Rapport - Eléments Logiciels pour le Traitement des données massives

Henri QIU - Julien TRAN

February 1, 2016

1 Introduction

Dans le cadre de notre projet mené pour le cours d'Eléments Logiciels à l'ENSAE, nous avons choisi d'utiliser le langage PIG afin de pratiquer le calcul parallélisé. En guise de sujet, nous avons choisi un algorithme d'optimisation très populaire à l'heure actuelle - L-BFGS - qui trouve notamment son utilité pour l'estimation de paramètres dans la pratique du Machine Learning. A l'aide du *cluster* Azure mis à notre disposition, nous avons donc implémenté l'algorithme en question, tout en mettant en pratique une innovation proposée par l'article qui nous a été présenté (1). Dans ce rapport, nous préciserons dans un premier temps notre compréhension de l'article, puis dans un second temps nous détaillerons l'implémentation de l'algorithme en question. Enfin, nous présenterons nos résultats avec quelques notes critiques, avant de parcourir l'intégralité du code tel qu'il a été écrit sur le *notebook*.

2 Analyse de l'article; objectif du rapport

2.1 L'algorithme L-BFGS et ses limites

La méthode d'optimisation L-BFGS fait partie de la famille des méthodes "Quasi-Newton", dont la caractéristique est qu'elles ne nécessitent pas d'évaluer directement la matrice Hessienne, celle-ci étant déterminée par une relation linéaire du gradient de la fonction objectif, estimée avec réactualisation à chaque étape, jusqu'à atteindre la convergence. L'article de Chen, Wang et Zhou (1) rappelle que L-BFGS est un algorithme codé dans de nombreux *packages* et qui sert surtout pour les problèmes qui requièrent de travailler avec un grand nombre de variables. Or, peu d'études ont été faites sur la scalabilité de l'algorithme, c'est-à-dire sur son extensibilité à un nombre potentiellement très grand (de l'ordre du milliard) de variables. A l'âge du *Big Data*, une telle question devient très importante puisque L-BFGS, qui conserve en mémoire les états dits "historiques" du processus de convergence, rencontre fatalement des problèmes de stockage qui rend ledit algorithme inapplicable sur un noeud computationnel unique.

Algorithm 1: L-BFGS Algorithm Outline

Input: starting point x_0, integer history size $m > 0$, k=1;
Output: the position x with a minimal objective function
while *no converge* **do**
  Calculate gradient $\nabla f(x_k)$ at position x_k ;
  Compute direction p_k using Algorithm 2 ;
  Compute $x_{k+1} = x_k + \alpha_k p_k$ where α_k is chosen to satisfy Wolfe conditions;
  if $k > m$ **then**
    Discard vector pair s_{k-m}, y_{k-m} from memory storage;;
  end
  Update $s_k = x_{k+1} - x_k, y_k = \nabla f(x_{k+1}) - \nabla f(x_k), k = k + 1$;
end

Figure 1: Algorithme 1 - L-BFGS

Algorithm 2: L-BFGS two-loop recursion

Input: $\nabla f(x_k)$, s_i, y_i where $i = k - m, ..., k - 1$
Output: new direction p
$p = -\nabla f(x_k)$;
for $i \leftarrow k - 1$ **to** $k - m$ **do**
  $\alpha_i \leftarrow \frac{s_i \cdot p}{s_i \cdot y_i}$;
  $p = p - \alpha_i \cdot y_i$;
end
$p = (\frac{s_{k-1} \cdot y_{k-1}}{y_{k-1} \cdot y_{k-1}})p$
for $i \leftarrow k - m$ **to** $k - 1$ **do**
  $\beta = \frac{y_i \cdot p}{s_i \cdot y_i}$;
  $p = p + (\alpha_i - \beta) \cdot s_i$;
end

Figure 2: Algorithme 2 - L-BFGS
Calcul de la direction de convergence

La figure 1 donnée dans l'article en question donne la structure du L-BFGS : le gradient de la fonction objectif est calculé pour tous les inputs, puis une direction est calculée avec l'algorithme 2 (2) et enfin l'itération est effectuée à partir des conditions de Wolfe. Ces dernières retournent une longueur de pas qui réduit suffisamment la fonction objectif dans la direction précédemment calculée. Le champ d'action pour MapReduce est ici clair : il est tout à fait possible de paralléliser le calcul du gradient et celui des itérations, de même que la direction de convergence (étape Map) puis de sommer *in fine* (étape Reduce). Problème : l'algorithme 2, qui calcule la direction, nécessite des ressources de stockage énormes : pour m variables de longueur d, l'algorithme enregistre $2m+1$ vecteurs ce qui le rend de complexité égale à $2md$; cela impacte considérablement les capacités de mémoire d'un noeud computationnel unique lorsque m est de l'ordre du milliard.

2.2 L'algorithme VL-BFGS

Rappelons quelques notations utilisées dans l'article:

$$\begin{aligned} s_k &= x_{k+1} - x_k \\ y_k &= \nabla f(x_{k+1}) - \nabla f(x_k) \end{aligned} \tag{1}$$

Dans l'esprit, VL-BFGS s'inscrit dans la continuité des méthodes "*Mini-Batch Gradient Descent*" : il s'agit de ne conserver en mémoire qu'une fenêtre de taille limitée sur les données historiques pour n'optimiser que sur ces données et éviter ainsi une surcharge de capacité. Comme l'algorithme 2 ne nécessite que $(2m+1)$ variables, on conserve un historique m des anciennes itérations $(s_{k-m}, ..., s_{k-1})$ et des différences de gradient $(y_{k-m}, ..., y_{k-1})$ ainsi que le gradient de l'étape en cours $\nabla f(x_i)$. Synthétiquement, on note b le vecteur qui agrège tous ces inputs :

$$(b_1, ..., b_m, b_{m+1}, ..., b_{2m}, b_{2m+1}) = (s_{k-m}, ..., s_{k-1}, s_{k-m}, ..., s_{k-1}, \nabla f(x_i)) \tag{2}$$

De plus, remarquant que la direction p est linéaire en les inputs, le but de l'algorithme devient de déterminer entièrement la forme linéaire :

$$p = \sum_{k=1}^{2m+1} \delta_k b_k \tag{3}$$

où les (δ_k) sont les paramètres d'intérêt et p la direction souhaitée.

Enfin, l'opération principale de l'algorithme 2 étant le produit scalaire, la propriété de linéarité de p en tant que fonction des b_k rend d'autant plus pertinent la parallélisation du calcul du produit scalaire.

2.3 Notre approche

Il faut, pour notre algorithme, calculer des gradients en input; le problème est qu'il n'y a pas de formule fermée qui permette de réaliser cela pour tout type de fonction. Nous nous sommes donc restreints à la fonction des carrés :

$$f(x_1, ..., x_k) = \sum_{k=1}^{N} x_k^2 \tag{4}$$

et cherchons à présent à déterminer le (x_k) optimal qui minimise cette fonction objectif.

Précisons que nous avons tout d'abord travaillé sans notebook, en local sur Linux pour implémenter l'algorithme. Le passage entre les deux plateformes s'est avéré déroutant dans la mesure où certaines pratiques différaient d'un système à l'autre.

3 Notre implémentation du vector-free VL-BFGS

Pour mener à bien notre implémentation, nous avons travaillé avec le langage Pig Latin qui nous a été enseigné lors des séances de travaux pratiques, en parallèle avec le langage Python qui fournit les fonctions nécessaires pour procéder aux tâches plus élémentaires (boucles, lecture des data frames avec pandas etc.).

3.1 Construction de l'arborescence (en local)

Pour rester fidèle aux notations de l'article, nous avons construit les points X, S, Y, G : le premier désigne la variable itérée, S est la différence entre un X itéré et la valeur précédente, Y est la différence de gradient et G est le gradient à la nouvelle période (après itération). Plutôt que de travailler sur une matrice agrégée, nous avons procédé colonne par colonne afin de pouvoir nous débarrasser plus facilement des données qui ne seront pas utilisées pour le mini-batch lors d'une itération.

Nous créons ensuite un dossier Data dans lequel seront stockées toutes les données issues de l'initialisation et des itérations. Ainsi, pour les S, un dossier est créé à chaque itération, durant laquelle un nouveau point X_k est calculé par PIG, et celui-ci est enregistré dans un dossier *Data/X/X-k*. La procédure est la même pour Y, S et G. Chaque itération nécessite de connaître l'information des inputs (S et Y) des m périodes précédentes. A l'initialisation ($t = 0$), donc, il nous faut générer des données que l'on stocke dans les dossiers *Data/S/S--k* (notez le signe "-" (moins) pour les étapes avant 0) et qu'on a pris égales à 0. Ce qui s'applique à S s'applique aussi à Y, mais pas à X (puisqu'on ne s'intéresse pas aux points antérieurs) ni à G (qui n'a besoin que d'un calcul par itération). Enfin il faut aussi générer une valeur de départ pour X que l'on stocke dans le dossier *Data/X/X-0* et qui cette fois-ci est générée de manière aléatoire.

Permettons une remarque sur la génération aléatoire des termes initiaux : n'ayant pas creusé outre mesure les conditions de Wolfe, nous avons posé arbitrairement un $\alpha = 0.1$; mais pour cela nous devons initialiser avec précaution le premier point X_0 pour la raison suivante: l'idée de l'algorithme VLBFGS consiste à trouver la direction du gradient grâce aux m états historiques. En conséquence, les m premières itérations sont susceptibles de ne pas fonctionner correctement et donc de retourner une mauvaise direction. Ce problème est censé être compensé par le choix dynamique de pas α très petits voire nuls par les conditions de Wolfe. Or nous avons pris un α constant. La solution optée pour éviter de trop diverger durant les m premières itérations a été de générer une petite valeur pour le point de départ X_0

Enfin, un dossier temporaire *Data/Tmp* est créé. Ce dossier servira pour la communication des résultat entre pig et python. En effet, une fois les produits scalaires calculés par pig, ils seront écris sur le disque à l'emplacement *Data/Tmp/DotProd*. C'est donc dans ce répertoire que python viendra récupérer les résultats sur les produits scalaires. Un dossier *Data/Tmp/Delta* est aussi créé pour un rôle similaire concernant le calcul des deltas. Enfin un dossier pour le débogage à aussi été laissé dans l'arborescence.

3.2 Corps de l'algorithme

3.2.1 Le traitement des produits scalaires ("*dot products*")

Les produits scalaires sont centraux dans l'algorithme VL-BFGS et il faut les gérer de manière optimale.

- Gain de mémoire: dans l'article il nous est précisé que seuls les produits scalaires des m étapes précédentes sont utiles; il n'y a donc pas besoin de conserver en mémoire les produits des étapes antérieures : nous utiliserons donc des matrices (*array*) de taille $m \times m$ pour stocker les produits scalaires. Nous distinguons les produits entre les S, entre les Y, entre S et Y, et les produits scalaires faisant intervenir G (entre les G, entre G et S, et entre G et Y). Quatre matrices seront donc nécessaires. Dans le code ces matrices sont notées SSArr, YYArr, SYArr et GArr. SSArr est une matrice (*array*) de dimensions $m \times m$ car on a besoin à chaque itération k des produits scalaires $< S_i, S_j >, i, j \in \{k - m + 1, ..., k\}$. De la même manière les matrices YYArr et SYArr sont de dimensions $m \times m$. Mais il faut noter qu'on a seulement besoin d'une matrice de dimension $3 * m$ pour GArr: à l'itération k, la première ligne stocke les produits scalaires $< G_i, G_i >, i \in \{k-m+1, ..., k\}$, la deuxième contient $< G_k, S_j >, j \in \{k - m + 1, ..., k\}$, et la troisième contient $< G_k, Y_j >, j \in \{k - m + 1, ..., k\}$. On remarque qu'un seul élément de la première ligne de GArr est en fait nécessaire: $< G_k, G_k >$. Évidemment les matrices SSArr et YYArr sont symétriques, on se contentera donc de remplir qu'une moitié de la matrice.

- Gain de temps de calcul: l'article nous fait aussi remarquer qu'une grande partie des produits scalaires utilisés à l'itération k-1 sont aussi réutilisés à l'étape k. En effet à l'itération k, le *dotprod* $\langle S_{k-m}; S_{k-m} \rangle$ (à voir comme les *dotprod* a l'itération k des *historical state* de S numéro m) par exemple correspond au *dotprod* $\langle S_{(k-1)-(m-1)}; S_{(k-1)-(m-1)} \rangle$ (à voir comme les *dotprod* a l'itération k-1 des historical state de S numero m-1). On a donc fait attention à chaque itération k à ne calculer seulement les produits scalaires faisant intervenir les nouveaux points: S_k, Y_k et G_k. Ces nouveaux produits scalaires sont stockés dans les matrices en faisant attention à écrire dans les bonnes cases c'est-à-dire à la place des produits scalaires qui ne sont plus utiles à l'itération k (ne surtout pas écrire par dessus les produits scalaires qu'on a voulu conserver).

Nous venons d'expliquer dans les deux points précédents que les produits scalaires sont stockés dans des matrices de taille $m \times m$ au plus. Et qu'à chaque itération, de nouveaux produits scalaires sont calculés et sauvegardés dans les matrices en venant écraser les produits scalaires qui ne sont plus utiles. Il y a donc une logique d'indexation des matrices à comprendre pour savoir à chaque itération quels sont les emplacements où écrire les produits scalaires. Cette logique d'indexation est expliquée à la section suivante.

3.2.2 Un mot sur l'indexation des matrices

Les données issues du calcul du produit scalaire sont enregistrées dans les matrices notées SSArr, YYArr, SYArr et GArr. Il convient de revenir sur un point technique intéressant sur l'indexation de ces matrices : en effet, on se souvient qu'à chaque itération nous n'avons besoin que de calculer les produits scalaires nécessaires, car nous pouvons "recycler" des anciens résultats de précédentes itérations sur une fenêtre de taille m. Où est-ce que ces nouveaux produits scalaires seront alors stockés? Il nous faut donc préciser la méthode d'indexation de ces matrices.

Pour cela nous avons choisi une indexation dite "relative": par exemple, plutôt que d'enregistrer $\langle S_{100}, S_{98} \rangle$ à la ligne 100 et à la colonne 98, nous allons plutôt enregistrer à la ligne $100\%m$ et à la colonne $98\%m$ (% pour "modulo"). La conséquence est que la cellule de référence en (0,0) d'une matrice ne restera pas constante (en haut à gauche) et se "déplacera" sur la diagonale au fur et à mesure des itérations.

En annexe, nous avons illustré cette indexation par un cas concret. A noter que dans le code, la fonction python qui permet à partir d'un couple (i,j) de retrouver l'emplacement du bon produit scalaire est la fonction `convertTriMat` dans le script `vlbfgs.py`.

3.2.3 Intervention des scripts PIG

PIG intervient pour la parallélisation, qui comme précisé dans l'article intervient pour 3 tâches précises:

1. Calcul des gradients aux points x_i: `gradient.pig`;
2. Calcul des produits scalaires: `dotPords.pig`;
3. Calcul du nouveau point actualisé: `updateXnS.pig`.

A chaque itération, un nouveau gradient est calculé, ce qui nous permet ensuite d'actualiser les produits scalaires. Comme expliqué plus haut, l'actualisation des produits scalaire signifie que l'on calcule seulement les produits scalaires nécessaires: on profite d'une part du fait que les matrices de S et Y sont symétriques et d'autre part du fait que de nombreux produits scalaire nécessaires à l'itération k ont déjà été calculés à l'itération k-1. Notons bien que les résultats des produits scalaires sont stockés dans le dossier temporaire : une fois le calcul fait, Python récupère les données pour les écrire dans les matrices correspondant aux produits scalaires, puis les efface. La raison de cette dernière action est que PIG est incapable d'écraser un fichier avec le même nom (*overwrite*), donc l'itération suivante risque de connaître un problème (le job s'arrête automatiquement sur le cluster).

3.2.4 Fin de l'algorithme

Enfin, la fonction vlbfgs prend en argument tous les tableaux de produits scalaires historiques pour en retourner les deltas (paramètres de linéarité de p). L'algorithme suit la logique donnée par l'article:

Enfin, un dernier script Pig permet de calculer la valeur du nouveau point après itération.

Algorithm 3: Vector-free L-BFGS two-loop recursion

Input: $(2m+1)*(2m+1)$ dot product matrix between b_i

Output: The coefficients δ_i where $i = 1, 2, ...2m+1$

for $i \leftarrow 1$ **to** $2m+1$ **do**

  $\delta_i = i \leq 2m\ ?\ 0 : -1$

end

for $i = k-1$ **to** $k-m$ **do**

  $j = i-(k-m)+1$;

  $\alpha_i \leftarrow \frac{s_i \cdot p}{s_i \cdot y_i} = \frac{b_j \cdot p}{b_j \cdot b_{m+j}} = \frac{\sum_{l=1}^{2m+1} \delta_l b_l \cdot b_j}{b_j \cdot b_{m+j}}$;

  $\delta_{m+j} = \delta_{m+j} - \alpha_i$;

end

for $i \leftarrow 1$ **to** $2m+1$ **do**

  $\delta_i = (\frac{b_m \cdot b_{2m}}{b_{2m} \cdot b_{2m}})\delta_i$

end

for $i \leftarrow k-m$ **to** $k-1$ **do**

  $j = i-(k-m)+1$;

  $\beta = \frac{b_{m+j} \cdot p}{b_j \cdot b_{m+j}} = \frac{\sum_{l=1}^{2m+1} \delta_l b_{m+j} \cdot b_l}{b_j \cdot b_{m+j}}$;

  $\delta_j = \delta_j + (\alpha_i - \beta)$

end

Figure 3: Algorithme 3 - VL-BFGS

3.3 Passage sur cluster Azure; difficultés rencontrées

Nous sommes parvenus à faire fonctionner l'algorithme sur le plan local. Lors du passage sur cluster, en revanche, nous avons rencontré les quelques difficultés suivantes que nous listons brièvement dans cette partie.

3.3.1 De la difficulté d'itérer

Notre algorithme est itératif et nécessite de faire fonctionner 3 scripts de PIG à chaque itération - or cela devient très ardu d'aboutir à plusieurs itérations dans la mesure où les jobs PIG subissent un lag après la commande de lancement et mettent du temps à s'exécuter - il est donc très compliqué de faire "tourner" une boucle sachant que celle-ci doit renvoyer le résultat d'un job PIG, et nous n'avons pas trouvé la commande pour geler les exécutions en attendant que le job termine. De même, comme le cluster est aussi usité par d'autres utilisateurs, nous craignions que notre itération ne paralyse le cluster entier, empêchant les autres utilisateurs de lancer leurs commandes de job correctement.

In fine, ceci explique pourquoi nous n'avons effectué qu'une seule itération sur le rendu notebook - celui-ci ne sert véritablement qu'à présenter les fonctions principales de notre algorithme; le cadre local nous paraît de meilleure facture pour une architecture plus claire et rigoureuse.

3.3.2 Des difficultés d'adaptation

Le code local s'avérait plutôt intuitif étant donné le langage Linux qui nous était davantage familier. La migration sur le cluster a notamment posé un problème lors du commencement, puisque nous n'avions pas trouvé la commande magique de *pyensae* pour créer des dossiers (et ainsi mettre en place notre arborescence) - nous doutons à ce jour qu'elle existe. Nous avons également rencontré des problèmes dans l'exécution des codes PIG : seule une calibration minutieuse nous a permis finalement de nous rendre compte que les chemins devaient être *loadés* d'une certaine façon (en tenant compte du *$CONTAINER/*), que les fonctions Jython devaient être considérées comme dépendances...

4 Résultats; quelques notes critiques

Malgré les quelques entraves à une implémentation fluide de l'algorithme sur le cluster, nous avons réussi à faire fonctionner l'ensemble et même à aboutir à des résultats de convergence intéressants en local.

4.1 Algorithme et convergence

Pour nous assurer que l'algorithme convergeait bien, nous avons tracé la trajectoire de chaque composante de X afin d'évaluer le comportement de chacune.

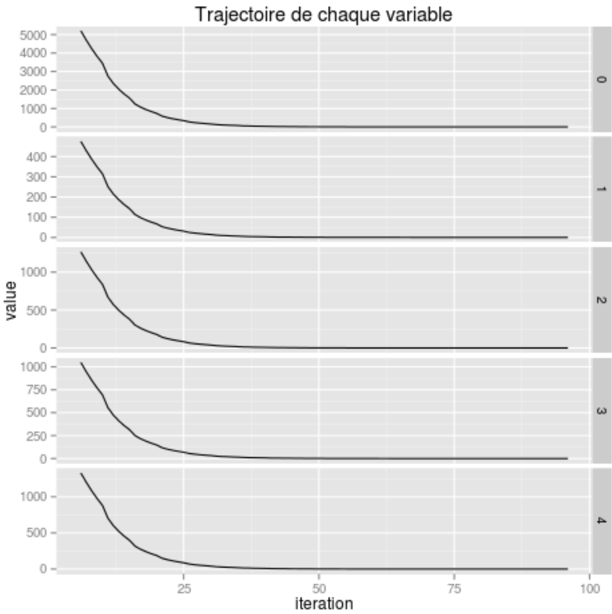

Figure 4: Trajectoires de convergence pour 5 composantes de X

Le graphique 4 montre bien que les composantes tendent toutes vers zéro après un grand nombre d'itérations. Il faut bien noter qu'un tel comportement ne s'observe qu'à partir de la sixième itération - puisque $m = 5$ états historiques sont nécessaire pour le bon fonctionnement de l'algorithme (i.e. les itérations 1 à 5 divergent).

L'on notera également que la convergence se fait de manière lente. Ceci est dû en grande partie au coefficient α que nous avons choisi arbitrairement petit, alors que celui-ci devrait être déterminé par les conditions de Wolfe qui permettent d'accélérer la convergence. En réalité, nous avons pris un pas $\alpha = 0.1$ qui peut être considéré comme prudent : si celui-ci était trop grand, on aurait pu effectuer un "pas" itératif trop grand - or même si nous étions dans la bonne direction c'est-à-dire la direction opposée du gradient, nous aurions pu avancer "trop loin" et donc passer de l'autre côté du 0 et de nous y éloigner. Mais en prenant un pas trop petit, nous nous sommes contraints à mettre en place une convergence de petits pas alors que les conditions de Wolfe nous auraient permis de détecter les cas où l'on aurait pu se permettre de faire un grand pas. *In fine*, nous reconnaissons que l'implémentation desdites conditions aurait pu nous permettre d'aboutir à plus d'efficacité.

4.2 Problèmes non résolus - quelques *hindsights*

Quelques problèmes d'exécution du code ont pu avoir lieu sans que l'on ne trouve la racine du problème :

- A l'initialisation, une erreur de division aléatoire fait apparaître un problème de division par zéro (*float division error*). Il s'agit d'une division qui survient dans l'algorithme 3. Il convient de préciser que ce problème semble disparaître dès la seconde itération si l'erreur ne survient pas à la première itération. Il semble donc que la probabilité de rencontrer cette erreur devient négligeable dès lors que les états historiques sont utilisés pour l'algorithme 3. Et dans ce cas, un moyen de régler le problème aurait été de randomiser les états -5, -4... -1 qui précèdent la première itération (0). Cela augmenterait les degrés de liberté, ce qui fait qu'on aurait moins de risque d'avoir cette erreur de division par zéro. Néanmoins l'origine de l'erreur n'a pas été clairement identifiée, et cette erreur intervient de manière aléatoire.

- La convergence (en local) connaît un bogue à l'itération 96 et n'atteint pas les 100 prévus. Nous ignorons la raison de ce problème.

Une remarque finale : bien entendu, l'exemple de la fonction-objectif ne constitue pas la fin ultime de ce travail - elle n'est là que pour les besoins d'illustration. En effet il est possible de faire une optimisation sur cette fonction de manière très aisée avec des fonctions précodées dans Python ou dans R - ce qui peut éviter le coût d'entrée immense de devoir maîtriser PIG. Nous rappellerons en guise de conclusion que la qualité de cet algorithme tient dans sa capacité à être "*scalable*", i.e. utilisable sur une très grande base de données, avec un nombre de variables de l'ordre du milliard.

5 Code

5.1 Code Local

Dans le dossier zip joint avec le mail de rendu, l'on pourra trouver:

- Un dossier principal: `VLBFGS/`
- Le dossier VLBFGS contient deux dossiers: `Datas/` et `Source/`
- Dans `VLBFGS/Source/` : l'intégrale des codes
- Dans `VLBFGS/Datas/` : on a laisse l'arborescence obtenue apres avoir lance le code en local sur 100 iterations. Remarque : il y a un bogue à partir de l'itération 96.

Enfin, pour lancer le code:

- Decompresser `QIU_TRAN.tar.gz`
- Se placer dans `VLBFGS/`
- lancer la commande: `pig -x local Source/main.py`

5.2 Code Notebook

In [1]:
```
import pyensae
%nb_menu
```

Out[1]:
```
<IPython.core.display.HTML object>
```

In [2]:
```
## Connexion au cluster

### blob : hdblobstorage
### hadoop : clusterensaeazure1
### pass 1 : jQIPVO/T54w8X49UPIbzAVvaNO3wmuUwI4/o9AJnCaPTHoCQnsaGBUkT4eIyi0BRQavgc/TAQMQwy8eu19
### pass 2 : 2azureENSAE;

import os
import pyquickhelper
blobhp = {}
if "HDCREDENTIALS" in os.environ:
    blobhp["blob_storage"], blobhp["password1"], blobhp["hadoop_server"], blobhp["password2"], b
        os.environ["HDCREDENTIALS"].split("**")
    r = type(blobhp)
else:
    from pyquickhelper.ipythonhelper import open_html_form
    params={"blob_storage":"", "password1":"", "hadoop_server":"", "password2":"", "username":"a
    r = open_html_form(params=params,title="server + hadoop + credentials", key_save="blobhp")
r
```

```
Out[2]: <IPython.core.display.HTML object>

In [3]: import pyensae
        blobstorage = blobhp["blob_storage"]
        blobpassword = blobhp["password1"]
        hadoop_server = blobhp["hadoop_server"]
        hadoop_password = blobhp["password2"]
        username = blobhp["username"] + "az"
        client, bs =  %hd_open
        client, bs

Out[3]: (<pyensae.remote.azure_connection.AzureClient at 0x7aea908048>,
         <azure.storage.blob.blobservice.BlobService at 0x7aea908080>)

In [4]: ##### Fenêtre d'importations de packages

        import pandas
        import os
        import datetime
        import numpy
        import random
        import sys
        import csv

In [5]: #### Dossiers sources

        sourceDir = '/$PSEUDO/'
        dataDir = sourceDir + 'Data/'
        XDir        = dataDir    + 'X/'
        SDir        = dataDir    + 'S/'
        GDir        = dataDir    + 'G/'
        YDir        = dataDir    + 'Y/'
        TmpDir      = dataDir    + 'Tmp/'
        dotProdDir  = TmpDir     + 'DotProd'
        arrDir      = TmpDir     + 'Arr/'
        deltaDir    = TmpDir     + 'Delta/'

In [6]: ## Facteurs initiaux
        ## On ne teste que sur une itération

        m = 5
        nvar = 5
        k = 100
        alpha = 0.1
        i=0

In [7]: ## On initialise les entrées initiales

        def init(umin,umax,d,t,n,nvar):
            path = d + t + '-' + str(n) + '/part-r-00000'
            pnom = 'part-r-00000'
            with open(pnom, "w") as f :
                for i in range(nvar):
                    f.write(t+'\t')
                    f.write(str(n)+'\t')
```

```
            f.write(str(i)+'\t')
            f.write(str(random.uniform(umin,umax))+'\n')

    X_i = pandas.read_csv(pnom,header = None, sep = "\t", engine = "python")
    X_i.columns = ['type','n','var','x']
    nom_csv = pnom + ".csv"
    X_i.to_csv(nom_csv, sep='\t', encoding='utf-8', index = False, header = False)

    return nom_csv, path
```

In [9]:
```
## On uploade les fichiers générés aléatoirement pour X_0, S_0; le reste on remplit de 0

name, path = init(umin = -10, umax = 10, d = dataDir + 'X/',t = 'X', n=0, nvar = nvar)
%blob_up name path

name, path = init(umin = -10, umax = 10, d = dataDir + 'S/',t = 'S', n=0, nvar = nvar)
%blob_up name path

name, path = init(umin = 0, umax = 0, d = dataDir + 'G/',t = 'G', n=-1, nvar = nvar)
%blob_up name path

for i in range(m):
    name, path = init(umin = 0, umax = 0, d = dataDir + 'Y/',t = 'Y', n=-1-i, nvar = nvar)
    %blob_up name path

for i in range(m):
    name, path = init(umin = 0, umax = 0, d = dataDir + 'S/',t = 'S', n=-1-i, nvar = nvar)
    %blob_up name path
```

In [10]:
```
%blob_ls /$PSEUDO
```

Out[10]:
```
                                 name                   last_modified  \
0   tranqqaz/Data/G/G--1/part-r-00000  Mon, 01 Feb 2016 03:48:31 GMT
1   tranqqaz/Data/S/S--1/part-r-00000  Mon, 01 Feb 2016 03:48:32 GMT
2   tranqqaz/Data/S/S--2/part-r-00000  Mon, 01 Feb 2016 03:48:32 GMT
3   tranqqaz/Data/S/S--3/part-r-00000  Mon, 01 Feb 2016 03:48:33 GMT
4   tranqqaz/Data/S/S--4/part-r-00000  Mon, 01 Feb 2016 03:48:33 GMT
5   tranqqaz/Data/S/S--5/part-r-00000  Mon, 01 Feb 2016 03:48:33 GMT
6    tranqqaz/Data/S/S-0/part-r-00000  Mon, 01 Feb 2016 03:48:31 GMT
7    tranqqaz/Data/X/X-0/part-r-00000  Mon, 01 Feb 2016 03:48:31 GMT
8   tranqqaz/Data/Y/Y--1/part-r-00000  Mon, 01 Feb 2016 03:48:32 GMT
9   tranqqaz/Data/Y/Y--2/part-r-00000  Mon, 01 Feb 2016 03:48:32 GMT
10  tranqqaz/Data/Y/Y--3/part-r-00000  Mon, 01 Feb 2016 03:48:32 GMT
11  tranqqaz/Data/Y/Y--4/part-r-00000  Mon, 01 Feb 2016 03:48:32 GMT
12  tranqqaz/Data/Y/Y--5/part-r-00000  Mon, 01 Feb 2016 03:48:32 GMT

                content_type  content_length  blob_type
0   application/octet-stream              60  BlockBlob
1   application/octet-stream              60  BlockBlob
2   application/octet-stream              60  BlockBlob
3   application/octet-stream              60  BlockBlob
4   application/octet-stream              60  BlockBlob
5   application/octet-stream              60  BlockBlob
6   application/octet-stream             128  BlockBlob
7   application/octet-stream             130  BlockBlob
```

```
8   application/octet-stream              60  BlockBlob
9   application/octet-stream              60  BlockBlob
10  application/octet-stream              60  BlockBlob
11  application/octet-stream              60  BlockBlob
12  application/octet-stream              60  BlockBlob
```

```
In [11]: %%PIG gradient.pig

         A = LOAD '$XInput' AS (type:chararray, n:int, var:int, x:double);

         B = FOREACH A GENERATE 'G' as type:chararray, n as n:int, var as var:int, 2*x as g:double;

         STORE B INTO '$GOutput' using PigStorage('\t');

         C = LOAD '$GInput' as (type:chararray, n:int, var:int, g:double);

         D = JOIN B BY var, C BY var;

         E = FOREACH D generate 'Y' as type:chararray, B::n as n:int, B::var as var:int, B::g - C::g as

         STORE E INTO '$YOutput' using PigStorage('\t');
```

```
In [12]: ## On lance la commande pour soumettre gradient.pig et exécuter

         client.pig_submit(bs,client.account_name,"gradient.pig",
                           params=dict(XInput = '$CONTAINER/$PSEUDO/Data/X/X-0',
                                       GOutput = '$CONTAINER/$PSEUDO/Data/G/G-0',
                                       GInput = '$CONTAINER/$PSEUDO/Data/G/G--1',
                                       YOutput = '$CONTAINER/$PSEUDO/Data/Y/Y-0'
                                      ),
                           stop_on_failure=True)

Out[12]: {'id': 'job_1451961118663_6708'}
```

```
In [135]: st0 = %hd_job_status job_1451961118663_6708
          st0["id"], st0["percentComplete"], st0["status"]["jobComplete"]

Out[135]: ('job_1451961118663_6708', '100% complete', True)
```

```
In [132]: ## On génère les indices des matrices qu'on veut utiliser (avec les paths correpondants)

          def SSLR_id(SDir,m,i):
              nSDir = '$CONTAINER' + SDir
              right  = '%s%s%d' % (nSDir,'S-',i)
              ileft = '{'
              for k in range(m):
                  ileft += str(i-m+1+k)
                  if k == m-1:
                      ileft += '}'
                  else:
                      ileft += ','
              left = '$CONTAINER' + SDir + 'S-' + ileft
              return [left,right]

          def YYLR_id(YDir,m,i):
```

```
        nYDir = '$CONTAINER' + YDir
        right  = '%s%s%d' % (nYDir,'Y-',i)
        ileft = '{'
        for k in range(m):
            ileft += str(i-m+1+k)
            if k == m-1:
                ileft += '}'
            else:
                ileft += ','
        left = '$CONTAINER' + YDir + 'Y-' + ileft
        return [left,right]

    def SYLR_id(SDir,YDir,m,i):
        #### Right1 Left2
        nYDir = '$CONTAINER' + YDir
        right1  = '%s%s%d' % (nYDir,'Y-',i)
        nSDir = '$CONTAINER' + SDir
        left2 = '%s%s%d' % (nYDir,'S-',i)

        #### Right2 Left1
        mm = m-1
        ileft = '{'
        for k in range(mm):
            ileft += str(i-mm+k)
            if k == mm-1:
                ileft += '}'
            else:
                ileft += ','

        left1 = '$CONTAINER' + SDir + 'S-' + ileft

        iright = '{'
        for k in range(m):
            iright += str(i-m+1+k)
            if k == m-1:
                iright += '}'
            else:
                iright += ','

        right2 = '$CONTAINER' + YDir + 'Y-' + iright

        return [left1,right1,left2,right2]

    def GLR_id(i):
        left = '$CONTAINER/$PSEUDO/Data/{G,S,Y}/{G,S,Y}-' + str(i)
        right = '$CONTAINER/$PSEUDO/Data/G/G-' + str(i)
        return [left,right]
```

In [136]:
```
#### On a codé les deux fonctions jython.py et jython0.py pour pouvoir nous en servir dans le
```

In [18]:
```
%%PYTHON jython.py

@outputSchema("joinedBag:{t:(type:chararray, n:int, var:int, x:double)}")
def joinBags(b1,b2):
    return b1+b2
```

In [19]: %%PYTHON jython0.py

```
@outputSchema("dot:double")
def dot(bag):
    out = 1.0
    for t in bag:
        out = out * t[0]
    return out
```

In [20]: i=0

In [22]: %%PIG dotProds.pig

```
REGISTER '$CONTAINER/$SCRIPTPIG/jython.py' USING jython AS myfuncs;
REGISTER '$CONTAINER/$SCRIPTPIG/jython0.py' USING jython AS myfuncs0;

DEFINE DUPLICATE(IN) RETURNS OUT
{
    $OUT = FOREACH $IN GENERATE *;
};

A1 = LOAD '$LEFT' AS (type:chararray, n:int, var:int, x:double);
A2 = LOAD '$RIGHT' AS (type:chararray, n:int, var:int, x:double);
B1 = GROUP A1 BY (type, n);
B2 = GROUP A2 BY (type, n);
CC = CROSS B1, B2;
C  = FOREACH CC GENERATE * AS (gr1:(type:chararray, n:int), b1:{t:(type:chararray, n:int, var::
            gr2:(type:chararray, n:int), b2:{t:(type:chararray, n:int, var:int, x:double)});
D  = FOREACH C GENERATE gr1, gr2, myfuncs.joinBags(b1,b2) AS b;
E  = FOREACH D GENERATE gr1, gr2, FLATTEN(b) AS (type: chararray, n:int, var:int, x:double);
F  = FOREACH E GENERATE gr1, gr2, var, x;
GG = GROUP F BY (gr1, gr2, var);
G  = FOREACH GG GENERATE FLATTEN(group) AS (gr1:(type:chararray, n:int), gr2:(type:chararray, 1

H  = FOREACH G GENERATE gr1, gr2, myfuncs0.dot(b);

II = GROUP H BY (gr1, gr2);

I  = FOREACH II GENERATE FLATTEN(group) AS (gr1:(type:chararray, n:int), gr2:(type:chararray, 1

J = FOREACH I GENERATE FLATTEN(gr1) AS (type1:chararray, n1:int), FLATTEN(gr2) AS (type2:chara:

STORE J INTO '$OUT' USING PigStorage('\t');
```

In [24]:
```
## On veut connaître les paths des inputs des produits scalaires de SSArr

left,right = SSLR_id(SDir,m,i)
print(left)
print(right)
```

```
$CONTAINER/$PSEUDO/Data/S/S-{-4,-3,-2,-1,0}
$CONTAINER/$PSEUDO/Data/S/S-0
```

In [25]:
```
client.pig_submit(bs,client.account_name,"dotProds.pig",
                  dependencies = ["jython.py","jython0.py"],
```

```
                     params=dict(LEFT = left,
                                 RIGHT = right,
                                 OUT = '$CONTAINER/$PSEUDO/Data/Tmp/DotProd/SS'
                                ),
                     stop_on_failure=True)
```

Out[25]: {'id': 'job_1451961118663_6711'}

```
In [46]: st0 = %hd_job_status job_1451961118663_6711
         st0["id"], st0["percentComplete"], st0["status"]["jobComplete"]
```

Out[46]: ('job_1451961118663_6711', '100% complete', True)

```
In [31]: ## On veut connaître les paths des inputs des produits scalaires de YYArr

         left,right = YYLR_id(YDir,m,i)
         print(left)
         print(right)
```

```
$CONTAINER/$PSEUDO/Data/Y/Y-{-4,-3,-2,-1,0}
$CONTAINER/$PSEUDO/Data/Y/Y-0
```

```
In [33]: client.pig_submit(bs,client.account_name,"dotProds.pig",
                     dependencies = ["jython.py","jython0.py"],
                     params=dict(LEFT = left,
                                 RIGHT = right,
                                 OUT = '$CONTAINER/$PSEUDO/Data/Tmp/DotProd/YY'
                                ),
                     stop_on_failure=True)
```

Out[33]: {'id': 'job_1451961118663_6715'}

```
In [53]: st0 = %hd_job_status job_1451961118663_6715
         st0["id"], st0["percentComplete"], st0["status"]["jobComplete"]
```

Out[53]: ('job_1451961118663_6715', '100% complete', True)

```
In [133]: ## On veut connaître les paths des inputs des produits scalaires de SYArr

          left1,right1,left2,right2 = SYLR_id(SDir,YDir,m,i)
          print(left1)
          print(right1)
          print(left2)
          print(right2)
```

```
$CONTAINER/$PSEUDO/Data/S/S-{-4,-3,-2,-1}
$CONTAINER/$PSEUDO/Data/Y/Y-0
$CONTAINER/$PSEUDO/Data/Y/S-0
$CONTAINER/$PSEUDO/Data/Y/Y-{-4,-3,-2,-1,0}
```

```
In [90]: client.pig_submit(bs,client.account_name,"dotProds.pig",
                     dependencies = ["jython.py","jython0.py"],
                     params=dict(LEFT = left1,
                                 RIGHT = right1,
                                 OUT = '$CONTAINER/$PSEUDO/Data/Tmp/DotProd/SY_001'
                                ),
                     stop_on_failure=True)
```

```
Out[90]: {'id': 'job_1451961118663_6743'}

In [108]: st0 = %hd_job_status job_1451961118663_6743
          st0["id"], st0["percentComplete"], st0["status"]["jobComplete"]

Out[108]: ('job_1451961118663_6743', '100% complete', True)

In [94]: client.pig_submit(bs,client.account_name,"dotProds.pig",
                          dependencies = ["jython.py","jython0.py"],
                          params=dict(LEFT = left2,
                                      RIGHT = right2,
                                      OUT = '$CONTAINER/$PSEUDO/Data/Tmp/DotProd/SY_002'
                                     ),
                          stop_on_failure=True)

Out[94]: {'id': 'job_1451961118663_6745'}

In [112]: ## Seul "fail" du code : ce job ne parvient pas à s'exécuter, on ne sait pas pourquoi
          st0 = %hd_job_status job_1451961118663_6745
          st0["id"], st0["percentComplete"], st0["status"]["jobComplete"]

Out[112]: ('job_1451961118663_6745', '0% complete', True)

In [41]: ## On veut connaître les paths des inputs des produits scalaires de GArr

         left, right =  GLR_id(i)
         print(left)
         print(right)

$CONTAINER/$PSEUDO/Data/{G,S,Y}/{G,S,Y}-0
$CONTAINER/$PSEUDO/Data/G/G-0

In [42]: client.pig_submit(bs,client.account_name,"dotProds.pig",
                          dependencies = ["jython.py","jython0.py"],
                          params=dict(LEFT = left,
                                      RIGHT = right,
                                      OUT = '$CONTAINER/$PSEUDO/Data/Tmp/DotProd/G'
                                     ),
                          stop_on_failure=True)

Out[42]: {'id': 'job_1451961118663_6724'}

In [134]: st0 = %hd_job_status job_1451961118663_6724
          st0["id"], st0["percentComplete"], st0["status"]["jobComplete"]

Out[134]: ('job_1451961118663_6724', '100% complete', True)

In [47]: #### On lance ensuite les tableaux

         import sys
         import csv

         # initialise an array with nrow and ncol
         def initArr(nrow,ncol):
             r = list()
             for i in range(nrow):
                 inner = list()
```

```
        for j in range(ncol):
            inner.append(0.0)
        r.append(inner)
    return r

# initialize the arrays for <S,S>, <Y,Y>, ...
def initArrs(nrow,nrowG,ncol):
    SSArr = initArr(nrow,ncol)
    YYArr = initArr(nrow,ncol)
    SYArr = initArr(nrow,ncol)
    GArr  = initArr(nrowG,ncol)
    return [SSArr,YYArr,SYArr,GArr]

# Permet de stocker les produits scalaires dans les tableaux désignés
def storeDotProds(filePath,m,type,arr):
    f = csv.reader(open(filePath, 'r'), delimiter='\t')
    if type == 'G':
        for l in f:
            i = int(0+1*(l[0]=='S')+2*(l[0]=='Y'))
            j = int(l[3])%m
            arr[i][j] = float(l[4])
    else:
        for l in f:
            i = int(l[1])%m
            j = int(l[3])%m
            arr[i][j] = float(l[4])
    return arr
```

In [61]:
```
## Initialisation des tableaux

SSArr,YYArr,SYArr,GArr = initArrs(m,3,m)
```

In [62]:
```
## Téléchargement des fichiers SS du cluster

%blob_downmerge /$PSEUDO/Data/Tmp/DotProd/SS SS_part-r-00000.csv --overwrite
f = "SS_part-r-00000.csv"
X = pandas.read_csv(f,header = None, sep = "\t", engine = "python")
f1 = csv.reader(open(f,'r'), delimiter = '\t')
SSArr = storeDotProds(f,m,'SS',SSArr)
```

In [63]:
```
SSArr
```

Out[63]:
```
[[82.58847528505197, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0]]
```

In [64]:
```
## Téléchargement des fichiers YY du cluster

%blob_downmerge /$PSEUDO/Data/Tmp/DotProd/YY YY_part-r-00000.csv --overwrite
f = "YY_part-r-00000.csv"
X = pandas.read_csv(f,header = None, sep = "\t", engine = "python")
f1 = csv.reader(open(f,'r'), delimiter = '\t')
YYArr = storeDotProds(f,m,'YY',YYArr)
```

```
In [65]: YYArr

Out[65]: [[1008.2175113083107, 0.0, 0.0, 0.0, 0.0],
          [0.0, 0.0, 0.0, 0.0, 0.0],
          [0.0, 0.0, 0.0, 0.0, 0.0],
          [0.0, 0.0, 0.0, 0.0, 0.0],
          [0.0, 0.0, 0.0, 0.0, 0.0]]

In [116]: ## Téléchargement des fichiers SY du cluster

          %blob_downmerge /$PSEUDO/Data/Tmp/DotProd/SY_001 SY001_part-r-00000.csv --overwrite
          f = "SY001_part-r-00000.csv"
          X = pandas.read_csv(f,header = None, sep = "\t", engine = "python")
          f1 = csv.reader(open(f,'r'), delimiter = '\t')
          SYArr = storeDotProds(f,m,'SY',SYArr)

In [117]: SYArr

Out[117]: [[-115.17129360983034, 0.0, 0.0, 0.0, 0.0],
           [0.0, 0.0, 0.0, 0.0, 0.0],
           [0.0, 0.0, 0.0, 0.0, 0.0],
           [0.0, 0.0, 0.0, 0.0, 0.0],
           [0.0, 0.0, 0.0, 0.0, 0.0]]

In [72]: ## Téléchargement des fichiers G du cluster

         %blob_downmerge /$PSEUDO/Data/Tmp/DotProd/G G_part-r-00000.csv --overwrite
         f = "G_part-r-00000.csv"
         X = pandas.read_csv(f,header = None, sep = "\t", engine = "python")
         f1 = csv.reader(open(f,'r'), delimiter = '\t')
         GArr = storeDotProds(f,m,'G',GArr)

In [73]: GArr

Out[73]: [[1008.2175113083107, 0.0, 0.0, 0.0, 0.0],
          [-115.17129360983034, 0.0, 0.0, 0.0, 0.0],
          [1008.2175113083107, 0.0, 0.0, 0.0, 0.0]]

In [76]: ## Fonctions d'implémentation du vlbfgs

         # dot products array are triangular
         # convert (i,j) to (min(i,j),max(i,j))
         # actually, not so simple... see report for explanations
         def convertTriMat(i,j,m,k):
             r  = k-min(k,m-1)
             i1 = (i-r)%m
             j1 = (j-r)%m
             i2 = min(i1,j1)
             j2 = max(i1,j1)
             i3 = (i2+r)%m
             j3 = (j2+r)%m
             return (i3,j3)

         # implement deltas (see paper for notation)
         # follow the algorithm given in the paper
         def vlbfgsDeltas(SSArr,YYArr,SYArr,GArr,m,k):
```

```
    mk = min(m,k+1)
    km = k%m

    deltaB = list()
    deltaC = list()
    deltaD = -1.0
    alpha = list()
    for i in range(mk):
        deltaB.append(0.0)
        deltaC.append(0.0)
        alpha.append(0.0)


    for j in range(mk):
        jm = (k-j)%m
        alpha[jm] = 0.0
        for u in range(mk):
            um = (k-u)%m
            c = convertTriMat(um,jm,m,k)
            alpha[jm] += SSArr[c[0]][c[1]] * deltaB[um]
        for u in range(mk):
            um = (k-u)%m
            alpha[jm] += SYArr[jm][um] * deltaC[um]
            ### alpha[jm] += SYArr[um][jm] * deltaC[um]
        alpha[jm] += GArr[1][jm] * deltaD
        alpha[jm] /= SYArr[jm][jm]
        deltaC[jm] -= alpha[jm]

    for i in range(mk):
        deltaB[i] *= SYArr[km][km] / YYArr[km][km]
        deltaC[i] *= SYArr[km][km] / YYArr[km][km]
    deltaD *= SYArr[km][km] / YYArr[km][km]

    for i in range(mk):
        j = mk-1-i
        jm = (k-j)%m
        beta = 0
        for u in range(mk):
            um = (k-u)%m
            beta += SYArr[um][jm] * deltaB[um]
            ### beta += SYArr[jm][um] * deltaB[um]
        for u in range(mk):
            um = (k-u)%m
            c = convertTriMat(um,jm,m,k)
            beta += YYArr[c[0]][c[1]] * deltaC[um]
        beta += GArr[2][jm] * deltaD
        beta /= SYArr[jm][jm]
        deltaB[jm] += alpha[jm] - beta

    return {'deltaS': deltaB, 'deltaY':deltaC, 'deltaG':deltaD}
```

In [118]:
```
# Utilisation de l'algorithme pour calculer les deltas optimaux
deltas = vlbfgsDeltas(SSArr,YYArr,SYArr,GArr,m,i)
```

In [137]: deltas

Out[137]:
```
{'deltaG': 0.11423258604225058,
 'deltaS': [-1.0, 0.0, 0.0, 0.0, 0.0],
 'deltaY': [-0.11423258604225058, 0.0, 0.0, 0.0, 0.0]}
```

In [120]:
```python
i=0
```

In [122]:
```python
## On créer le fichier avec le deltas pour ensuite l'uploader sur le cluster

deltaS = deltas['deltaS']
deltaY = deltas['deltaY']
deltaG = deltas['deltaG']

for j in range(max(0,m-len(deltaS))):
    deltaS.append(0.0)
for j in range(max(0,m-len(deltaY))):
    deltaY.append(0.0)

deltasFile = 'deltas'

with open(deltasFile,"w") as file:
    for j in range(len(deltaS)):
        file.write('S\t')
        file.write(str(i-j))
        file.write('\t')
        file.write(str(deltaS[j]))
        file.write('\n')
    for j in range(len(deltaY)):
        file.write('Y\t')
        file.write(str(i-j))
        file.write('\t')
        file.write(str(deltaY[j]))
        file.write('\n')
    file.write('G\t')
    file.write(str(i))
    file.write('\t')
    file.write(str(deltaG))
    file.close()

Input = '$CONTAINER/$PSEUDO/Data/{S/S-{'
for j in range(i-m+1,i+1):
    if j == i:
        Input = Input + str(j) + '}'
    else:
        Input = Input + str(j) + ','
Input = Input + ',Y/Y-{'
for j in range(i-m+1,i):
    if j == i-1:
        Input = Input + str(j) + '}'
    else:
        Input = Input + str(j) + ','
Input = Input + ',G/G-' + str(i) + '}'
print(Input)
```

```
$CONTAINER/$PSEUDO/Data/{S/S-{-4,-3,-2,-1,0},Y/Y-{-4,-3,-2,-1},G/G-0}
```

```
In [123]: path = "/$PSEUDO/Data/Tmp/Delta"
          %blob_up deltasFile path

Out[123]: '$PSEUDO/Data/Tmp/Delta'

In [124]: %%PIG updateXnS.pig

          A = load '$deltasFile' AS (type:chararray, n:int, delta:double);

          B = load '$Input' AS (type:chararray, n:int, var:int, x:double);

          C = join B by (type, n), A by (type, n);

          D = foreach C generate B::type AS type:chararray, B::n AS n:int, B::var AS var:int, B::x AS x

          E = foreach D generate type, n, var, x*delta AS s:double;

          F =group E by var;

          G = foreach F generate 'S' AS type:chararray, '$n' AS n:int, group AS var:int, 0.1*SUM(E.s) AS

          store G into '$SOutput' using PigStorage('\t');

          H = load '$XInput' AS (type:chararray, n:int, var:int, x:double);

          I = join H by var, G by var;

          J = foreach I generate H::type AS type:int, '$n' AS n:int, H::var AS var:int, H::x + G::x AS :
          dump J;

          store J into '$XOutput' using  PigStorage('\t');

In [125]: ## Soumission du job d'actualisation du X dans l'itération

          client.pig_submit(bs,client.account_name,"updateXnS.pig",
                            params=dict(deltasFile = '$CONTAINER/$PSEUDO/Data/Tmp/Delta',
                                        n = str(i+1),
                                        Input = '$CONTAINER/$PSEUDO/Data/{S/S-{-4,-3,-2,-1,0},Y/Y-{-4,-:
                                        XInput = '$CONTAINER/$PSEUDO/Data/X/X-0',
                                        SOutput = '$CONTAINER/$PSEUDO/Data/S/S-1',
                                        XOutput = '$CONTAINER/$PSEUDO/Data/X/X-1'),
                           stop_on_failure=True)

Out[125]: {'id': 'job_1451961118663_6753'}

In [128]: st0 = %hd_job_status job_1451961118663_6753
          st0["id"], st0["percentComplete"], st0["status"]["jobComplete"]

Out[128]: ('job_1451961118663_6753', '100% complete', True)

In [129]: %blob_ls /$PSEUDO

Out[129]:                                          name  \
          0              tranqqaz/Data/G/G--1/part-r-00000
          1                          tranqqaz/Data/G/G-0
          2                 tranqqaz/Data/G/G-0/_SUCCESS
```

3	tranqqaz/Data/G/G-0/part-m-00000
4	tranqqaz/Data/S/S--1/part-r-00000
5	tranqqaz/Data/S/S--2/part-r-00000
6	tranqqaz/Data/S/S--3/part-r-00000
7	tranqqaz/Data/S/S--4/part-r-00000
8	tranqqaz/Data/S/S--5/part-r-00000
9	tranqqaz/Data/S/S-0/part-r-00000
10	tranqqaz/Data/S/S-1
11	tranqqaz/Data/S/S-1/_SUCCESS
12	tranqqaz/Data/S/S-1/part-r-00000
13	tranqqaz/Data/Tmp
14	tranqqaz/Data/Tmp/Delta
15	tranqqaz/Data/Tmp/DotProd
16	tranqqaz/Data/Tmp/DotProd/G
17	tranqqaz/Data/Tmp/DotProd/G/_SUCCESS
18	tranqqaz/Data/Tmp/DotProd/G/part-r-00000
19	tranqqaz/Data/Tmp/DotProd/SS
20	tranqqaz/Data/Tmp/DotProd/SS/_SUCCESS
21	tranqqaz/Data/Tmp/DotProd/SS/part-r-00000
22	tranqqaz/Data/Tmp/DotProd/SY1
23	tranqqaz/Data/Tmp/DotProd/SY1/_SUCCESS
24	tranqqaz/Data/Tmp/DotProd/SY1/part-r-00000
25	tranqqaz/Data/Tmp/DotProd/SY_001
26	tranqqaz/Data/Tmp/DotProd/SY_001/_SUCCESS
27	tranqqaz/Data/Tmp/DotProd/SY_001/part-r-00000
28	tranqqaz/Data/Tmp/DotProd/SY_01
29	tranqqaz/Data/Tmp/DotProd/SY_01/_SUCCESS
..	...
33	tranqqaz/Data/Tmp/DotProd/YY/part-r-00000
34	tranqqaz/Data/X/X-0/part-r-00000
35	tranqqaz/Data/X/X-1
36	tranqqaz/Data/X/X-1/_SUCCESS
37	tranqqaz/Data/X/X-1/part-r-00000
38	tranqqaz/Data/Y/Y--1/part-r-00000
39	tranqqaz/Data/Y/Y--2/part-r-00000
40	tranqqaz/Data/Y/Y--3/part-r-00000
41	tranqqaz/Data/Y/Y--4/part-r-00000
42	tranqqaz/Data/Y/Y--5/part-r-00000
43	tranqqaz/Data/Y/Y-0
44	tranqqaz/Data/Y/Y-0/_SUCCESS
45	tranqqaz/Data/Y/Y-0/part-r-00000
46	tranqqaz/scripts/pig/dotProds.pig
47	tranqqaz/scripts/pig/dotProds.pig.log
48	tranqqaz/scripts/pig/dotProds.pig.log/exit
49	tranqqaz/scripts/pig/dotProds.pig.log/stderr
50	tranqqaz/scripts/pig/dotProds.pig.log/stdout
51	tranqqaz/scripts/pig/gradient.pig
52	tranqqaz/scripts/pig/gradient.pig.log
53	tranqqaz/scripts/pig/gradient.pig.log/exit
54	tranqqaz/scripts/pig/gradient.pig.log/stderr
55	tranqqaz/scripts/pig/gradient.pig.log/stdout
56	tranqqaz/scripts/pig/jython.py
57	tranqqaz/scripts/pig/jython0.py
58	tranqqaz/scripts/pig/updateXnS.pig

```
59        tranqqaz/scripts/pig/updateXnS.pig.log
60    tranqqaz/scripts/pig/updateXnS.pig.log/exit
61  tranqqaz/scripts/pig/updateXnS.pig.log/stderr
62  tranqqaz/scripts/pig/updateXnS.pig.log/stdout

                    last_modified              content_type  content_length  \
0   Mon, 01 Feb 2016 03:48:31 GMT  application/octet-stream              60
1   Mon, 01 Feb 2016 03:49:44 GMT                                         0
2   Mon, 01 Feb 2016 03:49:44 GMT  application/octet-stream               0
3   Mon, 01 Feb 2016 03:49:43 GMT  application/octet-stream             129
4   Mon, 01 Feb 2016 03:48:32 GMT  application/octet-stream              60
5   Mon, 01 Feb 2016 03:48:32 GMT  application/octet-stream              60
6   Mon, 01 Feb 2016 03:48:33 GMT  application/octet-stream              60
7   Mon, 01 Feb 2016 03:48:33 GMT  application/octet-stream              60
8   Mon, 01 Feb 2016 03:48:33 GMT  application/octet-stream              60
9   Mon, 01 Feb 2016 03:48:31 GMT  application/octet-stream             128
10  Mon, 01 Feb 2016 04:38:38 GMT                                         0
11  Mon, 01 Feb 2016 04:38:38 GMT  application/octet-stream               0
12  Mon, 01 Feb 2016 04:38:38 GMT  application/octet-stream             133
13  Mon, 01 Feb 2016 03:57:17 GMT  application/octet-stream               0
14  Mon, 01 Feb 2016 04:36:03 GMT  application/octet-stream             139
15  Mon, 01 Feb 2016 03:57:17 GMT  application/octet-stream               0
16  Mon, 01 Feb 2016 04:10:01 GMT                                         0
17  Mon, 01 Feb 2016 04:10:01 GMT  application/octet-stream               0
18  Mon, 01 Feb 2016 04:10:01 GMT  application/octet-stream              82
19  Mon, 01 Feb 2016 03:57:36 GMT                                         0
20  Mon, 01 Feb 2016 03:57:36 GMT  application/octet-stream               0
21  Mon, 01 Feb 2016 03:57:35 GMT  application/octet-stream              78
22  Mon, 01 Feb 2016 04:05:17 GMT                                         0
23  Mon, 01 Feb 2016 04:05:17 GMT  application/octet-stream               0
24  Mon, 01 Feb 2016 04:05:17 GMT  application/octet-stream              52
25  Mon, 01 Feb 2016 04:29:42 GMT                                         0
26  Mon, 01 Feb 2016 04:29:42 GMT  application/octet-stream               0
27  Mon, 01 Feb 2016 04:29:41 GMT  application/octet-stream              80
28  Mon, 01 Feb 2016 04:25:40 GMT                                         0
29  Mon, 01 Feb 2016 04:25:40 GMT  application/octet-stream               0
..                            ...                       ...             ...
33  Mon, 01 Feb 2016 04:01:29 GMT  application/octet-stream              79
34  Mon, 01 Feb 2016 03:48:31 GMT  application/octet-stream             130
35  Mon, 01 Feb 2016 04:42:47 GMT                                         0
36  Mon, 01 Feb 2016 04:42:47 GMT  application/octet-stream               0
37  Mon, 01 Feb 2016 04:42:47 GMT  application/octet-stream             123
38  Mon, 01 Feb 2016 03:48:32 GMT  application/octet-stream              60
39  Mon, 01 Feb 2016 03:48:32 GMT  application/octet-stream              60
40  Mon, 01 Feb 2016 03:48:32 GMT  application/octet-stream              60
41  Mon, 01 Feb 2016 03:48:32 GMT  application/octet-stream              60
42  Mon, 01 Feb 2016 03:48:32 GMT  application/octet-stream              60
43  Mon, 01 Feb 2016 03:50:22 GMT                                         0
44  Mon, 01 Feb 2016 03:50:22 GMT  application/octet-stream               0
45  Mon, 01 Feb 2016 03:50:21 GMT  application/octet-stream             129
46  Mon, 01 Feb 2016 04:25:56 GMT  application/octet-stream            1402
47  Mon, 01 Feb 2016 04:30:58 GMT                                         0
48  Mon, 01 Feb 2016 04:30:58 GMT  application/octet-stream               3
49  Mon, 01 Feb 2016 04:30:48 GMT  application/octet-stream           27373
```

```
50  Mon, 01 Feb 2016 04:30:48 GMT  application/octet-stream               0
51  Mon, 01 Feb 2016 03:48:45 GMT  application/octet-stream             472
52  Mon, 01 Feb 2016 03:50:43 GMT                                          0
53  Mon, 01 Feb 2016 03:50:43 GMT  application/octet-stream               3
54  Mon, 01 Feb 2016 03:50:33 GMT  application/octet-stream           28488
55  Mon, 01 Feb 2016 03:50:33 GMT  application/octet-stream               0
56  Mon, 01 Feb 2016 04:25:56 GMT  application/octet-stream             139
57  Mon, 01 Feb 2016 04:25:57 GMT  application/octet-stream             144
58  Mon, 01 Feb 2016 04:36:47 GMT  application/octet-stream             819
59  Mon, 01 Feb 2016 04:43:10 GMT                                          0
60  Mon, 01 Feb 2016 04:43:11 GMT  application/octet-stream               3
61  Mon, 01 Feb 2016 04:43:00 GMT  application/octet-stream          104506
62  Mon, 01 Feb 2016 04:43:00 GMT  application/octet-stream             138

    blob_type
0   BlockBlob
1   BlockBlob
2   BlockBlob
3   BlockBlob
4   BlockBlob
5   BlockBlob
6   BlockBlob
7   BlockBlob
8   BlockBlob
9   BlockBlob
10  BlockBlob
11  BlockBlob
12  BlockBlob
13  BlockBlob
14  BlockBlob
15  BlockBlob
16  BlockBlob
17  BlockBlob
18  BlockBlob
19  BlockBlob
20  BlockBlob
21  BlockBlob
22  BlockBlob
23  BlockBlob
24  BlockBlob
25  BlockBlob
26  BlockBlob
27  BlockBlob
28  BlockBlob
29  BlockBlob
..        ...
33  BlockBlob
34  BlockBlob
35  BlockBlob
36  BlockBlob
37  BlockBlob
38  BlockBlob
39  BlockBlob
40  BlockBlob
```

```
41  BlockBlob
42  BlockBlob
43  BlockBlob
44  BlockBlob
45  BlockBlob
46  BlockBlob
47  BlockBlob
48  BlockBlob
49  BlockBlob
50  BlockBlob
51  BlockBlob
52  BlockBlob
53  BlockBlob
54  BlockBlob
55  BlockBlob
56  BlockBlob
57  BlockBlob
58  BlockBlob
59  BlockBlob
60  BlockBlob
61  BlockBlob
62  BlockBlob

[63 rows x 5 columns]
```

In [131]:
```
## On regarde ce que X1 donne

%blob_downmerge /$PSEUDO/Data/X/X-1/ X1.csv --overwrite
%head X1.csv
```

Out[131]: <IPython.core.display.HTML object>

References

[1] Weizhu Chen, Zhenghao Wang, Jingren Zhou "Large-scale L-BFGS using MapReduce" 2012.

A Un exemple illustratif de l'indexation relative

```
Evolution de l'array SSArr au cours des itérations
Initialement SSArr est une matrice de taille m*m (m=5) remplie de 0

-----------------------------------------------------
Itération: 0
Arr: SSArr

A la première iteration (iteration 0), l'algorithme vlbfgs nous dit qu'on a besoin de <S_0,S_0>

Donc pig calcule <S_0,S_0> et écrit le resultat dans un fichier temporaire

Ensuite python va lire ce fichier et enregistre cette valeur dans la matrice SSArr

On rappelle que pour i et j qcq, <S_i,S_j> est stocké dans la cellule SSArr(i%m,j%m)
```

```
Donc <S_0,S_0> est stocké dans SSArr(0,0)

[39.6, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]


-----------------------------------------------------
Iteration: 1
Arr: SSArr

A l'itération I=1, l'algo vlbfgs a besoin de <S_1,S_1> <S_0,S_1> <S_1,S_0> et <S_0,S_0>

Comme <S_0,S_0> est déjà calculé à l'itération précédente,
on ne va ni le recalculer ni le déplacer dans la matrice.

Il nous faut donc une formule (fonction de I et de m)  qui nous donne l'emplacement de <S_0,S_0>
à l'itération I=1.

On sait que <S_0,S_0> est a la cellule (0,0) = (I-1,I-1) = (I-1%m,I-1%m).

On sait qu'a l'itération I=0, <S_0,S_0> est l'historical state numero -1
(on regarde une date en arriere I-1 = 0)

Donc la formule générale est (0,0) = (I-1,I-1) = (I-1%m,I-1%m) = (I-k%m,I-k%m)
où k l'indice d'historical state

On a encore besoin de <S_1,S_1> <S_0,S_1> <S_1,S_0>
Comme le produit scalaire est symétrique on sait que SSArr sera symétrique
Donc inutile de calculer <S_1,S_0>
(on ne remplit que la partie triangulaire supérieure de SSArr)

Donc on demande à pig de les calculer et de les stocker dans un fichier temporaire
Python va lire ce fichier et ajouter ces valeurs dans la matrice SSArr

A l'itération I=1, S_1 est l'historical state 0 (car 1 = I-0)
et S_0 est l'historical state -1 ( car 0 = I-1)
Si S_i est l'historical state -i et S_j est l'historical state -j
alors <S_i,S_j> sera ecrit dans SSArr(I-i%m,I-j%m)

On trouve bien la matrice suivante:

[39.6, -4.2, 0.0, 0.0, 0.0]
[0.0, 0.5, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]


-----------------------------------------------------
Iteration: 2
```

```
Arr: SSArr

Idem - Idem - Idem

[39.6, -4.2, -719.5, 0.0, 0.0]
[0.0, 0.5, 77.9, 0.0, 0.0]
[0.0, 0.0, 13320.7, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]

-----------------------------------------------------
Iteration: 3
Arr: SSArr

Idem - Idem - Idem

[39.6, -4.2, -719.5, 2319.8, 0.0]
[0.0, 0.5, 77.9, -251.3, 0.0]
[0.0, 0.0, 13320.7, -42964.5, 0.0]
[0.0, 0.0, 0.0, 138579.7, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0]

-----------------------------------------------------
Iteration: 4
Arr: SSArr

Arrive a l'iteration I=4, on a alors remplit la matrice entierement pour la premiere fois.
Et a l'iteration I=4, l'algo a besoin de <S_i,S_j> pour i,j dans [I-m+1=0;I]=[0;4]
Donc l'algo a besoin de tous les dot products stockés dans cette matrice SSArr.

On rappelle que SSArr est une matrice symétrique,
donc tous les 0 sont virtuellement remplis par la valeur
à l'emplacement symétrique.

[39.6, -4.2, -719.5, 2319.8, 11757.6]
[0.0, 0.5, 77.9, -251.3, -1272.2]
[0.0, 0.0, 13320.7, -42964.5, -217609.7]
[0.0, 0.0, 0.0, 138579.7, 701875.6]
[0.0, 0.0, 0.0, 0.0, 3554938.5]

Qu'est ce qui se passera a l'iteration suivante I=m=5?
L'algo aura besoin de <S_i,S_j> pour i,j dans [I-m+1=0;I]=[1;5]
Donc on voit qu'on aura a calculer tous les dotprod <S_5,S_j> pour j=1...5
Or la matrice est déjà pleine. Où va-t-on stocker les dotprod?
Comme l'algo n'a besoin que de <S_i,S_j> pour i,j dans [I-m+1=0;I]=[1;5]
la solution consiste donc à retirer de la matrice les dotproducts <S_0,S_j> pour j=0....4
qu'on avait calculés à l'itération 4
(on va les écraser (overwrite))

-----------------------------------------------------
Iteration: 5
Arr: SSArr

On va donc appliquer cette solution pour l'iteration I=5
```

Cherchons les emplacements qui correspondent à <S_0,S_j> pour j=0...4
et qu'on a le droit d'écraser car l'algo n'en a pas besoin.

Ce sont les cellules de la premiere ligne et aussi de la premiere colonne car SSArr est symétrique

Donc à l'itération I=5 on peut écrire aux emplacements SSArr(0,j)
pour i=0...4 et SSArr(i,0) pour i=0...4

Cela correspond exactement à écrire
<S_5,S_5> = <S_I,S_I> à l'emplacement SSArr(I-0%m,I-0%m)=SSArr(0,0)

De manière générale, cela correspond à écrire <S_5,S_j> = <S_I,S_I-k> pour j=1...5 (k=0,...4)
aux emplacements SSArr(I-0%m,I-k%m) = SSArr(0,I-k%m) (il s'agit de la premiere colonne de SSArr)

De même on peut vérifier que les emplacements SSArr(I-i%m,I-j%m) pour i,j=0...4 c
contiennent bien <S_I-i,S_I-j>
ie le produit scalaire entre l'historical state -i
et l'historical state -j

[84301934.0, -4.2, -719.5, 2319.8, 11757.6]
[6196.6, 0.5, 77.9, -251.3, -1272.2]
[1059695.7, 0.0, 13320.7, -42964.5, -217609.7]
[-3417965.8, 0.0, 0.0, 138579.7, 701875.6]
[-17311417.8, 0.0, 0.0, 0.0, 3554938.5]

On remarque qu'on a écrasé la première colonne mais pas la première ligne.

Mais ce n'est pas grave: on sait que SSArr est
symétrique donc il n'est pas nécessaire d'écraser la ligne 1.

Cela fait écho aux itérations précédentes : on n'avait pas
écrasé la partie inférieure de SSArr non plus.
(on avait laissé les 0).

La question est de savoir comment python saura,
lorsqu'il faut lire la matrice SSArr,
s'il lit SSArr(i,j) ou SSArr(j,i).

Pour l'iteration 1 par exemple, on avait vu que <S_0,S_1> = <S_1,S_0> était stocké dans SSArr(0,1)
i.e. dans la partie supérieure de la matrice.

Mais pour l'itération 5 on voit que <S_5,S_1> = <S_1,S_5> est stocké dans SSArr(1,0)
c'est-a-dire dans la partie inferieure de la matrice.

On a donc besoin d'une règle pour dire a python s'il faut aller lire SSArr(i,j) ou SSArr(j,i).
Cette règle sera evidemment fonction de i et j.
Elle sera aussi fonction de I et de m.

La solution à cette question est donnee par le code suivant :

Le code prend en input des indices (i,j) et donne en output des (i',j').

Si python veut lire <S_i,S_j> alors il ira lire a l'emplacement SSArr(i',j').
Les deux autres paramètres sont m qui est le nombre de historical states qu'utilise l'algo m=5,
et k est le numero d'iteration (que je notais I plus haut).

L'idée est qu'une solution possible consiste a chaque iteration d'identifier SSArr
par une cellule de reference qui est REF=SSArr(I%m,I%m).

Puis on fait la transformation suivante:
- on déplace la 1e colonne de SSArr à droite de SSArr
c'est a dire que (1|2|3|4|5) devient (2|3|4|5|1).

- on repete jusqu'à ce que REf se retrouve a la dernière colonne de SSArr.

- on fait de meme avec les lignes.

Apres cette transformation, REF se retrouve donc en bas a droite de SSArr.
Les cellules à lire sont donc celles dans la partie triangulaire
supérieure de cette matrice SSArr apres transformation:
donc <S_i,S_j> est dans SSArr_transformed(min(i,j),max(i,j)).

Une fois qu'on a lu le dotproduct qu'on voulait,
on fait la transformation inverse pour remettre SSArr dans l'etat initial.

Evidemment, c'est une perte de temps de faire ces transformations a la matrice SSArr...
Le code ci-dessous retrouve les bons indices sans faire ces transformations.

```
def convertTriMat(i,j,m,k):
    r  = k-min(k,m-1)
    i1 = (i-r)%m
    j1 = (j-r)%m
    i2 = min(i1,j1)
    j2 = max(i1,j1)
    i3 = (i2+r)%m
    j3 = (j2+r)%m
    return (i3,j3)
```

Iteration: 6
Arr: SSArr

On donne la matrice a l'iteration suivante I=6 pour le fun

```
[84301934.0, -12938419.5, -719.5, 2319.8, 11757.6]
[6196.6, 1985755.7, 77.9, -251.3, -1272.2]
[1059695.7, -162638.7, 13320.7, -42964.5, -217609.7]
[-3417965.8, 524581.1, 0.0, 138579.7, 701875.6]
[-17311417.8, 2656894.7, 0.0, 0.0, 3554938.5]
```