Rapport de projet :
Analyse en composante principale

Chabroulin Sylvain Anoman Don

Janvier 2015

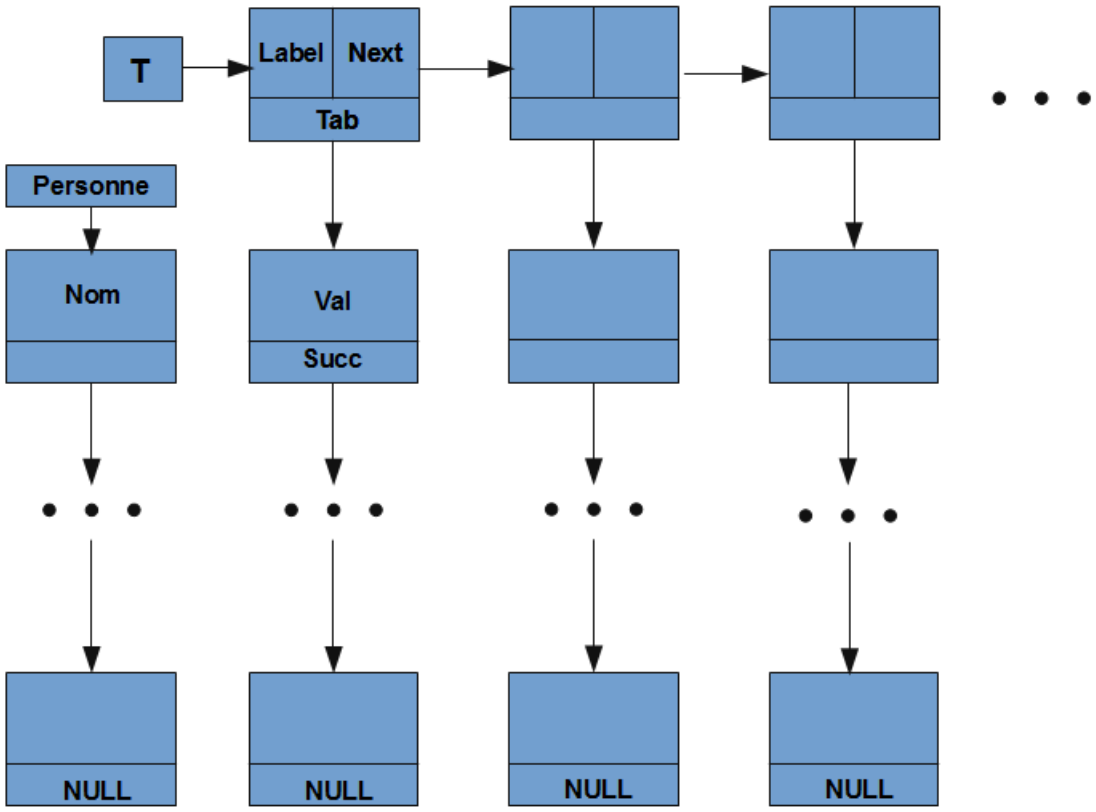
INTRODUCTION

Dans le cadre de ce projet nous vous présentons un programme capable de réaliser la partie mathématique d' une analyse en composante principale (les commentaires sont laissés aux soins de l'expert qui utilise le programme). Une ACP a pour but de determiner, avec en entrée une matrice de donnée, les nouvelles variables qui expliquent au mieux la dispersion de l'ensemble des données (les variables les plus significatives). Celles qu'on trouve sont combinaisons linéaires des variables initiales. En générale , on considère comme plus importantes les deux premières variables qui sont ici les deux 1ères colonnes de la matrice des vecteurs propres de la matrice de correlation. En effet , étant donnée une matrice de donnée M et un vecteur (colonne) U (une combinaison linéaire des variables initiales), pour avoir la dispersion des individus suivant cet axe on calcule : Leur projection sur cet axe M*U , qui donne un vecteur colonne avec, sur chaque ligne, la projection de chaque individu ; on mesure ensuite la variance de ces projections (l'écart par rapport à la moyenne). Il est prouvé que cet écart est plus grand pour un vecteur propre relatif à la plus grande des valeurs propres de la matrice de correlation de M. Et il décroit avec celles ci (si on prend un vecteur propre de la 2eme valeur propre il aura une dispersion moindre que le 1er mais plus importante que le 3ème). L'objectif ici est de programmer l'analyse en composante principale en prenant les données dans un fichier puis en effectuant les traitements neccessaires à la réalisation de l'ACP, d'ajouter ,supprimer, sauvegarder des données dans un fichier binaire et enfin d'avoir une sortie gnuplot. Cette dernière présentera la projection de chaque individu sur les 2 axes les plus importants.

1 Choix de la structure de donnée

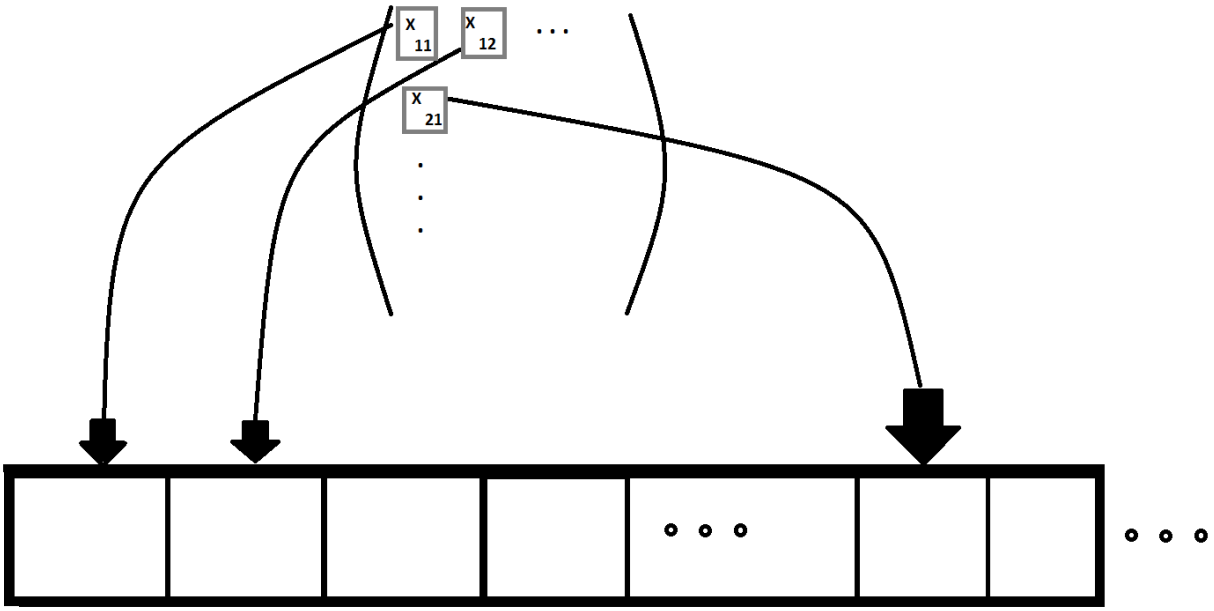
Nous avons choisi d'utiliser deux structures de données différentes pour le programme. La première(structure post-lecture) est utilisée lors de la lecture des données du fichier d'entrée void lecture(FILE* fichier, elemtab** ini, personne** pers).Elle est faite pour faciliter les modifications des données initiales comme l'ajout ou la suppression de ligne ou de colonne de données. Elle se présente de la manière suivante :


elemtab :
Label
Next
Tab
Cellule :
Val
Succ


T
Label
Next
Tab
Personne
Nom
Val
Succ
NULL
NULL
NULL
NULL

Tout d'abord il y a une liste chainée qui sert à sauvegarder les labels de chaque ligne que l'on remplira au fur et à mesure de la lecture. Puis nous avons une seconde liste chainée dont chaque élément contient le label de la colonne si présent, un pointeur sur l'élément suivant et un pointeur sur le début de la liste chainée des valeurs de la colonne qui est créée au fur et à mesure de la lecture. Cela est géré par les structures "elemtab" et "cellule" représenté en début de l'image.

Par la suite, nous avons opté pour une structure de donnée en tableau pour simplifier les calculs typedef struct matrix
int nl,nc ;
float* m ;
matrix ;.

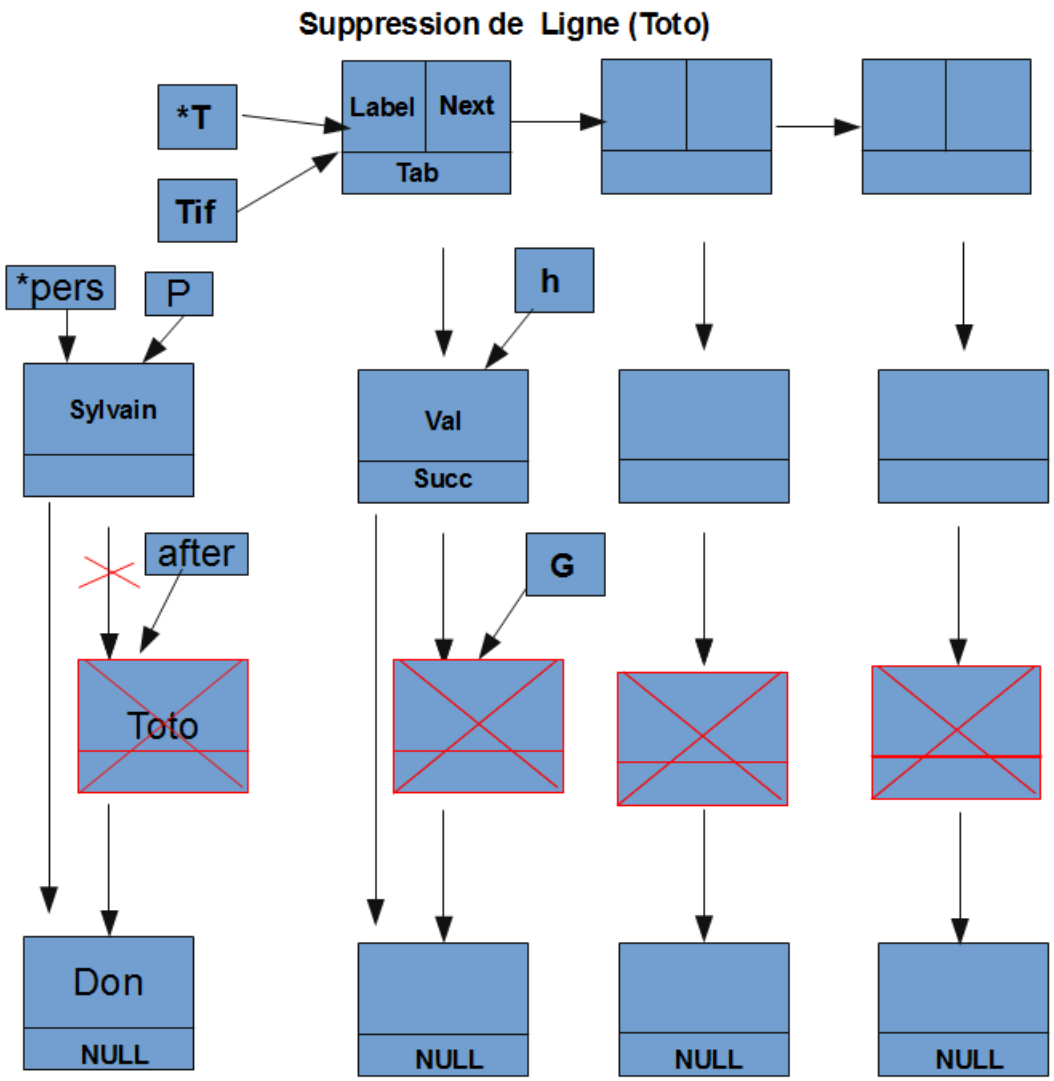


Cette structure contient le nombre de colonnes ainsi que celui de lignes et le tableau de données(1 seule dimension) dont les éléments sont d'abord ceux de la première ligne puis la seconde etc. Une autre raison du choix de cette structure fut notre melleure habileté à manipuler des tableaux ce qui à permis une écriture plus aisée des fonctions tel que la multiplication de matrice ou la diagonalisation de la matrice. Cette structure bien que plus légère , est très male adaptée à la gestion dynamique de donnée. Pour éviter au maximum des pertes en mémoire, nous la désallouons après chaque calcul (on la crée avec *newmat* au début du calcul avec notre 1ère structure).

2 Diagonalisation : Procédé de Jacobi

La matrice de donnée étant créée, nous utilisons succéssivement les fonctions centré reduit ($void\, centrereduit(matrix*)$) correlation ($matrix * correlation(matrix*)$) et la fonction Jacobi ($matrix * Jacobi(matrix*)$) sur le pointeur sur cette matrice de telle sorte à utiliser le moins de mémoire possible. Pour diagonaliser nous utilisont la fonction Jacobi qui fonctionne sur le principe suivant : Sachant que la matrice de correlation est carrée et symétrique , elle admet toujours une diagonalisation à l'aide de matrices orthogonales. Ainsi le but de la méthode de la méthode de Jacobi est, en partant d'une matrice symétrique $A_0 = A$, construire une suite de matrices orthogonales Q_k telles que : $A_{k+1} = Q_{k+1}^T A_k Q_{k+1}$ soit symétrique et converge vers une matrice diagonale D, ayant les mêmes valeurs propres que A. Il faut noter que toutes les matrices A_k ont même valeurs propres que A. On pose $O_k = Q_1 Q_2 ... Q_k$ tels que $A_k = O_k^T A O_k$. L'astuce ici est de construire des Q_{k+1} afin d'annuler un coéficient i,j non diagonal qu'on choisit. La construction de la matrice Q_{k+1} se fait avec les formules utilisées dans le programme. Dans celui ci , l'on ne s'attarde pas sur la multiplication des matrices qui aurait été trops couteuse , mais on modifie directement la matrice ce qui revient moins chère. Des résultats de convergence de cette méthode ont été démontrés et nous constatons que l'exécution de notre programme se fait en temps convenable (quelque centièmes de seconde). A la fin du programme nous retournons la matrice dans le bon ordre de vecteurs propres (de la plus grande valeur propre à la plus petite) grace à la fonction *rangeons* .

3 Modification des données de base



La suppression void supligne(elemtab** T, personne** pers,char* name) ; void supcol(elemtab** T,int num) ; . La suppression de la ligne (illustrée ci dessus) se fait en enlevant le nom demandé (s'il y est) et en supprimant la ligne de données qui lui est relative grace à des $free$. Le même principe est utlisé pour le programme de suppression de colonnes.
L'ajout void ajoutcolonne(elemtab**T) ; void ajoutligne(elemtab**T,personne**pers) ; .Elle se fait au début de la structure et on rajoute la ligne (avec le nom de la personne) ou la colonne en fonction des données déjà présentes (pour une ligne on rajoute exactement le nombre de colonnes présentes) et pour cela on alloue au fur et à mesure la mémoire nécessaire.
Ainsi lorsque l'utilisateur fait le choix de supprimer une ligne ou une colonne , l'on modifie la structure post-lecture et on fait agir le programme *traitement*
$void\ traitement(elemtab * T)$ sur notre structure . Ce programme utilise la structure *matrix* moins gourmande en mémoire et plus maniable pour les calculs. Il la supprime juste après les affichages de telle sorte qu'on ait en mémoire que la structure post - lecture qui est plus adaptée pour l'ajout et la suppression. Cela offre une grande maniabilité au projet dans le sens où il n'utilise que la mémoire qui lui est nécessaire et libère celle qu'il n'utilise plus.

4 GNUPLOT et Difficultés rencontrées

Dans le programme nous créons un fichier script pour Gnuplot qui sert à afficher les points qui ont pour coordonnées les deux premières colonnes de la matrice de donnée initiale dans la base des vecteurs propres . Cela correspond à la projection de chaque individu sur les 2 axes principaux. Le résultat est sauvegardé dans un fichier .png. Plus précisément le programme crée le script gnuplot et un fichier de donnée qui contient les données a afficher dans gnuplot.

Une difficultés rencontrées lors du projet fut la structure de données initiale, en effet celle-ci était à la base plus hybride(entre liste chainée et tableau) mais elle posa problème pour la suite du projet ce qui nous a amené à la structure présente.

Conclusion

Aux termes de ce projet , le prétexte d'étude statistique (ACP) nous a permis de réutiliser les notions éssentielles de programmation que nous avons apprises au cours du semestre. En effet l'utilisation de la programmation dynamique et des structures d'objets nous a permis de reproduire un outil fort utile d'analyse financière. Ainsi ce semestre a été une occasion pour toucher d'un peu plus près l'utilité de nos connaissances.