PROJET D'EA RECHERCHE

Méthodes constructives pour l'analyse d'algorithmes d'optimisation stochastique

17 décembre 2024

Elias Ben Rhouma et Victor Galmiche
Tuteur : Adrien Taylor

TABLE DES MATIÈRES

1 INTRODUCTION

L'optimisation convexe est une branche fondamentale de l'optimisation mathématique dans laquelle la fonction objectif ainsi que l'ensemble admissible sont convexes. Cette propriété de convexité, bien que restrictive, est cruciale car elle permet de garantir que tout minimum local est un minimum global. Cette caractéristique rend ces problèmes particulièrement attractifs tant d'un point de vue théorique que pratique.

L'optimisation convexe joue un rôle central dans de nombreux domaines scientifiques et industriels, particulièrement en apprentissage automatique où elle constitue un fondement mathématique de l'entraînement des modèles. Bien que de nombreux algorithmes d'optimisation soient largement utilisés en pratique, leurs garanties théoriques de convergence et leurs comportements dans les pires cas restent des sujets d'étude actifs. En effet, l'établissement de bornes théoriques de convergence permet non seulement de mieux comprendre les limites fondamentales de ces algorithmes, mais aussi constitue une aide à la conception de ceux-ci.

Ce projet se concentre sur l'analyse de pire cas de trois algorithmes fondamentaux en optimisation stochastique convexe : la descente de gradient classique, la descente de gradient stochastique (SGD), particulièrement adaptée aux problèmes dans lesquels les jeux de données sont de grande taille, et l'algorithme SAGA, une méthode plus récente qui est un algorithme stochastique dit de réduction de variance par rapport à SGD.

La suite de ce rapport s'organise comme suit : dans un premier temps, nous présenterons l'état de l'art sur l'analyse de pire cas en optimisation convexe, en mettant l'accent sur une approche constructive. La section 3 introduira en détail les trois algorithmes étudiés, leurs fondements théoriques et leurs caractéristiques distinctives. La section 4 développera ensuite l'analyse de pire cas pour chacun d'entre eux à l'aide d'une méthode constructive. Ces analyses nous permettront de dériver des bornes théoriques de convergence précises. Une dernière visera à expérimenter numériquement sur deux cas simples (régression linéaire et régression logistique) la convergence de ces 3 algorithmes.

Notations : Durant ce projet, nous nous intéressons à des problèmes d'optimisation convexe dans l'espace euclidien $\mathbb{R}^d$. Tout au long de ce rapport, on considère la norme euclidienne sur cet espace, noté $\|\cdot\|$ et le produit scalaire associé, que l'on note $\langle\cdot;\cdot\rangle$. Pour une fonction différentiable $f : \mathbb{R}^d \to \mathbb{R}$, on notera $\nabla f(x) \in \mathbb{R}^d$ son gradient au point $x \in \mathbb{R}^d$. On note $\mathbb{S}^n$ l'ensemble des matrices symétriques de $\mathbb{R}^{n\times n}$. Pour $A \in \mathbb{S}^n$, $A \succcurlyeq 0$ signifie que A est semi-définie positive. Par ailleurs, on notera e_i le i-ème vecteur de la base canonique de $\mathbb{R}^n$, pour lequel tous les composants sont nuls sauf le i-ème qui est égal 1. Pour deux vecteurs $x, y \in \mathbb{R}^d$, on note $x \odot y$ la matrice de $\mathbb{S}^d$ égale à $\frac{1}{2}(xy^T + yx^T)$. On remarque alors que pour $S \in \mathbb{S}^n$ et $x, y \in \mathbb{R}^n$, on a $\mathrm{Tr}((x \odot y)S) = x^T S y$. Ce résultat sera utile lors de reformulation de problème sous forme semi-définie.

2 APPROCHES CONSTRUCTIVES POUR L'ANALYSE DE PIRE CAS

2.1 Performance Estimation Problem (PEP)

Durant ce projet, nous nous sommes particulièrement intéressés à une approche introduite par Yoel Drori et Marc Teboulle dans [6] appelé *Performance Estimation Problem* (PEP). Cette approche repose sur le fait que le comportement dans le pire cas d'un algorithme d'optimisation est lui même un problème d'optimisation : c'est le PEP. Cette approche a été reprise dans [12] sur lequel nous nous sommes basés pour notre projet.

Formellement, on considère un algorithme d'optimisation du premier ordre $\mathcal{A}$ et le comportement dans le pire cas après N appels de l'algorithme consiste à étudier le problème suivant :

$$\begin{aligned} \sup_{f,(x_i),x_\star} \quad & f(x_N) - f(x_\star) \\ \text{s.t.} \quad & f \in \mathcal{F} \\ & x_{i+1} = \mathcal{A}(x_0, ..., x_i, f(x_0), ..., f(x_i), f'(x_0), ..., f'(x_i)) \\ & f \text{atteint son minimum en } x_\star \\ & \|x_0 - x_\star\| \leq R \end{aligned}$$

où $\mathcal{F}$ est la classe de fonctions objectifs sur laquelle on cherche une borne théorique. La dernière contrainte est nécessaire afin que cette borne supérieure soit bien finie. La difficulté de résolution de ce problème provient notamment du fait que l'espace $\mathcal{F}$ est de dimension infinie. L'idée pour pouvoir gérer cette difficulté est une refomulation des contraintes utilisant des théorèmes d'interpolation. La reformulation de cette contrainte fonctionnelle permet ensuite d'aboutir à la résolution d'un problème de programmation semi-définie. Ces travaux ont alors montré qu'il est possible de dériver des bornes théoriques de convergence pour des algorithmes du premier ordre en résolvant cette reformulation du PEP.

2.2 Fonctions fortement convexes, lisses et interpolation convexe

Durant ce projet, la classe de fonctions principalement étudiée ($\mathcal{F}$ dans le PEP) est l'ensemble des fonctions fortement convexes et lisses.

$F : \mathbb{R}^d \to \mathbb{R}$ est dite μ-fortement convexe si : $F - \frac{\mu}{2}\|\cdot\|^2$ est convexe.

$F : \mathbb{R}^d \to \mathbb{R}$ est dite L-lisse si : ∇F est L-Lispchitz. Cela est aussi équivalent au fait que $\frac{L}{2}\|\cdot\|^2 - F$ est convexe.

Dans la suite, on notera $\mathcal{F}_{\mu,L}$ l'ensemble des fonctions μ-fortement convexes et L-lisses.

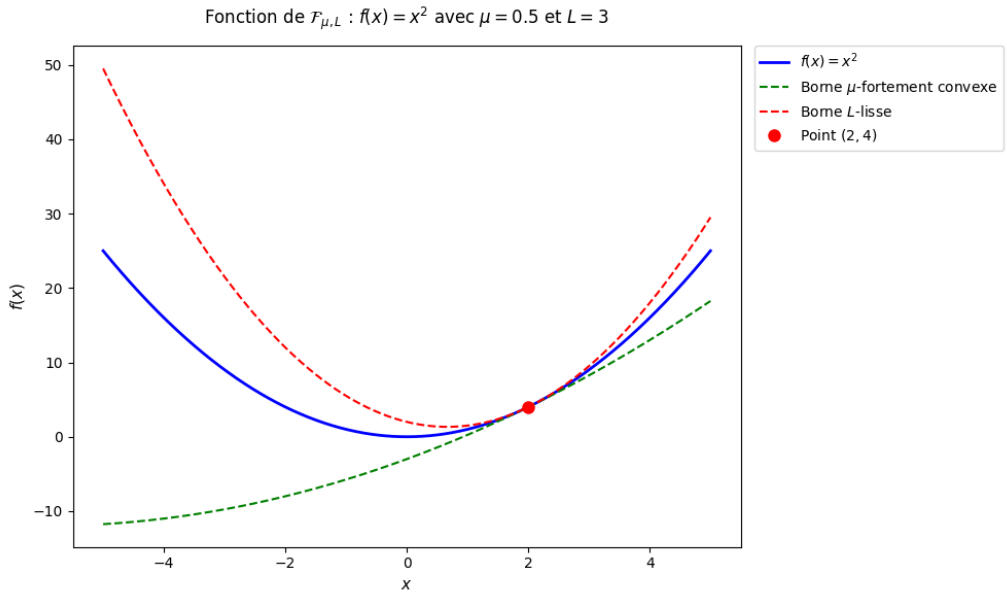

Figure 1 – Représentation d'une fonction fortement convexe et lisse

Graphiquement, la Figure 1 représente ce type de fonctions. Il est en effet possible d'encadrer la valeur de la fonction par deux fonctions quadratiques (de courbures μ et L). Sur ce graphe, cet encadrement se fait autour du point $x = 2$ mais cela est possible pour n'importe quel point $x \in \mathbb{R}$.

Comme évoqué précédemment, la résolution du PEP passe par une reformulation des contraintes. C'est notamment la contrainte $f \in \mathcal{F}$ qui est reformulé à l'aide du théorème d'interpolation sur les fonctions (fortement) convexes (et lisses). En effet, l'idée est de commencer par remplacer la contrainte fonctionnelle $f \in \mathcal{F}$ en interpolant f ainsi que son gradient sur des variables dans $\mathbb{R}^d$ à savoir les points x_i, $x_\star$. Schématiquement, cela revient à discrétiser la fonction f sur un ensemble de points. Il reste cependant maintenant à déterminer sous quelles conditions pour un ensemble de triplets $\{(x_i, g_i, f_i)\}$, il existe une fonction $f \in \mathcal{F}$ telle que :

$$\forall i \in I, \begin{cases} f(x_i) = f_i \\ \nabla f(x_i) = g_i \end{cases}$$

Cette problématique est notamment traitée dans [12]. Notamment, le Théorème 4 énoncé dans [12] nous assure que, si on se donne un ensemble de triplets $\{(x_i, g_i, f_i)\}_{i \in I}$:

$$\exists f \in \mathcal{F}_{\mu,L}, \forall i \in I, f(x_i) = f_i \text{ et } \nabla f(x_i) = g_i$$
$$\Longleftrightarrow$$
$$\forall i, j \in I, f_i - f_j \geq \langle g_j; x_i - x_j \rangle + \tfrac{1}{2L}\|g_i - g_j\|^2 + \tfrac{\mu}{2(1-\mu/L)}\|x_i - x_j - \tfrac{1}{L}(g_i - g_j)\|^2$$

Ce théorème dit d'interpolation lisse et fortement convexe nous permet ainsi de passer d'un problème d'optimisation sur un espace de dimension infinie à un problème d'optimisation sur un espace de dimension finie.

2.3 Programmation semi-définie - *Semi-Definite Programming (SDP)*

Au cours de l'approche PEP, le problème d'optimisation obtenu correspond à un problème de programmation semi-définie après avoir reformulé la contrainte fonctionnelle comme évoqué en section 2.2. Dans ce type de problème, l'inconnue est une matrice symétrique semi-définie positive. La fonction à optimiser est, quant à elle, linéaire, tout comme les contraintes. Ce type de problème est connu pour être relativement facile à résoudre (voir notamment [13]).

Pour $A_1, \ldots, A_m$ des matrices symétriques et $b_1, \ldots, b_m$ dans $\mathbb{R}$, la formulation standard d'un SDP est :

$$\begin{aligned} \min_{X \succcurlyeq 0} \quad & \mathrm{Tr}(CX) \\ \text{s.t.} \quad & \mathrm{Tr}(A_1 X) = b_1, \\ & \mathrm{Tr}(A_2 X) = b_2, \\ & \vdots \\ & \mathrm{Tr}(A_m X) = b_m. \end{aligned}$$

Des outils numériques ont par ailleurs été développé ce type de problème. Durant ce projet, nous avons notamment utilisé la librairie CVXPY pour Python [4, 1]. Ce module, en utilisant le solveur SCS [9, 8] nous a permis de résoudre les SDP obtenus afin d'obtenir les bornes de convergence des algorithmes étudiés.

2.4 Métriques d'études de convergence

Il convient de remarquer que la convergence d'un algorithme d'optimisation vers un optimum peut se traduire de plusieurs manières. Dans le PEP générique présenté en section 2.1, la métrique traduisant la convergence de l'algorithme est la précision sur la valeur de la fonction $f(x_N) - f(x_\star)$. Cependant, d'autres métriques peuvent être prises en compte comme la distance à l'optimum $\|x_N - x_\star\|$ ou encore la norme du gradient $\|\nabla f(x_N)\|$. Il n'existe pas véritablement de métriques plus légitime que l'autre et il est tout à fait possible de s'intéresser à certains critères spécifiques en fonction du problème.

Dans notre étude, nous avons choisi pour les algorithmes de descente de gradient et de descente de gradient stochastique de nous intéresser essentiellement à la distance à l'optimum en raison de sa simplicité.

Pour l'algorithme SAGA, nous utilisons un autre critère de convergence comme suggéré dans [3]. Ces critères sont ce qu'on appelle des fonctions de Lyapunov.

2.5 PEPit

L'étude numérique de problèmes de type PEP a par ailleurs été largement implémenté dans Python dans la librairie PEPit [7]. Le but de ce package est de simplifier l'analyse de pire cas pour une grande gamme d'algorithmes d'optimisation du premier ordre. Ce package implémente donc déjà l'analyse de nombreux algorithmes, dont les trois étudiés dans notre projet. L'idée est donc, dans ce projet, de comparer les garanties données par PEPit avec nos garanties mais aussi avec des garanties théoriques de convergence des algorithmes.

3 ALGORITHMES ÉTUDIÉS

Durant ce projet, nous avons particulièrement étudié trois algorithmes. Chacun de ces algorithmes sont dits du premier-ordre, c'est à dire qu'ils n'utilisent uniquement des calculs de gradient. Il sera donc naturel, lors de l'étude de la convergence de ces algorithmes de comparer ces convergences en fonction du nombre de gradients calculés.

3.1 Descente de gradient

L'algorithme de descente de gradient est un algorithme d'optimisation. On considère un problème de minimisation de type :

$$\min_x F(x)$$

où $F \in \mathcal{F}_{\mu,L}$.

Une étape de l'algorithme est décrite par : $x_{k+1} = x_k - \gamma \nabla F(x_k)$ où γ est le pas de l'algorithme. Il est supposé ici constant mais la plupart des variantes pratiques de l'algorithme consiste à faire varier ce pas. Pour un pas assez faible, cet algorithme converge bien vers une solution du problème [6]. La descente de gradient est un algorithme simple à implémenter et à analyser ce qui rend son utilisation encore aujourd'hui intéressantes.

3.2 SGD

Pour l'algorithme de descente de gradient stochastique (*Stochastic Gradient Descent* - SGD), on considère le problème de minimisation suivant :

$$\min_x \left\{ F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) \right\}$$

où chacune des fonctions f_i est μ-fortement convexe et L-lisse.

Une étape de l'algorithme correspond à :

— On tire un i_k uniformément au hasard parmi $\{1, ..., n\}$

— $x_{k+1} = x_k - \gamma \nabla f_{i_k}(x_k)$

Sur ce problème d'optimisation, la descente de gradient stochastique a une meilleure efficacité computationnelle que la descente de gradient classique, car pour chaque itération on ne calcule qu'un gradient au lieu de n pour la descente de gradient. Par ailleurs, le nombre total de calculs de gradient est en général plus faible que pour la descente de gradient, car les paramètres sont mis à jour à chaque calcul de gradient.

Cependant, cet algorithme ne converge pas réellement vers un optimum mais plutôt vers un voisinage d'un tel optimum. En effet, si ∇F s'annule au minimum, ce n'est pas le cas des gradients ∇f_i. Nous retrouverons ce résultat avec l'apparition d'un terme de variance dans notre étude de pire cas.

3.3 SAGA

L'algorithme SAGA est décrit initialement dans [3] qui en dérive une borne théorique de convergence. SAGA est un algorithme stochastique qui, au cours de son exécution, garde en mémoire les derniers gradients calculés. L'idée est d'ajouter de l'inertie dans l'algorithme de SGD.

Algorithm 1 Algorithme SAGA

Require: Vecteur initial $\mathbf{x}_0 \in \mathbb{R}^d$, table des dérivées initiales $\nabla f_i(\phi_i^0)$ pour $i = 1, \dots, n$, taille de pas γ

1: **for** itération $k = 0, 1, 2, \dots$ **do**

2: **Étape 1** : Sélectionner un indice j uniformément au hasard dans $\{1, 2, \dots, n\}$

3: **Étape 2** : Mettre à jour $\phi_j^{k+1} = \mathbf{x}^k$ et stocker $\nabla f_j(\phi_j^{k+1})$ dans la table. Les autres entrées de la table restent inchangées.

4: Mettre à jour $\mathbf{x}^{k+1}$ en utilisant :

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \gamma \left(\nabla f_j(\phi_j^{k+1}) - \nabla f_j(\phi_j^k) + \sum_{i=1}^{n} \nabla f_i(\phi_i^k) \right)$$

5: **end for**

L'algorithme présenté ici est légèrement différent de celui présenté en [3] par souci de simplifications. Initialement, l'algorithme utilise un opérateur de proximalité permettant une régularisation. Pour simplifier notre étude, nous avons décidé de ne pas inclure cet opérateur proximal.

Il est possible de montrer que l'algorithme SAGA est non biaisé dans le sens où :
$\mathbb{E}_k \left[\nabla f_j(\phi_j^{k+1}) - \nabla f_j(\phi_j^k) + \sum_{i=1}^n \nabla f_i(\phi_i^k) \right] = \nabla F(x^k)$.

Si SAGA peut être plus efficace que la descente de gradient stochastique (car il a tendance à moins osciller autour d'un minimum), il peut nécessiter un coût mémoire plus élevé. En effet, il faut stocker n gradients durant toute l'exécution de l'algorithme.

4
UNE APPROCHE CONSTRUCTIVE POUR L'ANALYSE DE PIRE CAS

Maintenant que les algorithmes étudiés ont été présentés, nous nous penchons plus particulièrement sur leur analyse de convergence dans les pires cas. L'idée est de dériver le PEP correspondant à chaque algorithme et de résoudre numériquement ce PEP, notamment en utilisant les outils présentés en section 2. On commencera par l'étude de descente de gradient, qui est la plus simple pour ensuite appliquer les mêmes outils aux cas plus compliqués de SGD et SAGA.

4.1 Descente de gradient

Le PEP de l'algorithme de descente de gradient a déjà été largement étudié, notamment dans [11] qui détaille l'analyse de pire cas. Cette partie constitue donc une simple reformulation de ce qui a été fait et est surtout utile pour comprendre la démarche de l'analyse de pire cas.

Ecriture du PEP et relaxation de la contrainte fonctionnelle : Le PEP étudié ici est :

$$\begin{array}{ll} \sup\limits_{F,x_0,x_\star} & \|x_1 - x_\star\|^2 \\ \text{s.t.} & F \in \mathcal{F}_{\mu,L} \\ & x_1 = x_0 - \gamma \nabla F(x_0) \\ & \nabla F(x_\star) = 0 \\ & \|x_0 - x_\star\|^2 \leq R^2 \end{array}$$

On considère ici une unique itération de l'algorithme. On pourra ensuite en déduire une borne générale pour un nombre quelconque d'itérations.

On simplifie ensuite ce PEP en y retirant x_1 et en relaxant la contrainte fonctionnelle par interpolation convexe. On va pour cela noter $f_0 = F(x_0)$, $g_0 = \nabla F(x_0)$, $f_\star = F(x_\star)$ et $g_\star = F(x_\star)$. On utilise alors le théorème d'interpolation convexe énoncé en 2.2 en se rappelant que $g_\star = 0$. On obtient alors ce problème d'optimisation équivalent :

$$\begin{array}{ll} \sup\limits_{f_0,g_0,f_\star,x_0,x_\star} & \|x_0 - \gamma g_0 - x_\star\|^2 \\ \text{s.t.} & f_\star - f_0 \geq \langle g_0; x_\star - x_0 \rangle + \frac{1}{2L}\|g_0\|^2 + \frac{\mu}{2(1-\mu/L)}\|x_0 - x_\star - \frac{1}{L}(g_0)\|^2 \\ & f_0 - f_\star \geq \frac{1}{2L}\|g_0\|^2 + \frac{\mu}{2(1-\mu/L)}\|x_0 - x_\star - \frac{1}{L}(g_0)\|^2 \\ & \|x_0 - x_\star\|^2 \leq R^2 \end{array}$$

Reformulation sous forme de SDP : On remarque que la fonction objectif ainsi que les contraintes sont linéaires en les coefficients de la matrice de Gram suivante : $G = \begin{pmatrix} \|x_0 - x_\star\|^2 & \langle g_0; x_0 - x_\star \rangle \\ \langle g_0; x_0 - x_\star \rangle & \|g_0\|^2 \end{pmatrix} \succcurlyeq 0$

Il existe A, A_0, A_1 et A_2 des matrices telles que le problème se réécrive :

$$\begin{array}{ll} \sup\limits_{f_0,f_\star,G\succcurlyeq 0} & \mathrm{Tr}(AG) \\ \text{s.t.} & f_0 - f_\star + \mathrm{Tr}(A_1 G) \leq 0 \\ & f_\star - f_0 + \mathrm{Tr}(A_2 G) \leq 0 \\ & \mathrm{Tr}(A_0 G) \leq R^2 \end{array}$$

On prend par exemple :

$$A = \begin{bmatrix} 1 & -\gamma \\ -\gamma & \gamma^2 \end{bmatrix}, A_0 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} \frac{L\mu}{2(L-\mu)} & -\frac{L}{2L-2\mu} \\ -\frac{L}{2L-2\mu} & \frac{1}{2(L-\mu)} \end{bmatrix}, A_2 = \begin{bmatrix} \frac{L\mu}{2(L-\mu)} & -\frac{\mu}{2L-2\mu} \\ -\frac{\mu}{2L-2\mu} & \frac{1}{2(L-\mu)} \end{bmatrix}$$

On peut alors utiliser le package CVXPY avec le solveur SCS pour résoudre ce SDP. On peut par ailleurs comparer avec la valeur théorique déterminée en [11] : $\max \|x_1 - x_\star\|^2 = \max\left((1-\mu\gamma)^2, (1-L\gamma)^2\right) R^2$

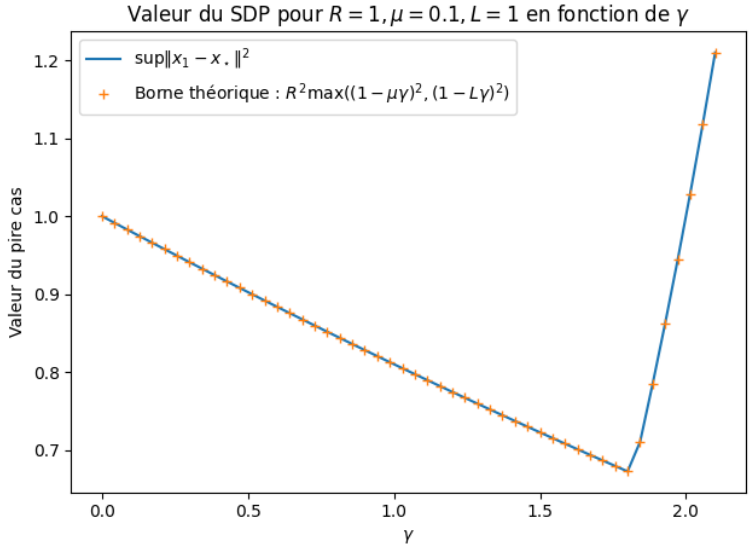

Figure 2 – Résolution numérique du SDP pour l'algorithme de descente de gradient

Nous avons effectué cette résolution numérique pour différentes valeurs de γ. Pour tracer la Figure 2, on a fixé les paramètres $\mu = 0.1$ et $L = 1$. On a pris une condition initiale $R = 1$ et on fait varier γ entre 0 et 2.1. On a aussi tracé sur ce graphe la valeur théorique du sup $\|x_1 - x_\star\|^2$ ce qui permet de confirmer la validité de notre résolution. On peut quantifier l'erreur en calculant la moyenne des carrés des différences entre les valeurs théoriques et celles renvoyées par CVXPY. Ici, cette erreur vaut 10^{-10} ce qui est très faible au vu de l'envergure des valeurs prises par la valeur du pire cas.

Par ailleurs, on constate que le maximum calculé est supérieur à $R^2 = 1$ pour les γ supérieurs à 2 ce qui justifie le choix d'un γ assez petit pour s'assurer de la convergence de l'algorithme. En effet, si le maximum est supérieur à R^2, on pourrait, après une itération de l'algorithme se retrouver plus loin de l'optimum que précédemment.

On remarque qu'ici, les points admissibles correspondent à certaines configurations initiales et donc constitue une borne inférieure sur le taux de convergence. Par le théorème de dualité faible, s'intéresser au problème dual permet d'obtenir naturellement des bornes supérieures sur ce taux de convergence.

Problème dual : On s'intéresse au problème dual du SDP précédent. On introduit donc des multiplicateurs de Lagrange pour chacune des contraintes λ_1, λ_2 et τ. Le Lagrangien s'écrit alors :

$$\begin{aligned}\mathcal{L} &= \mathrm{Tr}(AG) - \lambda_1(f_0 - f_\star + \mathrm{Tr}(A_1 G)) - \lambda_2(f_\star - f_0 + \mathrm{Tr}(A_2 G)) - \tau(\mathrm{Tr}(A_0 G) - R^2) \\ &= \mathrm{Tr}(SG) + (\lambda_1 - \lambda_2)(f_\star - f_0) + \tau R^2\end{aligned}$$

en notant $S = A - \lambda_1 A_1 - \lambda_2 A_2 - \tau A_0$ qui est symétrique.

$$\sup_{f_0, f_\star, G \succcurlyeq 0} \mathcal{L} = \begin{cases} +\infty & \text{si } S \not\preccurlyeq 0 \\ +\infty & \text{si } \lambda_1 \neq \lambda_2 \\ \tau R^2 & \text{sinon} \end{cases}$$

Il vient donc que le problème dual s'écrit :

$$\begin{aligned} \inf_{\lambda_1, \lambda_2, \tau \geq 0} \quad & \tau R^2 \\ \text{s.t.} \quad & S \preccurlyeq 0 \\ & \lambda_1 = \lambda_2 \end{aligned}$$

Tout point admissible nous donne alors une borne supérieure sur le taux de convergence. Par ailleurs, on voit ici directement apparaître une dépendance en R^2. Celle-ci permet donc d'obtenir une borne plus générale pour

un nombre quelconque d'itérations. En effet, si on note $\tau_\star$ une solution admissible du problème de minimisation précédent, on a $\|x_1 - x_\star\|^2 \leq \tau_\star R^2$. En itérant le processus, il vient que $\|x_N - x_\star\|^2 \leq \tau_\star^N R^2$.

On peut, de la même manière que pour le problème primal, résoudre numériquement le dual pour trouver ce $\tau_\star$ en fonction de γ.

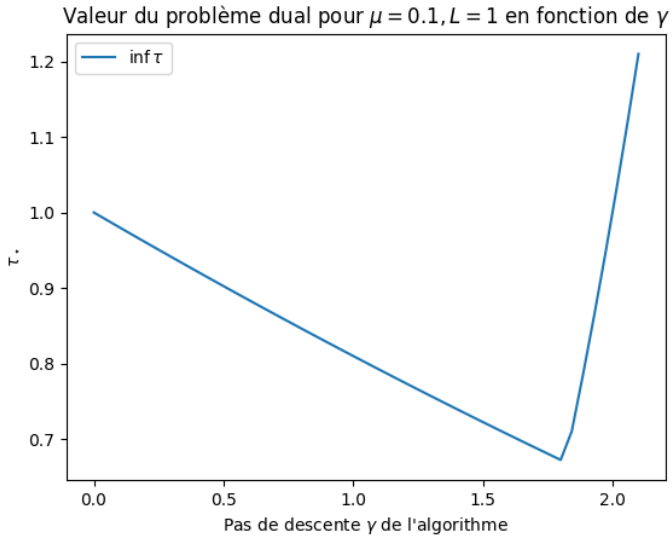

Figure 3 – Résolution numérique du problème dual pour la descente de gradient

Graphiquement, on observe qu'on obtient les mêmes résultats en Figure 3 qu'en Figure 2. Cela se traduit par le fait que le saut de dualité dans ce cas particulier est nul.

On retrouve donc bien la convergence exponentielle connue pour l'algorithme de descente de gradient, à condition que $\tau_\star < 1$, ce qui est bien le cas pour un pas de descente γ assez petit : il faudrait prendre $\gamma \in (0, 2/L)$ pour avoir un $\tau_\star < 1$ et donc une garantie de convergence de l'algorithme.

Le γ optimal permettant d'obtenir la meilleure convergence de l'algorithme de descente de gradient est $\gamma_\star = \frac{2}{\mu+L}$.

4.2 SGD

Maintenant qu'on a déroulé les étapes de l'approche constructive de l'analyse de pire cas, comme présentées dans [11] pour le cas simple de la descente de gradient, il est intéressant d'adapter la même analyse pour des algorithmes plus complexes, comme l'algorithme de descente de gradient stochastique.

Etablissement du PEP : On s'intéresse ici au calcul de la garantie de pire cas pour une seule étape de l'algorithme SGD. On cherche donc le

$$\max \mathbb{E}\left[\|x_1 - x_\star\|^2\right]$$

avec les contraintes suivantes :

$$\begin{cases} f_i \in \mathcal{F}_{\mu,L} \\ \|x_0 - x_\star\|^2 \leq R^2 \\ \nabla F(x_\star) = 0 \end{cases}$$

Une hypothèse essentielle pour assurer la convergence est une variance bornée des gradients à l'optimum (noté $x_\star$) i.e : $\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x_\star)\|^2 \leq v^2$. Cette condition va donc être ajoutée aux contraintes du problème.

Par ailleurs, on peut aussi écrire $\mathbb{E}\left[\|x_1 - x_\star\|^2\right] = \frac{1}{n}\sum_{i=1}^{n}\|x_0 - \gamma\nabla f_i(x_0) - x_\star\|^2$.

Une fois ces simplifications effectuées, on utilise la propriété d'interpolation convexe : on ne considère que les valeurs des f_i et ∇f_i aux points x_0 et $x_\star$. On notera ainsi $f_i = f_i(x_0)$, $g_i = \nabla f_i(x_0)$, $f_i^\star = f_i(x_\star)$ et $g_i^\star = \nabla f_i(x_\star)$. On obtient alors ce problème d'optimisation :

$$\begin{aligned}
\sup_{(f_i),(g_i),(f_i^\star),(g_i^\star),x_0,x_\star} \quad & \frac{1}{n}\sum_{i=1}^{n}\|x_0 - \gamma g_i - x_\star\|^2 \\
\text{s.t.} \quad & f_i^\star \geq f_i + \langle g_i; x_\star - x_0\rangle + \tfrac{1}{2L}\|g_i - g_i^\star\|^2 + \tfrac{\mu}{2(1-\mu/L)}\|x_0 - x_\star - \tfrac{1}{L}(g_i - g_i^\star)\|^2 \\
& f_i \geq f_i^\star + \langle g_i^\star; x_0 - x_\star\rangle + \tfrac{1}{2L}\|g_i - g_i^\star\|^2 + \tfrac{\mu}{2(1-\mu/L)}\|x_0 - x_\star - \tfrac{1}{L}(g_i - g_i^\star)\|^2 \\
& \|x_0 - x_\star\|^2 \leq R^2 \\
& \tfrac{1}{n}\sum_{i=1}^{n}\|g_i^\star\|^2 \leq v^2 \\
& \tfrac{1}{n}\sum_{i=1}^{n} g_i^\star = 0
\end{aligned}$$

La dernière contrainte permet d'assurer que $x_\star$ est bien un minimiseur de F. On va retirer cette contrainte dans la suite en fixant $g_n^\star = -\sum_{i=1}^{n-1} g_i^\star$, ce qui permet déjà de réduire la dimension de l'espace sur lequel on résout notre problème d'optimisation mais surtout d'obtenir des contraintes qualifiées, afin de garantir notamment la dualité forte pour ce problème d'optimisation.

Reformulation sous forme de SDP : Pour reformuler le PEP sous forme de SDP, on commence par introduire quelques notations, inspirées de [5]. Soient $P \in \mathbb{R}^{d\times 2n}$ et $F \in \mathbb{R}^{2n}$ définies par :

$$P = \left[x_0 - x_\star \mid g_1 \dots g_n \mid g_1^\star \dots g_{n-1}^\star\right]$$

$$F = \left[f_1 \dots f_n \mid f_1^\star \dots f_n^\star\right]$$

La matrice de Gram introduite alors, de la même manière que pour la descente de gradient, est $G = P^T P \succcurlyeq 0$. On a donc la fonction objectif ainsi que les contraintes du PEP linéaires en les coefficients de G et de F, ce qui permet de reformuler le problème sous forme de SDP.

On remarque que :

$$\begin{aligned}
x_0 - x_\star &= Pe_1 \\
g_i &= Pe_{i+1} && \text{pour } i = 1,\dots,n \\
g_i^\star &= Pe_{n+i+1} && \text{pour } i = 1\dots n-1 \\
f_i &= Fe_i && \text{pour } i = 1,\dots,n \\
f_i^\star &= Fe_{n+i} && \text{pour } i = 1,\dots n
\end{aligned}$$

Cette reformulation s'écrit donc :

$$\begin{aligned}
\sup_{G\succcurlyeq 0, F} \quad & \mathrm{Tr}(AG) \\
\text{s.t.} \quad & F(e_i - e_{n+1}) + \mathrm{Tr}(A_{1,i}G) \leq 0 \\
& F(e_{n+i} - e_i) + \mathrm{Tr}(A_{2,i}G) \leq 0 \\
& \mathrm{Tr}(A_0 G) \leq R^2 \\
& \mathrm{Tr}(A_{\mathrm{var}} G) \leq v^2
\end{aligned}$$

qui correspond bien à un SDP où l'on choisit les matrices $A, \dots$ pour coder les précédents termes :

$$
\begin{aligned}
A = &\quad \frac{1}{n}\sum_{i=1}^{n}(e_1 - \gamma e_{i+1}) \odot (e_1 - \gamma e_{i+1}) \\
A_{1,i} = &\quad e_{i+1} \odot e_1 + \tfrac{1}{2L}(e_{i+1} - e_{n+i+1}) \odot (e_{i+1} - e_{n+i+1}) \\
&\quad + \tfrac{\mu}{2(1-\mu/L)} \left(e_1 - \tfrac{1}{L}(e_{i+1} - e_{n+i+1})\right) \odot \left(e_1 - \tfrac{1}{L}(e_{i+1} - e_{n+i+1})\right) \\
A_{2,i} = &\quad e_{n+i+1} \odot e_1 + \tfrac{1}{2L}(e_{i+1} - e_{n+i+1}) \odot (e_{i+1} - e_{n+i+1}) \\
&\quad + \tfrac{\mu}{2(1-\mu/L)} \left(e_1 - \tfrac{1}{L}(e_{i+1} - e_{n+i+1})\right) \odot \left(e_1 - \tfrac{1}{L}(e_{i+1} - e_{n+i+1})\right) \\
A_0 = &\quad e_1 \odot e_1 \\
A_{\text{var}} = &\quad \frac{1}{n}\sum_{i=1}^{n} e_{n+i+1} \odot e_{n+i+1}
\end{aligned}
$$

On peut maintenant résoudre numériquement ce SDP à l'aide de CVXPY et du solveurs SCS, avec une certaine configuration initiale.

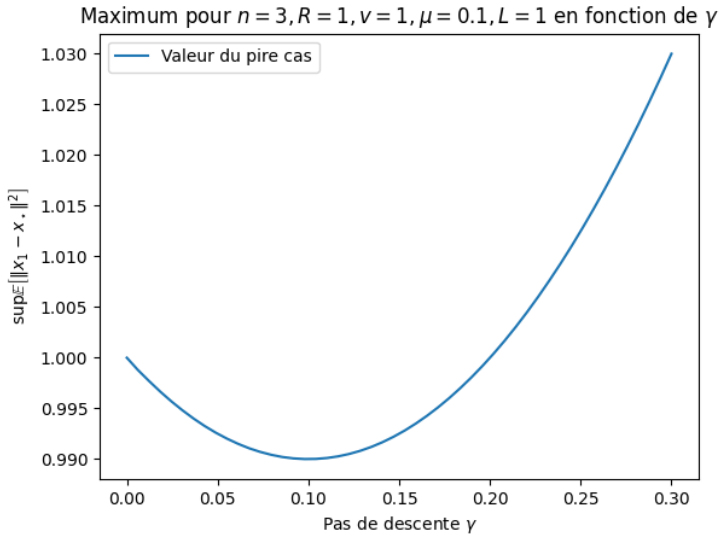

Figure 4 – Résolution numérique du SDP pour SGD

La Figure 4 montre la valeur du SDP en fonction de γ. Comme pour l'algorithme de descente de gradient, pour que l'algorithme ait des garanties de convergence, il faut s'assurer de prendre un pas de descente γ assez petit.

Pour tracer la Figure 4, on a choisi un certain nombre de fonctions $n = 3$. Cependant, il peut aussi être intéressant d'étudier la dépendance de la valeur du pire cas en fonction de ce n.

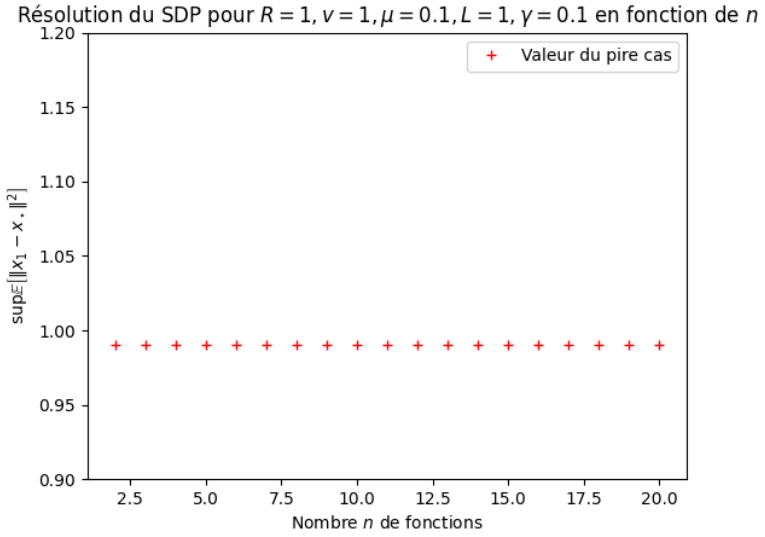

Figure 5 – Résolution numérique du SDP de SGD en fonction du nombre n de fonctions

La Figure 5 montre bien que la borne théorique calculée est indépendante de n. Cela peut sembler étonnant au premier abord, mais en fait s'explique par les nombreuses symétries présentes dans le SDP. Cela permet de justifier l'utilisation de SGD dans le cas d'un jeu de données de grande taille.

Problème dual : Comme pour l'algorithme de descente de gradient, il peut aussi être intéressant d'étudier le problème dual de ce SDP. On introduit donc des multiplicateurs de Lagrange pour chacune des contraintes précédentes et on obtient un lagrangien de la forme :

$$\begin{aligned}\mathcal{L} &= \mathrm{Tr}(AG) - \sum_{i=1}^{n} \lambda_{1,i}(f_i - f_i^\star + \mathrm{Tr}(A_{1,i}G)) - \sum_{i=1}^{n} \lambda_{2,i}(f_i^\star - f_i + \mathrm{Tr}(A_{2,i}G)) - \rho(\mathrm{Tr}(A_0 G) - R^2) \\ &\quad -\tau(\mathrm{Tr}(A_{\mathrm{var}}G) - v^2) - \nu \mathrm{Tr}(A_{\mathrm{sum}}G) \\ &= \mathrm{Tr}(SG) + \sum_{i=1}^{n} (\lambda_{1,i} - \lambda_{2,i})(f_i^\star - f_i) + \rho R^2 + \tau v^2\end{aligned}$$

où l'on a noté $S = A - \sum_{i=1}^{n} \lambda_{1,i} A_{1,i} - \sum_{i=1}^{n} \lambda_{2,i} A_{2,i} - \rho A_0 - \tau A_{\mathrm{var}}$

A partir de ce lagrangien, on peut donc dériver le problème dual qui s'écrit :

$$\begin{aligned}\inf_{(\lambda_{1,i}),(\lambda_{2,i}),\rho,\tau} \quad & \rho R^2 + \tau v^2 \\ \text{s.t.} \quad & \lambda_{1,i} = \lambda_{2,i} \\ & S \preccurlyeq 0 \\ & \lambda_{1,i}, \lambda_{2,i}, \rho, \tau \geq 0\end{aligned}$$

Sous cette forme on voit apparaître une borne dépendant de R^2 et de v^2. La résolution de ce problème dual permet donc d'étudier plus précisément la dépendance en chacun de ces deux paramètres. La dépendance en R^2 est en fait cruciale pour pouvoir, comme dans la descente de gradient, déterminer des bornes de convergence après plusieurs itérations.

Pour calculer un tel point admissible, on peut par exemple minimiser τ en forçant une certaine valeur pour ρ. On va utiliser ce qu'on a déjà fait pour la descente de gradient pour fixer la valeur de ρ. On va contraindre $\rho = \rho_\star = \left(1 - \frac{\mu}{L}\right)^2$ qui correspondrait à $\gamma = \frac{1}{L}$ et résoudre le problème :

$$\begin{aligned} \inf_{(\lambda_{1,i}),(\lambda_{2,i}),\tau} \quad & \tau \\ \text{s.t.} \quad & \lambda_{1,i} = \lambda_{2,i} \\ & S \preccurlyeq 0 \\ & \lambda_{1,i}, \lambda_{2,i}, \tau \geq 0 \end{aligned}$$

Si on note $\tau_\star$ un point admissible, on aura alors : $\mathbb{E}\left[\|x_1 - x_\star\|^2\right] \leq \rho_\star R^2 + \tau_\star v^2$ et ceci pour tout R^2 donc on obtient en déroulant les inégalités :

$$\begin{aligned} \mathbb{E}\left[\|x_k - x_\star\|^2\right] &\leq \rho_\star^k \|x_0 - x_\star\|^2 + \tau_\star v^2 \sum_{i=0}^{k-1} \rho_\star^i \\ &\leq \rho_\star^k \|x_0 - x_\star\|^2 + \tau_\star v^2 \tfrac{1-\rho_\star^k}{1-\rho_\star} \\ &\leq \rho_\star^k \|x_0 - x_\star\|^2 + \tfrac{\tau_\star v^2}{1-\rho_\star} \end{aligned}$$

On retrouve bien le fait que SGD converge vers une boule autour d'un optimum.

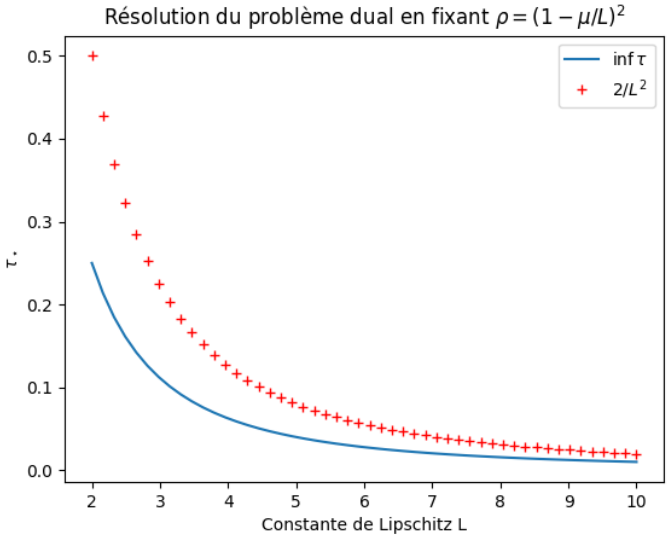

Figure 6 – Résolution numérique du dual pour l'algorithme SGD

Numériquement, on peut majorer, pour $\rho_\star = \left(1 - \frac{\mu}{L}\right)^2$, $\tau_\star$ par $\frac{2}{L^2}$ comme le montre la Figure 6.

4.3 SAGA

Métrique de convergence : Appliquons maintenant l'approche constructive précédente à l'algorithme SAGA. La principale différence avec les deux constructions précédentes est la métrique d'étude de la convergence de l'algorithme. Pour GD et SGD, nous avions fait le choix d'une métrique simple, la distance au carré à l'optimum recherché. Ici, comme proposé par [3], on va utiliser une fonction de Lyapunov. On reprend les notations introduites dans l'Algorithme 1. On pose alors :

$$T^k = \frac{1}{n} \sum_{i=1}^{n} \left(f_i(\phi_i^k) - f_i(x_\star) - \langle \nabla f_i(x_\star), \phi_i^k - x_\star \rangle \right) + c\|x^k - x_\star\|^2$$

Cette fonction de Lyapunov est bien positive grâce à la contrainte de convexité sur chacune des fonctions f_i. On a même $T^k \geq c\|x^k - x_\star\|$ qui permet tout de même de dériver une borne sur la distance à l'optimum.

Etablissement du PEP : Comme pour l'algorithme SGD, on va de nouveau étudier une itération de l'algorithme, on cherche donc

$$\max \mathbb{E}\left[T^1\right]$$

sous les contraintes :

$$\begin{cases} f_i \in \mathcal{F}_{\mu,L} \\ T^0 \leq T \\ \nabla F(x_\star) = 0 \end{cases}$$

Commençons par décrire plus en détail la fonction objectif :

$$\mathbb{E}\left[T^1\right] = \frac{1}{n}\mathbb{E}\left[\sum_{i=1}^n f_i(\phi_i^1)\right] - F(x_\star) - \frac{1}{n}\mathbb{E}\left[\sum_{i=1}^n \langle \nabla f_i(x_\star), \phi_i^1 - x_\star\rangle\right] + c\mathbb{E}\left[\|x^1 - x_\star\|^2\right]$$

Simplifions ensuite chacun des termes :

$$\mathbb{E}\left[\sum_{i=1}^n f_i(\phi_i^1)\right] = F(x^0) + \left(1 - \frac{1}{n}\right)\sum_{i=1}^n f_i(\phi_i^0)$$

$$\begin{aligned} \mathbb{E}\left[\sum_{i=1}^n \langle \nabla f_i(x_\star), \phi_i^1 - x_\star\rangle\right] &= \langle \nabla F(x_\star), x^0 - x_\star\rangle + \left(1 - \tfrac{1}{n}\right)\sum_{i=1}^n \langle \nabla f_i(x_\star), \phi_i^0 - x_\star\rangle \\ &= \left(1 - \tfrac{1}{n}\right)\sum_{i=1}^n \langle \nabla f_i(x_\star), \phi_i^0 - x_\star\rangle \end{aligned}$$

$$\mathbb{E}\left[\|x^1 - x_\star\|^2\right] = \tfrac{1}{n}\sum_{i=1}^n \|x^0 - \gamma(\nabla f_i(x^0) - \nabla f_i(\phi_j^0) + \bar{f}^0) - x_\star\|^2$$

Comme dans les deux cas précédents, il est maintenant possible de simplifier le PEP en relaxant la contrainte fonctionnelle à l'aide d'une interpolation en les ϕ_i, x^0 et $x_\star$. On va donc noter :

$$\begin{aligned} f_i &= f_i(x^0) \\ g_i &= \nabla f_i(x^0) \\ f_i^\star &= f_i(x_\star) \\ g_i^\star &= \nabla f_i(x_\star) \\ f_{\phi,i} &= f_i(\phi_i^0) \\ g_{\phi,i} &= \nabla f_i(\phi_i^0) \end{aligned}$$

Mise sous forme de SDP : On met alors sous forme de SDP en introduisant une matrice de Gram G. L'unique différence avec la résolution de SGD consiste en la taille de la matrice de Gram qui doit maintenant contenir aussi les $(\phi_i^0 - x_\star)$ et les $g_{\phi,i}$

Ainsi en reprenant les notations de la section précédente sur SGD, on pose

$$P = \left[x_0 - x_\star \mid \phi_1^0 - x_\star \dots \phi_n^0 - x_\star \mid g_1 \dots g_n \mid g_{\phi,1} \dots g_{\phi,n} \mid g_1^\star \dots g_{n-1}^\star\right] \in \mathbb{R}^{d\times 4n}$$

pour ensuite noter $G = P^T P \succcurlyeq 0$ afin d'écrire la fonction objectif sous la forme :

$$\frac{1}{n^2}\sum_{i=1}^n f_i + \left(1 - \frac{1}{n}\right)\frac{1}{n}\sum_{i=1}^n f_{\phi,i} - \frac{1}{n}\sum_{i=1}^n f_i^\star + \mathrm{Tr}(AG)$$

Les contraintes d'interpolation convexe s'écrivent alors :

$$\begin{aligned} f_i - f_i^\star + \mathrm{Tr}(A_{1,i}G) &\leq 0 \\ f_i^\star - f_i + \mathrm{Tr}(A_{2,i}G) &\leq 0 \\ f_i - f_{\phi,i} + \mathrm{Tr}(A_{3,i}G) &\leq 0 \\ f_{\phi,i} - f_i + \mathrm{Tr}(A_{4,i}G) &\leq 0 \\ f_{\phi,i} - f_i^\star + \mathrm{Tr}(A_{5,i}G) &\leq 0 \\ f_i^\star - f_{\phi,i} + \mathrm{Tr}(A_{6,i}G) &\leq 0 \end{aligned}$$

La contrainte sur la valeur initiale de la fonction de Lyapunov s'écrit simplement :

$$\frac{1}{n}\sum_{i=1}^{n}(f_{\phi,i}-f_i^\star)+\mathrm{Tr}(A_0G)\leq T$$

On résout alors le problème numériquement comme dans les deux cas précédents à l'aide du solveur SCS. On commence par s'intéresser à la dépendance de l'optimum en fonction du paramètre c utilisé dans la fonction de Lyapunov.

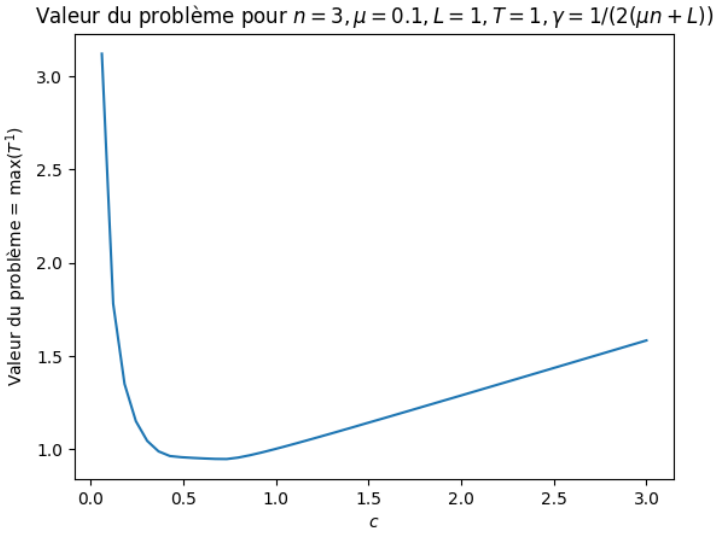

Figure 7 – Résolution du SDP de SAGA pour différentes valeurs de c

Pour cette résolution, on a pris $\mu = 0.1, L = 1, T = 1$ et $\gamma = \frac{1}{2(\mu n+L)}$ comme suggéré dans [3]. On a représenté la valeur du maximum de la fonction de Lyapunov en fonction du paramètre c. Dans [3], la constante c est fixée à $\frac{1}{2n\gamma(1-\mu\gamma)}$.

Cette valeur ne correspond pas exactement au minimum trouvé numériquement mais est plutôt par souci de simplicité pour effectuer les calculs théoriques de convergence comme dans le Théorème 1 de [3].

On va donc maintenant fixer le paramètre paramètre $c = \frac{1}{2n\gamma(1-\mu\gamma)}$ et étudier la valeur du problème en fonction de μ afin de comparer la borne de convergence de SAGA avec les algorithmes précédents. Théoriquement, [3] nous donne une borne théorique : $\max \mathbb{E}\left[T^1\right] = (1-\gamma\mu)T^0$.

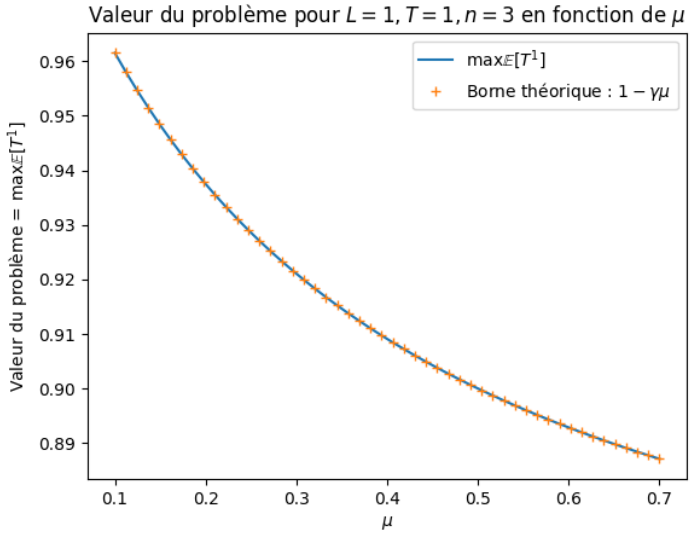

Figure 8 – Résolution du SDP de SAGA pour différentes valeurs de μ

Sur la Figure 8, on observe qu'on retrouve bien la borne théorique attendu. Ici, l'erreur est de l'ordre de 10^{-11} ce qui est bien négligeable. La borne théorique est donc atteinte. A partir de cette inégalité, il est alors facile d'étendre le résultat après plusieurs itérations :

$\mathbb{E}\left[T^k\right] \leq (1-\gamma\mu)^k T^0$ puis $\|x^k - x_\star\|^2 \leq \frac{(1-\gamma\mu)^k}{c} T^0$.

On obtient donc une borne théorique de convergence pour SAGA, qui converge de façon exponentielle sous l'hypothèse de bonnes conditions initiales et de choisir le pas de descente de l'algorithme de manière judicieuse. Cette borne est atteinte ce qui signifie qu'on ne peut faire mieux, on ne peut pas espérer une meilleure convergence pour cet algorithme.

5 EXPÉRIMENTATIONS NUMÉRIQUES

5.1 Régression linéaire

On considère un problème de régression linéaire : on dispose de données $(x_i) \in (\mathbb{R}^d)^I$ et $(y_i) \in \mathbb{R}^I$ et on cherche à expliquer Y à partir d'une fonction linéaire de X. Pour les expérimentations, on génère de manière aléatoire ces données à l'aide de la fonction `datasets.make_regression` du module scikit-learn [10] pour Python.

Le critère à minimiser ici est donc

$$F(\beta) = \frac{1}{n} \sum_{i=1}^{n} \|X_i\beta - Y_i\|^2$$

. On a alors $f_i = \|X_i\beta - Y_i\|^2$, $\nabla f_i(\beta) = 2X_i^\top (X_i\beta - Y_i)$ et la hessienne $\nabla^2 f_i(\beta) = 2X_i^\top X_i$. Les valeurs propres de $2X_i^\top X_i$ sont nulles, sauf une égale à $2\|X_i\|^2$.

Pour assurer que chaque f_i soit μ-fortement convexe, il est nécessaire d'ajouter un terme de régularisation à la fonction $F(\beta)$. La nouvelle fonction objectif régularisée devient alors :

$$F_{\text{reg}}(\beta) = \frac{1}{n}\sum_{i=1}^{n} \|X_i\beta - Y_i\|^2 + \frac{\lambda}{2}\|\beta\|^2,$$

où $\lambda > 0$ est le coefficient de régularisation contrôle l'importance de la pénalisation. Ce terme $\frac{\lambda}{2}\|\beta\|^2$ garantit que les fonctions f_i sont μ-fortement convexes, avec $\mu = \lambda$.

Chaque terme $\|X_i\beta - Y_i\|^2$ est alors L_i-lisse, avec $L_i = 2\|X_i\|^2 + \lambda$. On peut prendre donc

$$L = 2\max_i \|X_i\|^2 + \lambda.$$

Nous pouvons maintenant comparer les résultats théoriques de la partie précédente avec les résultats expérimentaux.

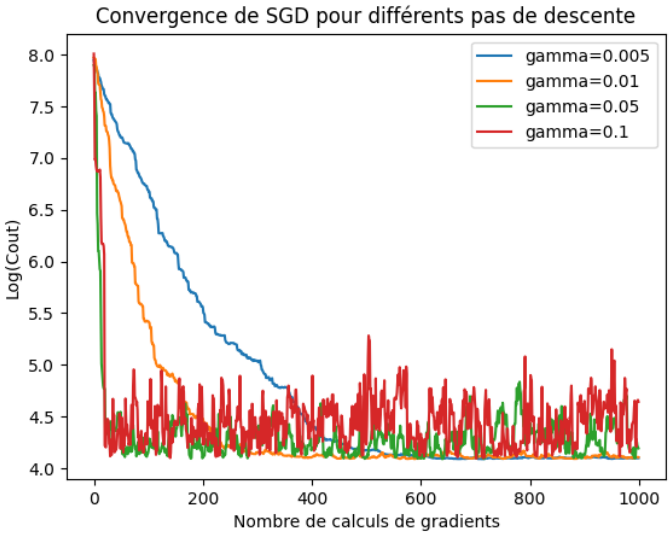

Figure 9 – Convergence de SGD en fonction du pas de descente γ

Sur la Figure 9, on a représenté la convergence de SGD pour notre problème de régression linéaire pour différentes valeurs du pas de descente. On retrouve bien la convergence dans une boule autour de l'optimum, ce qui se traduit par les oscillations que l'on observe sur le graphe.

Nous pouvons faire de même avec l'algorithme SAGA :

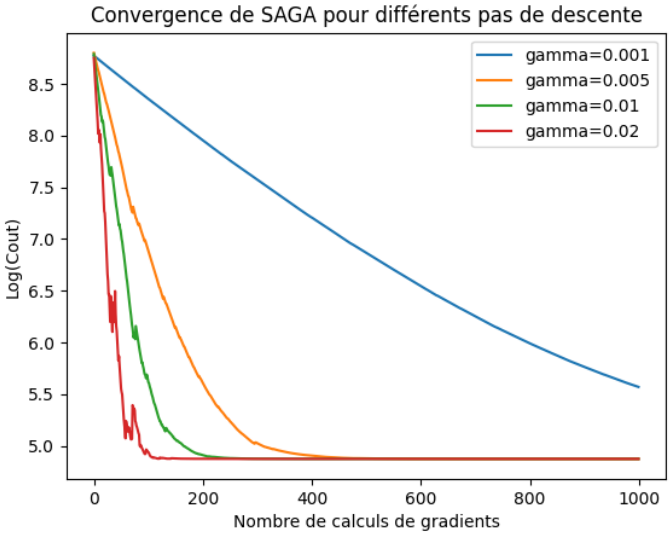

Figure 10 – Convergence de SAGA en fonction du pas de descente

Cette fois-ci on retrouve bien la convergence exponentielle que l'on avait trouvée dans la partie précédente.

5.2 Régression logistique

Considérons maintenant le problème de classification de la régression logistique. Pour la régression logistique, on traite un problème de classification binaire où $y_i \in 0, 1$. Le modèle cherche à estimer la probabilité d'appartenance à la classe 1 à travers une fonction sigmoïde appliquée à une combinaison linéaire des variables explicatives. La probabilité que $y_i = 1$ est modélisée par : $\mathbb{P}(y_i = 1|x_i) = \sigma(\beta_0 + x_i^T\beta)$ où $\sigma(z) = \frac{1}{1+e^{-z}}$ est la fonction sigmoïde. Pour les expérimentations on utilise le jeu de données "Mushrooms" de la librairie LIBSVM [2].

Le critère à minimiser est la log-vraisemblance négative (aussi appelée log loss ou cross-entropy) :

$$F(\beta) = \frac{1}{n}\sum_{i=1}^{n}\big[-Y_i\log(\sigma(X_i\beta)) - (1-Y_i)\log(1-\sigma(X_i\beta))\big],$$

On a alors

$$f_i = \big[-Y_i\log(\sigma(X_i\beta)) - (1-Y_i)\log(1-\sigma(X_i\beta))\big],$$

et son gradient est donné par :

$$\nabla f_i(\beta) = (\sigma(X_i\beta) - Y_i)X_i^\top.$$

La Hessienne de f_i s'écrit :

$$\nabla^2 f_i(\beta) = \sigma(X_i\beta)(1-\sigma(X_i\beta))X_i^\top X_i.$$

Les valeurs propres de $\sigma(X_i\beta)(1-\sigma(X_i\beta))X_i^\top X_i$ sont nulles, sauf une inférieure à $\frac{1}{4}\|X_i\|^2$, puisque $\sigma(z)(1-\sigma(z)) \leq \frac{1}{4}$.

Pour assurer que chaque f_i soit μ-fortement convexe, il est nécessaire d'ajouter un terme de régularisation à la fonction $F(\beta)$. La nouvelle fonction objectif régularisée devient alors :

$$F_{\text{reg}}(\beta) = \frac{1}{n}\sum_{i=1}^{n}\big[-Y_i\log(\sigma(X_i\beta)) - (1-Y_i)\log(1-\sigma(X_i\beta))\big] + \frac{\lambda}{2}\|\beta\|^2,$$

où $\lambda > 0$ est le coefficient de régularisation qui contrôle l'importance de la pénalisation. Ce terme $\frac{\lambda}{2}\|\beta\|^2$ garantit que les fonctions f_i sont μ-fortement convexes, avec $\mu = \lambda$.

Chaque f_i est alors L_i-lisse, avec :

$$L_i = \frac{\|X_i\|^2}{4} + \lambda.$$

On peut donc prendre une constante globale :

$$L = \frac{\max_i \|X_i\|^2}{4} + \lambda.$$

Numériquement, on compare la convergence de nos 3 algorithmes :

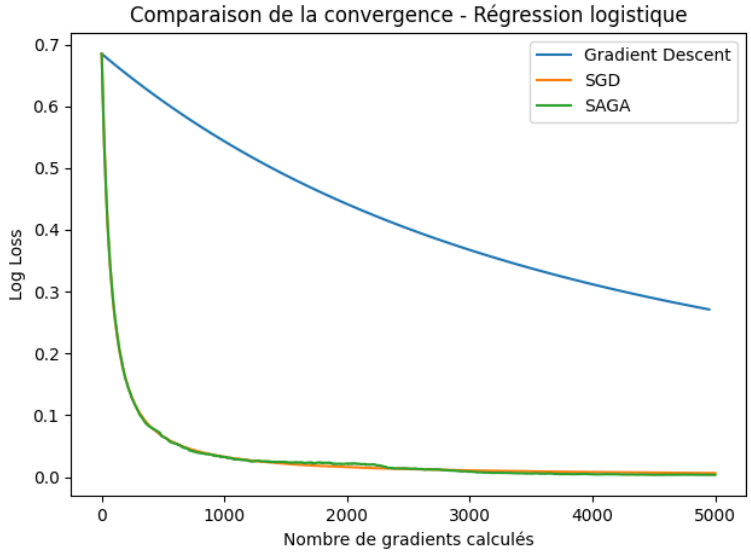

Figure 11 – Comparaison de convergence des 3 algorithmes sur le jeu de données "Mushrooms"

On retrouve le résultat connu dans lequel les algorithmes stochastiques convergent beaucoup plus rapidement que l'algorithme de descente de gradient classique.

Numériquement, sur ce jeu de données on a $\max_i \|X_i\|^2 \approx 4.7$. On fixe le terme de régularisation avec $\lambda = 0.1$ ce qui nous donne $\mu = 0.1$ et $L \approx 1.28$. Pour la descente de gradient, le pas de descente doit donc être inférieur à environ 1.56 pour avoir des garanties de convergence.

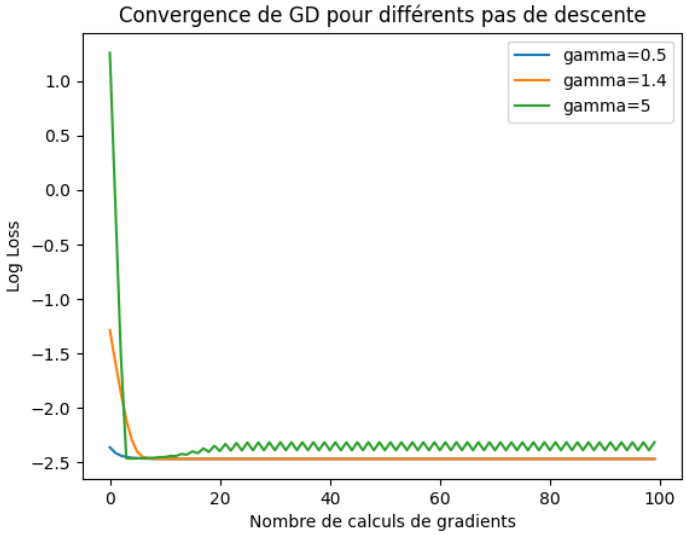

Figure 12 – Convergences de GD pour différents pas de descente

Numériquement, on voit sur la Figure 12, qu'on a bien convergence pour des γ plus petits que cette valeur numérique. Pour des valeurs plus grandes, on pert la garantie de convergence et par exemple, on voit ici pour $\gamma = 5$ qu'il n'y a plus convergence mais simplement une oscillation autour de l'optimum. Cependant, nous avons juste une analyse de pire cas et nous pourrions donc avoir tout de même convergence pour un pas de descente supérieur à 1.56.

6 CONCLUSION

Durant ce projet, il était question de l'étude de 3 algorithmes d'optimisation encore aujourd'hui massivement utilisés, notamment dans l'apprentissage automatique. Nous nous sommes plus particulièrement penchés sur une approche constructive pour l'analyse de pire cas de ces algorithmes. Cette approche montre comment les outils numériques permettent de confirmer la qualité des bornes de convergence théorique. En particulier pour l'algorithme SAGA, nous avons pu observer que la borne théorique de convergence déterminée dans [3] est bien atteinte, on ne peut pas espérer trouver une meilleure borne de convergence.

RÉFÉRENCES

[1] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1) :42–60, 2018.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM : A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2 :27 :1–27 :27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA : A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in neural information processing systems*, 27, 2014.

[4] Steven Diamond and Stephen Boyd. CVXPY : A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83) :1–5, 2016.

[5] Yoel Drori and Adrien Taylor. Efficient first-order methods for convex minimization : a constructive approach. *Mathematical Programming*, 184(1) :183–220, 2020.

[6] Yoel Drori and Marc Teboulle. Performance of first-order methods for smooth convex minimization : a novel approach. *Mathematical Programming*, 145(1) :451–482, 2014.

[7] Baptiste Goujaud, Céline Moucer, François Glineur, Julien Hendrickx, Adrien Taylor, and Aymeric Dieuleveut. Pepit : computer-assisted worst-case analyses of first-order optimization methods in python. *Mathematical Programming Computation*, 16(3) :337–367, 2024.

[8] Brendan O'Donoghue. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization*, 31 :1999–2023, August 2021.

[9] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3) :1042–1068, June 2016. URL : `http://stanford.edu/~boyd/papers/scs.html`.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830, 2011.

[11] Adrien Taylor. Computer-aided analyses in optimization, 2020. URL : `https://francisbach.com/computer-aided-analyses/`.

[12] Adrien Taylor, Julien Hendrickx, and François Glineur. Smooth strongly convex interpolation and exact worst-case performance of first-order methods. *Mathematical Programming*, 161 :307–345, 2017.

[13] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1) :49–95, 1996.