

Sorbonne Université

Master 1 Automatique, Robotique

Parcours Systèmes Avancés et Robotiques

Rapport de projet

---

# Optimisation en lien avec le Machine Learning

---

**Auteur :** Amine EL KHARRAS & Hao YUAN

**Encadrante :** Florence OSSART

*Année universitaire: 2022-2023*

*7 juillet 2023*

## Table des matières

<b>I. Introduction.....</b>	<b>4</b>
<b>II. les principes et les outils.....</b>	<b>6</b>
1. les principes des Stochastic Gradient Descent et Batch Gradient Descent.....	6
a. Batch Gradient Descent (BGD).....	6
b. Stochastic Gradient Descent (SGD).....	7
c. Comparaison.....	7
2. méthode pour calculer son gradient (méthode analytique et numérique ).....	8
a. C'est quoi le Pytorch?.....	9
b. Comment PyTorch réalise cette fonctionnalité de suivi des gradients?.....	9
<b>III. FITTER UN NUAGE DE POINTS PAR UNE HQ.....</b>	<b>10</b>
1. Définition de l'hyperquadrique et la fonction du coût à minimiser.....	11
a. Hyperquadrique.....	11
b. la fonction du coût à minimiser.....	11
2. Problème simplifié.....	13
a. Résultats obtenus par SGD et BGD (24 données dans le nuage) .....	13
b. Résultats obtenus par SGD et BGD (3000 données dans le nuage) .....	14
c. Amélioration de la méthode du gradient avec le taux d'apprentissage décroissant ou avec la méthode du moment (momentum).....	15
d. Importance du taux d'apprentissage et du momentum.....	16
e. L'impact de différents points de départ sur la convergence.....	17
<b>IV. classification des données relatives au cancer du sein.....</b>	<b>18</b>
1. Étapes pour programmer une régression logistique.....	18
2. Définition du modèle et la fonction du coût à minimiser.....	19
a. Le modèle (Fonction logistique).....	19
b. Fonction de coût et son gradient.....	20
b.1 BGD.....	20
b.2 SGD.....	21
3. résultat de la méthode BGD.....	22
4. résultat de la méthode SGD.....	22
<b>V. Perspective.....</b>	<b>23</b>
<b>VI. Conclusion et Commentaire.....</b>	<b>24</b>

## **Remerciements**

Nous tenons à exprimer nos sincères remerciements à Madame Florence OSSART pour sa contribution inestimable à notre projet.

Malgré son emploi du temps chargé, Madame Florence OSSART a pris le temps chaque semaine de nous rencontrer et d'orienter notre travail sur ce projet, qui a duré un mois. Son engagement et sa direction ont été essentiels pour notre progression et nos acquis.

Nous avons beaucoup appris grâce à cette expérience. Nous vous remercions, Madame Florence OSSART, pour cette opportunité.

## I. Introduction

Dans le monde de l'apprentissage automatique et de l'intelligence artificielle, l'optimisation et l'ajustement des modèles jouent un rôle crucial dans la résolution des problèmes et la prise de décisions informées. Dans ce rapport, nous abordons certaines des techniques les plus couramment utilisées pour optimiser les modèles d'apprentissage automatique. Nous nous concentrerons sur la descente de gradient en lots (Batch Gradient Descent, BGD) et la descente de gradient stochastique (Stochastic Gradient Descent, SGD), deux variantes populaires de la méthode de descente de gradient.

En abordant d'abord le fonctionnement de ces méthodes et leurs principes sous-jacents, nous établirons une compréhension solide des mécanismes qui régissent ces techniques d'optimisation. Nous expliquerons comment chaque méthode traite les données pour trouver le minimum global d'une fonction de coût. Le BGD traite toutes les données à la fois, tandis que le SGD le fait de manière incrémentielle, en utilisant un seul exemple à la fois. La compréhension de ces différences est essentielle pour apprécier le rôle que jouent ces méthodes dans le domaine de l'apprentissage automatique.

Au-delà de ces principes, nous explorerons également comment les gradients sont calculés pour ces méthodes d'optimisation. Il existe deux approches principales pour cela: analytique et numérique. L'approche analytique implique l'utilisation de dérivées, tandis que l'approche numérique implique l'utilisation d'approximations. Nous discuterons de chacune de ces approches et de leurs avantages et inconvénients respectifs.

Dans ce contexte, nous introduirons PyTorch, une bibliothèque d'apprentissage profond qui offre une fonctionnalité de suivi des gradients, permettant le calcul automatique des gradients. Nous examinerons en détail comment PyTorch réalise cette fonctionnalité, et comment elle peut être utilisée pour simplifier le processus d'optimisation dans l'apprentissage automatique.

Ensuite, nous mettrons en pratique ces concepts en utilisant un problème concret: ajuster un nuage de points à une hyperquadrique. Cela nous donnera l'occasion de voir comment les différentes méthodes d'optimisation peuvent être appliquées à des problèmes du monde réel. Nous comparerons les résultats obtenus par les méthodes BGD et SGD pour ce problème, en prenant en compte divers facteurs tels que la taille du nuage de points et le point de départ de l'optimisation.

Enfin, nous appliquerons ces concepts à un autre problème pratique très important: la classification des données relatives au cancer du sein. Nous utiliserons la régression logistique, un outil puissant pour la classification binaire. En définissant le modèle, la fonction de coût et son gradient, nous serons en mesure d'appliquer les techniques d'optimisation que nous avons discutées pour résoudre ce problème de classification.

Ainsi, ce rapport vise à fournir une compréhension approfondie des méthodes d'optimisation en apprentissage automatique, de leur fonctionnement à leur application à des problèmes du monde réel.

## **II. les principes et les outils**

### **1. les principes des Stochastic Gradient Descent et Batch Gradient Descent**

Les principes des SGD (Stochastic Gradient Descent) et BGD (Batch Gradient Descent) sont deux algorithmes d'optimisation couramment utilisés dans le domaine de l'apprentissage automatique (machine learning) pour minimiser les fonctions de coût lors de l'entraînement des modèles.

#### **a. Batch Gradient Descent (BGD)**

Le BGD est une méthode d'optimisation qui utilise l'ensemble complet des données d'entraînement à chaque itération pour mettre à jour les paramètres du modèle. Voici les principes clés du BGD :

**Calcul du gradient :** À chaque itération, le gradient de la fonction de coût par rapport à chaque paramètre du modèle est calculé en utilisant l'ensemble complet des données d'entraînement. Cela implique de parcourir tous les exemples d'entraînement, ce qui peut être coûteux en termes de temps de calcul pour les grands ensembles de données.

**Mise à jour des paramètres :** Une fois que le gradient est calculé, les paramètres du modèle sont mis à jour dans la direction opposée au gradient, en utilisant un taux

d'apprentissage fixe ou adaptatif. La mise à jour est effectuée simultanément pour tous les paramètres du modèle.

Itérations jusqu'à la convergence : Le processus de calcul du gradient et de mise à jour des paramètres est répété jusqu'à ce qu'un critère de convergence soit atteint, généralement lorsque la fonction de coût atteint un minimum local ou que la variation de la fonction de coût entre les itérations successives est suffisamment faible.

#### b. Stochastic Gradient Descent (SGD)

Le SGD est une variante du BGD qui utilise un seul exemple d'entraînement (ou un petit échantillon appelé mini-batch) à chaque itération plutôt que l'ensemble complet des données. Voici les principes clés du SGD :

Calcul du gradient stochastique : À chaque itération, un seul exemple (ou mini-batch) est utilisé pour calculer le gradient de la fonction de coût par rapport aux paramètres du modèle. Cela introduit un certain niveau de bruit dans le processus de mise à jour des paramètres.

Mise à jour des paramètres : Après avoir calculé le gradient stochastique, les paramètres du modèle sont mis à jour de la même manière qu'avec le BGD, en utilisant un taux d'apprentissage fixe ou adaptatif.

Itérations jusqu'à la convergence : Le processus de calcul du gradient et de mise à jour des paramètres est répété pour chaque exemple d'entraînement (ou mini-batch) jusqu'à ce qu'un critère de convergence soit atteint.

#### c. Comparaison

La principale différence entre le SGD et le BGD réside dans le fait que le SGD utilise un seul exemple (ou mini-batch) à chaque itération, tandis que le BGD utilise l'ensemble complet des données. En conséquence, le SGD est souvent plus rapide pour les grands ensembles de données, car il effectue moins de calculs par itération. Cependant, le SGD peut également être plus bruité et moins stable que le BGD en raison de l'approximation introduite par l'utilisation d'un seul exemple à la fois.

En résumé, SGD et BGD sont deux algorithmes d'optimisation courants utilisés pour minimiser la fonction de coût dans l'apprentissage automatique. SGD effectue à chaque itération le calcul du gradient et la mise à jour des paramètres en utilisant un échantillon individuel ou un mini-lot, ce qui le rend plus rapide mais moins stable. D'autre part, BGD effectue le calcul du gradient et la mise à jour des paramètres en utilisant l'ensemble complet des données, ce qui le rend plus stable mais plus coûteux en termes de calcul. Le choix de l'algorithme dépend de la taille spécifique du problème et de l'ensemble de données. En général, SGD convient aux grands ensembles de données, tandis que BGD convient aux ensembles de données plus petits ou lorsque la stabilité des paramètres est plus importante.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

FIGURE1 - BGD

```
for  $i := 1, \dots, m\{$ 
     $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ 
    (for every  $j = 0, \dots, n\)$ 
```

FIGURE2 - SGD

## 2. méthode pour calculer son gradient (méthode analytique et numérique )

Pour le domaine de l'optimisation convexe, le calcul du gradient d'une fonction de perte est nécessaire, mais toutes les fonctions ne disposent pas d'un gradient déterminé pour le calcul. Autrement dit, l'utilisation de méthodes traditionnelles est limitée dans ce cas. Dans de telles situations, nous pouvons utiliser des outils numériques pour calculer le gradient de la fonction de perte, parmi lesquels les plus couramment utilisés sont Keras, TensorFlow et PyTorch.

Dans le cadre de ce projet, nous avons essayé d'utiliser PyTorch pour calculer les gradients de la fonction. Bien sûr, l'utilisation de PyTorch n'est pas du tout nécessaire pour ce projet, mais c'était néanmoins une expérience intéressante. Par conséquent, je pense qu'il est nécessaire de commencer par une brève introduction sur le fonctionnement de PyTorch pour le suivi des gradients.

a. C'est quoi le Pytorch?

PyTorch est une bibliothèque open source utilisée pour construire des modèles d'apprentissage en profondeur. Elle a été développée par le Facebook AI Research (FAIR) et a été publiée en 2016. Aujourd'hui, elle est devenue l'une des bibliothèques les plus populaires dans le domaine de l'apprentissage en profondeur.

PyTorch est capable de suivre automatiquement les gradients grâce à une fonctionnalité appelée "Autograd". Autograd est une partie intégrée de PyTorch qui permet d'effectuer le suivi des opérations effectuées sur les tenseurs et de calculer les gradients de manière automatique.

b. Comment PyTorch réalise cette fonctionnalité de suivi des gradients?

- Lorsque vous créez un tenseur et effectuez des opérations mathématiques dessus, PyTorch enregistre ces opérations dans un graphe computationnel.
- Le graphe computationnel est une représentation des opérations appliquées sur les tenseurs, et il enregistre également les dépendances entre ces opérations.
- Lorsque vous effectuez un calcul qui nécessite le gradient, par exemple lors de l'optimisation d'une fonction de perte, PyTorch utilise le graphe computationnel pour calculer le gradient de la fonction par rapport aux paramètres.
- PyTorch utilise la méthode de rétropropagation (backpropagation) pour calculer les gradients à partir du graphe computationnel. La rétropropagation parcourt le graphe de manière récursive, en utilisant la règle de dérivation de la chaîne pour calculer les gradients des opérations intermédiaires.
- Une fois que les gradients ont été calculés, vous pouvez les utiliser pour mettre à jour les paramètres du modèle en utilisant un algorithme d'optimisation tel que la descente de gradient stochastique (SGD).

En résumé, PyTorch utilise le suivi automatique des gradients grâce à la fonctionnalité Autograd. Il enregistre les opérations effectuées sur les tenseurs dans un graphe computationnel, puis utilise

la rétropropagation pour calculer les gradients par rapport aux paramètres. Cela facilite grandement le processus d'optimisation des modèles de deep learning avec PyTorch.

D'après un exemple simple utilisant PyTorch pour calculer les gradients (*ANNEXE I*). Nous pouvons voir qu'il n'est pas nécessaire de dériver les équations mathématiques pour calculer les gradients spécifiques. Nous pouvons plutôt obtenir les gradients en un certain point en utilisant la méthode de rétropropagation (backward).

Maintenant, nous avons compris les différences fondamentales entre la descente de gradient stochastique et la descente de gradient par lots, ainsi que quelques méthodes de calcul du gradient. Maintenant, nous allons présenter le problème que nous devons traiter.

Tout d'abord, nous allons utiliser SGD (Stochastic Gradient Descent) et BGD (Batch Gradient Descent) pour résoudre un problème simple. Grâce à la résolution de ce problème, nous pouvons comprendre pleinement les méthodes de fonctionnement des deux approches. Ensuite, nous utiliserons SGD pour résoudre un problème de classification binaire dans l'apprentissage automatique.

### III. FITTER UN NUAGE DE POINTS PAR UNE HQ

L'objectif de ce projet est de modéliser un nuage de points à l'aide d'une hyperquadrique (HQ).

Pour commencer, nous collectons un ensemble de points dans un espace donné. Ces points peuvent être répartis dans plusieurs dimensions, et notre tâche consiste à trouver une hyperquadrique qui les représente au mieux.

Pour ce faire, nous utilisons des techniques d'optimisation pour ajuster les paramètres de l'hyperquadrique afin qu'elle corresponde étroitement aux points donnés. Les méthodes d'optimisation couramment utilisées incluent la descente de gradient stochastique (SGD) et la descente de gradient par lots (BGD).

Une fois que nous avons ajusté l'hyperquadrique aux points, nous pouvons l'utiliser pour prédire de nouveaux points ou pour obtenir une meilleure compréhension des relations entre les données.

Cela peut être particulièrement utile dans des domaines tels que la reconnaissance de formes, l'analyse de données et l'apprentissage automatique.

En résumé, ce projet consiste à utiliser une hyperquadrique pour modéliser un nuage de points en ajustant ses paramètres à l'aide de méthodes d'optimisation telles que SGD et BGD. Cela nous permet de représenter les données de manière plus précise et d'obtenir des informations significatives à partir de celles-ci.

## 1. Définition de l'hyperquadrique et la fonction du coût à minimiser

### a. Hyperquadrique

Une hyperquadrique (HQ) est un contour défini dans le plan  $x, y$  par l'équation implicite:

$$\psi(x, y, \lambda) = 0$$

$$\text{et : } \varphi(x, y, \lambda) = \sum_{k=1}^{Nh} |A_k \cdot x + B_k \cdot y + C_k|^{\gamma_k}$$

$$\text{avec : } \psi(x, y, \lambda) = \varphi(x, y, \lambda) - 1$$

Nh est le nombre de termes de la l'hyperquadrique HQ, supérieur ou égale à 2, et lambda={Ak, Bk, Ck, Gammak, 1<=k<=Nh}est le vecteur des paramètres de HQ avec Gammak> 0.

### b. la fonction du coût à minimiser

Les données à fitter sont un ensemble de points  $x, y$  . Le but du problème est de trouver le jeu de coefficients lambda([A, B, C, gamma]), tels que l'hyperquadrique passe au mieux par le nuage de points. Pour cela, il faut définir un critère de distance entre un point  $x, y$  et l'hyperquadrique HQ, dont on déduit un critère de distance entre l'ensemble des points et HQ.

On propose de travailler avec le critère quadratique ci-dessous.

$$J([A, B, C, Gamma]) = [\psi(xi, yi, [A, B, C, Gamma])]^2$$

La sgd et la bgd utilisent la même fonction de coût, la différence réside dans le fait que lors de la mise à jour des paramètres, la sgd utilise les valeurs de gradient d'un seul point de données, tandis que la bgd additionne les valeurs de gradient de chaque point de données dans le nuage de points ( $x, y$ ) et en fait la moyenne. Si nous devons ajuster un nuage de points contenant vingt-quatre données, la sgd n'a besoin de calculer le gradient qu'une fois (celui du point choisi au hasard dans le nuage de points), tandis que la bgd doit calculer le gradient vingt-quatre fois (en calculant le gradient de chaque point dans le nuage de points, puis en prenant la moyenne). En d'autres termes, la sgd nous offre une méthode d'itération plus rapide mais moins stable, tandis que la bgd est plus stable mais perd en vitesse d'itération.

En d'autres termes, pour le SGD (descente de gradient stochastique), la valeur du gradient utilisée pour mettre à jour les paramètres à chaque étape est calculée par l'équation suivante :

$$gran(J([A, B, C, Gamma])) = grad([\psi(xi, yi, [A, B, C, Gamma])]^2)$$

(Nous choisissons aléatoirement un  $xi$  et un  $yi$ , que nous considérons comme des constantes, puis nous calculons le gradient de la fonction de coût par rapport à chaque paramètre à ajuster ( $dA, dB, dC, dGamma$ ).)

Mais pour la méthode de la descente de gradient par lots (BGD), nous devons effectuer les calculs mentionnés précédemment pour chaque point, puis prendre la moyenne de ces gradients comme le gradient de mise à jour des paramètres. L'équation est la suivante :

$$gran(J([A, B, C, Gamma])) = \frac{\sum_{i=1}^N grad([\psi(xi, yi, [A, B, C, Gamma])]^2)}{N}$$

N est le nombre de données dans le nuage .

## 2. Problème simplifié

Le problème considéré est un cas particulier du problème général, avec  $N = 2$ . De plus, 6 des 8 paramètres sont connus. L'HQ est définie par l'équation ci-dessous, dans laquelle il n'y a que deux paramètres :  $a$  et  $b$ .

$$\psi(x, y, a, b) = 0$$

avec  $\psi(x, y, a, b) = (ax + by)^4 + (x + y)^4 - 1$

- a. Résultats obtenus par SGD et BGD (24 données dans le nuage)

Tout d'abord, nous avons testé l'ajustement d'un nuage de points composé de 24 points. ([ANNEXE2](#))

Tout d'abord, nous avons utilisé SGD ( $a_0=1, b_0=1, \alpha=0.005$ ) pour ajuster le nuage de points. L'ANNEXE 3 montre l'évolution des résultats d'ajustement au fil des itérations. Nous constatons qu'à mesure que les itérations progressent, la forme de l'ajustement se rapproche de plus en plus de la forme du nuage de points, ce qui indique que la méthode SGD fonctionne correctement.

Ensuite, nous ajustons à nouveau le nuage de points donné en utilisant la descente de gradient en batch (BGD)( $a_0=1, b_0=1, \alpha=0.005$ ). Les résultats obtenus ainsi que certaines figures d'analyse sont présentés en [ANNEXE 4-8](#).

L'ANNEXE 4 décrit le chemin parcouru par les paramètres ( $a_0, b_0$ ) au fur et à mesure des itérations(dans un graphique d'isocontours fixe(Le code impliquant le tracé des lignes de contour est en [ANNEXE 5](#))). Nous pouvons constater que les deux méthodes atteignent avec succès un minimum local, mais la différence réside dans le fait que le chemin parcouru par BGD est plus régulier. Cela est dû au fait que le chemin recherché par SGD est aléatoire (le gradient calculé à chaque itération est basé sur un point choisi au hasard).

L'ANNEXE 6 présente le temps nécessaire pour les deux méthodes. On peut constater que le temps requis pour BGD est nettement supérieur à celui de SGD, ce qui est tout à fait logique, car chaque itération de BGD nécessite beaucoup plus de calculs que SGD. Cela devient encore plus évident lorsque l'on ajuste un ensemble de points plus important. Nous discuterons ensuite du cas d'un ensemble de données de taille 3000.

L'ANNEXE 8 décrit les variations des paramètres ( $a_0, b_0$ ) et de l'erreur globale lors du processus itératif. Nous pouvons constater que l'erreur globale diminue progressivement au fur et à mesure des itérations, et que les deux paramètres convergent progressivement. Nous pouvons également observer qu'en termes de nombre d'itérations, le sgd nécessite plus d'itérations pour atteindre la

convergence, tandis que le bgd en nécessite moins. Cependant, à chaque itération, le temps nécessaire au bgd est bien plus long que celui du sgd.

De plus, l'erreur globale est calculée à l'aide de la formule suivante :

$$error_{global} = \frac{\sum_{j=1}^{len(x)} \left( \sum_{i=1}^{Nh} (A(i) \times x(j) + B(i) \times y(j) + C(i))^{\gamma(i)} - 1 \right)}{len(x)}$$

Une fois que des nouvelles valeurs A, B, C et gamma sont obtenues, toutes les données des points à ajuster sont successivement introduites dans la fonction de coût actuelle. Les résultats obtenus sont ensuite additionnés et la moyenne est calculée. Cela décrit la qualité globale de l'ajustement actuel entre la fonction et les données.

b. Résultats obtenus par SGD et BGD (3000 données dans le nuage)

Nous avons ensuite testé un ensemble de données plus important (3000 points) et les résultats obtenus sont présentés dans **les ANNEXES 9-12.**

Nous pouvons constater qu'en augmentant la densité de l'ensemble de données, c'est-à-dire en augmentant le nombre de points dans le nuage de points, le temps nécessaire pour le BGD augmente considérablement. Dans le cas précédent avec 24 points, cela prenait 3 secondes, mais lorsque le nombre de points à ajuster atteint trois mille, le temps requis pour le BGD atteint 200 secondes. En revanche, le SGD prend beaucoup moins de temps dans ces deux cas, et une raison importante est que le taux d'apprentissage du BGD ne peut pas s'ajuster de manière adaptative. Nous pouvons concevoir une fonction pour ajuster automatiquement le taux d'apprentissage en fonction du nombre d'itérations, mais cela présente un inconvénient.

c. Amélioration de la méthode du gradient avec le taux d'apprentissage décroissant ou avec la méthode du moment (momentum)

En général, nous pouvons introduire l'équation suivante pour réaliser un taux d'apprentissage adaptatif:

$$\alpha = A \times e^{-\gamma \times n_{iteration}}$$

Le taux d'apprentissage décroissant (Learning Rate Decay) :

Le taux d'apprentissage décroissant est une stratégie qui consiste à réduire progressivement la valeur du taux d'apprentissage lors du processus d'optimisation afin de contrôler la taille des mises à jour des paramètres. Le taux d'apprentissage initial est généralement fixé à une valeur élevée pour se rapprocher rapidement de la solution optimale. Au fur et à mesure des itérations, le taux d'apprentissage diminue progressivement, ce qui permet de raffiner les mises à jour des paramètres et de les rendre plus stables. La décroissance du taux d'apprentissage peut aider l'algorithme d'optimisation à explorer plus efficacement l'espace de recherche et à être plus précis lorsqu'il se rapproche de la solution optimale. Cependant, des stratégies de décroissance du taux d'apprentissage trop grandes ou trop petites peuvent entraîner une diminution des performances, il est donc important de choisir une stratégie de décroissance du taux d'apprentissage appropriée.

Nous pouvons également utiliser la méthode du moment (Momentum) pour accélérer l'itération.

la méthode du moment (Momentum)

La méthode du moment est une technique qui met à jour les paramètres en accumulant les informations de gradient précédentes. À chaque itération, en plus d'utiliser le gradient actuel pour mettre à jour les paramètres, on introduit également un terme de moment qui représente l'accumulation des directions et des magnitudes de gradient des itérations précédentes. La méthode du moment permet d'accélérer la descente de gradient dans l'espace des paramètres, en particulier en présence de zones plates ou de changements de direction de gradient lents. Elle améliore la stabilité de l'algorithme d'optimisation et réduit les oscillations lors du processus d'optimisation. Cependant, la méthode du moment peut entraîner un surajustement ou passer à côté des optimaux locaux. De plus, un paramètre de moment trop élevé peut rendre l'algorithme instable, il est donc important de choisir une valeur appropriée pour le paramètre de moment.

Pour notre projet, nous avons essayé d'utiliser la méthode du momentum pour réaliser des itérations plus rapides.

Dans L'ANNEXE 13, on donne un exemple simple de la méthode de momentum. C'est très simple. Nous introduisons un terme de momentum pour mettre à jour les paramètres, ce qui signifie que pour chaque variable, nous mettons à jour sa valeur  $v$  ( $v = \text{momentum} * v - \alpha * \text{grad}$ ). Ensuite, nous mettons à jour les paramètres en utilisant  $\text{var} = \text{var} + v_{\text{var}}$ . Si nous fixons le momentum à zéro, il ne tiendra pas compte des gradients précédents. Plus la valeur du momentum est grande, plus les gradients précédents auront un impact important sur la prochaine mise à jour des paramètres.

Nous avons placé les résultats obtenus en utilisant cette méthode en ANNEXE 14-15. Nous pouvons constater qu'après l'introduction de la méthode du moment pour aider à la convergence, avec le même ensemble de points de 3000 points à ajuster, la méthode BGD (descente de gradient en lot) ne prend que 20 secondes pour s'adapter parfaitement et le nombre d'itérations nécessaires est considérablement réduit à seulement 100. Cependant, les résultats précédents montrent que, sans utiliser la méthode du moment, il faut 200 secondes et 1000 itérations. Pour la méthode SGD (descente de gradient stochastique), nous constatons que l'introduction de la méthode du moment n'a pas vraiment aidé à accélérer les itérations. Cela est dû au fait que la vitesse d'itération est déjà suffisamment rapide sans utiliser la méthode du moment, ce qui rend l'introduction de la méthode du moment peu significative et peut même diminuer la stabilité de la convergence.

#### d. Importance du taux d'apprentissage et du momentum

Ensuite, nous essayons d'analyser l'impact de différents taux d'apprentissage sur la convergence. Nous fixons la valeur du momentum à 0,9 et utilisons deux taux d'apprentissage différents (0,05 et 0,0005) pour observer s'il y a une différence par rapport à la convergence précédente. De plus, nous utilisons ici la méthode du BGD (Batch Gradient Descent), car comme mentionné précédemment, l'introduction du momentum n'a pas beaucoup de sens pour le SGD (Stochastic Gradient Descent). Les résultats des tests sont présentés dans L'ANNEXE 16-17. Les résultats montrent qu'après avoir augmenté le taux d'apprentissage de 0,005 à 0,05, la vitesse de convergence de la fonction devient plus rapide, mais aussi plus instable, et même présente une certaine divergence. Lorsque nous réduisons le taux d'apprentissage à 0,0005, nous constatons que la vitesse de convergence de la fonction diminue considérablement, nécessitant plus de temps pour converger, mais elle devient plus stable.

e. L'impact de différents points de départ sur la convergence.

Ensuite, nous avons également analysé l'impact du choix de points de départ différents sur la convergence de la fonction. Nous avons utilisé les mêmes taux d'apprentissage et de momentum que précédemment ( $\alpha=0,005$ ,  $\text{momentum}=0,9$ ), mais nous avons uniquement modifié les valeurs de départ  $a_0$  et  $b_0$  (auparavant  $(1,1)$ , maintenant  $(-1,-1)$ ). Les résultats précis sont présentés dans ***L'ANNEXE 18***. Nous avons constaté que la convergence de la fonction cette fois-ci était différente de celle précédente. Elle converge vers un autre minimum local  $(-0,77, 0,35)$ . En ce qui concerne les autres valeurs, telles que le temps nécessaire pour l'itération, il n'y a pas de changement significatif. En fait, nous pouvons observer sur le graphe des courbes de niveau que ces deux résultats de convergence différents sont symétriques par rapport à une ligne. Cela indique que ces deux résultats de convergence différents sont équivalents. Cependant, cela nous montre également que le choix de points de départ différents a une importance significative sur la convergence de la fonction.

## **IV. classification des données relatives au cancer du sein**

Dans cette partie de notre projet, nous allons appliquer la méthode de BGD et SGD pour résoudre un problème de classification des données relatives au cancer du sein en utilisant la régression logistique.

La régression logistique est une technique d'apprentissage supervisée utilisée pour résoudre des problèmes de classification binaire. Dans notre cas, nous cherchons à prédire si un patient est atteint d'un cancer du sein positif ou négatif en fonction de certaines caractéristiques et attributs.

### **1. Étapes pour programmer une régression logistique**

Voici les étapes principales de la méthode utilisant la régression logistique et la méthode de gradient stochastique :

- Collecte de données : nous rassemblons un ensemble de données contenant des caractéristiques pertinentes pour chaque patient, telles que l'âge, la taille et le type de tumeur, etc. Chaque patient est étiqueté comme positif (cancer du sein) ou négatif (pas de cancer du sein).
- Préparation des données : nous préparons les données en les divisant en deux ensembles : un ensemble d'entraînement pour ajuster notre modèle et un ensemble de tests pour évaluer les performances du modèle.
- Régression logistique : nous utilisons la régression logistique comme modèle de classification . La régression logistique applique une fonction sigmoïde pour transformer la sortie en une probabilité comprise entre 0 et 1, ce qui représente la probabilité d'appartenance à la classe positive (cancer du sein).
- Fonction de coût : nous avons défini une fonction de coût, telle que l'entropie croisée, qui mesure l'écart entre les prédictions du modèle et les étiquettes réelles. Cette fonction de coût nous permet de quantifier l'erreur de prédiction de notre modèle.
- Optimisation : nous utilisons la méthode de descente de gradient stochastique (SGD) pour ajuster les paramètres du modèle afin de minimiser la fonction de coût. A chaque itération, le SGD met à jour les paramètres en calculant le gradient de la fonction de coût sur un exemple aléatoire de notre données d'entraînement.
- Répétition : nous répétons l'étape précédente jusqu'à ce que la fonction de coût converge vers un minimum local, indiquant que notre modèle est bien ajusté aux exemples d'entraînement.
- Évaluation : une fois que le modèle est entraîné, nous l'évaluons sur l'ensemble des données du test pour mesurer sa performance en termes de précision. Cela nous permet de déterminer l'efficacité de notre modèle dans la prédiction du cancer du sein.

En utilisant la régression logistique avec la méthode de descente de gradient stochastique , nous serons en mesure de développer un modèle de classification pour les données relatives au cancer du sein. Ce modèle pourrait être un outil précieux pour aider à la détection précoce et précise du cancer du sein, contribuant ainsi à des résultats de traitement et à une amélioration des soins aux patients.

## 2. Définition du modèle et la fonction du coût à minimiser

### a. Le modèle (Fonction logistique)

La régression logistique est un algorithme couramment utilisé pour la classification binaire. Son principe repose sur la fonction logistique, également appelée fonction sigmoïde.

L'objectif de la régression logistique est de prédire la probabilité d'une variable de sortie binaire en fonction des caractéristiques d'entrée fournies. Par exemple, déterminer si un e-mail est un spam ou non, ou prédire la présence d'une maladie en fonction de certaines caractéristiques du patient.

La régression logistique utilise la fonction logistique pour transformer la combinaison linéaire des caractéristiques d'entrée en une valeur de probabilité comprise entre 0 et 1. La formule de la fonction logistique est la suivante :

$$P(y=1|x) = 1 / (1 + e^{-(z)})$$

Dans cette formule,  $P(y=1|x)$  représente la probabilité de la sortie étant 1, étant donné les caractéristiques d'entrée  $x$ .  $e$  est la base du logarithme naturel, et  $z$  est la somme pondérée des caractéristiques d'entrée multipliées par leurs poids respectifs :

$$z = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

Ici,  $w_0, w_1, w_2, \dots, w_n$  sont les coefficients de régression, et  $x_1, x_2, \dots, x_n$  sont les caractéristiques d'entrée.

Pendant le processus d'entraînement, l'algorithme de régression logistique ajuste les valeurs des coefficients de régression en fonction des données d'entraînement, afin de mieux ajuster le modèle aux données d'entraînement et de prédire les données inconnues. Le processus d'entraînement utilise généralement des techniques d'optimisation telles que le maximum de vraisemblance ou la descente de gradient pour minimiser la fonction de perte.

Pendant la phase de prédiction, en donnant de nouvelles caractéristiques en entrée, la régression logistique calcule une valeur de probabilité correspondante et la transforme en une sortie de classification binaire en fonction d'un seuil prédéfini. Dans notre projet, nous utilisons un seuil de 0,5. En d'autres termes, nous utilisons l'ensemble de données de test avec  $P(y=1|x) = 1 / (1 +$

$e^{-z})$ ). Si le résultat du calcul est supérieur à 0,5, nous classons les données dans la catégorie '1'. S'il est inférieur, nous les classons dans la catégorie '0'. Enfin, nous comparons les différences entre la classification prédite et la classification réelle, puis calculons l'exactitude du modèle.

En résumé, la régression logistique utilise la fonction logistique pour transformer la combinaison linéaire des caractéristiques d'entrée en une valeur de probabilité, ajuste les coefficients de régression pour ajuster les données d'entraînement et est utilisée pour prédire des problèmes de classification binaire.

#### b. Fonction de coût et son gradient

##### b.1 BGD

la fonction de coût pour la régression logistique est définie par :

$$J(\theta) = -\frac{1}{N} \sum_{i=1}^N (y_{true}^i \log(y_{pred}^i) + (1 - y_{true}^i) \log(1 - y_{pred}^i))$$

Le gradient de la fonction de coût par rapport aux paramètres du modèle est donnée par :

$$\frac{\delta J(\theta)}{\delta \theta} = \frac{1}{N} \sum_{i=1}^N (y_{pred}^i - y_{true}^i) \cdot X$$

où : N est le nombre d'échantillons

$y_{true}^i$  est l'étiquette réelle de l'échantillon i

$y_{pred}^i = f(X, \theta)$  est la prédiction du modèle pour l'échantillon i

##### b.2 SGD

Pour SGD, nous avons choisi une fonction de coût plus intuitive :

$$J(w) = \text{abs}(y_s - \frac{\exp(w \times x_s)}{1 + \exp(w \times x_s)})$$

La fonction du coût calcule l'écart entre les valeurs prédictives et les valeurs réelles du modèle actuel. En utilisant la méthode de descente de gradient stochastique (SGD), nous réduisons

progressivement cet écart. Ainsi, nous obtenons un modèle capable de prédire avec précision les classifications réelles. À chaque itération, un ensemble ( $xs$ ,  $ys$ ) est sélectionné de manière aléatoire, avec la variable  $w$ .

### 3. résultat de la méthode BGD

Nous avons placé les résultats de la régression logistique de la méthode de la descente de gradient par lots à [\*l'Annexe 19\*](#).

La première figure représente l'évolution de la fonction de coût au fil d'itérations montre que notre fonction de coût diminue de manière significative et à se rapprocher à zéro. Cela suggère que la descente de gradient a progressé et se rapproche d'une solution optimale.

La deuxième figure de la précision de test montre une augmentation de la précision jusqu'à environ 94%. Cela indique que le modèle s'améliore durant l'entraînement et parvient à bien classer les exemples de test, nous pouvons donc dire que le modèle que nous avons obtenu est relativement précis.

La troisième figure représente la variation des paramètres au fil d'itération, on observe que les poids varient dans une plage restreinte. En revanche, le biais, qui influe sur le calcul de notre fonction de coût, présente une diminution qui suit un comportement similaire à celui de la fonction de coût.

### 4. résultat de la méthode SGD

Nous avons placé les résultats de la régression logistique de la méthode de la descente de gradient stochastique à [\*l'Annexe 20\*](#). La première image montre les variations des trente et un poids et biais à mesure que la fonction converge. Nous pouvons constater que la fonction converge avec succès au fil des itérations. À la fin de chaque itération, nous avons testé l'ajustement de classification du modèle actuel sur l'ensemble de test, comme le montre la deuxième image. Nous avons constaté qu'à la fin de la convergence, le modèle obtenu présente un taux de précision de prédiction d'environ 95% sur l'ensemble de test. Nous pouvons donc dire que le modèle que nous avons obtenu est relativement précis.

## V. Perspective

Perspectives pour les futures recherches :

Amélioration des algorithmes d'optimisation : Bien que nous ayons déjà étudié le BGD (batch gradient descent) et le SGD (stochastic gradient descent), il existe encore de nombreux autres algorithmes d'optimisation qui méritent d'être explorés, tels que AdaGrad, RMSprop, Adam, etc. Ces algorithmes cherchent tous à résoudre, à différents degrés, certains des défis de la descente de gradient, tels que l'optimisation dans des espaces de grande dimension ou l'optimisation de problèmes non convexes. Les recherches futures peuvent approfondir la compréhension de ces méthodes et les comparer et les analyser dans des problèmes concrets.

Choix et optimisation des modèles : Dans nos recherches, nous avons choisi les ajustements de courbe hyperquadratiques et la régression logistique comme bases expérimentales. Cependant, pour des problèmes différents, il peut être nécessaire d'utiliser des modèles différents. Comprendre comment choisir et optimiser les modèles est une partie importante de l'apprentissage automatique. Les recherches futures peuvent explorer des modèles plus adaptés à des problèmes spécifiques et étudier comment les optimiser.

Exploration des techniques d'ajustement automatique des paramètres : Nous avons discuté dans ce rapport de l'impact du taux d'apprentissage et de la quantité de mouvement sur le processus d'optimisation. Cependant, ajuster manuellement ces paramètres peut être très chronophage. Par conséquent, étudier l'automatisation de ce processus est également une direction significative, par exemple en utilisant des méthodes telles que la recherche en grille, la recherche aléatoire ou l'optimisation bayésienne pour trouver automatiquement les meilleurs paramètres.

Application des techniques d'apprentissage en profondeur : Bien que ce rapport se concentre principalement sur les méthodes d'apprentissage automatique de base, avec l'amélioration des ressources de calcul et l'émergence des mégadonnées, l'apprentissage en profondeur a montré une performance supérieure dans de nombreux domaines. Par conséquent, explorer comment appliquer les techniques d'apprentissage en profondeur aux problèmes existants ou comment combiner les méthodes traditionnelles d'apprentissage automatique et l'apprentissage en profondeur est également une direction de recherche intéressante.

Considération de facteurs pratiques supplémentaires : Dans les applications pratiques, il est possible de rencontrer de nombreux problèmes complexes tels que le bruit des données, le déséquilibre des classes et la sélection des caractéristiques. Les recherches futures peuvent prendre en compte ces problèmes et explorer comment optimiser les performances des modèles face à ces défis.

Nous espérons que ces perspectives fourniront des idées pour les travaux futurs.

## VI. Conclusion et Commentaire

Ce rapport explore en profondeur certains concepts clés de l'apprentissage automatique, notamment la descente de gradient par lots (BGD) et la descente de gradient stochastique (SGD), ainsi que leurs applications dans la résolution de problèmes concrets tels que l'ajustement de courbes hyperquadratiques et la classification des données sur le cancer du sein, approfondissant ainsi notre compréhension de ces techniques importantes.

Nous avons étudié les deux principales variantes de la descente de gradient, BGD et SGD, mettant en évidence le compromis entre l'efficacité de calcul et la précision. Nous avons constaté que, face à de grands ensembles de données, la SGD présente généralement de meilleures performances et une convergence plus rapide. De plus, nous avons également constaté que nous pouvons optimiser davantage les effets de la descente de gradient en ajustant correctement le taux d'apprentissage et le moment.

En utilisant l'exemple de l'ajustement de courbes hyperquadratiques à des ensembles de points, nous avons compris plus intuitivement le fonctionnement de la descente de gradient et comment le point initial, le taux d'apprentissage et le moment affectent les performances de l'algorithme. Ces facteurs jouent un rôle crucial dans l'efficacité et les résultats du processus d'optimisation.

Dans la pratique de la classification des données sur le cancer du sein, nous avons appliqué les connaissances théoriques susmentionnées et constaté que BGD et SGD peuvent tous deux être utilisés avec succès pour l'entraînement de modèles de régression logistique. Cependant, ces deux méthodes diffèrent en termes de vitesse de convergence, de coût de calcul et de performances

finales du modèle, soulignant ainsi l'importance du choix de la méthode appropriée pour s'adapter à un problème spécifique.

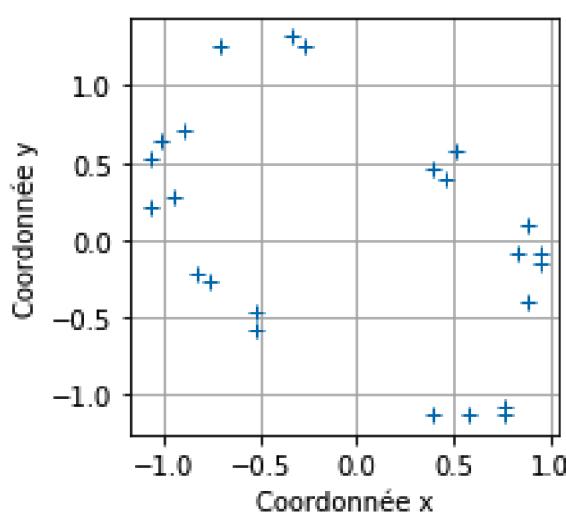
Enfin, nous avons étudié PyTorch, un outil puissant qui permet le calcul automatique des gradients. Cela nous a permis de nous concentrer davantage sur la conception et l'optimisation du modèle, sans avoir à effectuer directement de calculs différentiels complexes.

En résumé, en comprenant en profondeur les différentes techniques de descente de gradient et comment les choisir et les optimiser, nous pouvons résoudre plus efficacement divers problèmes d'apprentissage automatique. Cela met également en évidence l'importance de l'apprentissage continu et de l'adaptation aux dernières avancées technologiques, car il n'existe pas de méthode universelle pour résoudre tous les problèmes en apprentissage automatique. En acquérant une meilleure connaissance des outils et des méthodes disponibles, nous pouvons trouver les solutions les plus adaptées pour résoudre des problèmes spécifiques.

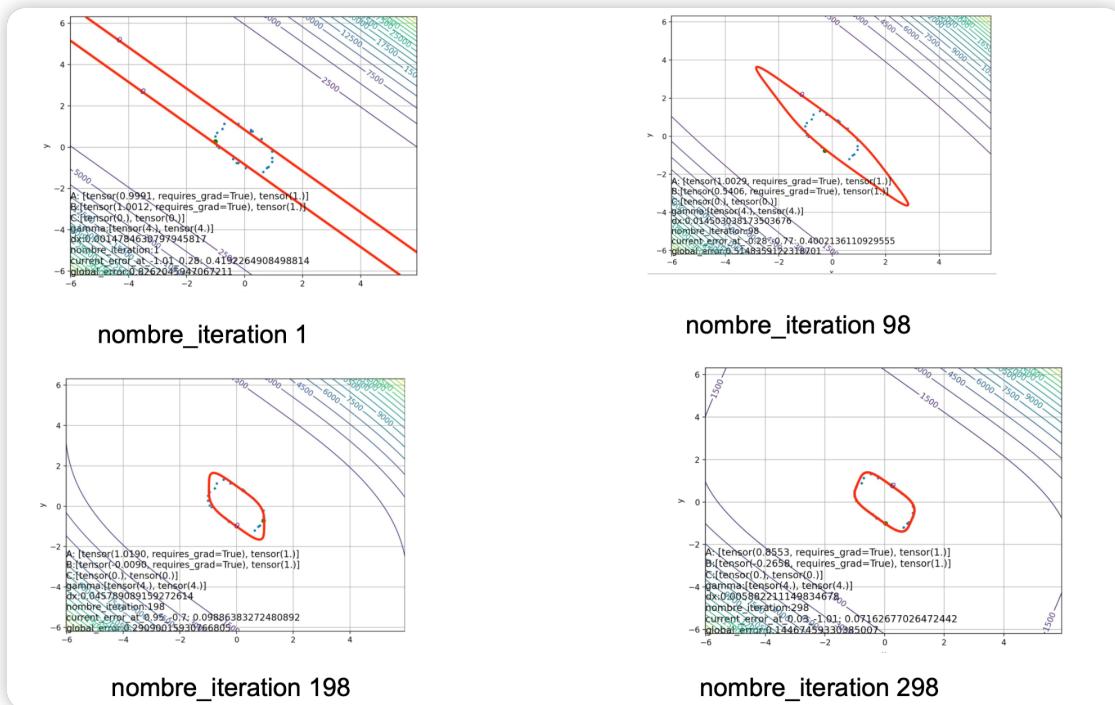
## Annexe

```
import torch
# Création d'un tenseur avec nécessité de suivi des gradients
x = torch.tensor(2.0, requires_grad=True)
# Effectuer une opération sur le tenseur
y = x**2 + 3*x + 1
# Calcul automatique des gradients
y.backward()
# Accéder au gradient du tenseur x
gradient = x.grad
print(gradient) #tensor(7.)
```

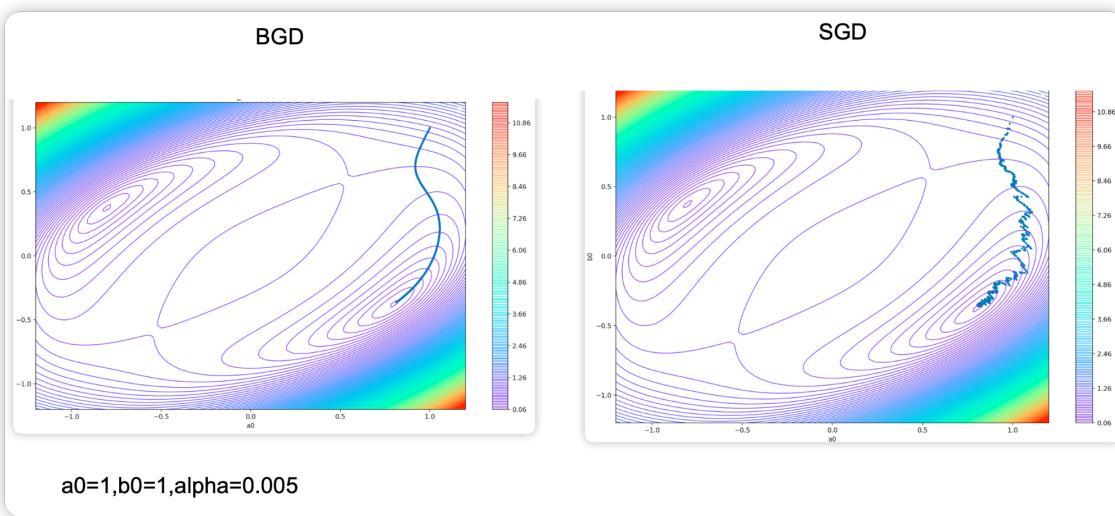
ANNEXE 1 - un exemple simple utilisant PyTorch pour calculer les gradients



ANNEXE 2 - L'image composée d'un nuage de points comprenant vingt-quatre points



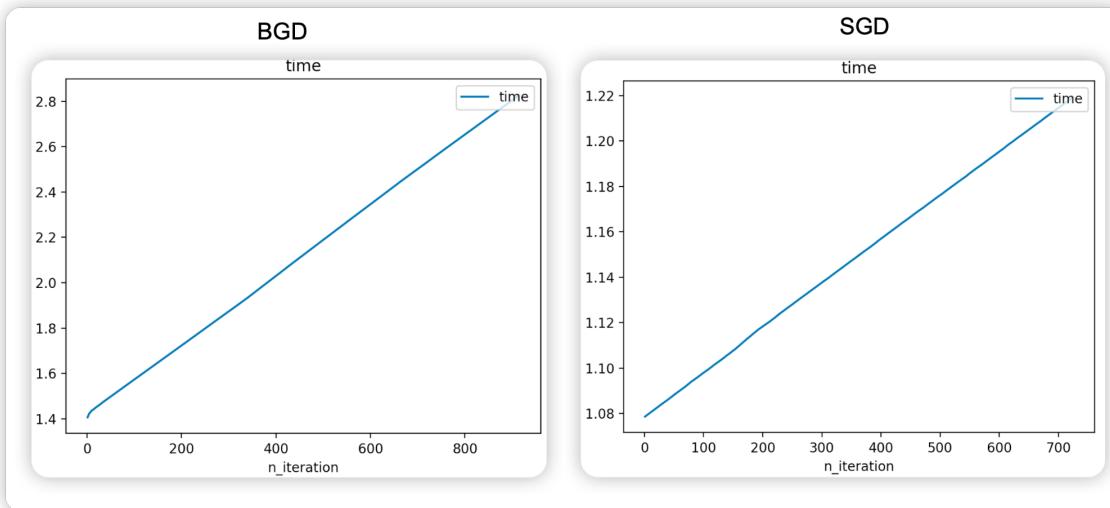
ANNEXE 3 - L'image ajustée varie avec l'itération(méthode sgd avec  
 $a_0=1, b_0=1, \alpha=0.005$ )(ajustement des 24 données)



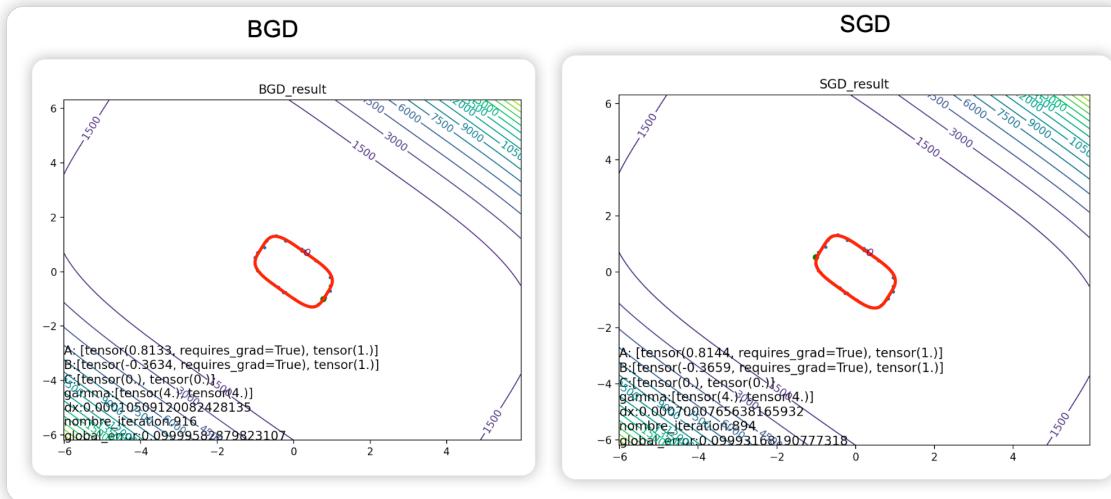
ANNEXE 4 - Le chemin de l'itération des paramètres (sgd et bgd avec a0=1 b0=1 et alpha=0.005)(ajustement des 24 données)

```
def isovaleur(a0,b0):
    phi=0
    for i in range (len(x)):
        phi+=((a0*x[i]+b0*y[i])**4+(x[i]+y[i])**4-1)**2
    return np.sqrt(phi/len(x))
a0 = np.linspace(-1.2, 1.2, 100)
b0 = np.linspace(-1.2, 1.2, 100)
A0, B0 = np.meshgrid(m, n)
fig, ax = plt.subplots()
ax.set_xlabel('a0')
ax.set_ylabel('b0')
fig.set_size_inches(15 ,15)
contour=ax.contour(A0, B0, isovaleur(A0,B0),levels=200,linewidths=1,cmap='rainbow')
```

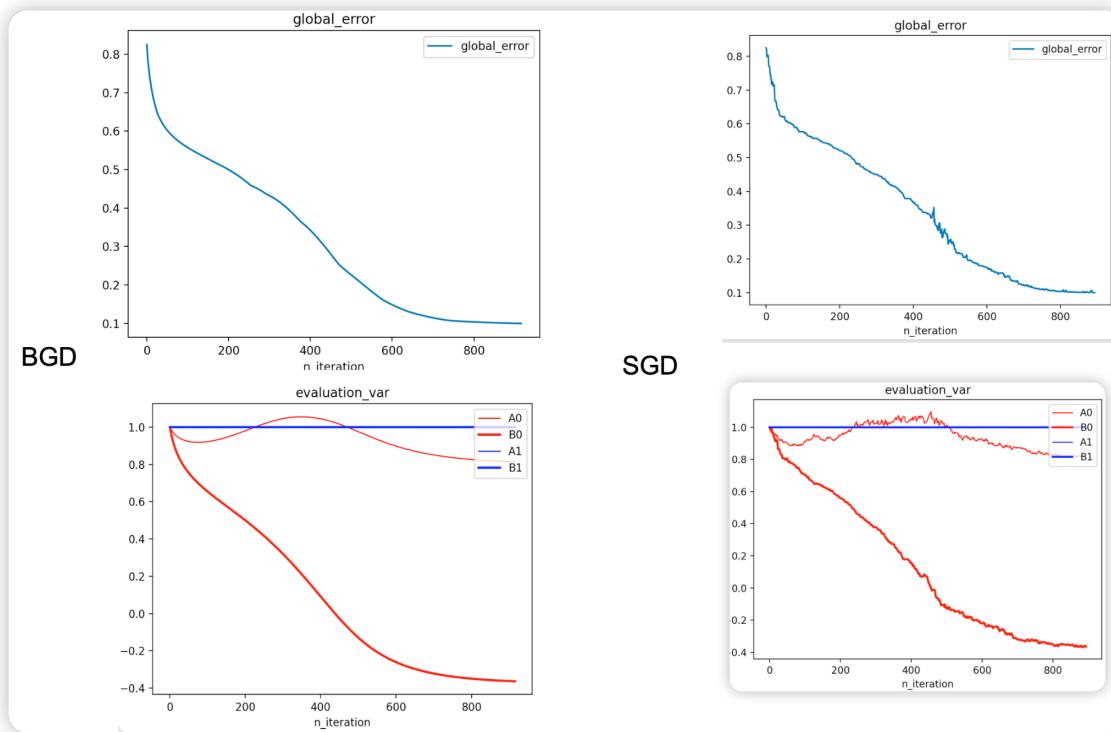
ANNEXE 5 - Le code impliquant le tracé des lignes de contour est en annexe



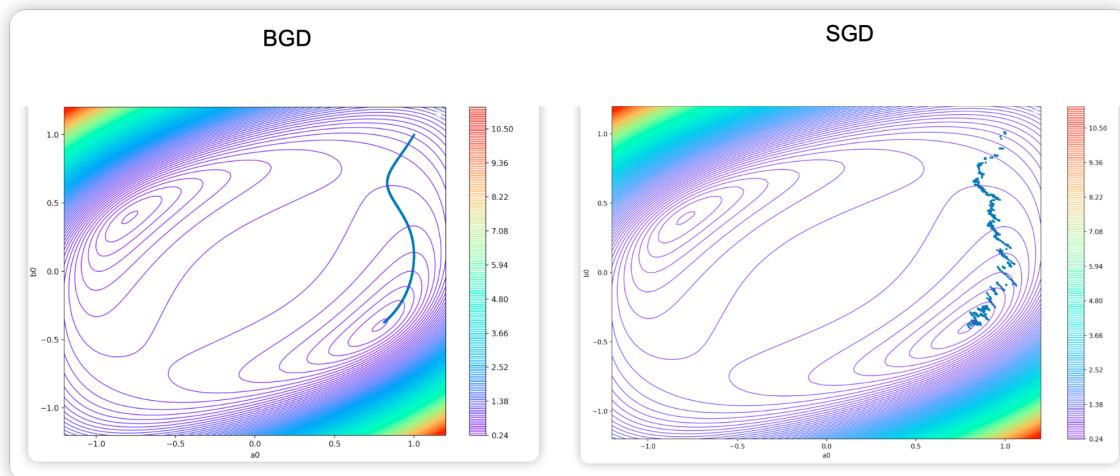
ANNEXE 6 - le temps brut qu'il a pris pour chaque méthode(sgd et bgd avec a0=1 b0=1 et alpha=0.005)(ajustement des 24 données)



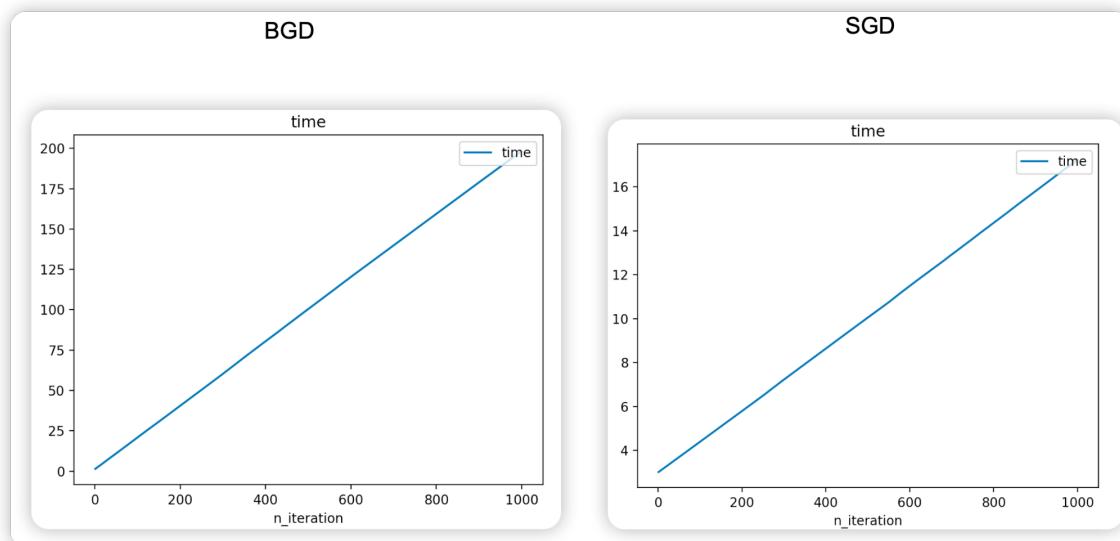
ANNEXE 7 - le resultat final apres la convergence(sgd et bgd avec  $a_0=1$   $b_0=1$  et  $\alpha=0.005$ )(ajustement des 24 donnees)



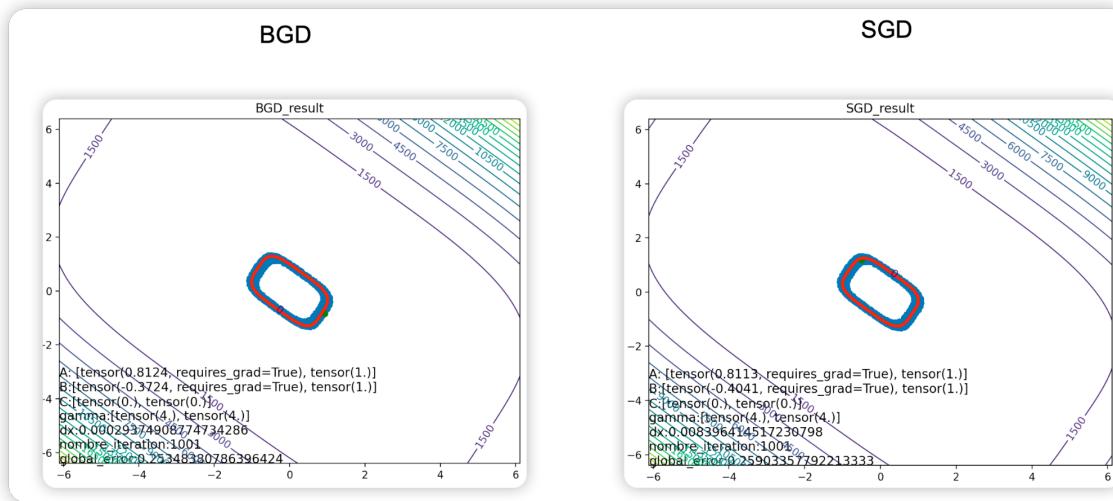
ANNEXE 8 -les schéma analytique pour illustrer l'évolution des paramètres et les erreurs (sgd et bgd avec  $a_0=1$   $b_0=1$  et  $\alpha=0.005$ )(ajustement des 24 données)



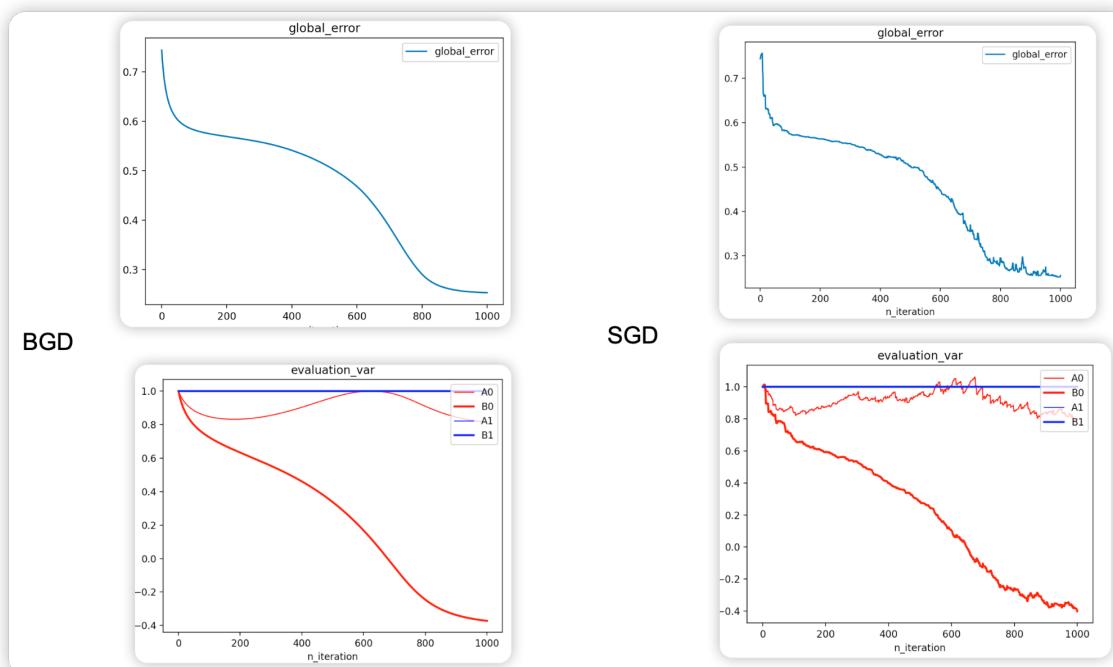
ANNEXE 9 - Le chemin de l'itération des paramètres (sgd et bgd avec  $a_0=1$   $b_0=1$  et  $\alpha=0.005$ )(ajustement des 3000 données)



ANNEXE 10 - le temps brut qu'il a pris pour chaque méthode(sgd et bgd avec a0=1 b0=1 et alpha=0.005)(ajustement des 3000 données)



ANNEXE 11 - le résultat final après la convergence(sgd et bgd avec a0=1 b0=1 et alpha=0.005)(ajustement des 3000 données)



ANNEXE 12 -les schéma analytique pour illustrer l'évolution des paramètres et les erreurs (sgd et bgd avec a0=1 b0=1 et alpha=0.005)(ajustement des 3000 données)

```

def J(x, y, z):
    return x**2 + y**3 + z**4
# mis à jour les paramètres
for epoch in range(num_epochs):
    # calcul du gradient
    grad_x = 2 * x
    grad_y = 3 * y**2
    grad_z = 4 * z**3

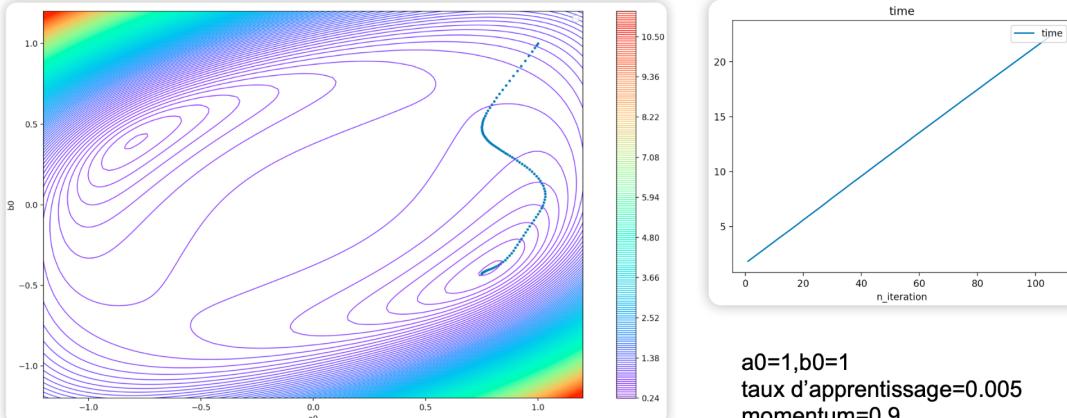
    # mis à jour le "moment" et les paramètres
    v_x = momentum * v_x - learning_rate * grad_x
    v_y = momentum * v_y - learning_rate * grad_y
    v_z = momentum * v_z - learning_rate * grad_z

    x = x + v_x
    y = y + v_y
    z = z + v_z

```

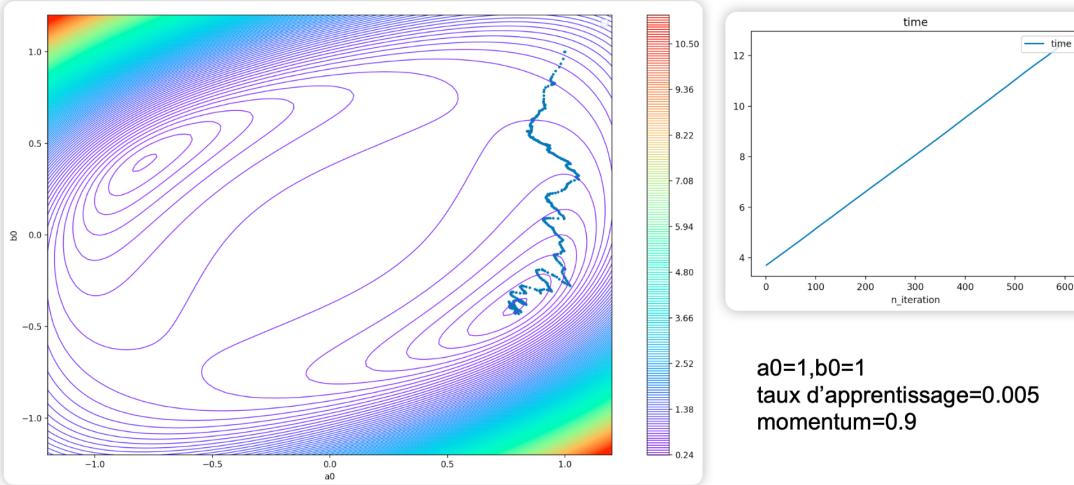
ANNEXE 13 - simple exemple pour illustrer le principe de la méthode du moment

BGD en utilisant le taux d'apprentissage avec moment

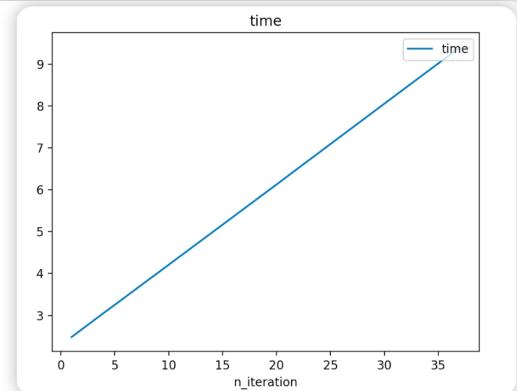
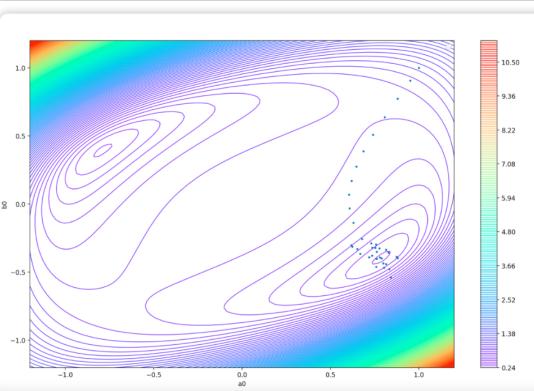


ANNEXE 14 - BGD en utilisant le taux d'apprentissage avec moment ( $a_0=1, b_0=1$  taux d'apprentissage=0.005 momentum=0.9)(3000 données)

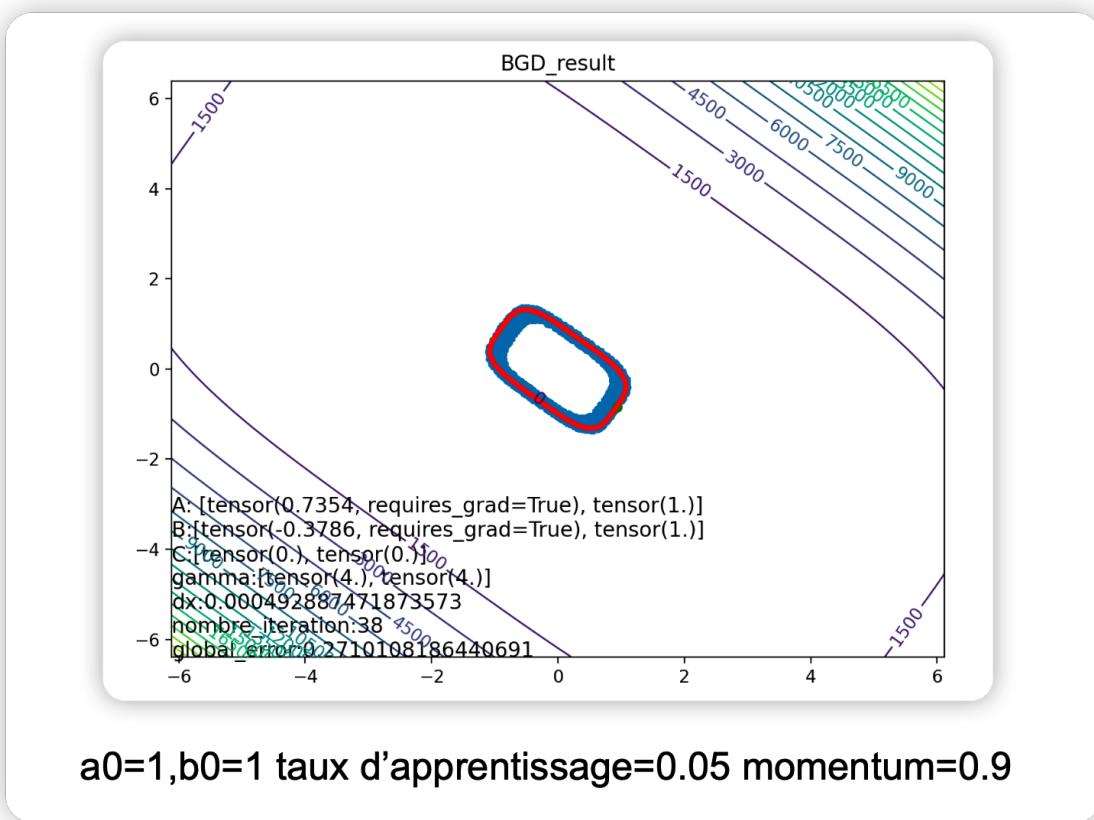
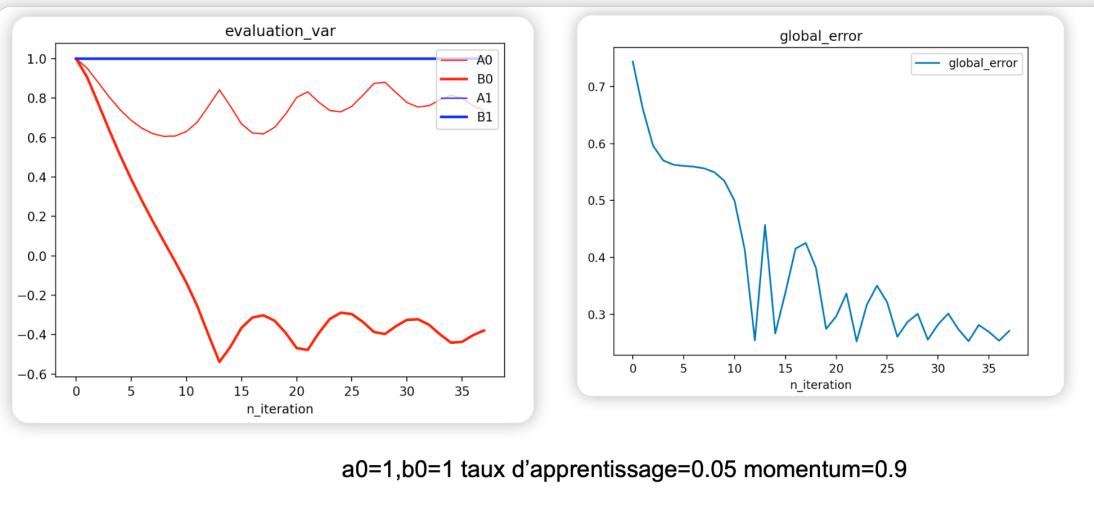
SGD en utilisant le taux d'apprentissage avec moment



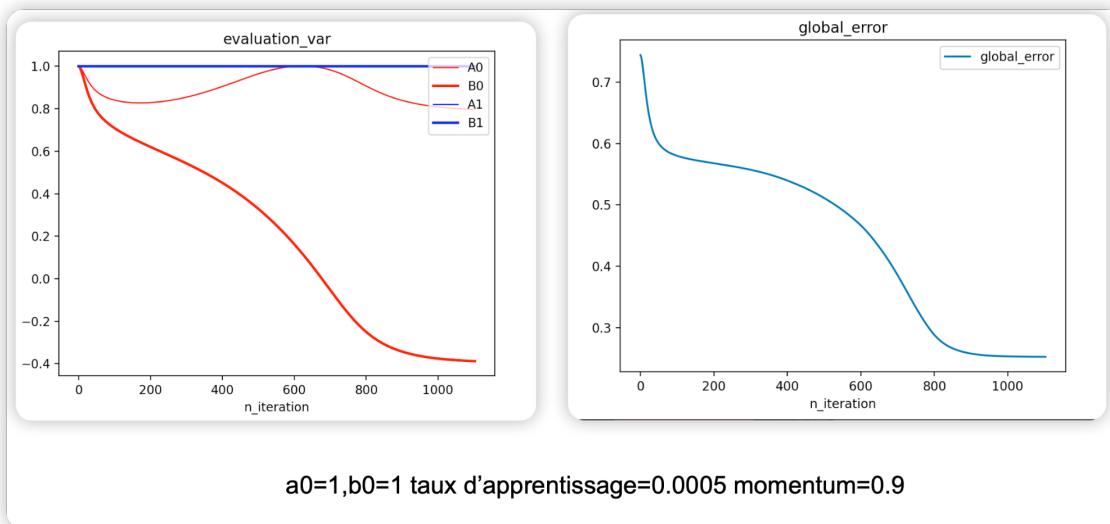
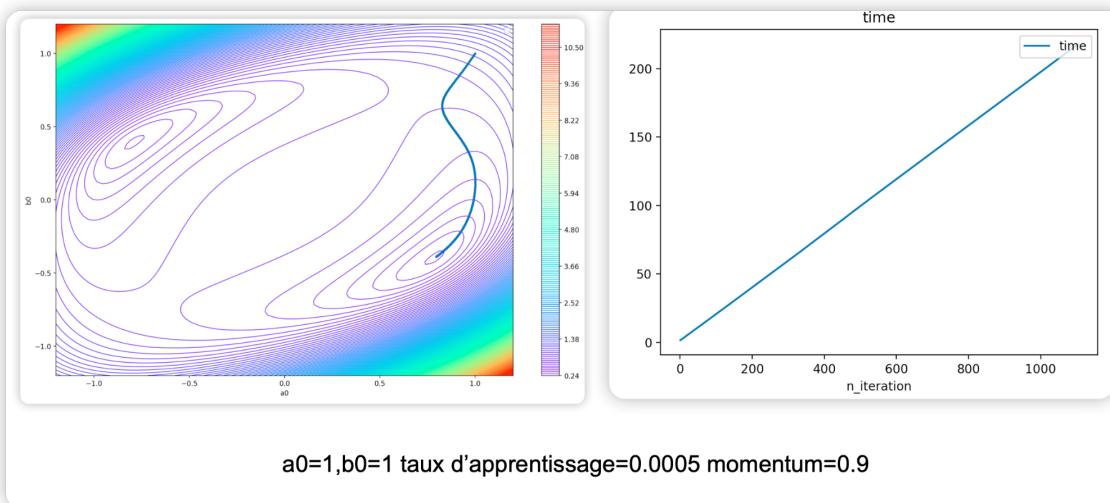
ANNEXE 15 - SGD en utilisant le taux d'apprentissage avec moment ( $a_0=1, b_0=1$  taux d'apprentissage=0.005 momentum=0.9)(3000 données)

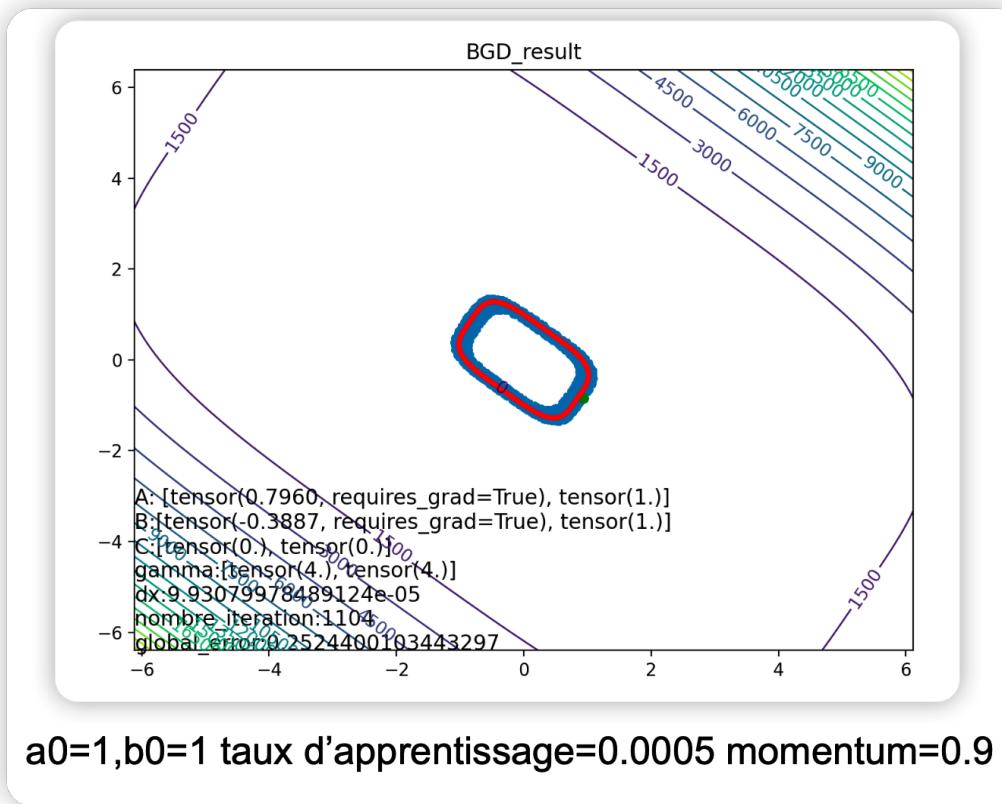


$a_0=1, b_0=1$  taux d'apprentissage=0.05 momentum=0.9

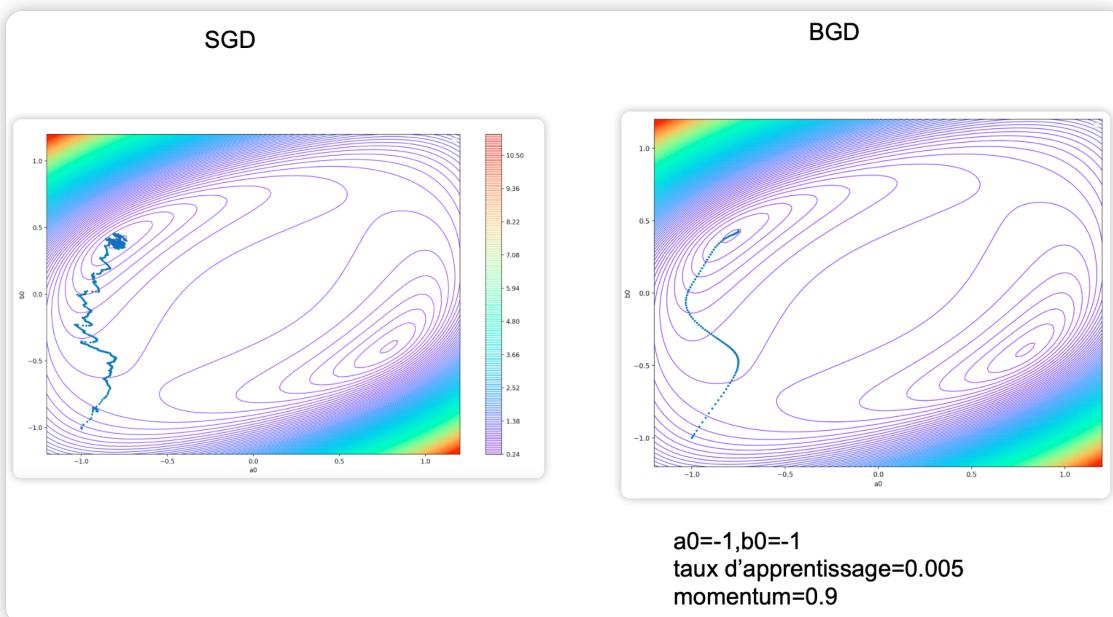


ANNEXE 16 - BGD en utilisant le taux d'apprentissage avec moment (a0=1,b0=1 taux d'apprentissage=0.05 momentum=0.9)(3000 données)

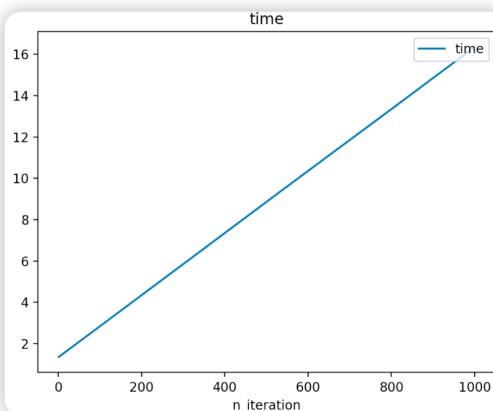




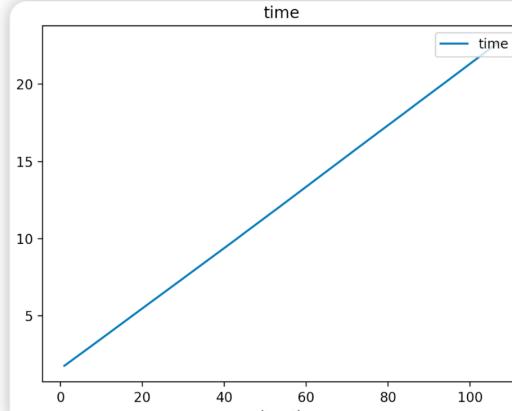
ANNEXE 17 - SGD en utilisant le taux d'apprentissage avec moment ( $a_0=1, b_0=1$  taux d'apprentissage=0.0005 momentum=0.9)(3000 données)



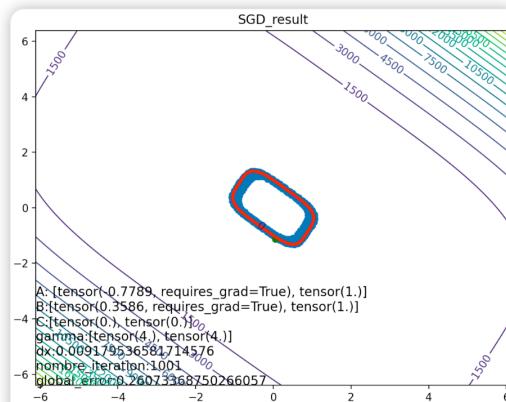
SGD



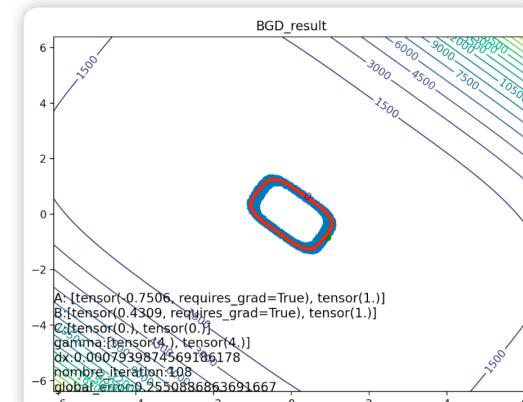
BGD

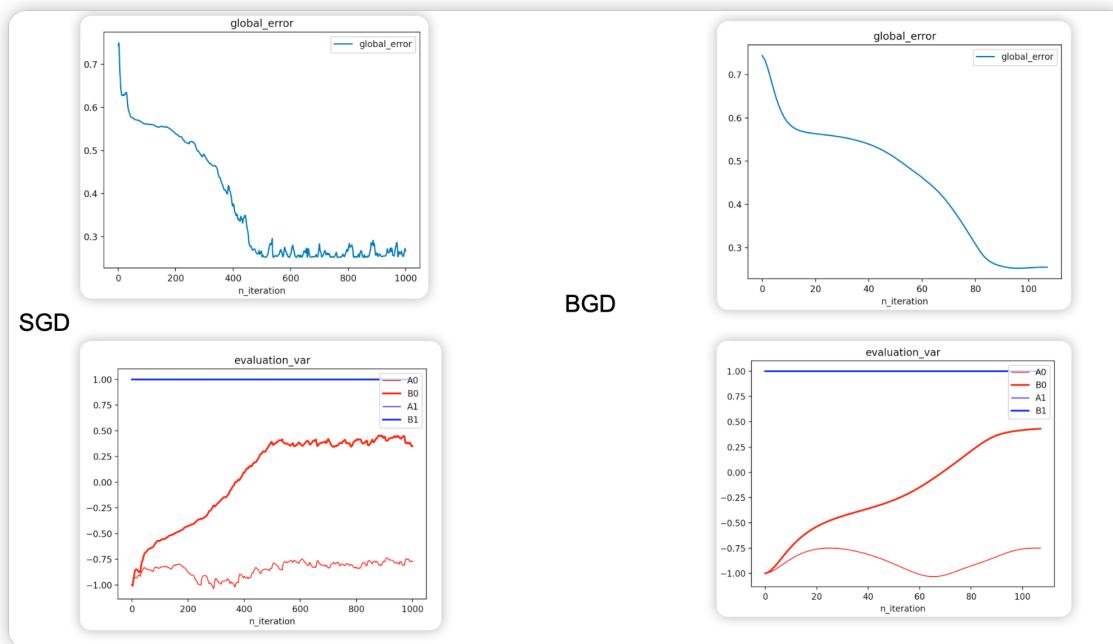


SGD

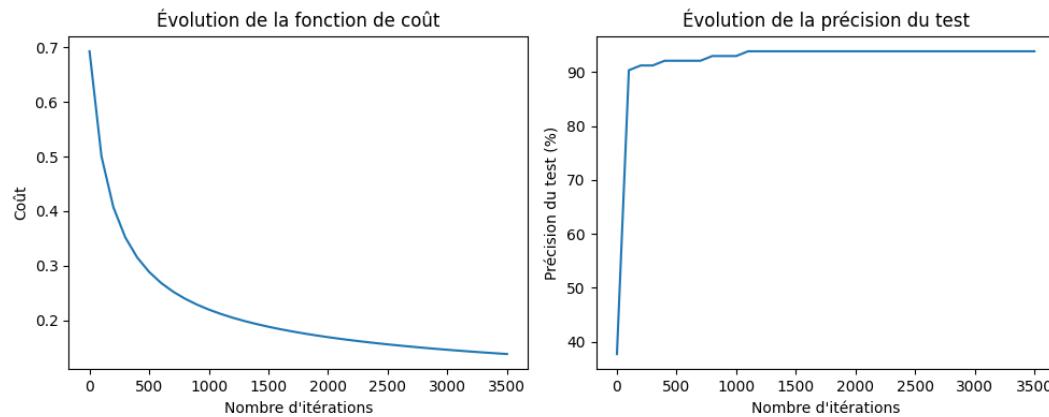


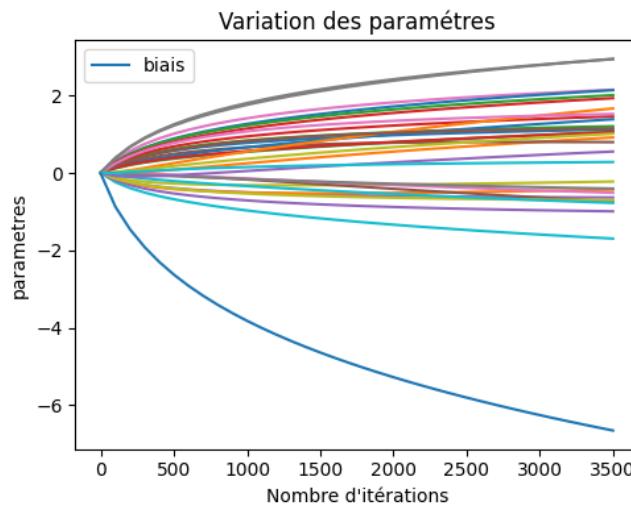
BGD



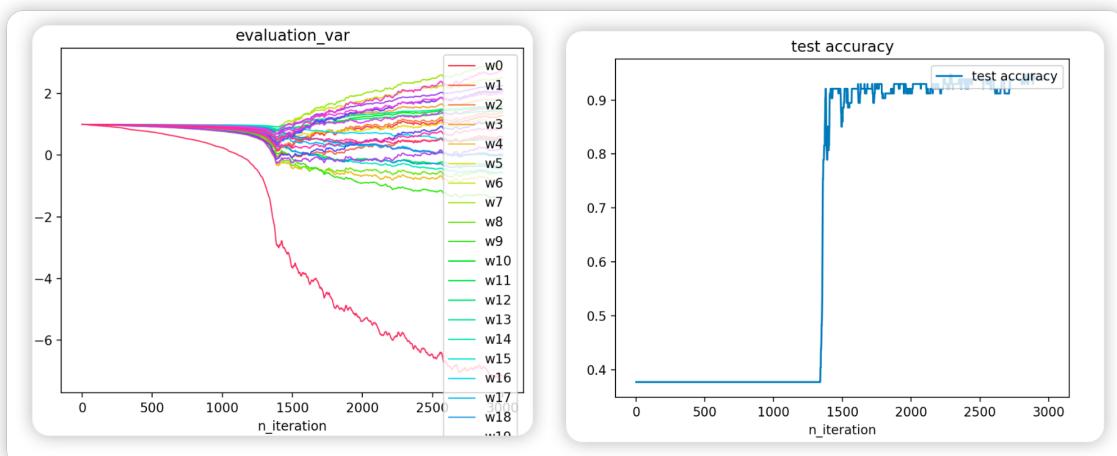


ANNEXE 18 - SGD et BGD en utilisant le taux d'apprentissage avec moment ( $a_0=-1, b_0=-1$  taux d'apprentissage=0.005 momentum=0.9)(3000 données)





ANNEXE 19 - Utilisant la méthode de la descente de gradient par lots (BGD), les résultats obtenus pour la classification logistique de type binomial sont réalisés (taux d'apprentissage = 0.1).



ANNEXE 20 - Utilisant la méthode de la descente de gradient par lots (SGD), les résultats obtenus pour la classification logistique de type binomial sont réalisés.

## Références

- [1] Brownlee, J. (2020, July 12). A Gentle Introduction to Gradient Descent for Machine Learning. Machine Learning Mastery.  
<https://machinelearningmastery.com/gradient-descent-for-machine-learning/>
- [2] Lutz, C. (2018, August 26). Understanding the Mathematics of Gradient Descent. Towards Data Science.  
<https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e>
- [3] PyTorch. (n.d.). PyTorch Tutorials. <https://pytorch.org/tutorials/>
- [4] PyTorch. (n.d.). Autograd: Automatic Differentiation.  
<https://pytorch.org/docs/stable/autograd.html>
- [5] Weisstein, Eric W. (n.d.). Hyperquadric. Wolfram MathWorld.  
<http://mathworld.wolfram.com/Hyperquadric.html>
- [6] UCI Machine Learning. (2017). Breast Cancer Wisconsin (Diagnostic) Data Set. Kaggle. <https://www.kaggle.com/uciml/breast-cancer-wisconsin-data>
- [7] Jack, M.T. (2019, May 14). Breast Cancer Classification with Logistic Regression. Medium.  
<https://medium.com/@mohammedterryjack/breast-cancer-classification-with-logistic-regression-568daf954c65>
- [8] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.
- [9] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Lin, Z. (2019). Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems (pp. 8026-8037).
- [10] Mirjalili, V., & Lewis, A. (2016). Quadratic programming in high dimensional feature spaces. arXiv preprint arXiv:1609.06973.
- [11] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.
- [12] Duda, R. O., Hart, P. E., & Stork, D. G. (2012). Pattern classification. John Wiley & Sons.