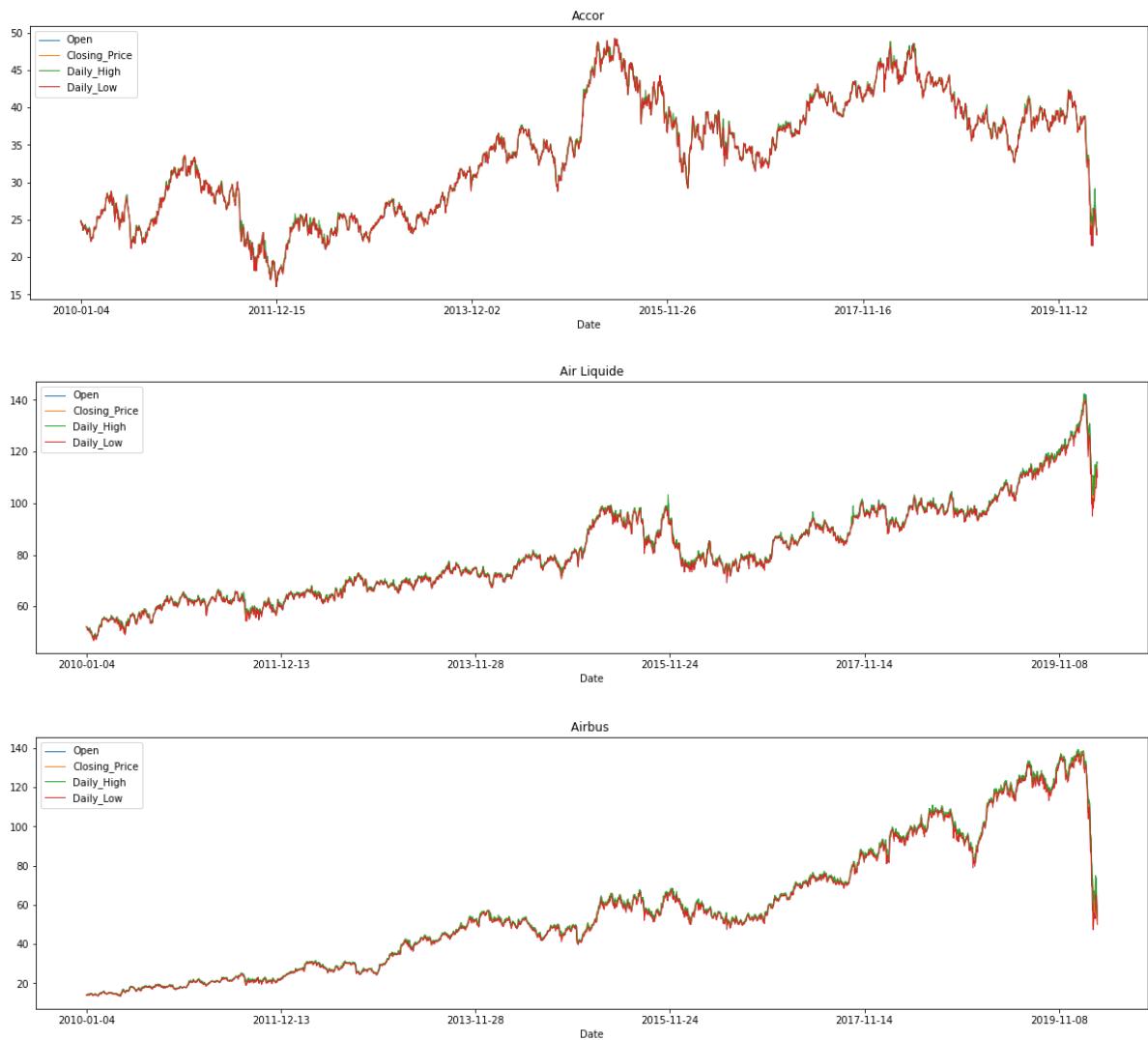
	Name	Date	Open	Closing_Price	Daily_High	Daily_Low	Volume
0	Accor	2020-04-03	22.99	23.40	23.40	22.99	67
1	Accor	2020-04-02	23.91	22.99	23.91	22.99	250
2	Accor	2020-04-01	24.10	23.83	24.10	23.83	37
3	Accor	2020-03-31	25.04	25.00	25.24	24.99	336
4	Accor	2020-03-30	26.50	25.02	26.50	24.99	415

```
In [3]: entreprises = df.Name.unique()
print(len(entreprises))
entreprises
```

38

```
Out[3]: array(['Accor', 'Air Liquide', 'Airbus ', 'ArcelorMittal', 'Atos',
   'AXA',
   'BNP Paribas', 'Bouygues', 'Cap Gemini', 'Crédit Agricole',
   'Danone', 'Dassault Systèmes', 'Engie (ex GDF Suez',
   'EssilorLuxottica', 'Hermès (Hermes International', 'Kering
   ',
   'LEGRAND', 'L'Oréal', 'LVMH Moet Hennessy Louis Vuitton',
   'Michelin (Compagnie Générale d Etablissements Michelin SCP
A',
   'Orange', 'Pernod Ricard', 'Peugeot', 'Publicis', 'Renault'
   ,
   'SAFRAN', 'Saint-Gobain', 'Sanofi', 'Schneider Electric',
   'Société Générale (Societe Generale', 'Sodexo',
   'STMicroelectronics', 'TOTAL', 'Unibail-Rodamco',
   'Veolia Environnement', 'VINCI', 'Vivendi', 'Worldline SA']
   ,
   dtype=object)
```

```
In [4]: i = 0
for entreprise in entreprises :
    data = df.loc[df['Name']==entreprise]
    data = data.iloc[::-1]
    data.plot(x = 'Date', y = ['Open','Closing_Price','Daily_High',
'Daily_Low'], title=entreprise, linewidth=1, figsize=(20,5))
    i+=1
    if i == 3 :
        break
```



Descente de gradient classique

Implémentation

```
In [5]: #Implémentation de la descente de gradient projeté
#On suppose que le gradient de la fonction f est donné par la fonction df
def Descente_Gradient(f, df, x0, pas, proj, erreur=1e-6, max_iter=1000):
    grad = df(x0)
    chemin = []
    chemin.append(x0)
    x=x0
    i = 0
    temps = []
    start = time.time()
    while abs(np.linalg.norm(grad)) > erreur and i < max_iter:
        grad = df(x)
        x = proj(x - pas[i]*grad)
        temps.append(time.time() - start)
        chemin.append(x)
        i+=1
        print('Le point final est ({},{}). \nLa valeur de f en ce point est :{}. \nLa valeur du gradient en ce point est ({},{}).'.format(x[0],x[1],f(x),grad[0],grad[1]))
        print('Nombre d\'itreration : {}'.format(i))
    print(len(temps))
    return x, chemin, temps
```

Exemple simple sur une fonction convexe classique

```
In [6]: #Implémentation des fonctions du problème

def f(x):
    return np.array(np.sin(x[0]**2 * x[1]**2))

def df(x):
    return np.array([2*x[0]*np.cos(x[0]**2 * x[1]**2), 2*x[1]*np.cos(x[0]**2 * x[1]**2)])

def proj(x):
    return x
```

```
In [7]: #Affichage de deux descentes de gradient pour deux points de départ distincts

fig = plt.figure(figsize=(13, 13), dpi=80)
ax = fig.add_subplot(111, projection='3d')

x0 = np.array((1,1)) # premier point de départ
max_iter=1000
pas = [0.003 for i in range(max_iter)] #suite de pas constant égaux
à 0.003
_, X, temps_dg = Descente_Gradient(f,df, x0, pas, proj, max_iter=max_iter)
X1 = X
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax.scatter(X, Y, Z, marker='x', color='r', label='Gradient_descent')

#affichage de la surface en trois dimensions
X = np.linspace(-1, 1, 200)
Y = np.linspace(-1, 1, 200)
Z = np.array([[f(np.array([X[i],Y[j]])) for i in range(len(X))] for j in range(len(Y))])
X, Y = np.meshgrid(X, Y)
ax.plot_wireframe(X, Y, Z, rstride=5, cstride=5, linewidth=1, alpha=0.6, label='f')

ax.legend()
ax.view_init(40, 110)
plt.show()
```

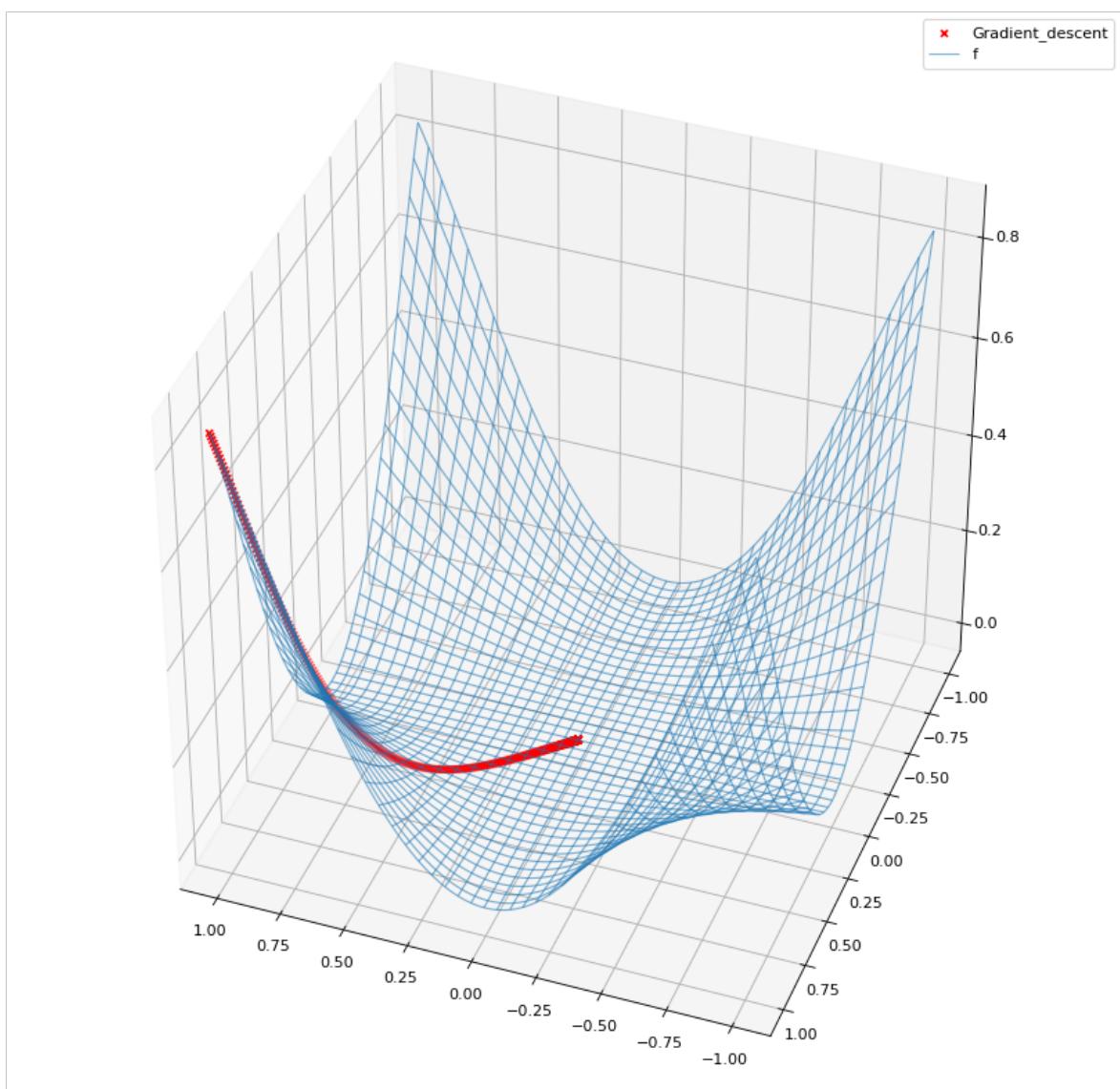
Le point final est (0.00264439613777353, 0.00264439613777353).

La valeur de f en ce point est : 4.889968446411675e-11.

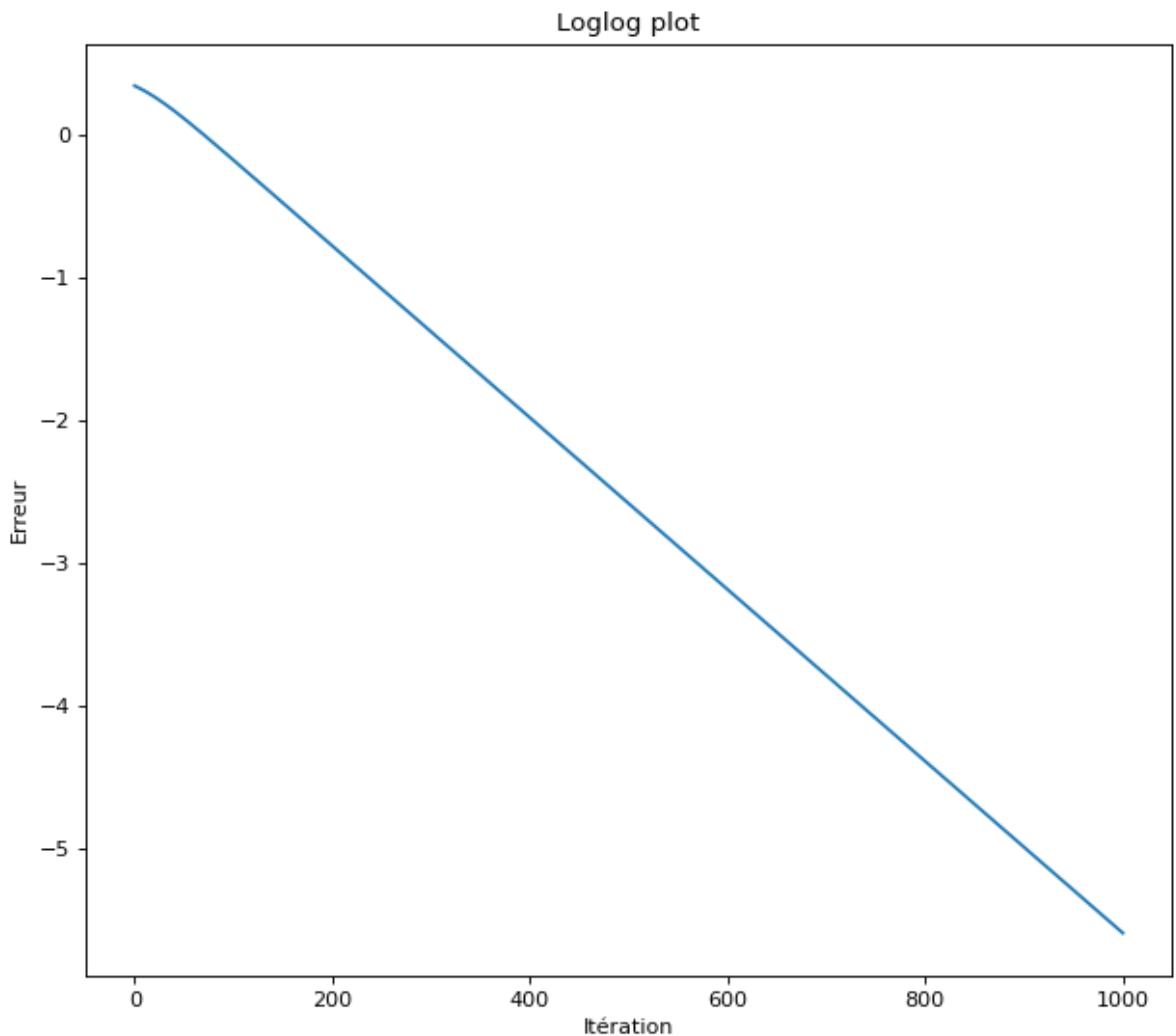
La valeur du gradient en ce point est (0.005320716574997042, 0.005320716574997042).

Nombre d'itération : 1000.

1000



```
In [8]: Erreur_dg = np.linalg.norm(X1, axis=1)
plt.figure(figsize=(20, 8), dpi=80)
ax1 = plt.subplot(121)
ax1.plot(np.arange(len(X1)), np.log(Erreur_dg))
ax1.set_title('Loglog plot')
ax1.set_xlabel('Itération')
ax1.set_ylabel('Erreur')
plt.show()
```



Exemple simple sur une fonction avec un point col

In [9]: #Implémentation des fonctions du problème

```
def f(x):
    return np.array(x[0]**2 - x[1]**2)

def df(x):
    return np.array([2*x[0], -2*x[1]])

def proj(x):
    return x
```

```
In [10]: #Affichage de deux descentes de gradient pour deux points de départ distincts

fig = plt.figure(figsize=(13, 13), dpi=80)
ax = fig.add_subplot(111, projection='3d')

x0 = np.array((9,0)) # premier point de départ
max_iter=100
pas = [0.03 for i in range(max_iter)] #suite de pas constant égaux à 0.03
_, X, temps_dg = Descente_Gradient(f,df, x0, pas, proj, max_iter=max_iter)
X1 = X
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax.scatter(X, Y, Z, marker='x', color='r', label='Gradient_descent')

x0 = np.array((9,1)) # deuxième point de départ
max_iter=36
pas = [0.03 for i in range(max_iter)] #suite de pas constant égaux à 0.03

_, X, t_ = Descente_Gradient(f,df, x0, pas, proj, max_iter=max_iter)
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax.scatter(X, Y, Z, marker='x', color='g', label='Gradient_descent')

#affichage de la surface en trois dimensions
X = np.linspace(-10, 10, 100)
Y = np.linspace(-10, 10, 100)
Z = np.array([[f(np.array([X[i],Y[j]])) for i in range(len(X))] for j in range(len(Y))])
X, Y = np.meshgrid(X, Y)
ax.plot_wireframe(X, Y, Z, rstride=5, cstride=5, linewidth=1, alpha=0.6, label='f')

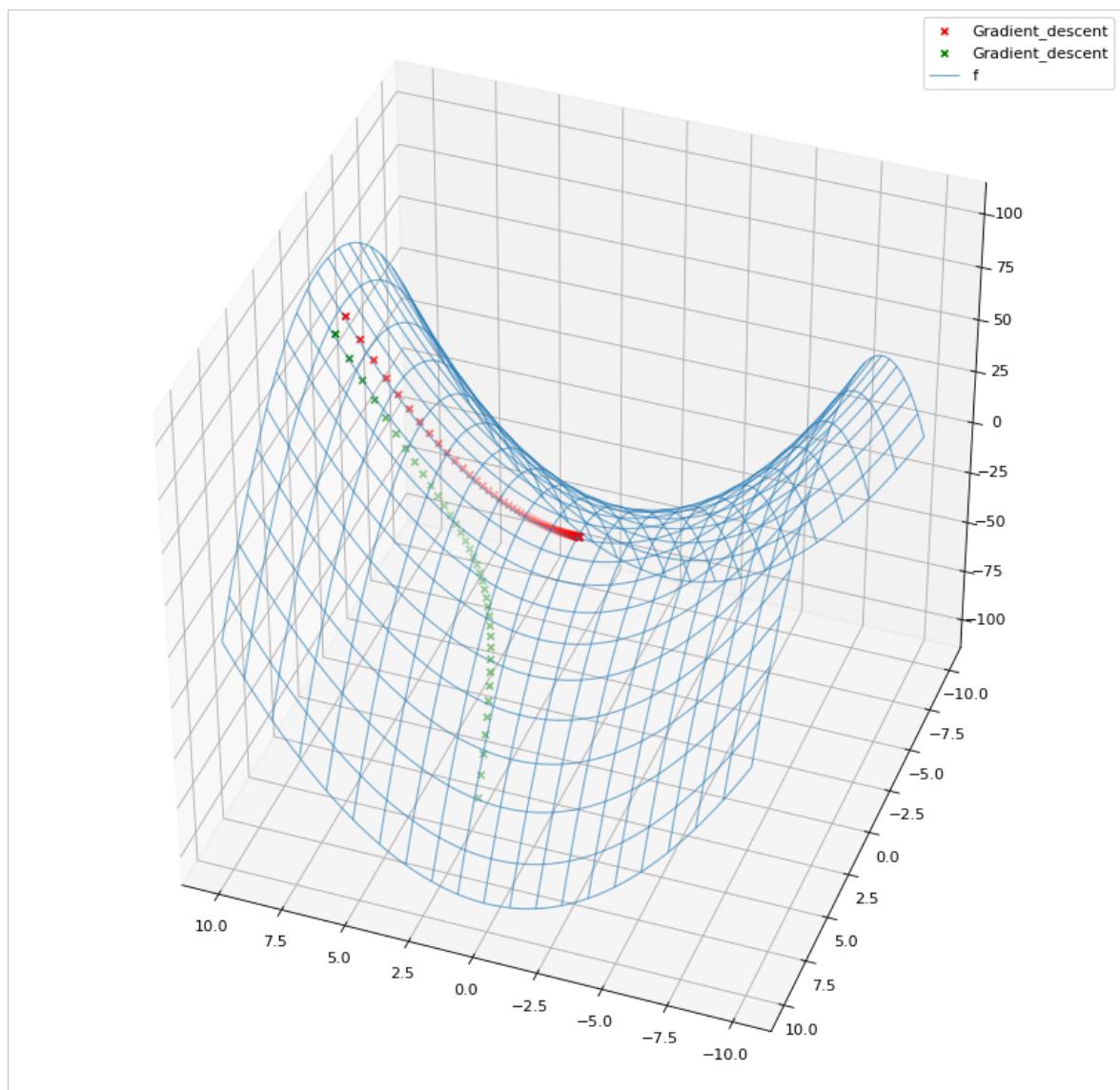
ax.legend()
ax.view_init(40, 110)
plt.show()
```

Le point final est (0.018493872934712382, 0.0).
La valeur de f en ce point est : 0.00034202333612528716.
La valeur du gradient en ce point est (0.03934866581853698, -0.0).
Nombre d'itération : 100.

100

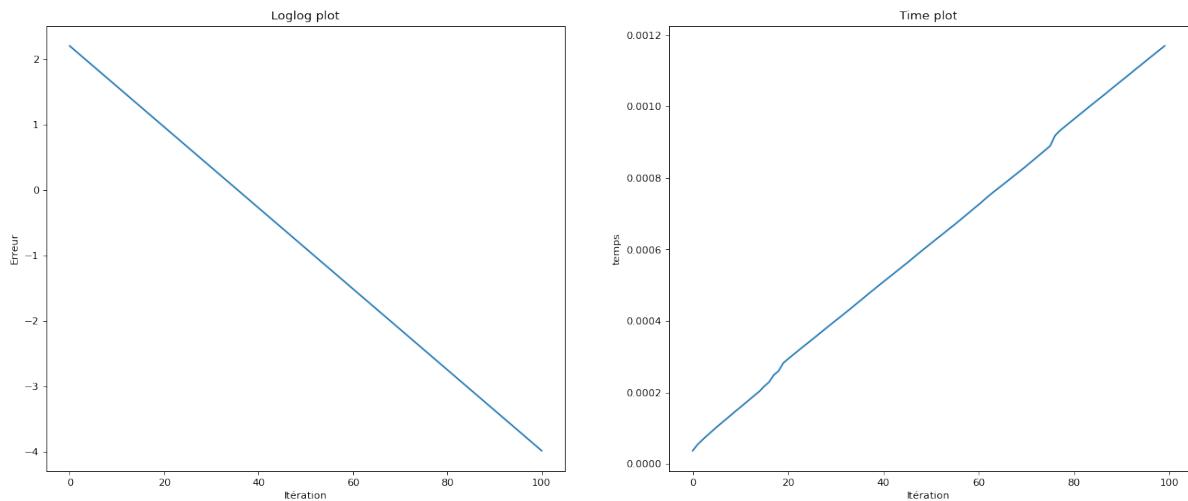
Le point final est (0.9701641873347264, 8.147251999851074).
La valeur de f en ce point est : -65.43649659869047.
La valeur du gradient en ce point est (2.0641791219887797, -15.3721
73584624669).
Nombre d'itération : 36.

36



Loglog plot de l'exemple précédent

```
In [11]: Erreur_dg = np.linalg.norm(X1, axis=1)
plt.figure(figsize=(20, 8), dpi=80)
ax1 = plt.subplot(121)
ax1.plot(np.arange(len(X1)), np.log(Erreur_dg))
ax1.set_title('Loglog plot')
ax1.set_xlabel('Itération')
ax1.set_ylabel('Erreur')
ax2 = plt.subplot(122)
ax2.set_title('Time plot')
ax2.plot(np.arange(len(X1)-1), temps_dg)
ax2.set_xlabel('Itération')
ax2.set_ylabel('temps')
plt.show()
```



On retrouve bien une vitesse de convergence exponentielle : cas de la minimisation d'une fonction strictement convexe.

Descente de gradient miroir

Rappel théorique

On cherche à déterminer :

$$x^* = \arg \min_{x \in X} f(x)$$

où X est un ensemble non-vide convexe compacte de \mathbb{R}^n et $f : X \rightarrow \mathbb{R}$: convexe Lipschitz continue.

On définit la divergence de Bregman $V : X \times X \rightarrow \mathbb{R}_+$:

$$V_w(x, y) = w(y) - w(x) - \nabla w(x)^T(y - x)$$

où $w : X \rightarrow \mathbb{R}$ est une fonction convexe.

L'algorithme de la descente miroir (MDA) est comme suit :

$$x_{t+1} = \nabla \phi^*(\nabla w(x_k) - \alpha_k \nabla f(x_k))$$

avec $\phi^* = \sup_{z \in X} [\langle z, x \rangle - w(z)]$

Implémentation

On considère pour les calculs la divergence de Bregman associée à la fonction convexe :

$$h(x) = \sum_{i=1}^n x_i \log(x_i) - x_i$$

également appelée entropie.

```
In [12]: def Descente_Miroir(f, df, grad_h, grad_h_inverse, x0, pas, proj,
erreur=1e-6, max_iter=1000):
    x=x0
    chemin = []
    chemin.append(x0)
    i = 0
    print(np.linalg.norm(df(x)))
    temps = []
    start = time.time()
    while i < max_iter and np.linalg.norm(df(x)) >= erreur :
        theta = grad_h(x)
        theta = theta - pas[i]*df(x)
        x1 = grad_h_inverse(theta)
        x = proj(x1)
        temps.append(time.time() - start)
        chemin.append(x)
        i+=1

    print('Nombre d\'itreration : ', i)
    return x, chemin, temps
```

In [13]: #implémentation des fonctions du problème

```
def f(x):
    return np.array(np.sin(x[0]**2 * x[1]**2))

def df(x):
    return np.array([2*x[0]*np.cos(x[0]**2 * x[1]**2), 2*x[1]*np.co
s(x[0]**2 * x[1]**2)])

def h(x) :
    return np.sum(x*np.log(x) - x)

def grad_h(x):
    return np.log(x)

def grad_h_inverse(x):
    return np.exp(x)

def Bregman(w, grad_w ,x,y):
    return w(y) - w(x) - grad_w(x).T@(y - x)

def proj(x):
    return x
```

```
In [14]: fig = plt.figure(figsize=(13, 13), dpi=80)
ax = fig.add_subplot(111, projection='3d')

x0 = np.array((1,1)) # point de départ

max_iter=100
pas = [0.003 for i in range(max_iter)]
_, X, temps_dm = Descente_Mirroir(f, df, grad_h, grad_h_inverse, x0
, pas, proj, 1e-3, max_iter=max_iter)
X1 = X
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax.scatter(X, Y, Z, marker='x', color='r', label='Mirror_descent')

x0 = np.array((1,0.5)) # point de départ

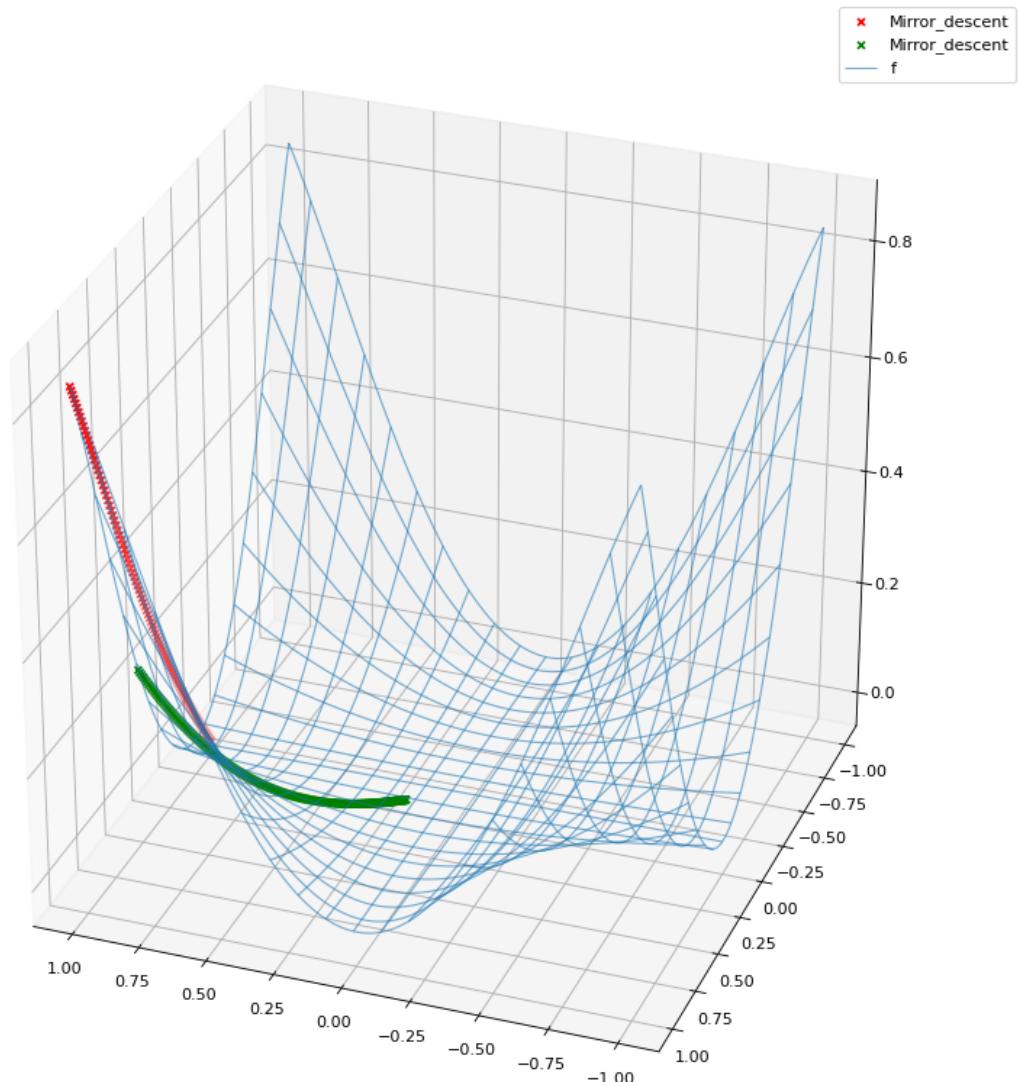
max_iter=1000
pas = [0.003 for i in range(max_iter)]
_, X, t1_ = Descente_Mirroir(f, df, grad_h, grad_h_inverse, x0, pas
, proj, 10**(-3), max_iter=max_iter)
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax.scatter(X, Y, Z, marker='x', color='g', label='Mirror_descent')

X = np.linspace(-1, 1, 100)
Y = np.linspace(-1, 1, 100)
Z = np.array([[f(np.array([X[i],Y[j]])) for i in range(len(X))] for
j in range(len(Y))])
X, Y = np.meshgrid(X, Y)

ax.plot_wireframe(X, Y, Z, rstride=5, cstride=5, linewidth=1, alpha=
0.6, label='f')

ax.legend()
ax.view_init(30, 110)
plt.show()
```

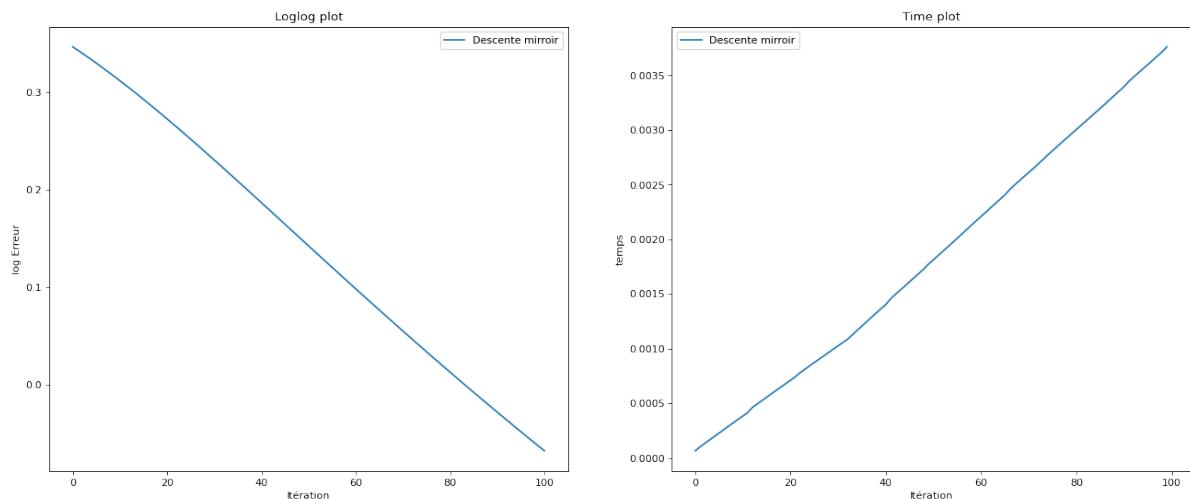
1.528205697480359
Nombre d'itreration : 100
2.1665540391889446
Nombre d'itreration : 1000



Log log plot de l'exemple précédent

```
In [15]: Erreur = np.linalg.norm(X1, axis=1)
plt.figure(figsize=(20, 8), dpi=80)
ax1 = plt.subplot(121)
ax1.plot(np.arange(len(X1)), np.log(Erreur), label='Descente miroir')
ax1.set_title('Loglog plot')
ax1.set_xlabel('Itération')
ax1.set_ylabel('log Erreur')
ax1.legend()

ax2 = plt.subplot(122)
ax2.set_title('Time plot')
ax2.plot(np.arange(len(X1)-1), temps_dm, label='Descente miroir')
ax2.set_xlabel('Itération')
ax2.set_ylabel('temps')
ax2.legend()
plt.show()
```



Comparaison des deux méthodes sur un cône

```
In [16]: def f(x) :
    return (1-x[0])**2 + (x[1] - x[0])**2

def df(x):
    return np.array([-2*(1 - x[0]) - 2*(x[1]-x[0]), 2*(x[1]-x[0])])
```

```
In [17]: def project_cone(y,p=0):
    ## p étant la direction du cône
    y_p = y[p]
    y_sp = np.concatenate((y[:p] , y[p+1:] ))
    if np.linalg.norm(y_sp)**2 <= y_p**2 :
        return y
    elif np.abs(y_p) < 1e-8:
        x = y/2
        x[p] = np.linalg.norm(np.concatenate((y[:p] , y[p+1:] )))/2
        return x
    c = np.linalg.norm(y_sp/y_p)
    if c - 1 < 0 :
        lamb = -(c+1)/(1-c)
    else :
        lamb = (c-1)/(1+c)
    x = y / (1+lamb)
    x[p] = y_p/(1-lamb)
    return x
```

```
In [18]: fig = plt.figure(figsize=(13, 13), dpi=80)
ax = fig.add_subplot(111, projection='3d')

x0 = np.array([1,4]) # point de départ

max_iter=100
pas = [0.01 for i in range(max_iter)]
_, X, temps_dg = Descente_Gradient(f,df, x0, pas, project_cone, max_iter=max_iter)
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax.scatter(X, Y, Z, marker='.', color='r', alpha = 1, label='Gradient_descent')

max_iter=100
pas = [0.01 for i in range(max_iter)]
_, X, temps_dm = Descente_Mirroir(f, df, grad_h, grad_h_inverse, x0, pas, project_cone, 1e-3, max_iter=max_iter)
X1 = X
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax.scatter(X, Y, Z, marker='x', color='g', alpha = 1, label='Mirroir_descent')

X = np.linspace(-4, 4, 100)
Y = np.linspace(-4, 4, 100)
Z_1 = np.array([f(np.array([X[i],Y[i]])) for i in range(len(X))])
Z_2 = np.array([f(np.array([X[i],-Y[i]])) for i in range(len(X))])
ax.plot(X,Y,Z_1, color = 'y')
ax.plot(X,-Y,Z_2,color = 'y')

#X = np.linspace(-2, 4, 100)
#Y = np.linspace(-2, 4, 100)
Z = np.array([[f(np.array([X[i],Y[j]])) for i in range(len(X))] for j in range(len(Y))])
X, Y = np.meshgrid(X, Y)

ax.plot_wireframe(X, Y, Z, rstride=5, cstride=5, linewidth=1, label='f')

ax.legend()
ax.view_init(30, 40)
plt.show()
```

Le point final est $(1.5545944564745904, 1.5545944564745904)$.

La valeur de f en ce point est : 0.3075750111523463 .

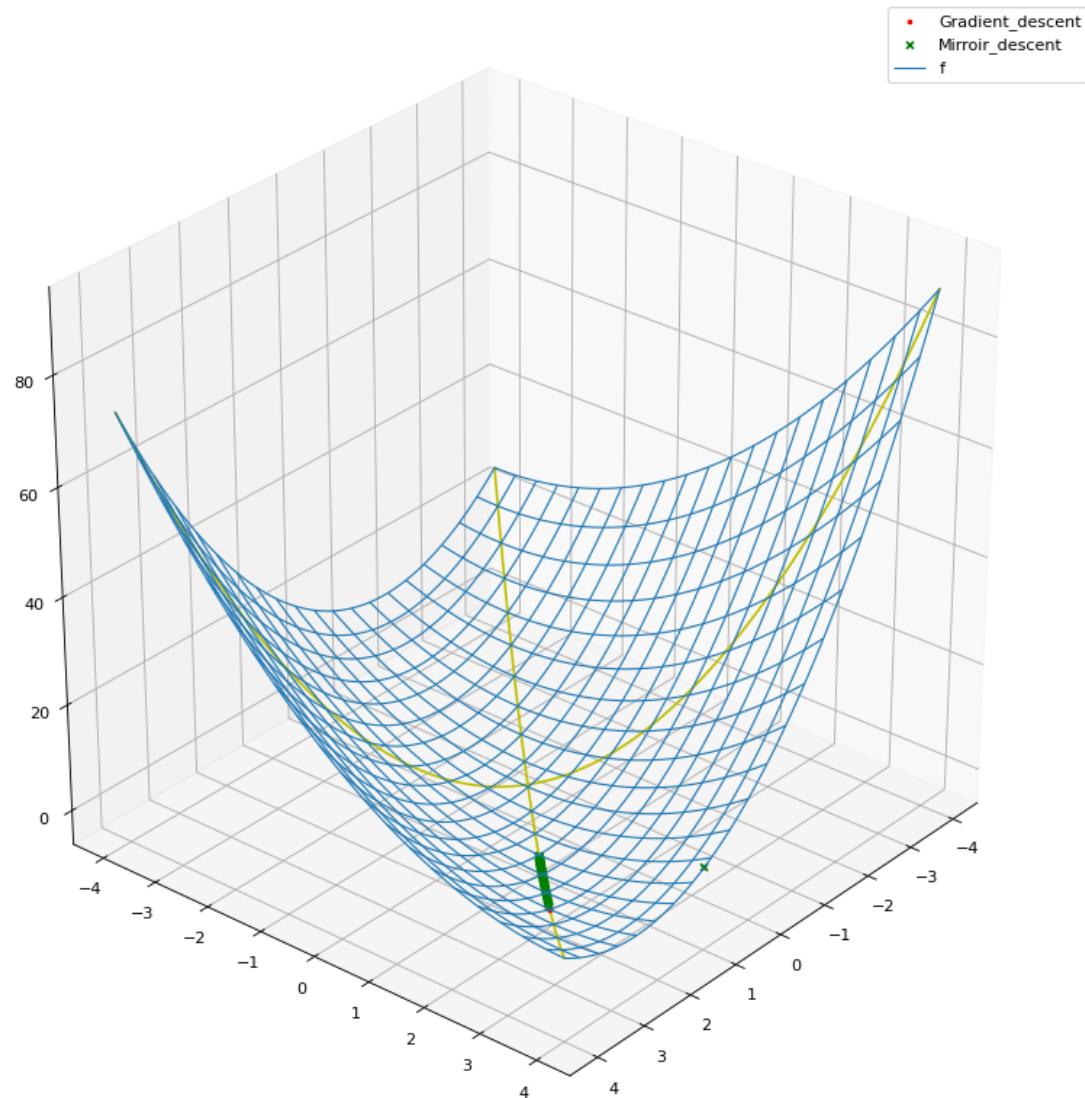
La valeur du gradient en ce point est $(1.1203928413628077, 8.881784197001252e-16)$.

Nombre d'itération : 100.

100

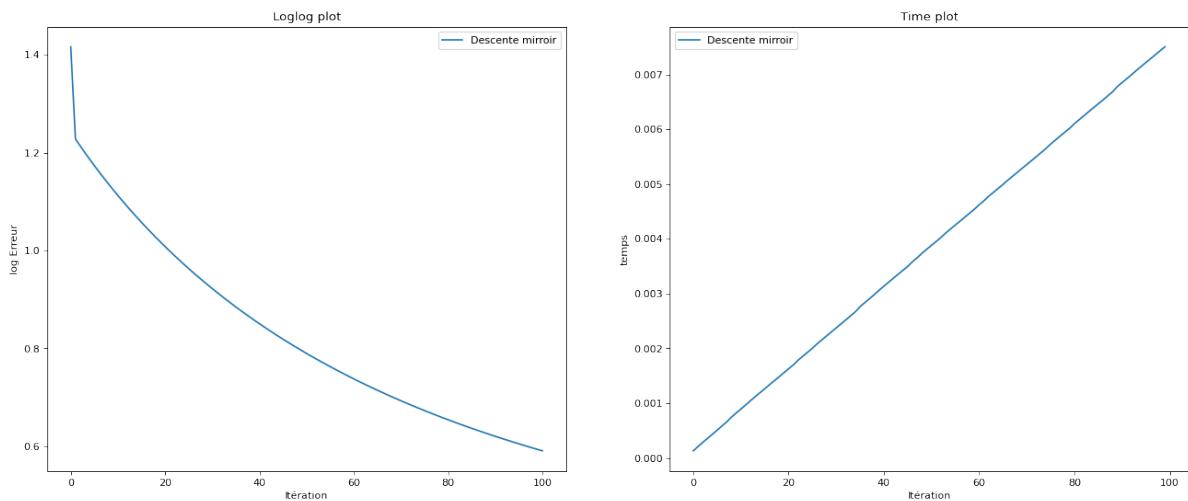
8.48528137423857

Nombre d'itération : 100



```
In [19]: Erreur = np.linalg.norm(X1, axis=1)
plt.figure(figsize=(20, 8), dpi=80)
ax1 = plt.subplot(121)
ax1.plot(np.arange(len(X1)), np.log(Erreur), label='Descente miroir')
ax1.set_title('Loglog plot')
ax1.set_xlabel('Itération')
ax1.set_ylabel('log Erreur')
ax1.legend()

ax2 = plt.subplot(122)
ax2.set_title('Time plot')
ax2.plot(np.arange(len(X1)-1), temps_dm, label='Descente miroir')
ax2.set_xlabel('Itération')
ax2.set_ylabel('temps')
ax2.legend()
plt.show()
```



Application au simplexe

On définit le simplexe par :

$$S = \{x \in \mathbb{R}^n : x \geq 0, \mathbf{1}^T x = 1\}$$

S peut être vu comme l'ensemble des mesures de probabilité discrètes de cardinal n .

On souhaite résoudre le problème d'optimisation :

$$x^* = \arg \min_{x \in S} f(x)$$

```
In [20]: #Implémentation d'une fonction problème
```

```
def f(x) :
    return (1-x[0])**2 + (x[1] - x[0])**2

def df(x):
    return np.array([-2*(1 - x[0]) - 2*(x[1]-x[0]), 2*(x[1]-x[0])])
```

Algorithme de projection euclidienne sur le simplexe S

On cherche à résoudre :

$$P(y) = \arg \min_{x \in S} \|x - y\|^2$$

Pour cela, on possède l'algorithme suivant :\

Input : $y \in \mathbb{R}^n$ \ -Sort y : $y_{i_1} \geq y_{i_2} \geq \dots \geq y_{i_n}$ \
 -Find $\rho = \max\{1 \leq j \leq n : y_{i_j} + \frac{1}{i_j}(1 - \sum_{k=1}^j y_{i_k}) > 0\}$ \ -Define $\lambda = \frac{1}{i_\rho}(1 - \sum_{k=1}^\rho y_{i_k})$ \
 -Define x by $x_{i_k} = \max\{y_{i_k} + \lambda, 0\}$ for $k = 0, \dots, n$. \ **Output**: x

```
In [21]: def project_simplexe(y):
    y = np.array(y)
    u = -np.sort(-y)
    rho = len( (u - 1 > 0).astype(int))
    mu = (1 - np.sum(u[:rho]))/rho
    x = y + mu
    x[x < 0] = 0
    return x
```

```
In [22]: fig = plt.figure(figsize=(13, 13), dpi=80)
ax = fig.add_subplot(111, projection='3d')

x0 = np.array([-1,0]) # point de départ

max_iter=2000
pas = [0.003 for i in range(max_iter)]
_, X, temps_dg= Descente_Gradient(f,df, x0, pas, project_simplexe,
max_iter=max_iter)
Erreur_dg = np.linalg.norm(X - np.array([0.6,0.4]), axis=1)
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax.scatter(X, Y, Z, marker='.',color='r', alpha = 1, label='Gradient descent partant de {}'.format(x0))

x0 = np.array([0,-1]) # point de départ

max_iter=2000
pas = [0.003 for i in range(max_iter)]
_, X, t_ = Descente_Gradient(f,df, x0, pas, project_simplexe, max_iter=max_iter)
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax.scatter(X, Y, Z, marker='x',color='g', alpha = 1, label='Gradient descent partant de {}'.format(x0))

X = np.linspace(0, 1, 100)
Y = np.linspace(0, 1, 100)
Z_1 = np.array([f(np.array([X[i],0])) for i in range(len(X))])
Z_2 = np.array([f(np.array([0,Y[i]])) for i in range(len(X))])
Z_3 = np.array([f(np.array([X[i],-Y[i] +1])) for i in range(len(X))])
ax.plot([0 for i in range(len(X))],Y,Z_2, color = 'y')
ax.plot(X,[0 for i in range(len(Y))],Z_1, color = 'y')
ax.plot(X,-Y +1,Z_3,color = 'y')

X = np.linspace(-1.5, 1.5, 100)
Y = np.linspace(-1.5, 1.5, 100)
Z = np.array([[f(np.array([X[i],Y[j]])) for i in range(len(X))] for j in range(len(Y))])
X, Y = np.meshgrid(X, Y)

ax.plot_wireframe(X, Y, Z, rstride=5, cstride=5, linewidth=1, label='f')

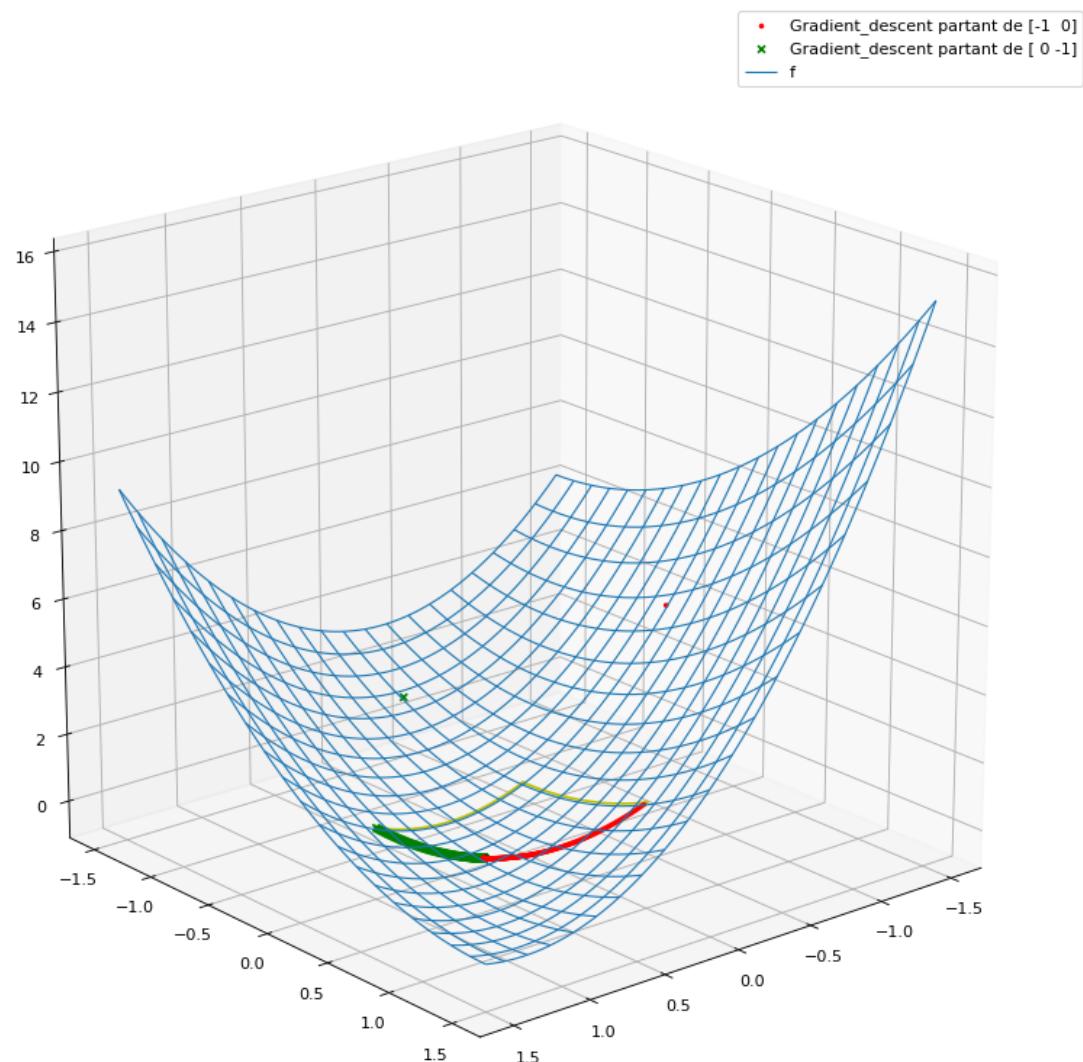
ax.legend()
ax.view_init(20, 50)
```

Le point final est $(0.599999999999556, 0.400000000000445)$.
La valeur de f en ce point est : 0.1999999999999999 .
La valeur du gradient en ce point est $(-0.4000000000002708, -0.399999998194)$.
Nombre d'itération : 2000.

2000

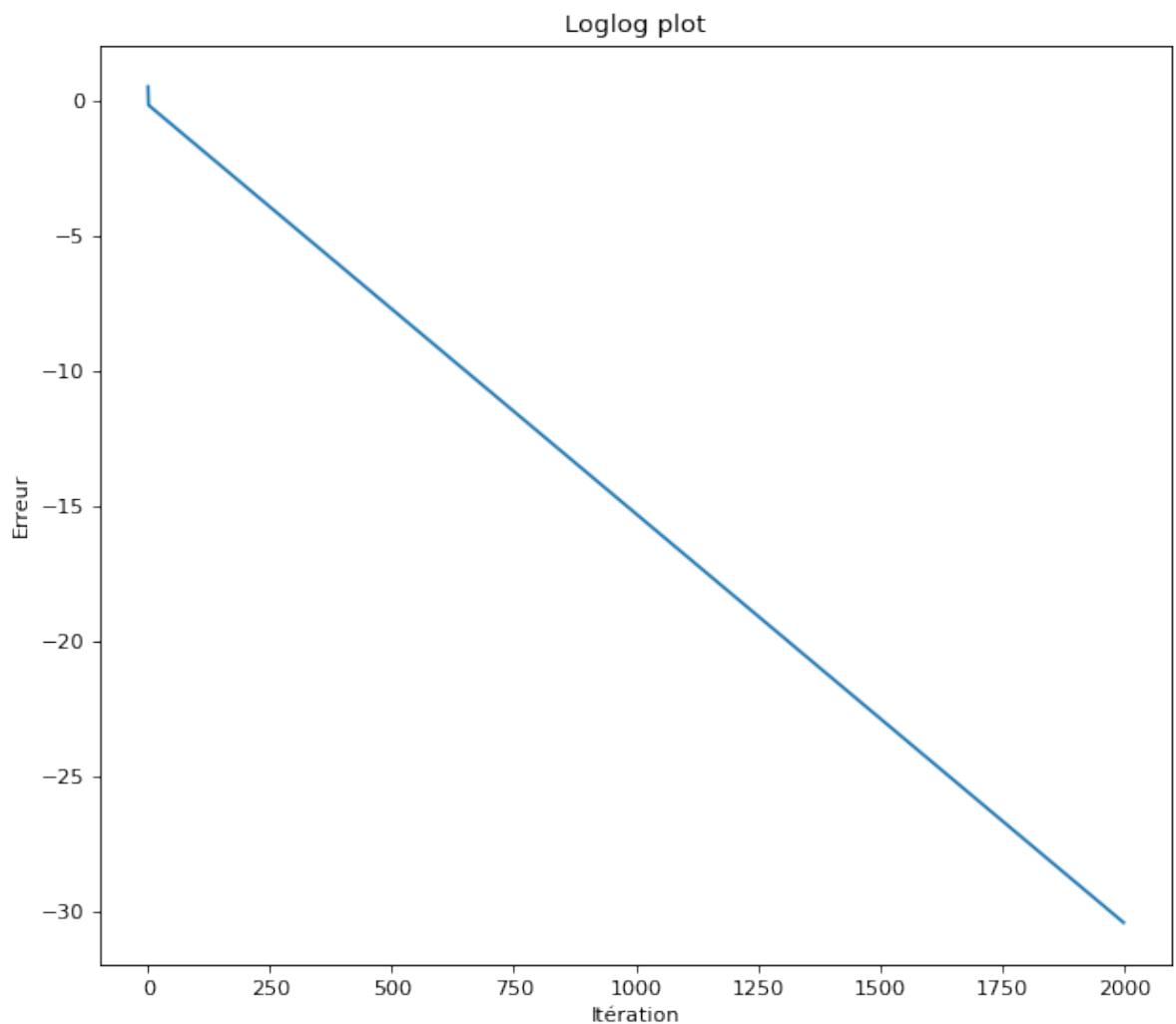
Le point final est $(0.600000000000301, 0.399999999999699)$.
La valeur de f en ce point est : 0.2 .
La valeur du gradient en ce point est $(-0.399999999998165, -0.40000000012226)$.
Nombre d'itération : 2000.

2000



```
In [23]: plt.figure(figsize=(20, 8), dpi=80)
ax1 = plt.subplot(121)
ax1.plot(np.arange(len(Erreur_dg)), np.log(Erreur_dg))
ax1.set_title('Loglog plot')
ax1.set_xlabel('Itération')
ax1.set_ylabel('Erreur')
"""
ax2 = plt.subplot(122)
ax2.set_title('Time plot')
ax2.plot(np.arange(len(Erreur_dg)-1), temps_dg)
ax2.set_xlabel('Itération')
ax2.set_ylabel('temps')
"""

plt.show()
```



Algorithme de projection selon la divergence de Bregman V sur le simplexe S

La projection sur le simplexe :

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} V(x, y) \\ & \text{s. c. } x^T \mathbb{1} = 1 \\ & \quad x \geq 0 \\ & P(y) = \operatorname{argmin}_{x \in S} V(x, y) \end{aligned}$$

On prend : $w : x \in \mathbb{R}_+^n \rightarrow \sum_i x_i \log(x_i) - x_i$

On écrit le lagrangien : $\mathcal{L}(x, \lambda, \mu) = w(x) - w(y) - \langle \log(y), x - y \rangle + \lambda(1 - \sum_i x_i) - \mu \cdot x$

$$\begin{cases} \mu \cdot x = 0 & (1) \\ \sum_i x_i = 1 & (2) \text{ En utilisant l'équation (3) on a :} \\ \log(x) - \log(y) + \lambda \cdot \mathbb{1} - \mu = 0 & (3) \end{cases}$$

$\forall i \in \{1, \dots, n\}, x_i = y_i e^{(\mu_i - \lambda)}$

- si $y_i = 0$, alors $x_i = 0$.
- sinon $y_i \neq 0$, alors $x_i \neq 0$ et donc d'après l'équation (1) $\mu_i = 0$.

Ainsi, $\forall i \in \{1, \dots, n\}, x_i = y_i e^{-\lambda}$ En utilisant l'équation (2) on a :

$$e^{-\lambda} = \frac{1}{\sum_i y_i}$$

Enfin, la solution générale du problème est :

$$\forall i \in \{1, \dots, n\}, x_i = \frac{y_i}{\sum_i y_i}$$

```
In [24]: def project_simplexe_bregman(y):
    y = np.array(y)
    if (y == 0).all():
        return np.zeros(y.shape)
    return y/np.sum(y)

def Descente_Mirroir_B(df, grad_h, grad_h_inverse, x0, pas, proj, e
rreur=1e-6, max_iter=1000):
    x, chemin = Descente_Mirroir(df, grad_h, grad_h_inverse, x0, pa
s, proj, erreur=1e-6, max_iter=1000)
    x = np.sum(np.array([chemin[i]*pas[i]/np.sum(pas) for i in ran
ge(len(chemin))]))
    return x, chemin

def h(x) :
    return np.sum(x*np.log(x) - x)

def grad_h(x):
    return np.log(x)

def grad_h_inverse(x):
    return np.exp(x)

def Bregman(w, grad_w ,x,y):
    return w(y) - w(x) - grad_w(x).T@(y - x)
```

```
In [25]: fig = plt.figure(figsize=(30, 13), dpi=80)
#ax1 = fig.add_subplot(121, projection='3d')
ax2 = fig.add_subplot(122, projection='3d')

...
x0 = np.array([0.1,0.4]) # point de départ

max_iter=10000
pas = [0.003 for i in range(max_iter)]
_, X, temps_dm = Descente_Mirroir(f, df, grad_h, grad_h_inverse, x0
, pas, project_simplexe_bregman, 10**(-3), max_iter=max_iter)
Z = np.array([f(X[i]) for i in range(len(X))])
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax1.scatter(X, Y, Z, marker='.',color='r', alpha = 1, label='Gradie
nt_descent')
...

x0 = np.array([0.2,1]) # point de départ

max_iter=200
pas = [0.03 for i in range(max_iter)]
_, X, temps_dm = Descente_Mirroir(f, df, grad_h, grad_h_inverse, x0
, pas, project_simplexe_bregman, 10**(-3), max_iter=max_iter)
Erreur_dg = np.linalg.norm(X - np.array([0.6,0.4]), axis=1)
Z = np.array([f(X[i]) for i in range(len(X))])
```

```
Y = np.array([X[i][1] for i in range(len(X))])
X = np.array([X[i][0] for i in range(len(X))])
ax2.scatter(X, Y, Z, marker='x', color='g', alpha = 1, label='Mirror
_descent')

X = np.linspace(0, 1, 100)
Y = np.linspace(0, 1, 100)
Z_1 = np.array([f(np.array([X[i],0])) for i in range(len(X))])
Z_2 = np.array([f(np.array([0,Y[i]])) for i in range(len(X))])
Z_3 = np.array([f(np.array([X[i],-Y[i] +1])) for i in range(len(X))
])
'''
ax1.plot([0 for i in range(len(X))],Y,Z_2, color = 'y')
ax1.plot(X,[0 for i in range(len(Y))],Z_1, color = 'y')
ax1.plot(X,-Y +1,Z_3,color = 'y')
'''

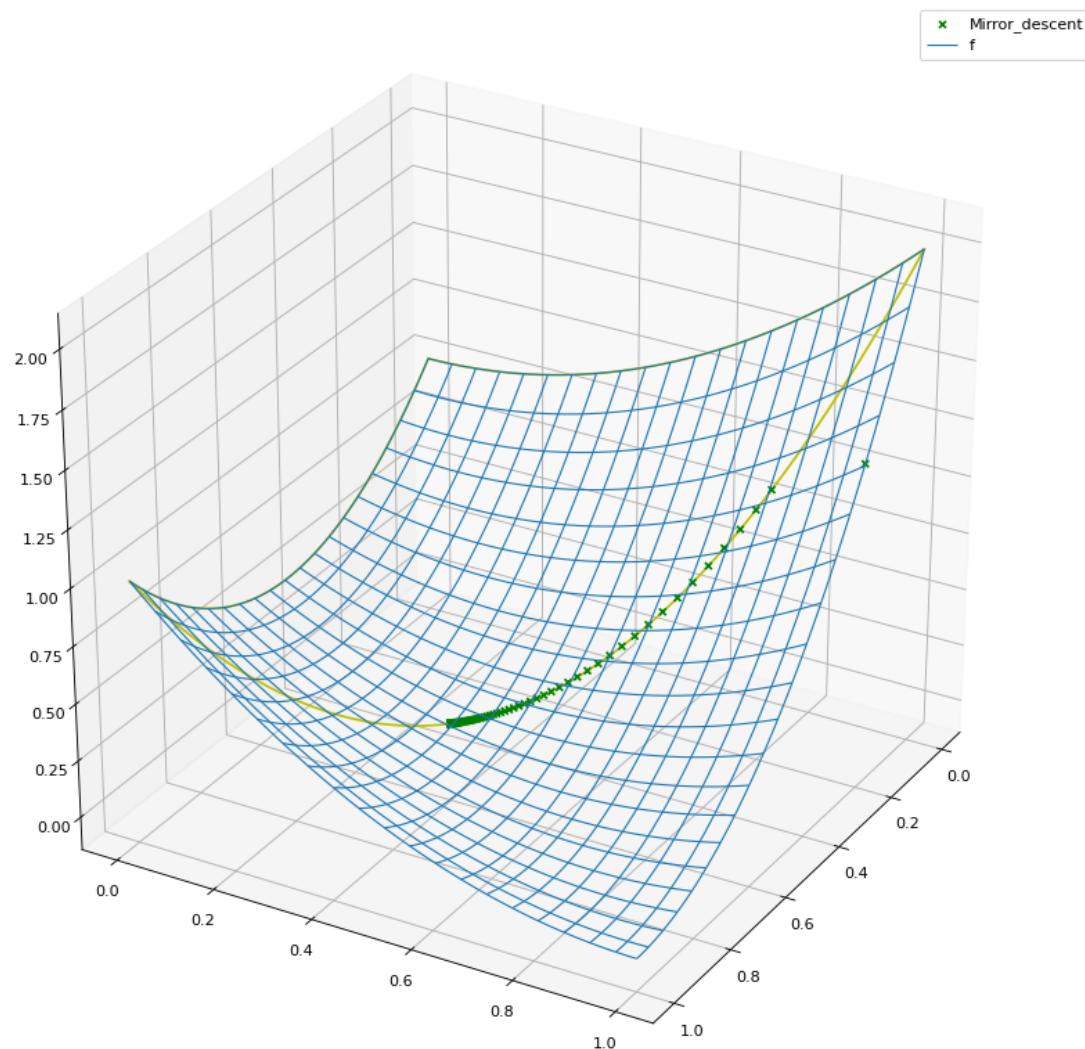
ax2.plot([0 for i in range(len(X))],Y,Z_2, color = 'y')
ax2.plot(X,[0 for i in range(len(Y))],Z_1, color = 'y')
ax2.plot(X,-Y +1,Z_3,color = 'y')

X = np.linspace(0, 1, 100)
Y = np.linspace(0, 1, 100)
Z = np.array([[f(np.array([X[i],Y[j]])) for i in range(len(X))] for
j in range(len(Y))])
X, Y = np.meshgrid(X, Y)

#ax1.plot_wireframe(X, Y, Z, rstride=5, cstride=5, linewidth=1, label
l='f')
ax2.plot_wireframe(X, Y, Z, rstride=5, cstride=5, linewidth=1, label
='f')

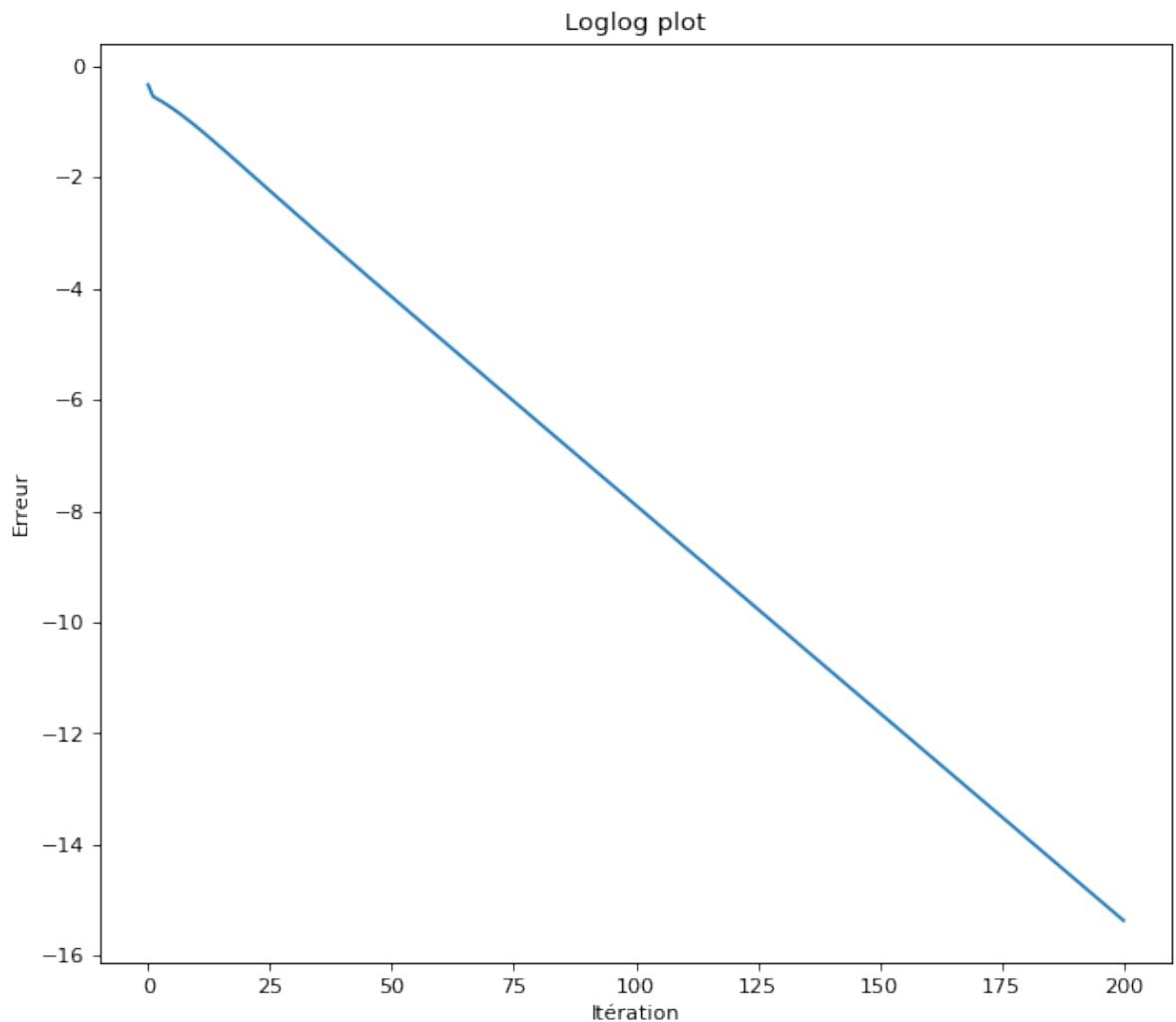
#ax1.legend()
ax2.legend()
##ax1.view_init(30, 30)
ax2.view_init(30, 30)
```

3.577708763999664
Nombre d'itreration : 200



```
In [26]: plt.figure(figsize=(20, 8), dpi=80)
ax1 = plt.subplot(121)
ax1.plot(np.arange(len(Erreur_dg)), np.log(Erreur_dg))
ax1.set_title('Loglog plot')
ax1.set_xlabel('Itération')
ax1.set_ylabel('Erreur')
"""
ax2 = plt.subplot(122)
ax2.set_title('Time plot')
ax2.plot(np.arange(len(Erreur_dg)-1), temps_dg)
ax2.set_xlabel('Itération')
ax2.set_ylabel('temps')
"""

plt.show()
```



Comparaison des deux méthodes

```
In [27]: x0 = np.array([0.2,1]) # point de départ
max_iter=2000
pas = [0.03 for i in range(max_iter)]

_, X_gradient, temps_dg = Descente_Gradient(f,dF, x0, pas, project_
simplexe, max_iter=max_iter)
f_gradient = np.array([f(X_gradient[i]) for i in range(len(X_gradie
nt))])
_, X_miroir, temps_dm = Descente_Mirroir(f, df, grad_h, grad_h_in
verse, x0, pas, project_simplexe_bregman, 10**(-3), max_iter=max_it
er)
f_miroir = np.array([f(X_miroir[i]) for i in range(len(X_miroir))
])

Erreurs_gradient = f_gradient - 0.2000000000000008
Erreurs_miroir = f_miroir - 0.2000000000000008
plt.figure(figsize=(8, 8), dpi=80)
plt.loglog(np.arange(1,len(X_gradient)), Erreurs_gradient[1:], color
= "r", label = "Méthode du gradient")
plt.loglog(np.arange(1,len(X_miroir)), Erreurs_miroir[1:], color
= "b", label = "Méthode de la descente miroir")
plt.title('Loglog plot')
plt.legend()
plt.xlabel('Itération')
plt.ylabel('Erreurs')
plt.show()
```

Le point final est (0.5999999999999999, 0.40000000000000013).

La valeur de f en ce point est : 0.2.

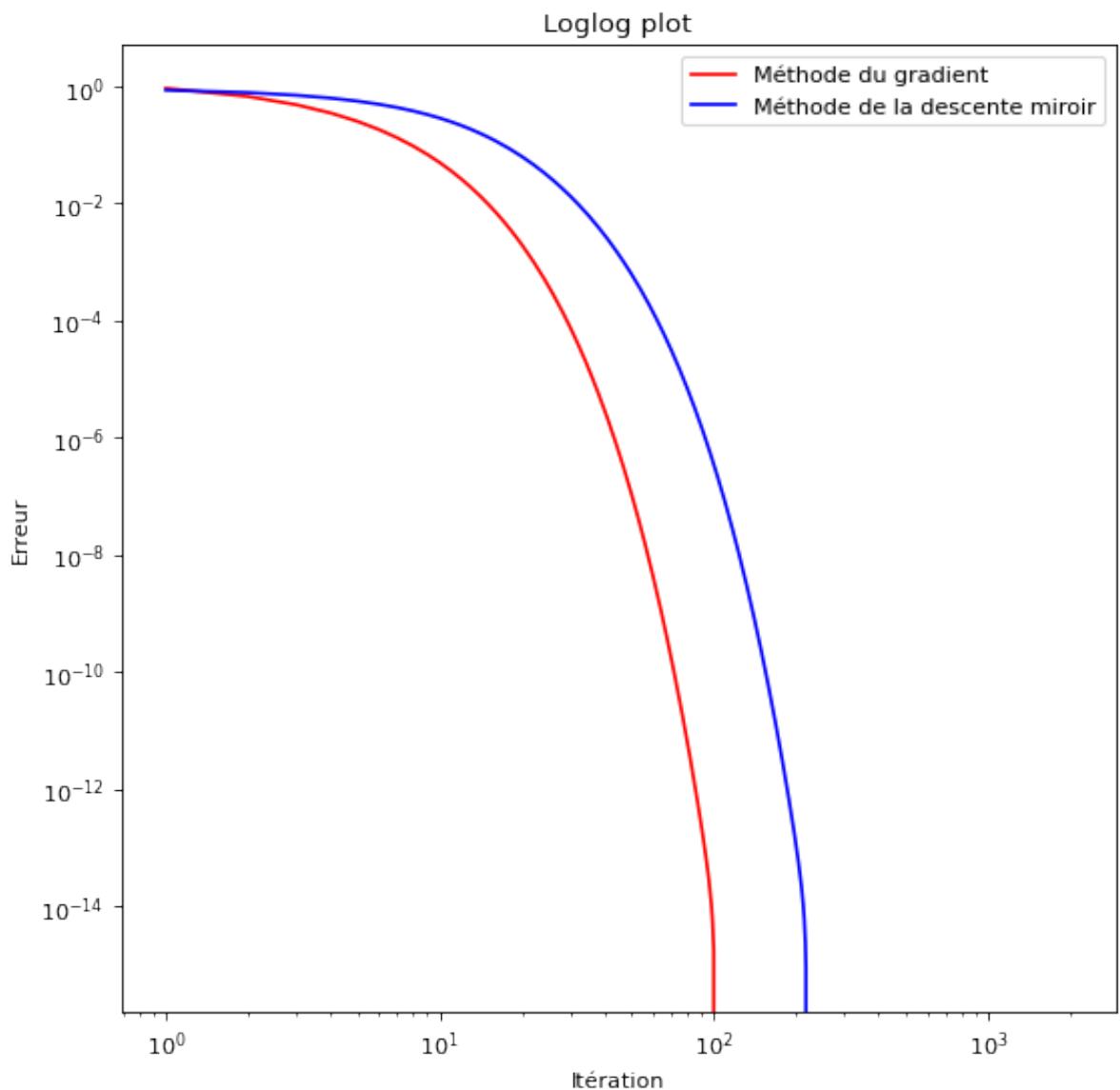
La valeur du gradient en ce point est (-0.4000000000000008, -0.3999
999999999947).

Nombre d'itération : 2000.

2000

3.577708763999664

Nombre d'itération : 2000



A commenter

Application à la simulation de portefeuille

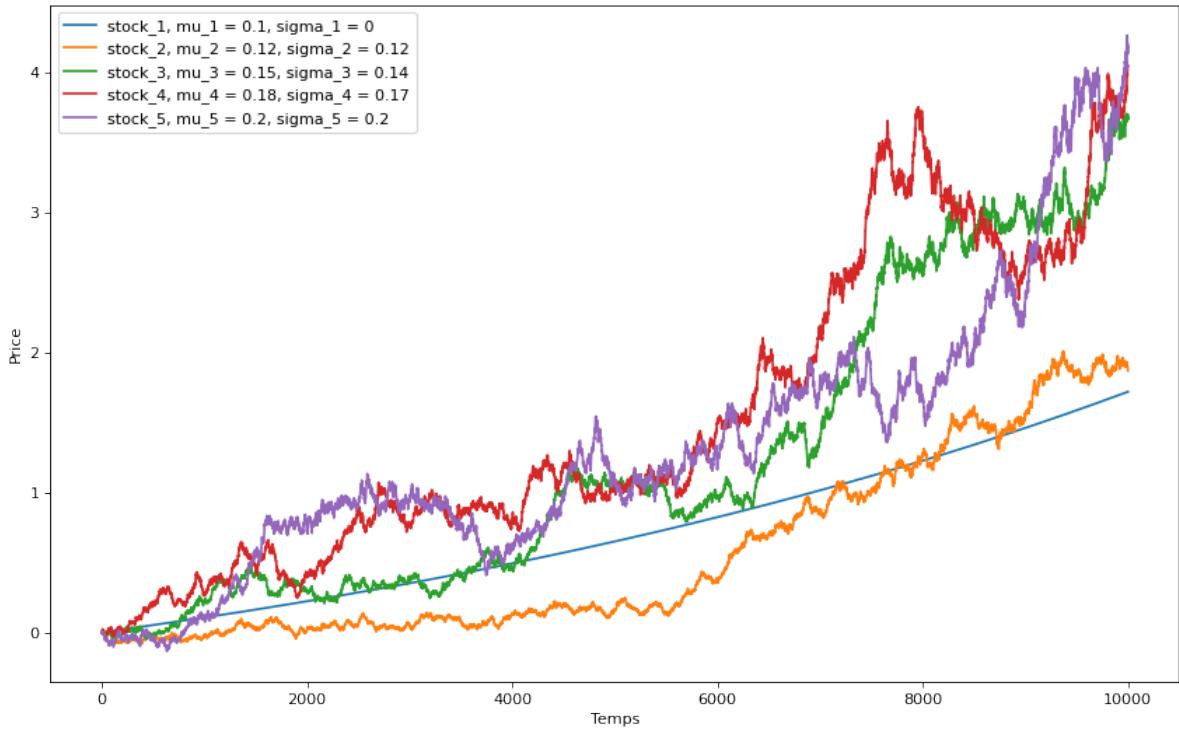
On simule la valeur de 10 actions pour un temps de 1000 à l'aide d'un mouvement brownien

```
In [28]: def simuler_action( mu, sigma, nombre=10, temps=1000):
    S_init = [1 for i in range(nombre)]
    T = np.linspace(0,1,temps)
    S = []
    delta_t = 1/1000
    for i in range(len(S_init)):
        s = S_init[i]
        B = np.cumsum(np.random.normal(loc=0, scale=np.sqrt(delta_t),
size=len(T)))
        S_i = []
        mu_i = mu[i]
        sigma_i = sigma[i]
        for t in range(len(T)):
            S_i.append(s*np.exp(mu_i*t*delta_t + sigma_i*B[t]) - s)
        S.append(S_i)
    return np.array(S)
```

```
In [29]: mu_list = [0.1,0.12,0.15,0.18,0.2,0.22]
sigma_list = [0,0.12,0.14,0.17,0.2,0.22]
S = simuler_action(mu_list, sigma_list, nombre = 5, temps = 10000)

def afficher(S):
    plt.figure(figsize=(13, 8), dpi=80)
    for i in range(S.shape[0]):
        plt.plot(np.arange(S.shape[1]), S[i], label=f'stock_{i+1}, mu_{i+1} = {mu_list[i]}, sigma_{i+1} = {sigma_list[i]}')
    plt.xlabel('Temps')
    plt.ylabel('Price')
    plt.legend()
    plt.show()

afficher(S)
```



Nous cherchons une solution du problème suivant :\

$$\max_{x \in S} \mathbb{E}[x \cdot u]$$

avec S le simplexe de dimension n , $u \in \mathbb{R}^n$ le prix des différentes actions.

```
In [30]: def simuler_action_2(n,S0_list,mu_list,sigma_list,M=1000,T=1):
    B = np.random.normal(loc = 0, scale = np.sqrt(T), size = (M,n))
    S = np.array([S0_list[i] * np.exp(sigma_list[i]*B[:,i] + (mu_list
    [i] - sigma_list[i]**2/2)*T) - S0_list[i] for i in range(n)])
    return S.T
```

```
In [31]: mu_list = [1, 0.0, 0.3, 0.5, 0.1]
sigma_list = [0.2, 0.5, 0.1, 0.4, 0.3]
S = simuler_action_2(5,[1,1,1,1,1],mu_list,sigma_list)
print(S.shape)

(1000, 5)
```

```
In [32]: def F(x,u):
    return x.T@u

def G(x,u):
    return u
```

```
In [33]: def Descente_Mirroir_sto(S, G, grad_h, grad_h_inverse, x0, pas, proj, erreur=1e-6, max_iter=1000):
    x=x0
    chemin = []
    chemin.append(x0)
    i = 0
    temps = []
    start = time.time()
    while True :
        theta = grad_h(x)
        chi = S[i,:]
        theta = theta + pas[i]*G(x,chi)
        x1 = grad_h_inverse(theta)
        x = proj(x1)
        temps.append(time.time()- start)
        chemin.append(x)
        if i >= max_iter-1 or np.linalg.norm(df(x)) < erreur :
            break
        i+=1
    print(len(chemin), len(pas))
    x = np.sum(np.array([chemin[j+1]*pas[j]/np.sum(pas) for j in range(len(pas))]), axis=0)
    print('Nombre d\'iteration : ', i)
    return x, np.array(chemin), np.array(temps)
```

Exemple simple de la descente miroir stochastique:

```
In [34]: mu_list = [3,1,1,1,1,1]
sigma_list = [1,1,1,1,1,1]
S0_list = [1,1,1,1,1,1]

S = simuler_action_2(6,S0_list,mu_list,sigma_list)

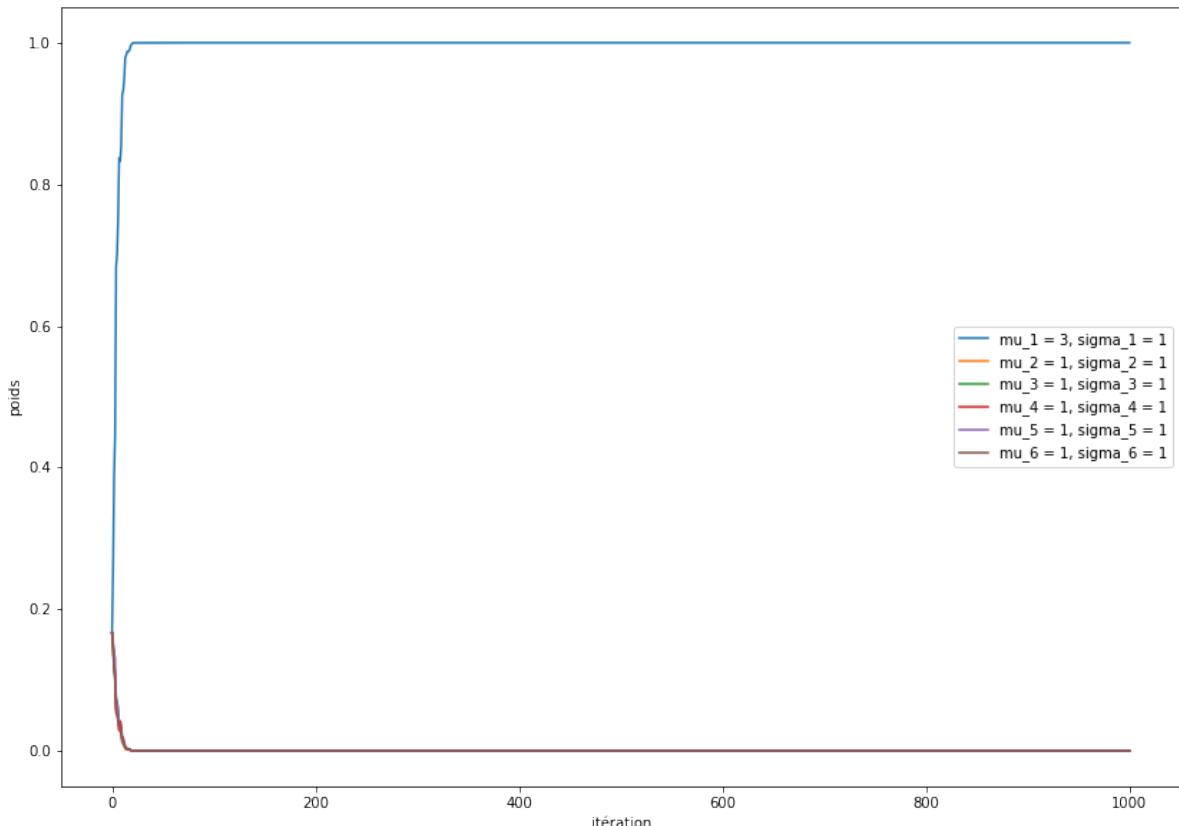
x0 = np.array([1,1,1,1,1,1])/6 # point de départ

max_iter=1000
pas = [0.03 for i in range(max_iter)]
opt, chemin, temps_dms = Descente_Mirroir_sto(S, G, grad_h, grad_h_inverse, x0, pas, proj=project_simplexe_bregman, erreur=1e-6, max_iter=1000)
print('Les poids attribués par le modèle sont respectivement :')
print('\t', opt)
print('La somme des poids : {:.3}'.format(np.sum(opt)))
plt.figure(figsize=(14,10))
for i in range(chemin.shape[1]):
    plt.plot(np.arange(chemin.shape[0]), chemin[:,i], label=f'mu_{i+1}')
    } = ' +str(mu_list[i]) + f', sigma_{i+1} = ' +str(sigma_list[i]))

plt.xlabel('itération')
plt.ylabel('poids')
plt.legend()
```

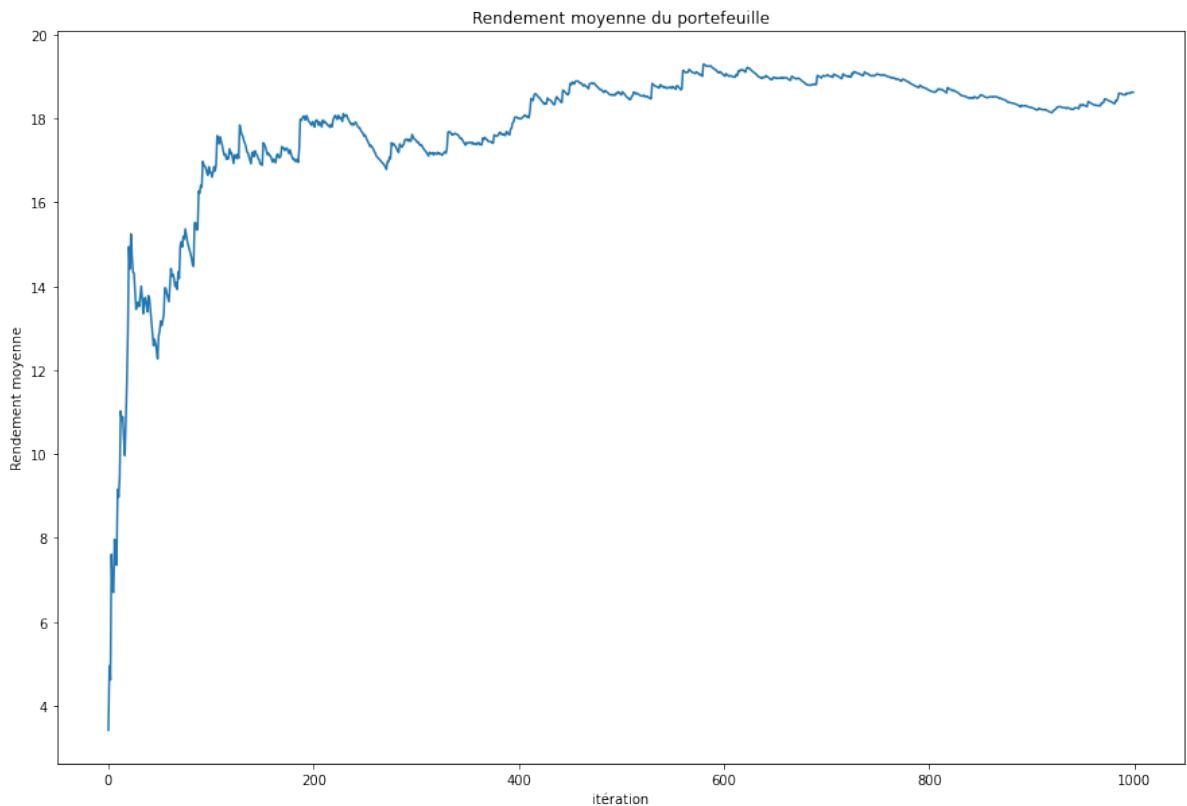
```
1001 1000
Nombre d'itreration : 999
Les poids attribués par le modèle sont respectivement :
[ 9.96468521e-01 6.54183049e-04 7.28818470e-04 7.17949641e
-04
 7.68401632e-04 6.62126025e-04 ]
La somme des poids : 1.0
```

Out[34]: <matplotlib.legend.Legend at 0x7f11d9047a30>



```
In [35]: vect = np.array([chemin[i,:]*S[i,:] for i in range(chemin.shape[0]-1)])
J = np.cumsum(vect)/np.arange(1,len(vect)+1)
plt.figure(figsize = (15,10))
plt.plot(J)
plt.title('Rendement moyenne du portefeuille')
plt.xlabel('itération')
plt.ylabel('Rendement moyenne')
```

Out[35]: Text(0, 0.5, 'Rendement moyenne')



```
In [36]: print('Rendement moyenne final : ',np.mean(np.sum([S[:,i]*opt[i] for i in range(5)], axis=0)))
```

Rendement moyenne final : 18.62787070491697

Descente miroire stochastique biaisée

```
In [37]: def biased_SMD(x0, theta_0, step_list, c, S0_list,mu_list,sigma_list, alpha=0.05,max_iter=1000):
    '''

    paramètres :
        x0 : np.array, point de départ
        theta_0 : float, représente la valeur moyenne de la perte
        alpha : float, le risque dans [0,1], sa valeur par défaut est 5
    %

        step_list : np.array, liste de pas de longueur max_iter
        c : float, lambda dans le papier à determiner
        S0_list : np.array, le prix initial des actions
        mu_list : np.array, shape = (nombre d'actions x0.shape, 100)
        sigma_list : np.array, shape = (nombre d'actions x0.shape, 100)
            !! Ces deux derniers coefficients doivent être estimés
            à partir du vraies actions. !!

    return :
        np.array representant les poids optimaux

    Cette algorithme est pris du papier "Manon Costa, Sébastien Gadat
    , Lorick Huang. Portfolio optimization under CV@R constraint with
    stochastic mirror descent. 2022. ffhal-0369723ff"

    Dans le drive (document.pdf page 7)
    '''

    i = 0
    u_k = x0
    theta_k = theta_0
    n = x0.shape[0]
    z_k = simuler_action_2(n,S0_list,mu_list,sigma_list,M=max_iter,T=
    1)# à revoir avec Sébastien
    theta = []
    U = []
    U.append(x0)
    theta.append(theta_0)

    while i < max_iter - 1:
        z_k = z_k[i,:]
        g_k_1 = (c/(1 - alpha)*(z_k@u_k >= theta_k).astype(int) - 1)*z_
        k
        g_k_2 = c * (1 - (z_k@u_k >= theta_k).astype(int)/(1 - alpha))
        u_k = u_k*np.exp(-step_list[i]*g_k_1)/np.sum(np.abs(u_k*np.exp(
        -step_list[i]*g_k_1)))
        theta_k -= step_list[i]*g_k_2
        i+=1
        U.append(u_k)
        theta.append(theta_k)

    return u_k, theta_k, z_k, U, theta
```

Exemple simple sur l'algorithme stochastique précédent

```
In [38]: x0 = np.array([1,1,1,1,1])/5 # point de départ
theta_0 = 0.2
lambda_ = 0.9
max_iter = 100000
pas = [i**(-0.75) for i in range(1,max_iter+1)]
S0_list = [1,1,1,1,1]
mu_list = [0.12,0.15,0.18,0.2,0.22]
sigma_list = [0.12,0.14,0.17,0.2,0.22]

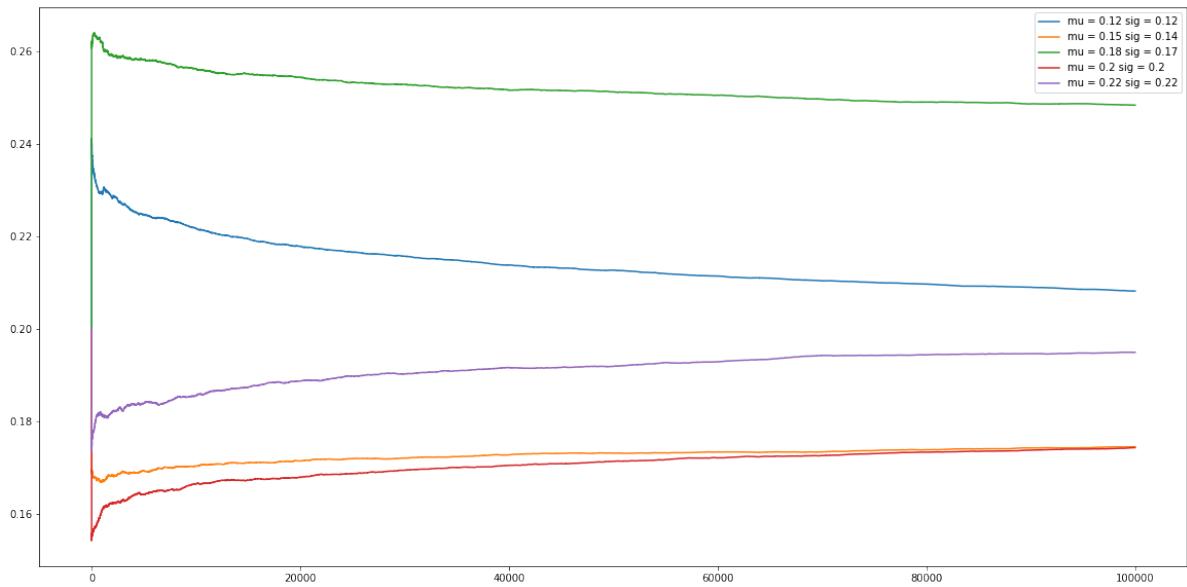
u, theta, z_k, U, theta = biased_SMD(x0, theta_0, pas, lambda_, S0_
list,mu_list,sigma_list, alpha=0.05,max_iter=max_iter)
```

```
In [39]: print('Les poids attribués par le modèle sont respectivement :')
print('mu \t', mu_list)
print('sigma\t', sigma_list)
print('poids\t', u)
print('La somme des poids : {:.3}'.format(np.sum(u)))
print(theta[-1])
U = np.array(U)

plt.figure(figsize=(20,10))
for i in range(U.shape[1]):
    plt.plot(np.arange(U.shape[0]), U[:,i], label='mu = ' +str(mu_lis
t[i]) + ' sig = ' +str(sigma_list[i]))
plt.legend()
```

```
Les poids attribués par le modèle sont respectivement :
mu      [0.12, 0.15, 0.18, 0.2, 0.22]
sigma   [0.12, 0.14, 0.17, 0.2, 0.22]
poids   [0.20812503 0.17442969 0.24835796 0.17423227 0.19485506]
La somme des poids : 1.0
0.04251086773884595
```

Out[39]: <matplotlib.legend.Legend at 0x7f11d90013a0>



```
In [40]: U = np.array(U)
theta = np.array(theta)
Z_k = np.array(Z_k)
vect = np.array([U[i,:]@Z_k[i,:] for i in range(U.shape[0])])
J = np.cumsum(vect)/np.arange(1,len(vect)+1)
CVAR = []
CVAR = np.cumsum(theta + np.maximum(vect-theta,0)/(1-0.05))/np.arange(1,len(vect)+1)

plt.figure(figsize=(30,10))
'''
ax1 = plt.subplot(121)
ax1.plot(np.arange(J.shape[0])[1000:], J[1000:])
ax1.set_title("CV@R")
'''

ax2 = plt.subplot(122)
ax2.plot(np.arange(CVAR.shape[0])[1000:], CVAR[1000:])
ax2.set_title("Valeur moyenne du Portefeuille")
```

Out[40]: Text(0.5, 1.0, 'Valeur moyenne du Portefeuille')

