OCVX1 RAPPORT
TP1

Alexandre Lemonnier - Sarah Gutierez - Victor Simonin

Promotion 2023

Table des matières

1 Partie I

1.1 Définition d'un ensemble de fonctions test

Cette première partie définit un ensemble de fonctions tests très utiles pour avoir une base solide de test pour nos algorithmes et pour avancer sereinement sur le projet. On veut représenter plusieurs situations et couvrir les cas qui nous intéressent.

Ceux-ci sont :

- Fonctions globalement convexes ou uniquement localement convexes
- Fonctions admettant ou non un minimum global
- Fonctions admettant ou non des minimums locaux

Ces différents cas nous permettent de modéliser nos situations de façons complètes. On a donc des fonctions quadratiques de toutes dimensions, des fonctions cubiques de dimension 1 et 2, des fonctions "multi-puits" de dimension 1 et 2 et enfin la banane de Rosenbrock qui est une fonction célèbre pour tester les algorithmes d'optimisation (cette fonction présente un minimum global unique qui se situe au fond d'une vallée très étroite et en forme de parabole).

2 Partie II

2.1 Calcul du gradient d'une fonction de manière approchée

Cette partie nous permet d'avoir des calculs de bases pour les modèles étudiés tout au long de ce projet. La première fonction **partial** nous permet d'avoir les dérivées partielles de la fonction et est ensuite nécessaire pour faire la fonction **gradient**. Les deux sont basés sur des calculs numériques et correspondent donc à des valeurs approchées des dérivées partielles.

3 Partie III

Cette partie a pour but d'implémenter nos premières descentes de gradients qui se font dans la direction du gradient, ici la descente à pas constant et une descente un peu plus optimisée grâce au Backtracking.

3.1 La descente dans la direction du Gradient à pas constant

Cette première descente est le cas générique de descente de gradient avec un pas constant. Cet algorithme consiste à prendre un point de départ au hasard et d'avancer pas à pas vers un point fixe (jusqu'à ce que cela converge) en calculant le gradient et en choisissant une direction grâce à lui.

Nous avons pu réaliser quelques tests afin de voir les performances de l'algorithme.
Tout d'abord, en effectuant de simples tests sur la fonction quadratique pour un point en **2** et **10** dimensions et un paramètre μ fixé à **0,001**, l'algorithme arrive à converger vers le point recherché mais demande autour des 7000 itérations. Cependant, pour la fonction Rosenbrock en **2** dimensions, l'algorithme n'arrive pas à converger en moins de 10 000 itérations, ce que nous considérons comme un inefficacité de l'algorithme.
Plus généralement, en faisant varier le paramètre μ, on observe que plus sa valeur est faible, meilleurs sont les résultats, mais qu'à partir d'un certain seuil il faut plus d'itérations pour une même erreur.

3.2 Optimisation du pas par Backtracking

L'optimisation du pas par backtracking consiste à adapter le pas au fur et à mesure de la descente en utilisant le critère d'Armijo. Le but est d'avoir un pas optimal à chaque étape. Pour cela, on choisit un α et un β tels que $0 < \alpha < 0.5$ et $0 < \beta < 1$. Et ensuite on cherche un μ qui vérifie :

$$J(\mathbf{p}_k + \mu \mathbf{d}_k) < J(\mathbf{p}_k) + \alpha\, \mu\, \mathbf{d}_k^T\, \nabla J(\mathbf{p}_k)$$

Cet algorithme dépend désormais des hyperparamètres α et β passés à la fonction de backtracking. En testant cet algorithme sur la fonction quadratique

en **2** et **10** dimensions ainsi que la fonction Rosenbrock en **2** dimensions, et en variant les hyperparamètres tels que $0 < \alpha < 0.5$ et $0 < \beta < 1$, on observe que l'impact est léger au niveau de la distance finale entre le point obtenu et le point exacte peu importe leurs valeurs. Les meilleurs résultats semblent être obtenus quand β vaut **0.5** et α entre **0.1** et **0.2**.

Cette méthode semble améliorer nos performances lorsqu'on observe rapidement le nombre d'itérations effectué pour que la valeur converge avec une fonction quadratique et de Rosenbrock. Mais nous allons comparer les deux méthodes plus en détails afin de voir réellement laquelle est meilleure et à quel point cela améliore les performances en général.

3.3 Comparaison entre les deux

3.3.1 Fonction quadratique : l'influence du conditionnement

Pour comparer les deux algorithmes il nous faut d'abord déterminer si le conditionnement impacte les performances de façon significative et savoir lequel des 2 est meilleur et avec quelle condition. Nous avons décidé de travailler avec 10 dimensions.

Valeur du conditionnement	Descente à pas constant	Descente avec backtracking
5	7186	40
10	7200	76
50	7272	327
100	7306	609
500	7366	2387

En observant ce tableau on peut vite remarquer que plus la condition est petite moins il faut d'itération pour que le résultat converge. De plus la descente avec backtracking est clairement plus efficace que la descente à pas constant en utilisant une fonction quadratique.

3.3.2 Fonction de Rosenbrock : l'influence de γ

Maintenant que nous avons vu l'impact de la condition sur les performances des descentes à pas constant et avec backtracking, nous allons voir si le paramètre γ

à un gros impact sur l'efficacité de ceux ci.

Valeur de γ	Descente à pas constant	Descente avec backtracking
5	10000+	584
10	10000+	1044
50	10000+	3921
100	10000+	7035

Dans notre cas ici, la descente à pas constant n'a pas convergé au bout de 10000 itérations alors que pour la descente avec backtracking on observe une convergence plus rapide que pour la fonction quadratique. On observe donc bien que l'algorithme de descente avec backtracking est plus performant que celui à descente à pas constant.

4 Partie IV

Cette partie nous permet d'explorer d'autres méthodes de descente de gradient, ici en choisissant d'autres directions de descentes. Nous ne nous déplacerons plus nécessairement dans la direction de $-\nabla J(\mathbf{p_k})$ mais dans une autre direction d_k qui vérifiera bien évidemment $\langle \nabla J(\mathbf{p_k}), d_k \rangle < 0$ afin que ce soit bien une direction de descente et non une direction de montée.

4.1 De plus forte pente en norme ℓ_1

La première méthode que nous allons implémenter consiste à choisir la descente de plus forte pente dans le cas de la norme ℓ_1 : la direction de descente d_k suit le vecteur de la base canonique de plus grande dérivée partielle en valeur absolue.

On a donc la direction :

$$\mathbf{d}_k = -\langle \nabla J(\mathbf{p}_k), e_i \rangle \, e_i$$

où i est le plus petit indice tel que : $\left| \dfrac{\partial J}{\partial x_i}(\mathbf{p}_k) \right| = \|\nabla J(\mathbf{p}_k)\|_\infty$

Nous avons donc deux fonctions, la première **dsgd** qui calcule notre direction selon le point actuel et la seconde **desc_l1_opti** qui contient l'implémentation de cette descente.

4.1.1 Comparaison entre la descente l_1 et la descente du gradient avec backtracking

- Fonction quadratique : l'influence du paramètre de condition

Pour comparer les deux algorithmes il nous faut d'abord déterminer si la condition impacte les performances de façon significative et savoir lequel des 2 est meilleur et avec quelle condition. Nous avons décidé de travailler avec 10 dimensions.

Valeur du conditionnement	Descente l_1	Descente avec backtracking
5	54	40
10	62	76
50	77	327
100	77	609
500	88	2387

Ce tableau nous permet de conclure que pour une fonction quadratique la descente avec backtracking est la solution qui est meilleure pour une condition très petite (de 5) par contre pour le reste elle est beaucoup moins performante que la descente l_1.

- Fonction de Rosenbrock : l'influence de γ

Maintenant que nous avons vu l'impact de la condition sur les performances des 2 descentes, nous allons observer si le paramètre γ à un gros impact sur leur efficacité.

Valeur de γ	Descente l_1	Descente avec backtracking
5	315	584
10	563	1044
50	2371	3921
100	4245	7035

Ici on voit clairement que peu importe le γ l'algorithme de descente l_1 est meilleur. On peut aussi conclure que le γ a un impact assez important sur le nombre d'itérations effectué par les 2 algorithmes.
En comparaison, la descente de gradient à pas constant fournis des résultats bien inférieur en termes de performances.
Par ailleurs, la modification du paramètre μ a les mêmes impacts que pour la descente de gradient à pas constant, en fournissant de meilleurs résultats dans le cas où celui-ci est faible mais il ne faut pas dépasser un certain seuil pour ne pas être contre-performant et itérer plus qu'il ne le faut.

4.2 Gradient conjugué

Les méthodes du gradient conjugué sont d'autres méthodes qui nous permettent de modifier la direction de descente en ajoutant à l'opposé du gradient un terme dépendant des directions de descente précédentes. Ce choix de descente est fait pour rendre deux directions de descentes orthogonales pour le produit scalaire qui vient de la Hessienne.

Ce calcul (qui est direct quand la fonctionnelle est quadratique) peut devenir compliqué quand la Hessienne n'est pas directement accessible.

Nous avons donc pu implémenter deux méthodes populaires, celle proposée par Fletcher-Reeves et celle proposée par Polack-Ribière.

4.2.1 Fletcher-Reeves

Pour la méthode de Fletcher-Reeves, la direction d_k est calculée de la manière suivante :

$$d_k = -\nabla J(\mathbf{p}_k) + \frac{\langle \nabla J(\mathbf{p}_k)^T, \nabla J(\mathbf{p}_k) \rangle}{\langle \nabla J(\mathbf{p}_{k-1})^T, \nabla J(\mathbf{p}_{k-1}) \rangle} d_{k-1}$$

4.2.2 Polack-Ribière

Pour la méthode de Polack-Ribière, la direction d_k est calculée de la manière suivante :

$$d_k = -\nabla J(\mathbf{p}_k) + \frac{\langle \nabla J(\mathbf{p}_k)^T, \nabla J(\mathbf{p}_k) - \nabla J(\mathbf{p}_{k-1}) \rangle}{\langle \nabla J(\mathbf{p}_{k-1})^T, \nabla J(\mathbf{p}_{k-1}) \rangle} d_{k-1}$$

4.2.3 Comparaison et tests

Pour comparer ces deux nouveaux algorithmes utilisant le gradient conjugué, nous avons comparé leur performance en dimensions **2** avec les fonctions tests quadratique et Rosenbrock, et nous avons observé leurs résultats vis-à-vis des autres algorithmes de descentes de gradient précédemment implémentés :

Fonction test	Algorithme GD	Itérations	Distance
quadraticn_	desc_grad_const	6960	0.001002
quadraticn_	desc_grad_opti	20	0.000001
quadraticn_	desc_l1_opti	8	0.000001
quadraticn_	desc_FR_opti	11	0.0
quadraticn_	desc_PR_opti	9	0.000001
Rosenbrock	desc_grad_const	10000	0.012517
Rosenbrock	desc_grad_opti	7035	0.000977
Rosenbrock	desc_l1_opti	4245	0.000764
Rosenbrock	desc_FR_opti	32	0.000166
Rosenbrock	desc_PR_opti	18	0.000062

A partir des résultats du tableau ci-dessus, on observe que les descentes de gradient conjugué obtiennent les mêmes résultats que la descente de gradient l_1 pour la fonction test quadratique, à 2 itérations près. Cependant, on observe que ces deux nouveaux algorithmes sont bien plus efficaces en ce qui concerne la fonction Rosenbrock, en passant de 4245 itérations pour la descente l_1 à 32 et 18 iterations pour la méthode Fletcher-Reeves et Polack-Ribière respectivement, avec une valeur d'erreur bien plus faible.

Si on test nos algorithmes avec les fonctions tests *multitrous2_* et *cubic2_*, on observe que nos algorithmes ne convergent pas pour n'importe quels points, et certains des algorithmes peuvent diverger à l'infini. Ainsi, nos algorithmes ont encore des limites en fonction de si la fonction test est convexe ou non.

5 Partie V

On nous propose ici de découvrir et d'implémenter plusieurs stratégies standards d'accélération de descentes de gradients. Nous avons choisi d'implémenter les 3 méthodes, donc la Momentum Optimisation, la Nesterov Optimisation et la Adam Optimisation.

5.1 Momentum Optimisation

L'optimisation du Momentum est une méthode qui permet d'accélérer la descente de gradient dans la direction la plus pertinente et d'amortir les oscillations. Pour ce faire, elle ajoute une fraction γ du vecteur de mise à jour du pas de temps passé au vecteur de mise à jour actuel.

On peut comparer le Momentum au fait de pousser une balle vers le bas d'une colline. La balle accumule de l'élan au fur et à mesure qu'elle roule vers le bas, devenant de plus en plus rapide sur le chemin. La même chose se produit pour nos mises à jour de paramètres : γ augmente pour les dimensions dont les gradients pointent dans les mêmes directions et réduit les mises à jour pour les dimensions dont les gradients changent de direction. En conséquence, nous obtenons une convergence plus rapide et une oscillation réduite.

5.2 Nesterov Optimisation

L'élan de Nesterov est une extension du Momentum qui consiste à calculer la moyenne mobile décroissante des gradients des positions projetées dans l'espace de recherche plutôt que les positions réelles elles-mêmes.

Cela a pour effet d'exploiter les avantages d'accélération de l'élan tout en permettant à la recherche de ralentir à l'approche de l'optimum et de réduire la probabilité de le manquer ou de le dépasser.

5.3 Adam Optimisation

Adam est une autre méthode qui calcule un taux d'apprentissage adaptatif pour chaque paramètre. Alors que la quantité de mouvement peut être vue comme une

balle descendant une pente, Adam se comporte comme une balle lourde avec frottement, qui préfère donc des minima plats.

5.4 Comparaison et tests

Nous avons implémenté les trois méthodes d'accélération et avons testé leur différents hyperparamètres.
Tout d'abord, pour l'optimisation *Momentum*, on peut faire varier γ et μ. On observe que pour la fonction test quadratique, la valeur de γ n'impacte que très légèrement l'erreur obtenue tandis que μ donne des résultats uniquement si celui-ci est inférieur à 1, et étonnamment de meilleurs résultats si celui-ci est grand, soit supérieur à 0.5. Pour Rosenbrock, la valeur de γ n'impacte pas mais il faut cependant que μ soit le plus petit possible pour que la convergence fonctionne.
Pour l'optimisation *Nesterov*, étant une extension du *Momentum*, les résultats obtenus en faisant varier les hyperparamètres γ et μ sont les mêmes que pour l'optimisation *Momentum*.
Enfin, pour l'optimisation *Adam*, nous pouvions faire varier trois hyperparamètres : μ, β_1, β_2. Nous avons observé qu'à partir du moment où μ était inférieur à 0.5, la distance d'erreur entre le point obtenu et le point exacte était identique peu importe la valeur de β_1 et β_2. Sinon, les meilleurs résultats étaient obtenus lorsque la valeur de β_1 et β_2 était fixée à 0.5.

Si l'on compare les accélérations avec les autres algorithmes de descentes de gradient, on observe que ces derniers nécessitent autant d'itérations peu importe le conditionnement donné à la fonction quadratique pour converger, et on besoin de plus d'itérations afin d'obtenir une erreur plus faible. Cependant, l'erreur obtenue pour la fonction test Rosenbrock est plus faible pour les fonctions d'accélérations vis à vis des autres algorithmes, malgré le fait qu'il est nécessaire de réaliser plus d'itérations.

6 Partie VI

6.1 La Méthode de Newton et la méthode de quasi-Newton

Dans cette partie nous avons pu implémenter la méthode de Newton et la méthode de quasi-Newton. La méthode de Newton consiste à minimiser le développement Limité d'ordre 2 de J au voisinage de $\mathbf{p}_k$: $h \mapsto J(\mathbf{p}_k) + \nabla J(\mathbf{p}_k)^T h + \frac{1}{2} h^T H_J(\mathbf{p}_k) h$ où $H_J(\mathbf{p}_k)$ est la Hessienne de J au point $\mathbf{p}_k$. On passe de la direction $\mathbf{d}_k$ à la direction $\mathbf{d}_{k+1}$ par la relation de récurrence :

$$\mathbf{d}_{k+1} = -H_J(\mathbf{p}_k)^{-1} \nabla J(\mathbf{p}_k).$$

La méthode de quasi Newton permet d'éviter les problèmes de la méthode de Newton, lorsque la Hessienne (H) n'est plus constante. L'implémentations se rapproche des descentes précédentes dans sa simplicité tout en ayant les avantages de la méthode de Newton.

6.2 Comparaison et tests

En réalisant des tests simples, on a pu observer que la méthode de Newton et de quasi-Newton permettaient de converger en très peu d'itération, tout en obtenant des résultats très correct pour la fonction test quadratique, mais moins précis pour la fonction test Rosenbrock.

En comparant tous les algorithmes implémenter jusqu'ici, on observe que ces deux dernières méthodes semblent effectivement très efficaces pour la fonction test quadratique et Rosenbrock pour une convergence en très peu d'itérations. Cependant, on peut quand même relever que elles obtiennent des valeurs de distances plus élevés pour la fonction test Rosenbrock, vis à vis des autres algorithmes.

Fonction test	Algorithme GD	Itérations	Distance
quadraticn_	desc_grad_const	6960	0.001002
quadraticn_	desc_grad_opti	20	0.000001
quadraticn_	desc_l1_opti	8	0.000001
quadraticn_	desc_FR_opti	11	0.0
quadraticn_	desc_PR_opti	9	0.000001
quadraticn_	momentum	433	0.000015
quadraticn_	nesterov	428	0.000021
quadraticn_	adam	336	0.000004
quadraticn_	newton	2	0.0
quadraticn_	desc_BFGS_opti	4	0.000108
Rosenbrock	desc_grad_const	10000	0.012517
Rosenbrock	desc_grad_opti	7035	0.000977
Rosenbrock	desc_l1_opti	4245	0.000764
Rosenbrock	desc_FR_opti	32	0.000166
Rosenbrock	desc_PR_opti	18	0.000062
Rosenbrock	momentum	838	0.000101
Rosenbrock	nesterov	857	0.000102
Rosenbrock	adam	624	0.000035
Rosenbrock	newton	2	1.000002
Rosenbrock	desc_BFGS_opti	3	0.913829

On peut retrouver plus de détails dans le Notebook lié à notre projet, notamment des graphiques, les détails de tous les tests ainsi que les algorithmes complets implémentés et détaillés dans ce rapport.