Grenoble INP - Ensimag
Projet de spécialité 2012/2013
Contribution à un logiciel libre : Eigen

GL27
BRUN Gauthier - CARRE Nicolas - CECCATO Jean - ZOPPITELLI Pierre

Juin 2013

Table des matières

1 Introduction

Le logiciel libre est une alternative au logiciel propriétaire qui laisse une libérté totale à l'utilisateur, que ce soit pour l'exploitation où la modification du logiciel. Souvent, ces logiciels font appel à leur communauté pour leur dévelloppement, ainsi toute personne qui possède les compétences, peut participer à sa réalisation et à son maintien. Dans ce processus les deux parties sont gagnantes, le logiciel n'arrête pas d'évoluer, le développeur, lui peut librement rajouter ou corriger des fonctionalités et être reconnu - cela n'est bien sur pas la raison principale à sa participation.
Notre projet de spécialité ayant pour but de mettre en application les connaisances obtenues lors de nos deux années à l'ENSIMAG, nous avons choisi de contribuer à un logiciel libre : Eigen. En plus de mobiliser nos connaissances mathématiques et informatiques, être impliqué dans un véritable projet nous a permis de sortir du cadre purement scolaire. Nous avons donc été confrontés à des problèmes qui ne se posent pas pour des projets habituels tels que la communication avec une communauté distante. Nous avons en outre profité de ce projet pour découvrir un nouveau type d'organisation : la méthode scrum, une méthode agile de plus en plus utilisée en entreprise.

2 Eigen

2.1 La bibilothèque

La bibliothèque *Eigen* est implémentée en template c++ permettant une grande souplesse d'utilisation :
Les classes conteneurs, telles que les vecteurs, les tableaux, les matrices... peuvent être utilisées pour contenir n'importe quel type de coefficients. Cela fournit un moyen simple d'appliquer des algorithmes indépendamment des types de données.
Se référer à l'annexe pour utiliser rapidement *Eigen*.

2.2 La communauté

La communauté de la bibliothèque *Eigen* est restreinte en comparaison de celles de *Firefox* et *Git* mais reste très réactive. Cette communauté dispose d'une mailing liste et d'un chat IRC sur lequel sont connectées une vingtaine de personnes en permanence, dont les principaux contributeurs : *Gael Guennebaud* et *Benoit Jacob*. Ce sont ces deux derniers qui ont été nos interlocuteurs privilégiés tout au long du projet.
Privilégiant la lisibilité du code, la fonctionalité et les performances, la communauté n'a pas de règles strictes de *coding style*.

3 Le problème à résoudre

L'augmentation du nombre de pixels dans les images numériques permet d'obtenir des photographies fidèles à la réalité. Ces images voient leur poids (en octet) sensiblement augmenté, ce qui devient un problème lorsqu'on désire stocker, mais surtout partager ces images. Il faut alors les compresser et c'est avec des algorithmes avec pertes (non réversibles) qu'on obtient les meilleurs résultats. La compression avec pertes nécessite de selectioner les infomations les plus pertinentes de l'image pour limiter au maximun la perte d'information.
Ainsi il existe des outils mathématiques permettant de quantifier la valeur des informations parmis un ensemble de données. Entre autres, le procédé d'algèbre linéaire de décomposition en valeurs singulières (ou SVD, de l'anglais : Singular Value Decomposition) d'une matrice permet de trier les données d'un ensemble selon la quantité d'informations que chacune apporte. Les applications de ce procédé s'étendent du traitement du signal aux statistiques, en passant par la météorologie.

3.1 Présentation de la SVD

Soit M une matrice de taille $m * n$ sur K :

$$M = \begin{pmatrix} m_{1,1} & ... & m_{1,n} \\ m_{2,1} & ... & ... \\ ... & ... & ... \\ m_{m,1} & ... & m_{m,n} \end{pmatrix}$$

Il existe alors une factorisation de M sous la forme :

$$M = U * \Sigma * V^*$$

Avec U une matrice unitaire de taille $m * m$ qui contient un ensemble de vecteurs formant une base orthonormale de K^n, dits de sortie.
Σ une matrice diagonale $m * n$ qui contient les valeurs singulières de la matrice M.
V une matrice $n * n$ qui contient un ensemble de vecteurs formant une base orthonormale K^m, dits d'entrée ou d'analyse. V^* est la matrice adjointe de V.
Le problème est de trouver ces trois matrices.

3.2 Problème de la performance :

Il existe actuellement une solution à ce problème dans *Eigen* : un algorithme basé sur celui de Jacobi.
Cet algorithme calcule le produit M^*M^T en utilisant une séquence de *rotation de Jacobi* pour le diagonaliser. Une *rotation de Jacobi* est une matrice rotation $2*2$ qui élimine les termes non diagonaux d'une matrice symétrique $2*2$. L'algorithme passe répétitivement dans la matrice $M * M^t$: il choisit deux couples d'indice (i, j) différents (i ligne et j colonne avec $i < j$), construit des sous matrices avec les intersections de ces colonnes et de ces lignes et applique une rotation de Jacobi pour retirer les termes non diagonaux.
Cependant, une rotation peut influencer les résultats d'une rotation précédente, donc ce processus doit être répété jusqu'à convergence. A convergence la matrice Σ de la SVD a été générée et les vecteurs U et V sont retrouvés en multipliant toutes les matrices de *rotationdeJacobi* ensemble.
La complexité de cet algorithme est en $O(n^3)$.

L'objectif global de ce projet est de participer à *Eigen* en implémentant un algorithme *Divide and Conquer*, plus performant en complexité que celui existant actuellement.

4 Solution proposée

4.1 Organisation

Ce projet a été réalisé en méthode agile *scrum*, en 3 sprints principaux. Nous avions prévu à l'origine de nous organiser de la manière suivante :

1) - du 21 au 23 Mai - Mise en place des outils de travail (dépot public Mercurial), appropriation de la bibliothèque, analyse de l'algorithme existant.
2) - du 3 au 7 Juin - Proposition d'une version fonctionnelle mais non optimisée de notre fonction à la communauté.
3) - du 8 au 12 Juin - Optimisation du code (minimisation des coûts en ressources + multithreading) et échanges avec la communauté.

Au cours des plannifications de sprint, au vu de la difficulté du projet et de notre avancement, nous avons revu nos objectifs à la baisse. Nous avons ainsi réalisé les sprints de la manière suivante :

1) - du 21 au 23 Mai - Mise en place des outils de travail (dépot public Mercurial), appropriation de la bibliothèque, analyse de l'algorithme existant.
2) - du 3 au 7 Juin - Implémentation de la partie Division et Fusion de l'algorithme.
3) - du 8 au 14 Juin - Soumission des parties Division, Fusion et Déflation à la communautée.

4.2 Etude mathématique

Implémentation d'un algorithme *Divide and Conquer* de complexité $O(n^2)$.
Pour étudier l'algorithme, nous avons utilisé les travaux de *Ming Gu* et *Stanley C. Eisensat* (cf : Références)
Cet algorithme se divise en cinq étapes distinctes :

1) Bidiagonalisation
2) Division du problème en problèmes de tailles inférieures
3) Fusion des résultats
4) Déflation de la matrice obtenue
5) Décomposition en valeurs singulières de la matrice simplifiée

4.2.1 Bidiagonalisation

L'algorithme *Divide and Conquer* que nous avons implémenté nécessite une matrice carrée bidiagonale.
Pour l'obtenir, nous avons utilisé la classe *UpperBidiagonalisation* qui avait déjà été implémentée par la communauté à cette fin. Partant d'une matrice $m * n$ avec $m \geq n$ quelconque :

$$M = \begin{pmatrix} m_{1,1} & ... & m_{1,n} \\ m_{2,1} & ... & ... \\ ... & ... & ... \\ m_{m,1} & ... & m_{m,n} \end{pmatrix}$$

La bidiagonalisation permet d'obtenir un matrice de taille $n+1, n$ de la forme :

$$B = \begin{pmatrix} \alpha_1 & & & & & \\ \beta_1 & \alpha_2 & & & & (0) \\ & \cdot & \cdot & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & (0) & & \beta_{N-1} & \alpha_n & \\ & & & & & 0 \end{pmatrix}$$

On a maintenant :

$$M = U_b * B * V_b^*$$

4.2.2 Division & Fusion

A partir de la matrice B on divise le problème en deux sous matrices B_1 et B_2 de taille $\frac{n}{2}, \frac{n}{2}$, pour lesquelles on calcule la SVD récursivement avec notre algorithme. La récursion s'arrête lorsque l'algorithme de Jacobi devient plus rapide pour la taille des sous-matrices.

On fusionne ensuite les résultats.
On a alors :

$$B = U_f * W * V_f^*$$

avec W une matrice de la forme suivante que nous nommons $v - inverse$:

$$W = \begin{pmatrix} \hat{z_1} & & & & \\ \hat{z_2} & \hat{d_2} & & & (0) \\ \cdot & & \cdot & & \\ & & & \cdot & \\ & (0) & & & \cdot \\ \hat{z_n} & & & & \hat{d_n} \end{pmatrix}$$

On a maintenant :

$$M = U_b * U_f * W * V_f^* * V_b^*$$

4.2.3 Déflation

Cette étape permet de modifier la matrice W de tel sorte que les éléments diagonaux $\begin{pmatrix} \hat{d_2} & \hat{d_3} & ... & \hat{d_n} \end{pmatrix}$ soient croissants tout en gardant sa forme en $v - inverse$. De plus, on annule les z_i qui ne respectent pas les conditions de stabilités numériques. On obtient W' :

$$Z = \begin{pmatrix} z_1 & & & & \\ z_2 & d_2 & & & (0) \\ \cdot & & \cdot & & \\ & & & \cdot & \\ & (0) & & & \cdot \\ z_n & & & & d_n \end{pmatrix}$$

Vérifiant :

$$W = U_d * W' * V_d^* \quad \text{et} \quad d_2 < d_3 < ... < d_n$$

On a alors :

$$M = U_b * U_f * U_d * W' * V_d^* * V_f^* * V_b^*$$

4.2.4 Décomposition

On applique une méthode de décomposition en valeurs singulières qui fonctionne en $O(n^2)$ sur les matrices de la forme de W'. Cette méthode consiste à résoudre l'équation :

$$1 + \sum_{k=1}^{n} \frac{z_k^2}{d_k^2 - w^2} = 0$$

Les n solutions de cette équation sont les valeurs singulières de M.
On a donc :

$$W' = U_s * D * V_s^*$$

Avec D la matrice diagonale contenant les valeurs singulières de M. Les calculs de U_s et V_s sont immédiats.
On a finalement :

$$M = U_b * U_f * U_d * U_s * D * V_s^* * V_d^* * V_f^* * V_b^*$$

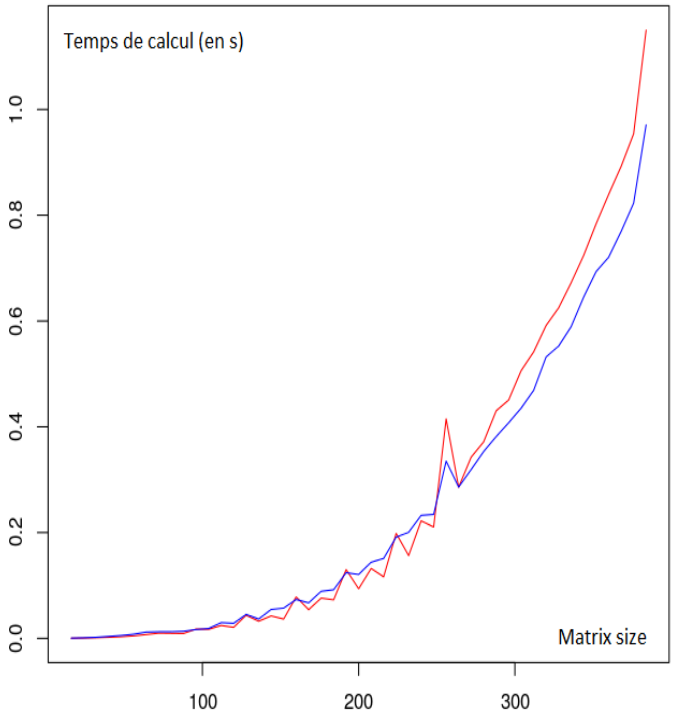
4.3 Réalisation dans Eigen

4.3.1 Implémentation

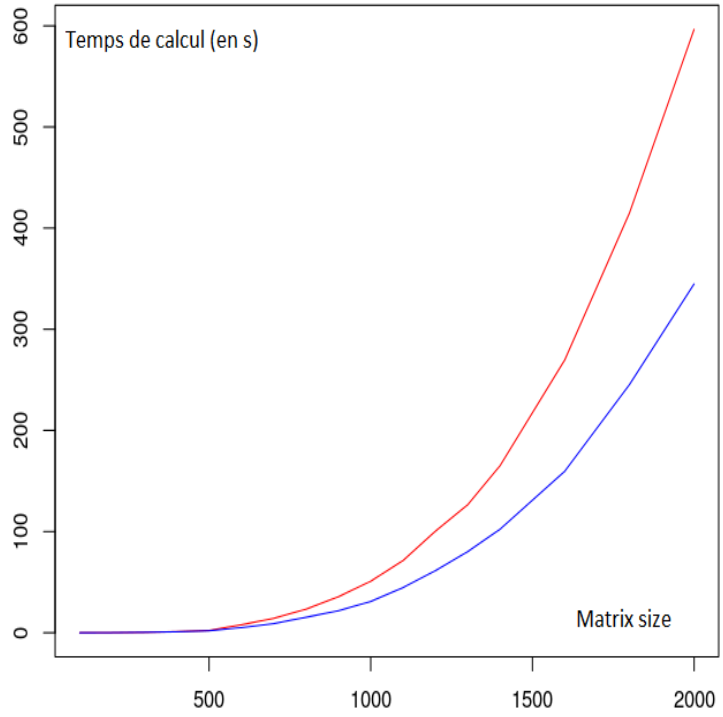
L'implémentation de l'algorithme est réalisée dans une nouvelle classe $BDCSVD$. Ayant de nombreux points communs avec la classe $JacobiSVD$, nous avons restructuré le code en créant une classe mère $SVDBase$ dont héritent les deux autres.

Nous avons implémenté les étapes de division, de fusion et de déflation de l'algorithme, celles-ci fonctionnant sur tous types de matrices sauf celles statiques ou partiellement statiques (Ces matrices sont limitées aux petites tailles ($n < 128$) où la différence de performance est de toute manière faible entre les deux algorithmes). La dernière partie de l'algorithme qui réalise une décomposition en valeurs singulières de la matrice W' n'étant pas implémentée, on utilise pour le moment JacobiSVD.

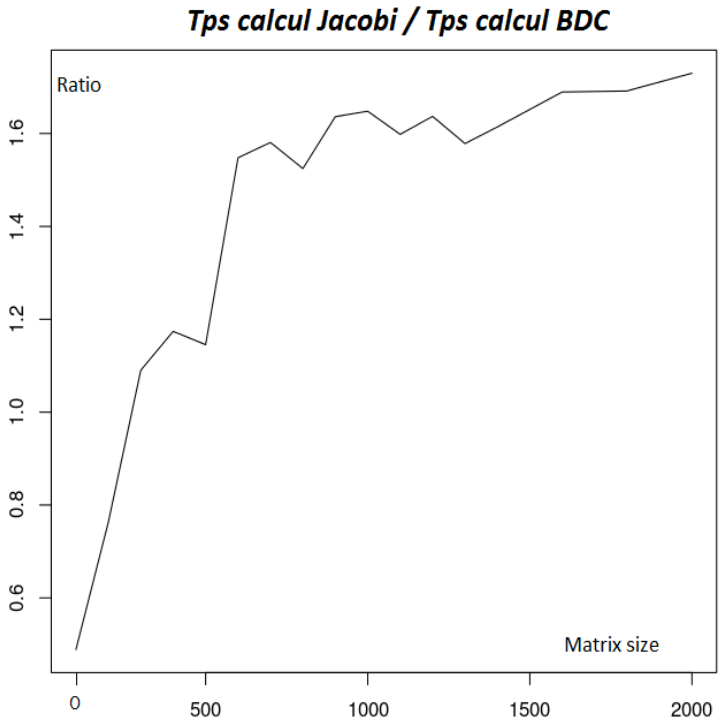
Notre implémentation est structurée, commentée et documentée pour permettre la réutilisation de notre travail, et plus particulièrement pour qu'il soit aisé de terminer l'étape de décomposition de la matrice après déflation.
Pour les matrices dont la taille est supérieure à 250 nous observons un gain de performance (calcul jusqu'à 1.7 fois plus rapide). (cf graphiques (1), (2) et (3) avec en bleu, le temps de calcul avec notre algorithme, en rouge celui de Jacobi)



Graphique (1)



Graphique (2)



Graphique (3)

Donc malgré le fait que nous n'ayons pas pu terminer l'algorithme et donc de ramener à $O(n^2)$ l'ordre de la complexité, nous avons tout de même un gain de performance.
Ceci peut s'expliquer par le fait que la matrice sur laquelle on applique la SVD après déflation contient beaucoup de zéros.

4.3.2 Tests et Benchmark

À l'image des tests existant pour l'algorithme Jacobi, nous avons créé une série de tests intégrés dans la bibliothèque. De nombreuses méthodes étant en commun avec les tests de Jacobi, nous avons dans un premier temps restructuré le code en créant un ensemble de fonction templaté dans *svd_common.h* permettant de factoriser le code de ces deux tests. L'appel à *bdcsvd* permet de lancer une quinzaine de tests, vérifiant la bonne construction, la compatibilité avec les options et l'exactitude des résultats de notre algorithme. Se référer à l'annexe pour lancer les tests de la bibliothèque.
Nous avons de plus créé un benchmark : *bench_svd* permettant de comparer les performances entre les implémentations des deux algorithmes.

4.3.3 Mise en évidence de bug

Nous avons réussi à isoler un bug dans la bibliothèque *Eigen*, à le reproduire précisément et à en informer la communauté. *Gael Guennebaud* a ainsi réussi à corriger rapidement ce bug et envisage d'intégrer cette correction rapidement dans la branche principale.

5 Difficultés rencontrées

5.1 Communication

La participation à un logiciel libre et le fait de communiquer avec une mailing liste complète en particulier a amené plusieurs difficultés : nous ne savions pas vraiment quel protocole suivre lors de nos échanges, et nous éprouvions des srupules à soumettre des éléments incomplets à la communauté. De plus, nous répugniions à poser des questions sur des problèmes que nous aurions pu résoudre par nous même. Cela n'a pas permis à la discussion d'avoir la fluidité espérée.
Cependant sur la fin du projet, nous avons réussi à fuidifier les échanges.

5.2 Mathématique

Malgré la grande difficulté de la partie mathématique du projet, nous avions bon espoir de la terminer. Cependant la dernière étape de notre algorithme était d'une complexité inattendue : à elle seule elle fait l'objet d'une étude théorique d'une vingtaine de pages. C'est ce qui nous a empeché de finaliser notre projet malgré une tentative poussée pour l'implémenter.

5.3 Technique

La bibliothèque étant implémentée en c++ templaté, nous avons eu de grandes difficultés en début de projet pour assimiler l'environnement et produire du code adapté. De plus, le compilateur par défaut que nous avons utilisé ($g++$ 4.7.3) gère très mal les messages d'erreur avec les templates, rendant le debuggage plus difficile.

6 Bilan

6.1 Bilans personnels

Brun Gauthier : Personnelement, le bilan de ce projet est trés positif car il m'a permis de mettre en pratique un grand nombre de mes connaissances informatiques et mathématiques mais surtout j'ai eu l'occasion d'apprendre énormément de choses, qu' elles soient techniques : le templated C++, cmake, gestionaires de version (hg), compilateurs multiples, la décomposition en valeurs singulières ... ou qui ont attrait à la gestion de projet : la méthode scrum, contribuer à un logiciel libre et la communication. Je retiendrai principalement de ce projet, la difficulté que l'on peut avoir à évaluer une tâche lors de la planification du travail, et l'importance de la comunication avec les autre personnes qui sont sur le projet (à l'intérieur ou à l'extérieur de l'équipe).

Carre Nicolas : Ayant déjà connu une bonne expérience avec cette équipe pour le projet GL, j'ai davantage choisi l'équipe que le réel sujet du projet. La partie mathématique m'a semblé extrêmement difficile et obscure, mais j'ai particulièrement apprécié le fait de participer au logiciel libre, d'avoir des retours positifs sur notre travail et de travailler dans un environnement peu contraignant grâce à la méthode agile.

Ceccato Jean : Les bons points seraient l'équipe, très motivée et prète à passer beaucoup de temps pour faire marcher le projet, et la communauté, très encourageante avec toujours des messages sympathiques du style "*Goodjob*, *keepupthehardwork*".
J'ai trouvé en revanche que la partie technique était difficile et les journées de debug m'ont semblé particulièrement pénibles. J'avais parfois l'impression de ne pas progresser dans le projet.

Zoppitelli Pierre : Après avoir passé un très bon projet gl, le choix de l'équipe semblait évident. Pour le sujet, on ne s'est pas vraiment concerté, un membre de l'équipe a proposé de travailler pour Eigen et on lui a fait confiance. La difficulté du projet m'a paru extrêmement élevée, dés le premier

abord, mais la motivation ne m'a jamais quitté, quitte à augmenter considérablement la quantité de travail. Le bilan a été très positif. J'ai apprécié l'experience d'un travail au sein d'une communauté.

6.2 Bilan global

Malgré toutes les difficultés rencontrées, nous avons implémenté la plus grande partie de l'algorithme en s'adaptant à l'architecture et aux contraintes de la bibliothèque. Notre équipe a su répondre avec efficacité aux obstacles, se mobilisant et s'investissant au maximum lors des périodes de crises ou de rush. Nous avons bon espoir que notre travail soit réutilisé par la communauté. Nous tenons à remercier nos encadrants *Mathieu Moy* et *Simon Courtemanches* pour leur soutient tout au long du projet, ainsi que la communauté *Eigen*.

7 Annexe

7.1 Utilisation d'Eigen

Pour utiliser rapidement *Eigen* :

- Pour télécharger le dépôt de notre travail finalisé : cloner notre répertoire bitbucket à l'aide de la commande : *hg clone https* : *//gl*27*@bitbucket.org/gl*27*/eigen − bdcsvd*
- Pour télécharger notre dépôt de travail : *hg clone https* : *//gl*27*@bitbucket.org/gl*27*/eigen*
- Créer un répertoire *build* au même niveau que le dossier du dépôt.
- Dans le répertoire *build* lancer la vérification des dépendances et la construction des Makefile grâce à la commande : *cmake ../Nom_du_depot*
- Lancer la compilation de tous les tests (Attention, prend un temps considérable) : *make build test*
- Lancer la compilation d'un test spécifique : *make Nom_du_Test* typiquement pour nos propres tests : *make bdcsvd* ou *make jacobisvd* pour les tests de non régression.
- Un test spécifique est lui-même divisé en sous-tests numérotés. Ainsi *make bdcsvd_7* lancera la compilation du sous-test 7.
- Les exécutables crés se trouvent dans le répertoire /build/test.
- *make test* dans le répertoire /build lance l'exécution de tous les tests en affichant leur réussite ou échec (même ceux qui n'ont pas été compilés, et qui vont donc échouer). Une méthode plus claire d'interpréter le résultat est de *grep* les tests voulus : *make test | grep svd*.
- Les tests de performance peuvent être lancé grâce aux benchmarks dans le répertoire /bench du dépôt.

Les tests sont implémentés dans le répertoire /test du dépot (/test/bdcsvd.cpp pour notre test), et le code principal des fonctions se trouve dans le répertoire /Eigen/src/ du dépot. Particulièrement, notre algorithme se trouve dans le dossier /Eigen/src/SVD/BDCSVD.h.

8 Références

"A Divide-And-Conquer Algorithm for the Bidiagonal SVD" de *Ming Gu* et *Stanley C. Eisensat* : http ://www.cs.yale.edu/publications/techreports/tr933.pdf