Interfaçage Web du programme Mixvol, optimisation, implantation du volume stable

Sommaire

INTRODUCTION GENERALE ET ENONCE DU SUJET

INTRODUCTION AU VOLUME MIXTE
- Polyèdre et polytope de Newton
- Subdivision et cellules mixtes
- Définition du volume mixte
- Résultant creux

ALGORITHME DE CALCUL DU VOLUME MIXTE
- Définitions supplémentaires
- Construction d'une subdivision mixte
- Calcul du volume mixte
- VM Contraint

INTERFACE WEB
- Présentation
- Architecture du site
- Page de saisie manuelle des données
- Création et traitement d'un fichier: les choix
- Lecture et traitement d'un fichier
- Gestion des erreurs
- Difficultés rencontrées et bilan

OPTIMISATION DU CODE
- Comment ?
- Quelles possibilités
- Les problèmes
- Bilan

VOLUME STABLE
- Appréhension du Volume Mixte
- Etude du volume stable
- Passage au codage
- Conclusions

REFERENCES

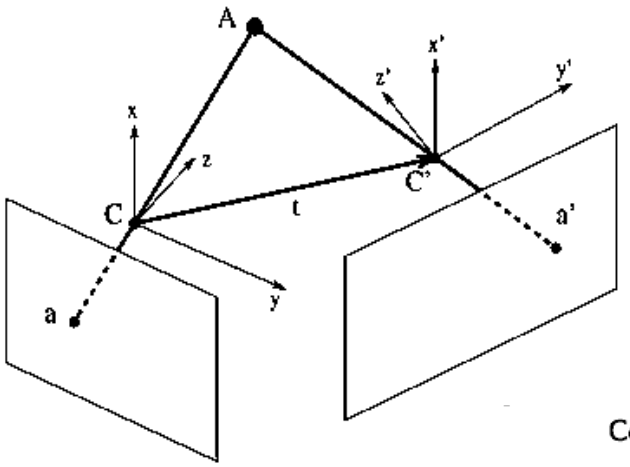
ANNEXE

Le volume mixte sert pour la résolution de systèmes d'équations non linéaires multivariées : il donne une borne fine sur le nombre de racines du système dans $(C–\{0\})^n$. Borne qui permet à certaines méthodes de résolution numériques de gagner en rapidité.

Les algorithmes de calcul du Volume Mixte peuvent donc être utiles à tous les problèmes (partiellement) réductibles à des systèmes d'équations. Nous pouvons donc citer quelques domaines d'application pratiques aussi variées que la biologie moléculaire ou la cinématique.

Pour citer deux exemples concrets :

Mouvement d'une caméra

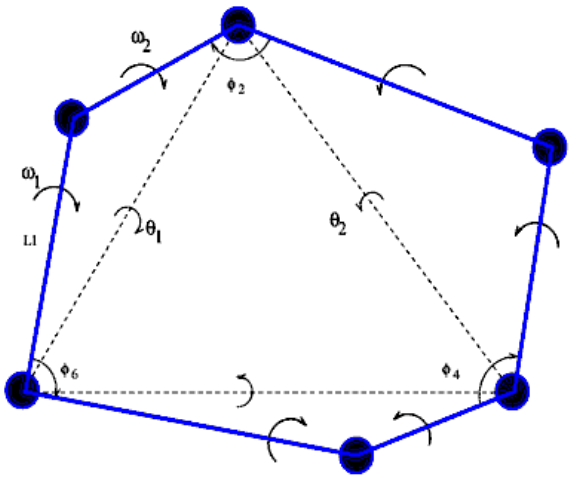


Le déplacement de la caméra à partir de 2 points correspondant dans 2 images est réduit à un système d'équations polynomiales.

Ce dernier est résolu en un temps inférieur à la seconde, pour une erreur inférieure à 10^{-4}.

Conformations moléculaires

Le problème est ici de déterminer les configurations possibles, étant données certaines contraintes d'angle et de longueur.



Une implémentation pour calculer le volume mixte a déjà été développée par Ioannis Emiris. Cette implémentation est très rapide pour les problèmes de petite dimension ; cependant, à partir de la dimension 10, le calcul prend une ou plusieurs heures. De plus, un groupe de chercheurs américains a implémenté une version plus rapide.

Le sujet du projet se compose de 2 parties :

La première, faite par Charles Gaudon consiste à trouver une solution pour optimiser l'implémentation existante de façon à réduire les temps de calcul. De plus, dans le but de faire tester ce programme, il faut créer une interface web pour utiliser celui-ci à distance via Internet. Cette partie comporte donc 2 sous-parties distinctes. L'implémentation existante est en C ; il faut donc pour la première partie du sujet, trouver un programme ou une librairie trouvant la solution à un problème d'optimisation linéaire et dont l'implémentation est optimale.

La seconde partie, faite par Pierre Mario découle du fait que le volume mixte possède une limitation importante : il ne donne d'informations que sur le nombre de racines situées dans $(C–\{0\})^n$. Plus clairement, celles dont aucune coordonnées n'est nulle. Mais il ne peut rien nous apprendre quant aux autres. Cela limite donc ses domaines d'application.

Ont été trouvées deux principales réponses à ce problème :

- La notion de volume mixte généralisé :

Il a été prouvé [4] qu'effectuer l'algorithme du volume mixte sur les ensembles de points auxquels ont aura rajouté l'origine donne une borne du nombre de racines dans $(C)^n$. Son application est donc simple, et la complexité additionnelle par rapport au volume mixte est négligeable.

- La notion de volume stable :

Des travaux récents ont définit le volume stable, et donné des manières de le calculer [5],[6]. C'est une extension du volume mixte permettant non seulement de borner les solutions sur $(C)^n$ de manière plus fine que le VMG, mais surtout de dissocier les racines selon leur type de stabilité. C'est à dire selon la nullité de leurs coordonnées.

Il possède tout de même un inconvénient majeur : une complexité additionnelle non négligeable par rapport au volume mixte.

Le but du travail de Pierre Mario est donc l'implantation d'un programme de calcul du Volume stable à partir des sources de la version du volume mixte développée par M. Emiris.

A noter pour souligner que le domaine est très récent et actif : à ce jour, seulement 3 ou 4 versions finalisées de calcul du volume mixte ont été délivrées par des équipes de chercheurs. Mais aucune qui calcule le volume stable.

INTRODUCTION AU VOLUME MIXTE

Il est important, avant de poursuivre, de définir quelques notions importantes.

1. Polyèdre et polytope de Newton

Un polynôme de Laurent est un polynôme qui admet des puissances entières négatives ; le support d'un tel polynôme est défini comme étant l'ensemble des vecteurs de puissances dont les coefficients sont non nuls.

Un polyèdre est l'intersection finie de demi-espaces fermés. Un polytope est un polyèdre borné. L'enveloppe convexe d'un ensemble E de points inclus dans Z^n est le plus petit ensemble convexe contenant E. Les polytopes de Newton sont les enveloppes convexes des supports des polynômes.

2. Subdivision et cellules mixtes

Soient les polytopes $Q_1,\ldots,Q_n \subset R^n$ dont les sommets appartiennent à Z^n ; une subdivision mixte Δ de la somme de Minkowski $Q=Q_1+\ldots+Q_n$ est un complexe polyédral de cellules dont l'union est Q ; chaque cellule σ de Δ s'exprime de façon unique comme une somme de faces $F_i \subset Q_i$ telle que :

$$\sigma = F_1 + \ldots + F_n, \quad dim\ \sigma = dim F_1 + \ldots + dim F_n$$

On appelle cellules mixtes les cellules σ telles que :

$$\forall i \in \{1,\ldots,n\} \quad dim F_i = 1$$

3. Définition du volume mixte

Le volume mixte $VM(Q_1,\ldots,Q_n)$ des polytopes $Q_1,\ldots,Q_n$ est défini de quatre façons équivalentes :

Définition n°1 : soient $\lambda_1,\ldots,\lambda_n \in R_{\geq 0}$, le volume $V=Vol(\lambda_1Q_1+\ldots+\lambda_nQ_n)$ est un polynôme en λ_i homogène de degré n. Le volume mixte est le coefficient du monôme $\lambda_1\ldots\lambda_n$ dans V.

Définition n°2 :

$$VM(Q_1,\ldots,Q_n) = \sum_{I \subset \{1,\ldots,n\}} (-1)^{n-|I|} Vol\left(\sum_{i \in I} Q_i\right)$$

Définition n°3 : le volume mixte est l'unique fonction à valeur dans N, symétrique et multilinéaire, par rapport à l'addition de Minkowski et à la multiplication par un scalaire, sur n polytopes, en dimension n, ie :

$$VM(Q_1,\ldots,\mu Q_k+\rho Q'_k,\ldots,Q_n)=\mu VM(Q_1,\ldots,Q_k,\ldots,Q_n)$$
$$+\rho VM(Q_1,\ldots,Q'_k,\ldots,Q_n),\quad \mu,\rho\in\mathbb{R}_{\geq 0}$$

et qui satisfait :

$$VM(\underbrace{Q_1,\ldots,Q_1}_{\text{n fois}})=n!\,Vol(Q_1).$$

Définition n°4 : soit Δ une subdivision mixte de la somme $Q=Q_1+\ldots+Q_n$.

$$VM(Q_1,\ldots,Q_n)=\sum_{\sigma\in E}Vol(\sigma),$$

où E est l'ensemble des cellules mixtes de Δ.

4. Résultant creux

Dans cette partie, nous allons voir que le volume mixte, ainsi qu'une subdivision mixte, permettent de calculer le résultant creux au moyen d'une matrice de Newton.

Le résultant, en général, est un polynôme qui exprime la solvabilité d'un système dans un corps algébriquement clos.

$$f_1,\ldots,f_{n+1}\in(\mathbb{Q}[c])[x_1^{\pm 1},\ldots,x_n^{\pm 1}]=(\mathbb{Q}[c])[x^{\pm 1}],$$

tel que $c=(c_{11},\ldots,c_{21},\ldots,c_{(n+1)1},\ldots)$ et tous les coefficients sont non nuls :

$$f_i=\textstyle\sum_j c_{ij}x^{a_{ij}},\ c_{ij}\neq 0.$$

Soient $K=Q[c]$, A_i le support de f_i, Q_i son polytope de Newton et m_i le cardinal de A_i.

$$Z_0=\{c\mid\exists\alpha\in\left(\overline{K^*}\right)^n:f_i(\alpha)=0,\forall i\}\subset\mathbb{P}^{m_1-1}\times\cdots\times\mathbb{P}^{m_{n+1}-1}\qquad Z=\overline{Z_0}$$

son adhérence de Zariski dans $P^{M1-1}{}_x\ldots{}_xP^{Mn+1-1}$

Définition : Le résultant creux est, à un signe près, le polynôme irréductible $R\in Z[C]$ qui définit Z, si codim(Z)=1. Si codim(Z)>1, R=1.
R est homogène en les coefficients c_i de f_i (pour $n+1\geq i\geq 1$) et son degré en les coefficients de f_i est $VM(Q_1,\ldots,Q_{i-1},Q_{i+1},\ldots Q_{n+1})$.

On voit ici le rôle que joue le volume mixte dans le calcul du résultant creux : il permet de connaître le degré du résultant en chaque vecteur c_i de coefficients du polynôme de départ.

Exemple :

$$f_1 = c_{10} + c_{11}y,$$
$$f_2 = c_{20} + c_{21}y^2,$$
$$f_3 = c_{30} + c_{31}x + c_{32}y$$

Ce système a comme volumes mixtes :2,1,0.
Ici Z_0=Z={c :$c_{10}^2c_{21}+c_{11}^2c_{20}$}. En fait, le résultant creux est R= $c_{10}^2c_{21}+c_{11}^2c_{20}$. Le degré du résultant en c_1 est donc bien de 2, celui en c_2 de 1 et celui en c_3 de 0.

Le but serait de construire des matrices dont le déterminant est le résultant R ou, si cela est impossible, un multiple non trivial du résultant. Ces multiples s'appellent des matrices du résultant et constituent des conditions nécessaires pour l'existence de racines communes.

Pour construire une matrice de Newton, on a besoin de supports B_i:
Si $B \subset Z^n$, P(B) = { $g \in K[x^{\pm 1}]$: supp(g) = B} est un espace vectoriel sur K de dimension |B|. Etant donnés $B_1,\ldots,B_{n+1} \subset Z^n$.

$$M : P(B_1) \times \cdots \times P(B_{n+1}) \to P\left(\bigcup_{i=1}^{n+1} A_i + B_i\right) \; : \; (g_1, \ldots, g_{n+1}) \mapsto \left[\sum_{i=1}^{n+1} f_i g_i\right]$$

$$\left[\sum_{i=1}^{n+1} f_i g_i\right] = [g_1, \ldots, g_{n+1}]M$$

Pour identifier les racines communes, M doit être génériquement surjective. La surjectivité de M implique une contrainte sur la dimension des domaines de départ et d'arrivée :

$$\sum_{i=1}^{n+1} \dim P(B_i) \geq \dim P\left(\bigcup_{i=1}^{n+1} A_i + B_i\right)$$

ie $nb_{lignes} \geq nb_{colonnes}$; donc R|D en Z[c], pour chaque mineur maximal D de M.

On peut donc obtenir un multiple du résultant creux. Le problème algorithmique qui se pose maintenant est de définir les supports B_i tels que M soit surjective. Une condition nécessaire est $nb_{lignes} \geq nb_{colonnes}$.

Une subdivision mixte peut être utilisée pour construire la matrice qui permettra de calculer le résultant creux. En effet, il existe un algorithme construisant une matrice dont le déterminant est un multiple du résultant creux. Cet algorithme consiste à construire une subdivision mixte Δ de la somme $Q=Q_1+\ldots+Q_{n+1}$ dans un premier temps. Ensuite de perturber la somme par un vecteur infinitésimal $\delta \in Q^n$. Et de regarder l'ensemble ε des points entiers, où ε est l'intersection entre le polytope Q perturbé $(Q+\delta)$ et Z^n, en définissant une fonction μ :

$$\mu : \varepsilon \to \bigcup_i A_i \; : \; p \mapsto a_{ij} \in A_i$$

si et seulement si $p \in \sigma = F_1+\ldots+a_{ij}+\ldots+F_{n+1}$ où σ est définie telle que $a_{ij} \in A_i$ et $\forall j \neq i$,Fj est une face de Q_j et tel que $\forall j>i$, dim $F_j>0$. On finit par construire une matrice M dont les lignes et les colonnes sont indexées par ε : pour $p,q \in \varepsilon$, l'élément (p,q) est le coefficient de x^q dans $x^{p-a_{ij}}f_i$, où $a_{ij}=\mu(p)$.

En calculant le déterminant de la matrice M ainsi construite, on obtient un multiple du résultant creux. La matrice M dépend fortement de la subdivision que l'on utilise, ainsi que du vecteur δ de perturbation. Et il se peut que l'on obtienne une matrice donnant exactement le résultant.

5. Homotopies creuses

Supposons que les cellules mixtes ont déjà été calculées, que la subdivision mixte a été obtenue grâce à un relèvement $l_1 \ldots L_n$, il ne reste plus qu'à trouver l'homotopie en considérant les polynômes relevés : $\widehat{f}_i(x,t) = \sum_{q \in \Lambda_i} c_{iq} x^q t^{l_i(q)}$

On a alors aussi le système : $\widehat{F}(x,t) = \left\{ \widehat{f}_i(x,t) \ \ pour\ 1 \leq i \leq n \right\}$

Pour définir le système initial de l'homotopie, on a besoin d'une information supplémentaire sur les cellules mixtes. En effet, il nous faut connaître la normale dont la dernière coordonnée vaut 1 pour chaque cellule mixte relevée (on sait déjà que l'ensemble orthogonal à chaque cellule mixte est de dimension 1). Pour trouver ces vecteurs, il suffit de connaître la normale sortante de chaque cellule et de la multiplier par le scalaire qui nous donnera ce que nous voulons. Une normale sortante d'une face d'un polytope est un vecteur normal à la face dont la direction est vers l'extérieur du polytope.

Une fois que l'on a obtenu toutes ces informations, on a donc pour chaque cellule les arêtes de celle-ci et le vecteur $\widehat{\gamma} = (\gamma, 1)$.

Pour chaque cellule mixte, on effectue le changement de variables $x_i = z_i t^{\gamma_i}, \quad \gamma = (\gamma_1, \ldots, \gamma_n)$ dans le système $\widehat{F}(x,t)$ pour obtenir un nouveau système $\widehat{F}(z,t)$ où $z = (z_1, \ldots, z_n)$

On peut diviser toutes les équations par la plus petite puissance de t qui apparaît. On a en fait :

$$\widehat{F}(x,t) = \left\{ \widehat{f}_1(x,t), \ldots, \widehat{f}_n(x,t) \right\}$$

d'où par le changement de variable :

$$\begin{aligned} \widehat{f}_i(z,t) &= \sum_{q \in \Lambda_i} c_{iq} z^q t^{\gamma \cdot q + l_i(q)} \\ &= \sum_{q \in \Lambda_i} c_{iq} z^q t^{(\gamma,1).(q,l_i(q))} \end{aligned}$$

Pour tous les $\widehat{f}_i(z,t)$ la puissance de t, $(\gamma,1).(q,l_i(q))$, vaut 0 si q est un point de la cellule mixte associée à $\widehat{\gamma} = (\gamma,1)$, et est positif sinon. On peut donc résoudre le système $\widehat{F}(z,0)$ en z et trouver ses solutions. On obtient donc le système S :

$$S = \begin{cases} \alpha_1(t) &= \zeta_1 t^{\gamma_1} \\ &\vdots \\ \alpha_n(t) &= \zeta_n t^{\gamma_n} \end{cases}$$

où ζ est solution du système $\widehat{F}(z,0)$, et α est une solution du système $\widehat{F}(x,t)$ pour t proche de 0.

Pour chaque solution que l'on a trouvé, tracer l'homotopie pour t variant de 0 à 1 .

Une fois que l'on a effectué le même travail pour toutes les cellules mixtes, on aura trouvé toutes les solutions du système F.

ALGORITHME DE CALCUL DU VOLUME MIXTE

1. Définitions supplémentaires

Enveloppe inférieure :l'enveloppe inférieure d'un polyèdre $P \subset R^n$ est l'union des facettes (ie les faces de dimension n-1) de P dont la dernière ($n^{ième}$) coordonnée de la normale sortant est négative.

Relèvement :Soient l1,…,ln $\in$ Z[x1,…,xn] des formes linéaires (ou simplement des vecteurs de Z^n ; alors l'évaluation des l_i se réduit à un produit scalaire). On définit un relèvement des polytopes par :

$$\hat{Q}_i = \{(a, l_i(a)) : a \in Q_i\} \subset \mathbf{R}^{n+1}, \quad i = 1, \ldots, n$$

Les $\hat{Q}_i$ sont les polytopes de Newton des $\hat{f}_i$ avec

$$F = \{f_1, \ldots, f_n\}, \ \text{où les } f_i \text{ sont définis par}$$

$$f_i = \sum_{q \in A_i} c_{iq} x^q \ \text{ pour } 1 \leq i \leq n.$$

$$\hat{f}_i(x,t) = \sum_{q \in A_i} c_{iq} x^q t^{l_i(q)}$$

$$\hat{F}(x,t) = \left\{\hat{f}_i(x,t) \ \text{ pour } 1 \leq i \leq n\right\}$$

L'enveloppe inférieure de la somme $\hat{Q} = \hat{Q}_1 + \ldots + \hat{Q}_n$ se projette de manière bijective sur Q. Les facettes de l'enveloppe inférieure correspondent aux cellules maximales de la subdivision induite de la somme $Q_1+\ldots+Q_n$.

2. Construction d'une subdivision mixte

Pour obtenir une subdivision mixte, il suffit de choisir les l_i du relèvement suffisamment générique, ie que les coordonnées des vecteurs l_i seront suffisamment génériques. Afin de vérifier que : si $\{p_1,\ldots,p_n\} \neq \{q_1,\ldots,q_n\}$, $p_i, q_i \subset Q_i$, les sommes relevées ne soient pas toutes les 2 sur l'enveloppe inférieure même si les sommes Σp_i et Σq_i sont égales. La généricité des l_i garantie l'unicité de l'expression pour chaque cellule σ comme une somme de faces F_i, ainsi que la propriété de la somme des dimensions.

On obtient donc le polytope $\hat{Q} = \hat{Q}_1 + \ldots + \hat{Q}_n$. On calcule les normales sortantes de toutes les facettes du polytope $\hat{Q}$. Et finalement, on projette toutes les facettes de l'enveloppe inférieure de $\hat{Q}$. On obtient ainsi une subdivision mixte.

3. Calcul du volume mixte

Entrée : $Q_1,\dots,Q_n \subset R^n$ aux sommets entiers
Sortie :$VM(Q_1,\dots,Q_n) \in Z_{\geq 0}$

1)calculer les ensembles des arêtes $E_1,\dots,E_n$; chaque ensemble E_i contient toutes les arêtes du polytope Q_i.

2)choisir un relèvement $l_1,\dots,l_n \in Z[x_1,\dots,x_n]$ au hasard, en espérant qu'il soit suffisamment générique.

3)appliquer l'algorithme VM-contraint sur $(E_1,\dots,E_n)$.

Par candidat mixte, on veut dire que la somme des arêtes $e_1,\dots,e_k$ est l'enveloppe inférieure de la somme des polytopes $Q_1,\dots,Q_k$, où $e_i \in Q_i$.

4. VM Contraint

Entrée : $(e_1,\dots,e_k,E'_{k+1},\dots,E'_n), \quad 0 \leq k \leq n,\ e_i \in E_i,\ E'_j \subset E_j, j > k$

Sortie : Σ vol(s) sur toute cellule mixte s exprimée (par rapport au relèvement choisi) comme $e_1+\dots+e_k+e'_{k+1}+\dots+e'_n \qquad e'_j \in E'_j, \quad j>k$

On initialise VMc à 0, VMc étant le volume mixte partiel pour une cellule.

1) si k=n et $(e_1,\dots,e_n)$ est candidat-mixte, ie la facette Σe_i a une normale sortante dont la dernière coordonnée est négative. Alors la séquence est mixte et $VMc=VMc+Vol(e_1,\dots,e_n)$.

2) si k=n mais $(e_1,\dots,e_n)$ n'est pas candidat-mixte, on ne fait rien.

3) Si k<n, pour chaque $e'_{k+1} \in E'_{k+1}$ tel que $(e_1,\dots,e_k,e'_{k+1})$ est candidat-mixte, $VMc=VMc+VM\text{-}contraint(e_1,\dots,e_k,e'_{k+1},E'_{k+2},\dots,E'_n)$.

L'algorithme se termine en renvoyant VMc.

Le principe de cet algo est donc de construire tous les ensembles $(e_1,\dots,e_n)$ d'arêtes, avec $e_i \in Q_i$, et d'éliminer les ensembles dont la somme de ses arêtes n'est pas sur l'enveloppe inférieure de $\hat{Q}$. Tout en calculant la surface des facettes se trouvant sur l'enveloppe inférieure. Pour calculer la surface, on calcule le déterminant de la matrice dont les colonnes sont les vecteurs formés par les arêtes de la cellule mixte (un vecteur correspond à une arête de la cellule).

On a également, à l'étape 3, une optimisation qui nous permet d'éliminer très tôt (en cours de construction) certains ensembles.

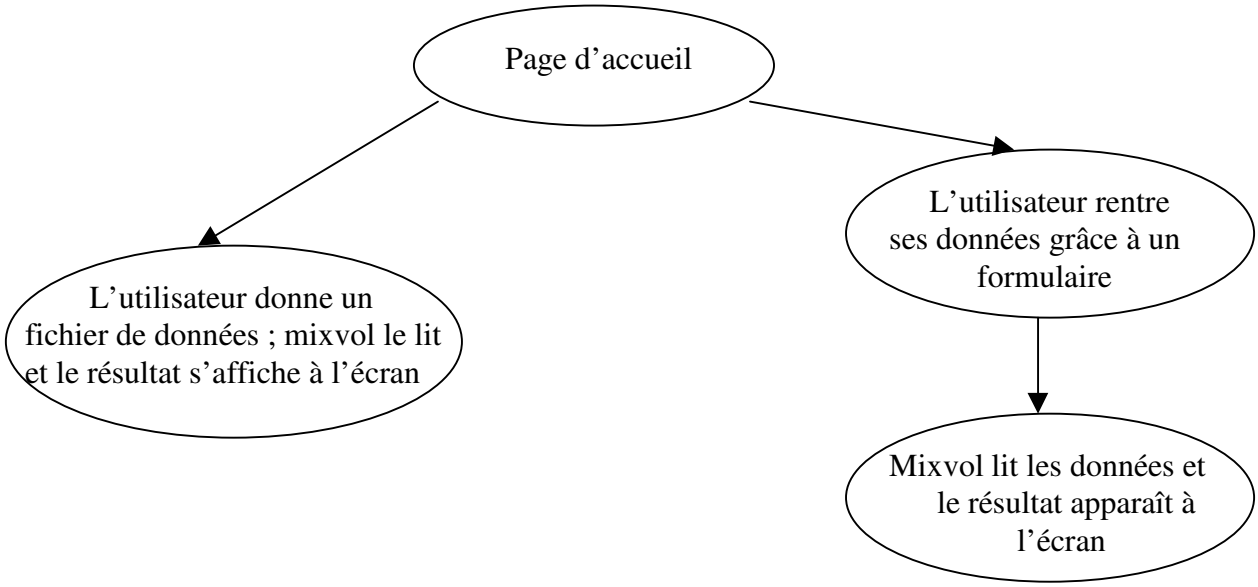
Interface Web

1. Présentation

La seconde partie du projet consiste à faire un site web permettant d'exécuter mixvol via Internet. Le but n'est pas d'en faire le seul moyen d'utilisation, mais plutôt de donner la possibilité aux utilisateurs potentiels de mixvol de pouvoir le tester simplement. De ce fait, la présentation du site se devait d'être claire et son utilisation, la plus simple possible.

2. Architecture du site

Pour la simplicité d'utilisation, le site ne comporte que 4 pages principales ; cependant, chaque page est faite de telle sorte que, dans un soucis de clarté, une partie du travaille soit fait par des fonctions (javascript) à la place de l'utilisateur.



La page d'accueil informe l'utilisateur sur le fonctionnement général du site, et offre quelques liens utiles.

mixed volume

MixVol is a *C* program that computes the mixed volume of *n* sets of points in general dimension *n*.
This web site allows you to test this code with your problems.
We have limited the dimension of the problem to 8 for now, so that the server might not crash. If you want to use it at a higher dimension, you may get the program or contact Ioannis Emiris.

You can test with your files following a specific format here
or you can enter your data manually here

Mixed subdivision of 3 polygons in 2 dimensions (used for constructing sparse resultants): colors show mixedness.

Mixed volume generalizes standard volume to sets of *n* point sets (or their convex hulls) in dimension *n*. By the BKK theorem and sparse elimination theory, mixed volume provides a bound on the number of solutions of a system of *n* polynomials (defined by their support or Newton polytope) in *n* variables. The algorithm and original implementation were proposed in this article by I.Z. Emiris and J.F. Canny. It is called the Lift-Prune algorithm and computes the mixed cells in some mixed subdivision of the Minkowski (vector) sum of the given convex hulls.

mixed volume

Home page

Please fill the windows below.
Each one represents a set of points (3 sets in total):
all the coordinates (3 for each point) of one point must be entered on the same line, with blank space between each coordinate;
type enter between each point.
Each coordinate must be a positive integer

Enter another dimension

Set number 1:

Set number 2:

Set number 3:

Compute

3. La page de saisie manuelle des données

Il ne faut jamais oublier que l'utilisateur du site peut être un novice ; il faut qu'il soit guidé et qu'il ne puisse pas commettre d'erreur. C'est pour cette raison que le javascript est omniprésent sur cette page.

```
function isValid(string,allowed) {
for (var j=0;j<string.length;j++)
{
if(allowed.indexOf
(string.charAt(j))==-1)
            return false;
     }
     return true;
}
```

Cette fonction permet de vérifier si une chaîne de caractère est valide. Sa validité dépend du paramètre « allowed » ; ce paramètre contient les caractères qui sont considérés comme valides.
exemple : `var validdim = '12345678';`
Cet exemple définit les entiers strictement positifs inférieurs à 9 comme valides.

La première chose que doit entrer l'utilisateur est la dimension de son problème. En effet, si le problème est en dimension n, il comportera n ensembles de points en dimension n. Cette information est donc incontournable pour la suite du programme. Ce renseignement est demandé grâce à la syntaxe suivante :

```
while (!isValid(x,validdim)||x==''){
x = prompt("You didn't enter a valid number (no coma,\n no
point and no letter) or you entered too\n high dimension;
enter another one","")
}
```

Ensuite, toujours grâce au javascript, il ne reste qu'à créer suffisamment de zones de texte pour entrer tous les points :

```
for (i=1;i<=x;i++){
document.write("Set number "+i+":<br><textarea name=set"+i+"
rows=6 cols="+10*x+" onKeydown='return handler(event);'>
</textarea><br><br>");
}
```

Cette commande permet de créer « x » zones de texte appelées seti. Ce détail est important dans la suite du déroulement de la saisie.

On peut remarquer qu'une option particulière est utilisée lors de la création des zones : `onKeydown='return handler(event);'`. Cette option permet de scruter les caractères tapés dans ces zone grâce à la fonction suivante :

```
document.onkeydown=handler;
                function handler(e) {
                    if (document.all)   {
                        e = window.event; }
                    var key;
                        if (document.layers)      {
                              key = e.which; }
                 else
                  {
                        if (document.all)
                            key = e.keyCode;
                  }
               if (key == 13 || key==32)
                        return true;
                 if (key==42 || key==43 || key==47 || key==9 || key==109)
                        return false;

               myChar = String.fromCharCode(key);

               if ('1234567890.'.indexOf(myChar) > -1)
                        return true;
                   if (document.all)
                        window.event.cancelBubble=true;
                   return false;
                }
```

De cette façon, automatiquement, directement au moment de la frappe d'une touche, on élimine une partie des sources d'erreur :

- l'utilisateur ne peut rentrer que des nombres
- pas de négatifs (touche 109 bloquée)

Une fois que les coordonnées sont rentrées ; il faut générer les données au bon format que le programme pourra traiter. Pour un problème de dimension 3, il faut générer un fichier de ce type là :

3	dimension
3	nombre d'ensembles (=dimension)
	ligne blanche
2	nombre de points dans le premier ensemble
3	nombre de points dans le second ensemble
3	nombre de points dans le troisième ensemble
	ligne blanche
0 0 0	coordonnées du premier point du premier ensemble
1 1 1	coordonnées du second point du premier ensemble
	ligne blanche
1 0 0	coordonnées du premier point du second ensemble
0 1 0	coordonnées du second point du second ensemble…

Bien évidemment, les premières lignes sont faciles à générer puisque les deux premières données sont identiques et ont été demandées clairement à l'utilisateur. Le problème devient plus complexe pour générer les lignes concernant les nombre de points de chaque ensemble. Il faut compter le nombre de lignes entrées par l'utilisateur dans chaque zone de texte.

```
function card() {
var f;
var y;
var blankLines;
file=''+x+"\n"+x+"\n\n";//génère le début du fichier

       for (i=1;i<=x;i++) {
                  var tmp=eval('document.sets.set'+i+'.value');
                tmp = tmp.split('\r\n').join('##')
                               .split('\n').join('##')
                                       .split('\r').join('##');
                y=tmp.split('##');
                blankLines = 0;
                for (f=0;f<y.length;f++) {
                    if (y[f] = '') {
                          blankLines++;
                    }
                }
                  f=y.length-blankLines;
                  file+=f+"\n";
        }
}
```

Cette fonction compte le nombre de lignes de chaque zone de texte et génère le début des données. On voit ici l'avantage de générer dynamiquement les zones de texte : il est tout aussi facile de les examiner une à une. Cependant, il demeure un problème : si l'utilisateur met une ligne blanche entre deux points d'un même ensemble ou s'il tape « enter » à la fin d'un ensemble, une ligne de plus est

comptabilisée…D'où l'idée de remplacer `tmp = tmp.split…` par `var z = onespace(tmp);z = z.split…` où onespace est une fonction qui élimine tous les espaces, retours, tabulations inutiles ; voilà cette fonction :

```
function onespace(x) {
var inc=0;

     while(x.substring(0,1)== "\n") {inc=1;x = x.substring(1);}
     while(x.substring(0,2)== "\r\n") {inc=1;x = x.substring(2);}
     while(x.substring(0,1)== "\r") {inc=1;x = x.substring(1);}

     while (x.substring(x.length-1,x.length) == "\n") {
      inc=1;
      x = x.substring(0,x.length-1);
      }
     while (x.substring(x.length-1,x.length) == "\r") {
      inc=1;
      x = x.substring(0,x.length-1);
      }
     while (x.substring(x.length-2,x.length) == "\r\n") {
      inc=1;
      x = x.substring(0,x.length-2);
      }

     while (x.substring(0,1) == ' ') {inc=1;x = x.substring(1);}
     while (x.substring(x.length-1,x.length) == ' ') {
      inc=1;
      x = x.substring(0,x.length-1);
      }
     for (k=0;k<x.length-2;k++){
     if (x.substring(k,k+2) == "  "){
         x = x.substring(0,k)+x.substring(k+1,x.length);
       inc=1;
     }
     }

     for (k=0;k<x.length-2;k++){
     if (x.substring(k,k+2) == "\r\r"){
         x = x.substring(0,k)+x.substring(k+1,x.length);
       inc=1;
     }
     }
     for (k=0;k<x.length-2;k++){
     if (x.substring(k,k+2) == "\n\n"){
         x = x.substring(0,k)+x.substring(k+1,x.length);
       inc=1;
     }
     }
     for (k=0;k<x.length-4;k++){
     if (x.substring(k,k+4) == "\r\n\r\n"){
         x = x.substring(0,k)+x.substring(k+2,x.length);
       inc=1;
     }
     }

     for (k=0;k<x.length-2;k++){
     if (x.substring(k,k+2) == " \n"){
         x = x.substring(0,k)+"\n"+x.substring(k+2,x.length);
       inc=1;
     }
     }
```

```
    for (k=0;k<x.length-2;k++){
    if (x.substring(k,k+2) == "\n "){
        x = x.substring(0,k)+"\n"+x.substring(k+2,x.length);
      inc=1;
    }
    }
    for (k=0;k<x.length-3;k++){
    if (x.substring(k,k+3) == " \r\n"){
        x = x.substring(0,k)+"\r\n"+x.substring(k+3,x.length);
      inc=1;
    }
    }
    for (k=0;k<x.length-3;k++){
    if (x.substring(k,k+3) == "\r\n "){
        x = x.substring(0,k)+"\r\n"+x.substring(k+3,x.length);
      inc=1;
    }
    }
    for (k=0;k<x.length-2;k++){
    if (x.substring(k,k+2) == " \r"){
        x = x.substring(0,k)+"\r"+x.substring(k+2,x.length);
      inc=1;
    }
    }
    for (k=0;k<x.length-2;k++){
    if (x.substring(k,k+2) == "\r "){
        x = x.substring(0,k)+"\r"+x.substring(k+2,x.length);
      inc=1;
    }
    }

    for (k=0;k<x.length;k++){
    if (x.substring(k,k+1) == "\t"){
        x = x.substring(0,k)+" "+x.substring(k+1,x.length);
      inc=1;
    }
    }
    if (inc==0) {
    return x;
    }else{
    return onespace(x);
    }
}
```

Cette fonction est récursive pour éviter tout problème ;si la variable inc vaut 1, c'est qu'un caractère inutile a été supprimé : il vaut mieux tout revérifier. De plus, les données générées sont plus soignées. La longueur de cette fonction est due au fait que les retours chariot sont gérés différemment selon la plate-forme(\n sous UNIX, \r sous Windows).

Il ne reste plus ensuite qu'à ajouter les coordonnées des points. En fait, tout se fait lors du submit qui lance une fonction :

```
function group(){
var tmp='';
card();
file+="\n";
for (i=1;i<=x;i++) {
         tmp=eval('document.sets.set'+i+'.value');
         file+=onespace(tmp)+"\n\n";
```

```
}
}
```

4. Création et traitement du fichier : les choix

Le fichier n'est pour l'instant pas créé, seul son contenu est terminé. Il faut maintenant créer un fichier pour le passer en paramètre au programme. Un bon moyen pour ce genre de travail est l'utilisation des CGI ; de plus, quand il sagit d'édition de fichier, le Perl est un langage simple et puissant. Voici la page qui apparaît après avoir rempli le formulaire avec les données d'un problème de dimension 3 :

Here are your data

Home page
dimension of your problem: 3

3
3

2
3
3

1 1 1
0 0 0

1 0 0
0 1 0
0 0 1

1 1 0
1 0 1
0 1 1

---

* Mixed volume computation.
* Accepted 3 point sets in 3 dimensions.

Computing vertices of polytope 1
Found 2 hull vertices
Computing vertices of polytope 2
Found 3 hull vertices
Computing vertices of polytope 3
Found 3 hull vertices

Computing Edges of Polytopes
Polytope 1, found 1 edges
Polytope 2, found 3 edges
Polytope 3, found 3 edges
Seed = 40.

took 0 minutes so far

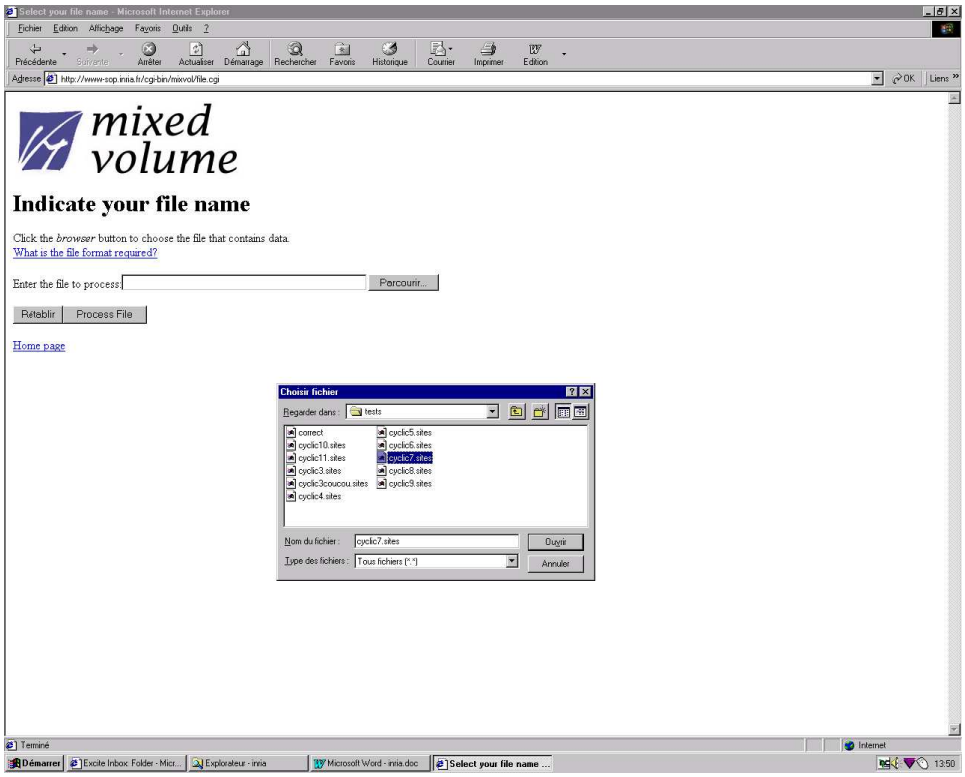
```
* Computing lower hull facets of partial Minkowski sums
found 1-th mixed cell, of vol 3.
found 2-th mixed cell, of vol 3.

Total number of mixed cells = 2.
Final mixed volume = 6.
Finished in time 0h 0m 0s.
```

L'utilisateur peut ainsi connaître le temps d'exécution précis et quelques détails du déroulement du programme.

En fait, au lancement du CGI, celui-ci récupère les données déjà triées du formulaire précédent ; ensuite, il crée un fichier temporaire dans le répertoire /tmp. Ce fichier est passé en paramètre à mixvol. Le CGI récupère tout ce qui arrive sur la sortie standard puis l'affiche. Le fichier temporaire est effacé.
Le code Perl du CGI est visible dans l'annexe.

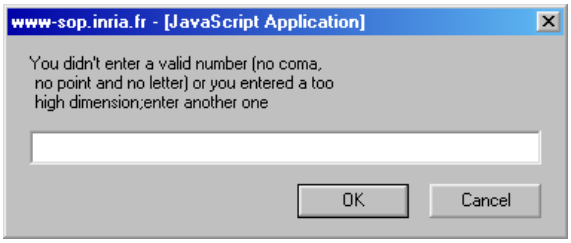
5. Lecture et traitement d'un fichier

La saisi manuelle est intéressante pour les problèmes de petite dimension et pour mémoriser le format des données de mixvol, cependant il n'est pas viable pour les problèmes de dimension plus importante. C'est pour cette raison qu'il y a, sur le site, la possibilité de donner directement a mixvol un fichier déjà prêt. Grâce au CGI présent, l'utilisateur peut sélectionner le fichier sur son ordinateur ; le CGI le lit, le copie sur le serveur et lance mixvol. La suite se déroule comme dans l'étape manuelle.



6. Gestion des erreurs

Dans le cas de la saisie manuelle, beaucoup d'erreurs possibles sont évitées grâce au javascript présent sur cette page mais dans le cas de la lecture d'un fichier, le CGI ne l'examine pas, il ne fait que le copier. C'est pour cette raison que toutes les erreurs de formats, de dimension trop élevée, de mauvais format des coordonnées sont prises en charge par le programme lui-même qui affiche le message d'erreur approprie. Celui-ci s'affiche alors a l'écran.



Si l'utilisateur essaye de rentrer une trop grande dimension lors de la saisie manuelle des données, il lui sera demandé d'en entrer une autre ; si lors de la lecture d'un fichier de l'utilisateur, mixvol trouve que la dimension est >8, c'est lui même qui affichera cette erreur et affichera un lien sur le mail de Ioannis Emiris.

```
* Mixed volume computation.

Reminder: 6 chars assumed per coordinate
Read error at 17-th nonempty line
mixed-volume program ERROR: Bad input: probably invalid format.
```

7. Difficultés rencontrées et bilan

Le serveur web de l'INRIA est Apache ;il gère plusieurs milliers de connections par jour. Pour cette raison, le timeout est configuré pour 5min. Un processus exécuté depuis une page web sur le site de l'INRIA ne peut donc pas excéder 5min. Le problème est que, étant donnée la puissance de ce serveur, au dessus de la dimension 8, mixvol prend plus de 5min pour trouver le résultat. Il a donc fallu installer un serveur apache sur une autre machine que le serveur de l'INRIA. Le problème fut de trouver la machine. Il fut d'abord installé sur un SUN Sparc20 et sur PC avec Linux. Le problème est que ces machines n'étaient pas accessibles depuis l'extérieur et les responsables du réseau n'étaient pas d'accord pour les rendre accessibles pour des raisons de sécurité contre le piratage. L'administration réseau m'a alors conseillé d'installer apache sur une machine expérimentale de type DEC laquelle est accessible depuis l'extérieur. J'ai donc installé un nième serveur apache sur DEC avec DecOS. L'autre problème est que les machines expérimentales étant rares et chères car souvent des machines très puissantes, il était dommage de monopoliser une telle machine pour le site de test. Les administrateurs réseaux ont alors finalement accepté d'héberger le site à condition que les processus n'utilisent pas trop de ressources et que la présence de ce site soit transparente pour les autres utilisateurs du site. Nous avons donc installé une version bridée de mixvol qui ne traite que les problèmes de dimension<9. De plus, étant donné que le site a été développé sous Netscape, il a fallu une longue séance de debuggage pour qu'il fonctionne parfaitement sous IE.

Etant donné que je n'avais jamais codé de CGI, il a fallu passer par une phase d'apprentissage. De plus, j'ai voulu coder en Perl, langage que je ne connaissais pas et qu'il m'a fallu apprendre. Finalement, tous les objectifs ont été atteints : le site est

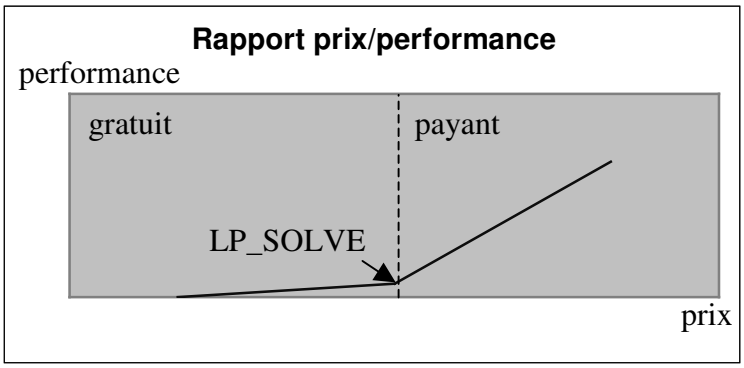
simple, bien expliqué et il fonctionne parfaitement. La fonction de site d'évaluation d'un programme est parfaitement remplie. En outre, il permettra de mettre en relation Ioannis Emiris avec des utilisateurs traitant de gros problèmes, ce qui peut être intéressant pour l'évolution de mixvol.

Optimisation du code

1. Comment ?

Pour calculer une enveloppe convexe, mixvol fait appel à un algorithme de programmation linéaire :simplex. Cet algorithme a été codé, en C à partir de « Numerical Recipes in C ». Cet algorithme est performant, cependant Ioannis Emiris pense que le code qu'il a produit n'est pas le plus performant qui puisse exister. Il faut donc trouver d'autres implémentations de cet algorithme, voir d'autres algorithmes, et les intégrer dans mixvol.

2. Quelles possibilités ?



Nous avons cherché sur Internet des sources en C de programmes de programmation linéaire. Lp_solve a retenu notre attention. C'est un des plus connus et des plus fiables. Après de multiples essais, il n'a pas été possible de l'intégrer dans mixvol. Cependant, en cherchant sur Internet, on s'aperçoit que les sources gratuites sont assez limitées en performance. En fait, les programmes payant, notamment CPLEX de ILOG, sont apparemment les plus performants.

3. Les problèmes

J'ai rencontré beaucoup de problèmes de compilation même lors de la compilation des sources originales non modifiées (erreurs dans des librairies…). J'ai quand même tenté d'intégrer le code dans mixvol et même si la compilation se déroulait sans problème, je n'ai jamais réussi à obtenir des résultats convenables.

4. Bilan

L'intégration de lp_solve dans mixvol n'est pas achevée. Cependant, je ne crois pas à un gain de performance important avec ce programme. Une autre piste qu'il faudrait suivre serait d'utiliser une librairie en Fortran. En effet ce langage se prête plutôt bien à la programmation linéaire. Je ne savais pas qu'il était possible d'appeler du Fortran depuis du C, c'est pour cette raison que je n'ai pas eu le temps de me plonger dans cette voie. J'ai quand même pu trouver une librairie Fortran dont les

auteurs et plusieurs articles font l'éloge. Cette librairie s'appelle SLATEC et se trouve à l'adresse suivante : ftp://netlib2.cs.utk.edu/slatec/ . Cette librairie est une autre implémentation de l'algorithme simplex. Il faudrait réussir à la faire fonctionner avec mixvol pour faire des essais. Son fonctionnement est tout de même assez complexe étant donné le fait qu'elle comporte 10 000 lignes de code et que la fonction à appeler prend 18 paramètres sur sa ligne de commande. Pour l'instant, seul le lien lors de la compilation de mixvol est fait. Enfin, une autre piste serait de générer du C à partir de Maple ; celui-ci possède la fonction de faisabilité de l'algorithme simplex et il est capable de générer un tel code, mais on ne sait rien sur ses performances.

VOLUME STABLE

Afin de pouvoir réaliser le travail imparti, il a fallu tout d'abord étudier la théorie derrière les notions de Volume mixte et stable. Il était évidemment hors de question de chercher à comprendre tous les théorèmes, mais plutôt de réussir à les admettre et s'en servir.

- La première étape du travail a donc été d'atteindre le niveau de compréhension du problème suffisant pour écrire une version personnelle de l'algorithme du volume mixte. Et qui serait suffisamment détaillé pour soulever tous les mécanismes rencontrés lors de l'étude du code existant. Pour ce faire, les publications [1] à [3] ont été utilisées.

- Une assez bonne vision de ce qu'impliquerait le volume stable comme modifications par rapport au VM. Est obtenue grâce aux articles [5], [6].

- A ensuite suivi une phase d'étude des sources durant laquelle furent repérées les différentes étapes de calcul, identifiées les diverses structures de données qu'il faudrait manipuler et repérés les endroits où devraient s'insérer les opérations nécessaires au calcul du volume stable.

- Enfin implantation proprement dite. Elle a compris plusieurs phases :
  - Implantation du calcul du Volume Mixte Généralisé,
  - Création de primitives nécessaires au Volume Stable,
  - Etude sur le choix d'un système de calcul de normale, puis implantation de la solution retenue.

1. Appréhension du Volume Mixte

La base de travail principale était la publication [1] qui donnait une version du 'Lift Prune Algorithm' :

Input: The vertex sets of convex polytopes $Q_1, \ldots, Q_n \subset \mathbb{R}^n$, which are all subsets of the integer lattice $\mathbb{Z}^n$.
Output: $MV(Q_1, \ldots, Q_n) \in \mathbb{Z}_{>0}$.
Lift-Prune algorithm:

1 Enumerate the edges of all polytopes $Q_1, \ldots, Q_n$ respectively into sets $E_1, \ldots, E_n$.
2 Compute random lifting vectors $l_1, \ldots, l_n \in \mathbb{Q}^n$.
3 Compute lifted edge $\widehat{e}_i$ for every edge $e_i \in E_i$, $i = 1, \ldots, n$.
4 Initialize the mixed volume to 0.
5 If $E_1 = \emptyset$ then terminate.
Otherwise, pick any edge $e_1 \in E_1$, remove it from E_1, create current tuple (e_1), let sets $E_2', \ldots, E_n'$ be copies of $E_2, \ldots, E_n$ and let $k = 1$.
6 Let i range from $k+1$ to n:
For every $e_i \in E_i'$, if $\sum_{j=1}^{k} \widehat{e}_j + \widehat{e}_i$ does not lie on the lower envelope of $\sum_{j=1}^{k} \widehat{Q}_j + \widehat{Q}_i$ then e_i is removed from E_i'.
7 Increment k.
8 If $k > n$, then add the volume of the Minkowski sum of $(e_1, \ldots, e_n)$ to the mixed volume; continue at step 5.
9 If $k \leq n$ and $E_k' = \emptyset$ then continue at step 5. If $k \leq n$ and $E_k' \neq \emptyset$ then add some edge $e_k \in E_k'$ to the current tuple $(e_1, \ldots, e_{k-1})$, remove e_k from E_k' and go to step 6.

Le papier mentionnait ultérieurement la remarque suivante *(traduction)* :

« L'algorithme donné ci-dessus n'exploite pas le fait que le volume mixte est invariant selon la permutation des polytopes. Dans notre implantation, nous changeons l'ordre des polytopes, ou plutôt de leur ensemble d'arêtes, de façon dynamique de la manière suivante : quand l'algorithme au pas 9 prend un nouvel ensemble d'arêtes, il choisit celui ayant la plus basse cardinalité. »

Il était nécessaire de transcrire cet algorithme de manière plus informatique. Ci-dessous en sont donc mentionnées les grandes lignes, et en annexe est fournie une note qui entre vraiment dans le détail.

- Initialisation des données : les ensembles E1 à En reçoivent les arêtes des polytopes de Newton associés au système d'équations.

- Relèvement scalaire des arêtes. A chaque ensemble est associé une fonction différente de relèvement aléatoire. Le nombre de coordonnées de chaque point passe de N à N+1. Pour exemple, un point de coordonnée

(x, y, z) devient (x, y, z, ax + by + cz).

- Lancement d'une fonction récursive de calcul du volume mixte.

Son but est de tester toutes les combinaisons d'arêtes susceptibles d'appartenir à une cellule mixte. Elles doivent être composées d'une arête de chaque ensemble, et être sur l'enveloppe inférieure de la Somme de Minkowski du tout.

Elle prend donc en paramètre :
- Les ensembles non encore atteints par la récursion,
- Une combinaison d'arêtes en cours de sélection,
- Le niveau de récursion.

Le premier appel se fait donc ainsi :

```
VolumeMixte(E1..En, NULL, 1)
```

Avec une fonction étant :

```
Fonction VolumeMixte(Ek..En, Tuple, k)
{
     VM=0
Si k >N   // cas terminal, Le tuple a réussi le test.
          Renvoyer SommeMinkowski(Tuple)
Sinon
Si k>1
     Filtrer les Arêtes des Ensembles Ek+1..En  // (*)
FinSi
Intervertir Ek et le plus petit ensemble de Ek+1..En //
l'optimisation mentionnée
TantQue Taille(Ek) >0
     Copier Tuple dans Tuple'      // (**)
     Enlever une Arete de Ek, pour la rajouter dans Tuple'
     Copier Ek+1..En dans E'k+1..E'n   // (**)
VM+=VolumeMixte(E'k+1..E'n,Tuple', k+1)
FinTantQue
Renvoyer VM
FinSi
}
```

(*) L'opération de filtrage est en fait un test qui regarde si la somme de Minkowski du tuple en cours avec chaque arête des ensembles suivants est susceptible de constituer une cellule mixte. Sinon, l'arête testée est supprimée.
Dans le codage, ce test est effectué par de la programmation linéaire.

(**) On peut noter que l'algorithme mentionne des copies de données avant de lancer la récursivité. Il est en effet nécessaire de restituer le même contexte de donnée pour chaque arête d'un même ensemble.

Par contre, au codage, cela se traduit par une manipulation de pointeurs pour une question d'optimisation en temps et en place.

2. Etude du volume stable

L'idée générale de l'extension du VM au VS est de rajouter l'origine aux ensembles de points caractérisant le système, mais en conservant la trace de cette opération ; cela permettra, au cas terminal de la récursion, de déterminer le type de cellule défini par la combinaison d'arêtes sélectionnées. Et d'obtenir les informations recherchées sur les racines qu'elles représentent. L'étude théorique est disponible dans [6]. Une notion importante qu'elle définit est celle de cellule I-stable, avec I un ensemble désignant, en fait, les coordonnées susceptibles d'être nulles des racines associées.

Par rapport a l'algorithme décrit ci-dessus, cela implique principalement les modifications suivantes :

Initialisation :

- Le calcul de l'enveloppe convexe s'effectue sur les ensembles de points originaux. Nous y effectuons ensuite un relèvement cette fois-ci affine.

- C'est seulement ensuite que se rajoute l'origine à tous les ensembles ne la possédant pas déjà. Nous lui définissons un relèvement 'Infini', et créons les arêtes qui découlent de son ajout.

- Nous ne devons conserver comme arêtes rajoutées que celles qui se trouvent sur l'enveloppe inférieure de la figure ainsi créée. Pour ceci, une solution simple : effectuer l'opération de filtrage de la fonction du volume mixte également au niveau de récursion 1.

- Lors du cas terminal, nous nous retrouvons donc avec une combinaison de n arêtes. Certaines connectent une origine rajoutée, d'autres non.

Nous effectuons les opérations suivantes :

- Pour chaque arête, recalculer le relèvement des points qui la composent ainsi :
  - 1 si le point est une origine artificielle,
  - 0 sinon.

- Calculer la normale de cette combinaison en la coordonnée du relèvement. Trois cas distincts :
  - Toutes les coordonnées sont nulles, c'est une cellule mixte.
  - Au moins une de ses coordonnées est négative, c'est une cellule non stable. La combinaison est rejetée.
  - Toutes ses coordonnées sont positives ou nulles, c'est donc une cellule stable.

Pour chaque coordonnée de la cellule nous avons comme information :

- Si elle vaut 0, les racines associées ont valeur non nulle sur cette dimension,
- Si elles sont strictement positives, la racine aura pour valeur de composante 0.

3. Passage au codage

Hygiène de programmation

Le code source du programme se révéla être conséquent et pour le moins optimisé. Pour mener à bien la tache impartie il était indispensable de repérer facilement les endroits où je grefferais du code. Pour ce faire, j'ai donc utilisé quelques conventions :

- Toutes les fonctions, structures et variables globales ajoutées commencent par les 2 lettres SV.
- Tout ajout / modification de code est entouré des commentaires suivants :

```
/* Modifs SV */
   …
/* Fin Modifs SV */
```

- La définition des corps des fonctions additionnelles est faite dans une bibliothèque séparée (LibSV).

Ainsi, cela donne par exemple pour l'opération de l'ajout de l'origine :

```
    …
  hull_edges();          /* Hull edges in AllEdges[0], number
                  AllCards[0] */
  lift();      /* lifts Npolytopes, prints lift-vectors */
/* Modifs SV */
 if (SVValiderModifs.Mode==SV)
    {
      SVAjouterZero(&NPointsTmp,&MaxPointsTmp);
    }
/* fin Modifs SV */
  mv = 0;
    …
```

Fonctions nécessaires au volume stable

Les fonctions qu'il a fallu créer sont :

```
ipoint SVCreerOrigineArtificielle();
char SVEstUneOrigine(ipoint Point);
char SVEstUneOrigineArtificielle(ipoint Point);
char SVContientOrigineArtificielle(ipoint* Ensemble,int
Taille);
void SVCreerAretesOrigineArtificielle(int NumEns);
void SVEstimerNbPoints(int *TotalNumberOfPoints,int
*MaxPointsPerPolytope);
```

```
void SVAjouterZero(int *TotalNumberOfPoints,int
*MaxPointsPerPolytope);
void SVSpecialLifting();
char SVEstMixte(earray tuple, int *inds);
void SVCalculerNormale(earray tuple,int *inds,fptype
*Normale,char Menage);
char SVTypeDeCellule(earray Tuple, int* inds,char
Afficher,char Menage);
```

Les modifications réalisées ont tenté de conserver l'esprit de ce qui existait déjà. Le maximum a été fait pour ajouter plus que transformer. Tant au niveau de l'affichage en sortie que dans les algorithmes en eux-mêmes.

Détermination de la normale

Une partie importante du calcul du volume stable est donc le calcul de la normale selon la coordonnée de relèvement d'une combinaison d'arêtes.

Le problème se réduit à la résolution du problème matriciel :

AX=V, avec :

A : la matrice des coordonnées des arêtes mis en ligne,

V : la négation de la valeur de relèvement spécial associée à chaque arête.

X représentera les n premières coordonnées de la normale, la $n+1^e$ valant 1.

Pour ceci, il était question au départ d'implanter une bibliothèque de calcul. Elle pouvait se baser sur 2 théories :

- la décomposition LU,
- l'arithmétique modulaire, en codant un algorithme proposé dans [7].

Il fallait donc en étudier les avantages et inconvénients pour déterminer le choix d'implantation.

Il a fallu prendre la décision d'implanter une bibliothèque se basant sur la décomposition LU, attendu que :

- Même si l'autre théorie paraissait avoir une moindre complexité, le gain apporté me semblait négligeable devant l'exponentialité du programme principal,
- La création d'une bibliothèque LU m'a paru être beaucoup plus simple que l'autre. Passer peu de temps dessus me permettrait donc éventuellement de changer le cas échéant.
- Mais surtout, cela répondait à un besoin impérieux de pouvoir tester 'en réel' toutes les modifications que j'avais apportées au programme.

Ainsi donc a pu être délivrée une version qui tournait du programme. Qui a fait apparaître un hic de taille provenant du calcul de la normale : sur des problèmes de taille respectable, la stabilité des calculs n'était pas assurée, le volume stable n'était en fait qu'une approximation.

C'est alors qu'a finalement été décidé d'utiliser une bibliothèque de calcul écrite en Fortran : Lapack.
Son intégration toute récente à donné des résultats tout a fait satisfaisants tant au niveau du temps que de la stabilité.

Utilisation

En entrée :

Le programme de calcul du volume stable s'utilise en ligne de commande. On lui passe en paramètres le nom d'un fichier caractérisant les données, éventuellement un fichier de rapport en sortie, et des options de fonctionnement. A donc été rajoutée la prise en compte de deux nouveaux paramètres :

-g : Calcul du volume mixte généralisé,
-G : Calcul du volume stable.

Ainsi, si le programme est lancé sans paramètres, sera affiché comme aide en ligne ceci :

```
mixed-volume program ERROR:
Command syntax is: mixvol [-switch]* site_file [output_file]
     switch:
  -g : computes generalized MV.
  -G : computes stable V.
  -a [Value] : Adds [Value] to every coordinate.
  -c : Checks Cells (Verification).
  -ch : Specifies entries are already convex hulls.
  -mo [Value] : Specifies [Value] as the display limit of the
monitor.
  -s : Suppresses printing of the tested edge tuple.
  -t : Prints all matrices.
  -S [Value] : Sets the random seed to [Value].
```

Affichage des résultats

Voici donc un exemple de fichier de résultats créée par le programme du volume mixte-stable :

```
This is the output on Bernstein.Test.
Stable mixed volume mode.

Seed = 38.
1-th lifting vector: [ 763364 78975 417877 ]
2-th lifting vector: [ 121526 589015 116147 ]

Took 0 minutes for hull vertices, edges

These are the stable/mixed cells of the Minkowski Sum.
each cell = sum of edges with indices in <...>.
```

```
Edge of polytope 1 polytope 2.

Heads [ 0 1 496852 ] [ 1 0 237673 ]
Tails [ 0 2 575827 ] [ 2 0 359199 ]
< 0  0 > Cell volume = 1.
Mixed Cell.

Heads [ 0 1 496852 ] [ 2 0 359199 ]
Tails [ 0 2 575827 ] [ 3 1 1069740 ]
< 0  2 > Cell volume = 1.
Mixed Cell.

Heads [ 0 1 496852 ] [ 1 0 237673 ]
Tails [ 0 2 575827 ] [ 0 0 1418167 ]
< 0  3 > Cell volume = 1.
{ 0 }-Stable Cell.

Heads [ 0 2 575827 ] [ 1 0 237673 ]
Tails [ 1 3 1418166 ] [ 3 1 1069740 ]
< 2  1 > Cell volume = 1.
Mixed Cell.

Heads [ 0 1 496852 ] [ 1 0 237673 ]
Tails [ 0 0 1418167 ] [ 2 0 359199 ]
< 3  0 > Cell volume = 1.
{ 1 }-Stable Cell.

Heads [ 0 1 496852 ] [ 1 0 237673 ]
Tails [ 0 0 1418167 ] [ 0 0 1418167 ]
< 3  3 > Cell volume = 1.
{ 0 1 }-Stable Cell.

Total number of stable/mixed cells = 6
Total sv = 6. Total time = 0h 0m 0s.
```

Par rapport au programme original de calcul du volume mixte, a donc été ajouté le type de chacune des cellules mixtes / stables.

Tests effectués

Pour tester mon programme, je suis donc parti sur deux extrapolations de l'exemple publié dans les papiers parlant du volume stable. Le but étant de le tester sur des données conséquentes. Ces batteries de tests ont été choisies parce qu'il proposent des solutions à coordonnées nulles, et ainsi mettent en valeur les différences entre les différents calculs.

L'exemple original étudiait le système :

$ay + by^2 + cxy^3 = 0$
$ex + bx^2 + cx^3y = 0$

Ce système était donc réduit à deux ensembles de 3 points : {(1,0),(2,0),(3,1)} et {(0,1),(0,2),(1,3)}.

Le calcul du volume stable nous informe qu'il possède, dans le cas générique:
- Une racine nulle,
- Une racine ayant pour valeur en x=0,
- Une racine ayant pour valeur en y=0,
- Trois racines dont aucune coordonnée n'est nulle.

- Première extrapolation :

Intéressons-nous dans un premier temps à la famille des systèmes :

S_n : $a_1x_1 + b_1x_1^2 + c_1x_1^3x_2 \ldots x_n$
$\ldots$
$a_nx_n + b_nx_n^2 + c_1x_1 \ldots x_{n-1}x_n^3$

Ils présentent la particularité d'avoir un volume mixte égal à n+1, et de posséder une racine unique pour chaque ensemble de stabilité possible.

Soit un volume stable de $(n+1)+(2^n -1)$.

- Deuxième extrapolation :

Là nous avons des systèmes d'équations plus conséquents :

S'_n : $a_1x_1 + b_1x_1^2 + c_{1,1}x_1^3x_2 + \ldots + c_{1,n}x_1^3x_n$
$\ldots$
$a_nx_n + b_nx_n^2 + c_{n,1}x_n^3x_1 + \ldots + c_{n,n}x_n^3x_{n-1}$

Les tests ont été effectués sur Nessie, le serveur (solaris) de l'ESSI. Est affiché le temps CPU.

Résultats :

		S'_5	S'_6	S'_7	S'_8
MV	Résultat	123	441	1515	5025
	Temps	0s.	1s.	8s.	36s.
GMV	Résultat	484	2152	9580	42208
	Temps	0s.	1s.	8s.	45s.
SV	Résultat	459	2027	9220	40953
	Temps	2s.	9s.	1m 4s.	6m 15s.

		S_8	S_9	S_{10}	S_{11}
MV	Résultat	9	10	11	12
	Temps	0s.	0s.	0s.	0s.
GMV	Résultat	1280	2816	6144	13312
	Temps	0s.	0s.	0s.	0s.
SV	Résultat	264	521	1034	2059
	Temps	6s.	18s.	0m 55s.	2m 34s.

Enseignements :

- On peut noter que la complexité des algorithmes dépend fortement du nombre de monômes des fonctions à étudier de par la différence de performances sur S et S'.
- Et aussi que le surcoût en temps impliqué par le volume stable est loin d'être négligeable. De par ses initialisations, il pourrait correspondre plutôt à un problème de volume mixte de dimension immédiatement supérieure.

Et ceci nous amène donc à ne pas définitivement 'enterrer' la notion de volume mixte généralisé. En fait, ces trois différents calculs peuvent être utilisables selon les besoins que posent le problème modélisé.

4. Conclusions

Le but du projet a été mené à bien. Il existe maintenant une première version de calcul du volume stable. Effectuant mon stage en continuation à l'Inria, j'aurai pour premier objectif de l'intégrer dans une bibliothèque de calcul pour qu'il soit utilisable. Et aussi de le faire répondre à des besoins ciblés. Comme par exemple lui faire calculer des bornes sur seulement certaines catégories de racines.

D'une manière plus générale, ce projet m'a permis de travailler sur un domaine de recherche, ce qui en soit est très enrichissant. Il présentait de plus un intérêt algorithmique certain. Et il m'a aussi amené à découvrir combien le travail de compréhension de publications scientifiques pouvait être difficile...

REFERENCES

[1] Efficient Incremental Algorithms for the sparse resultant and the mixed volume, Ioannis Z. Emiris & John F. Canny.

[2] Notes creuses sur résultant creux, I. Z. Emiris.

[3] Implantation distribuée du volume mixte, I.Z. Emiris & T. Giordano.

[4] The BKK root count in C^n, T. Y. Li & X. Wang.
Manuscript, 1994.

[5] Bernstein's theorem in affine space, B. Huber & B. Sturmfels.
Discr. And Computational geometry, 1995.

[6] Counting affine roots of polynomial systems, I. Z. Emiris & J. Verschelde.

[7] Computing exact geometric predicates using modular arithmetic with single precision, H. Bronniman, I. Z. Emiris, V. Y. Pan, S. Pion.

Les documents 1, 2, 3 6 & 7 sont disponibles à partir de :
http://www-sop.inria.fr/saga/personnel/Ioannis.Emiris/index.html

ANNEXE

CGI

```
Cgi qui ouvre, copie puis traite un fichier
#!/usr/local/bin/perl5 -w

                use CGI qw(:standard);
                use CGI::Carp;

                print header();
                print start_html("Select your file name");
         print "<body bgcolor='white'>";
         print "<img src='../../saga/demo/mixvol/logo_vm.jpg'
border='0'></img><br>";
                print h1("Indicate your file name"), p,
                   'Click the ',cite('browser'),' button to choose the
file
                   that contains data.';
                print "<br><a href='javascript:var
pop=open("../../saga/demo/mixvol/file_help.html");'>What is the file format
required?</a><br>";
                   print start_multipart_form(),
                   "Enter the file to process:",
                   filefield('filename','',45),
                   br,
                   #checkbox_group('count',@types,@types),
                   p,
                   reset,submit('submit','Process File'),
                   endform;
                print "<a href='../../saga/demo/mixvol/index.html'>Home
page</a><BR>";
                  if ($file = param('filename')) {
                   print hr();
                   $timenow = time;
            open MYFILE, ">/tmp/mixvol$timenow.sites";
                   while (<$file>) {
            print MYFILE;
                   }
            close $file;
                   close MYFILE;
            print "computing in progress, please wait...",br;
            $ result='/u/www/0/www/tools/mixvol/mixvol -s
/tmp/mixvol$timenow.sites';
                   ($result = $result) =~ s/\n/\n<BR>/g;
                   print $result;
            unlink ("/tmp/mixvol$timenow.sites");

                }

CGI qui crée puis traite un fichier en functions des parameters reçus
#!/usr/local/bin/perl5 -w
                use CGI qw(:standard);
                use CGI::Carp;
require "cgi-lib.pl";

&ReadParse(*input);


 print &PrintHeader;
```

```
 print "<body bgcolor='white'>";
 print "<img src='../../saga/demo/mixvol/logo_vm.jpg'
border='0'></img><br>";
 print &HtmlTop ("Here are your data");
print "<a href='../../saga/demo/mixvol/index.html'>Home page</a><BR>";
  ($text = $input{'pb'}) =~ s/\n/<BR>/g;

print "dimension of your problem: ",$input{'dimension'},"<br><hr>";
$timenow = time;
open MYFILE, ">/tmp/mixvol$timenow.sites";
print MYFILE $input{'pb'};
close MYFILE;

if ($input{'dimension'}<7){
}else {
print "Warning:it may take a long time to compute such a big
problem...<br>";
}

print $text,hr;
$ result='/u/www/0/www/tools/mixvol/mixvol -s /tmp/mixvol$timenow.sites';
($result = $result) =~ s/\n/\n<BR>/g;
print $result;
unlink ("/tmp/mixvol$timenow.sites");

print end_html;
```

Volume Stable

Note préliminaire :

Ceci est l'extrait d'un document que Pierre Mario a remis a M. Emiris décrivant de manière détaillée l'algorithme du volume mixte tel qu'il l'aurait écrit. C'est une interprétation personnelle qui présente de fortes similitudes avec le programme existant, mais aussi des différences.

…

2. Données utilisées et remarques.

Sont mentionnées dans cette partie des définitions de types de données utilisés dans mon algorithme. Elle n'ont pour but que de définir un accès aux variables de manière 'stricte'. Mais elles ne tiennent en aucun cas compte de la manière dont elles pourraient être implantées.

Si ces définitions ont de grands airs de ressemblance avec du C, ce n'est évidemment pas une coïncidence.

- Rappelons que n est le nombre d'ensembles que nous avons en entrée.
- Définir un type Point paraît indispensable. C'est un tableau d'entiers à n+1 coordonnés. La n+1-ième servira en fait pour stocker le relèvement fait dans LPA.3.

TypePoint : Entier[1..N+1]

On accèdera à la i-ième coordonnée d'un point Point par

Point[i]

* On a défini le point pour parler en fait d'arêtes. Une arête peut être définie simplement par 2 points.

```
TypeArete : Structure
{
        TypePoint Origine, Fin.
}
```

On accèdera par exemple à la coordonnée de relèvement du premier point d'une arête Arete par :

Arete.Origine[N+1]

* Maintenant nous pouvons définir les ensembles d'arêtes mentionnés dans LPA.1. Ce seront donc des tableaux d'arêtes de taille variable. Nous les définirons sans se préoccuper des problèmes de codage, donc ainsi :

```
TypeEnsemble : Structure
{
        Arete : TypeArete[1..Variable]
        Taille : Entier
}
```

Nous pourrons obtenir la taille d'un ensemble E par :

E.taille,

et nous aurons le droit d'accéder de

E.Arete[1] à E.Arete[E.Taille]. // Sauf si E.Taille est nulle.

* Reste maintenant à créer un type pour grouper les N ensembles de données.

TypeGroupeDEnsemble : Ensemble[1..N].

Accéder a la coordonnée *Coord* de l'origine de l'arête *Arete* de l'ensemble *Ens* du groupe Groupe se fera donc ainsi :

Groupe[Ens][Arete].Origine[Coord].

* Un point moins important : L'opération de lifting de LPA.3 nécessite un vecteur de N+1 coordonnées différent pour chacun des N ensembles. Nous supposerons l'existence d'une matrice pré-initialisée d'entiers aléatoires appelée **Random** dont l'accès aux éléments se fera ainsi :

Random[Ens][Coord].

* Nous définirons aussi une constante de confort pour rajouter un peu de lisibilité :

 # definir COORDLIFT = N+1

3. Fonctions prédéfinies

a. Fonctions non développées

L'algorithme décrit dans la partie suivante utilise des fonctions de manipulation de données qui ne sont pas détaillées, car elles ne représentent qu'un intérêt limité pour le problème qui nous concernent, ou alors dépendent de la manière dont sont implémentées les données.

* **TypeEnsemble DupliquerEnsemble(E : TypeEnsemble)**
Fonction qui retourne une copie de l'ensemble passé en paramètre.

* **TypeArete ExtraireAreteNumero(E : TypeEnsemble, Num : Entier**)
Fonction qui retourne la Num-ième arête de l'ensemble E, et qui l'enlève de l'ensemble.

* **AjouterAreteAEnsemble(E : TypeEnsemble,A : TypeArete)**
Fonction qui rajoute à la fin de l'ensemble E l'arête A.

* **TypeGroupeDEnsemble DupliquerGroupeAPartirDe(G : TypeGroupeDEnsemble, I : Entier)**
Fonction qui renvoie un groupe d'ensembles comme ci-après :
- Les ensembles de 1 à I-1 son vides,
- Les ensembles de I à N sont les copies de ceux du groupe passé en paramètre.

L'utilité de cette fonction est une optimisation qui permet surtout de montrer que l'algorithme récursif ne travaille que sur une partie des ensembles.

* **PermuterEnsembles(D : TypeGroupeDEnsemble, I : Entier, J :Entier)**
Fonction qui permute les ensemble numéro I et J du groupe D. Utilisé pour l'optimisation.

* **Entier Minkowski(E : TypeEnsemble)**
Fonction qui renvoie le Volume de l'ensemble des arêtes contenues dans E. Consiste en une opération sur des vecteurs. Ne devrait pas être modifiée par une implantation du volume stable.

b. Fonction a développer.

* **Booléen SurEnveloppInf(T : TypeEnsemble, A : Arete, D : TypeGroupeDEnsemble, NumEns : entier)**

C'est la fonction qui permet de réaliser le Test mentionné dans LPA.6. Elle regarde si, en Minkowski-ajoutant au tuple T l'arête A, on obtient une face qui se trouve sur l'enveloppe inférieure de la somme de Minkowski d'un sous-ensemble des polytopes relevés du départ. Pour les détails de précision, voir LPA.6.

Ce qui est sûr, c'est qu'elle a donc besoin d'un tuple, d'une arête, des données de départ, et de deux indices pour lui stipuler sur quoi faire les différentes sommes. Elle obtiendra le 'k', niveau de la récursion en regardant la taille du tuple, et l'indice variable 'i' lui sera indiqué par NumEns.

En tout cas, c'est cette fonction qui utilise les bibliothèques de programmation linéaire. Il est probable que la greffe du calcul du volume stable la modifiera.

4. Algorithme LPA orienté programmation.

```
// Definition de 'Donnee', Le groupe qui contiendra tous les ensembles sur lesquels on va travailler.
Donnee : TypeGroupeDEnsemble

VolumeMixteTotal : Entier
//LPA.1, LPA.2
//      Ils ne sont pas développés : il ne devraient pas être concernés par le rajout du volume stable.
//      Par contre nous considérons que Donnees a été rempli par ces étapes.

// LPA.3, relevement des arètes
//      Entraînement un peu barbare a la manipulation des notations.
Pour NumEns=1..N
  Pour NumAr=1..Donnee[NumEns].Taille
          Pour Coord=1..N
          Donnee[NumEns][NumAr].Origine[COORDLIFT]+=
                              Donnee[NumEns][NumAr].Origine[Coord]*Random[NumEns][Coord]
                  Donnee[NumEns][NumAr].Fin[COORDLIFT]+=
                              Donnee[NumEns][NumAr].Fin[Coord]*Random[NumEns][Coord]
```

```
                FinPour         // Coord
                Donnee[NumEns][NumAr].Origine[COORDLIFT]+=Random[NumEns][COORDLIFT]
                Donnee[NumEns][NumAr].Fin[COORDLIFT]+= Random[NumEns][COORDLIFT]
        FinPour                 // NumAr
      FinPour                   // NumEns

// LPA.4, LPA.5, LPA.6, LPA.7, LPA.8, LPA.9.
// Obligation de les regrouper car, à moins d'utiliser un type de programmation dit 'spaghetti', il paraît
// impossible de garder leur ordonnancement rigoureux.
VolumeMixteTotal=CalculerVolumeMixte(Donnees, NULL, 1)


Fonction CalculerVolumeMixte(D : TypeGroupeDEnsemble, Tuple : TypeEnsemble, NiveauRecursion :
Entier)
{
        VMTemp : Entier  // Variable qui stockera les valeurs partielles du Vol. Mixte
        VMTemp=0
// Variables globales utilisées :
//   - Donnee
// Vérités :
//   - Le tuple est exactement composé de [NiveauRecursion -1] arètes.
//   - Le filtrage peut modifier D à partir de [NiveauRecursion], mais pas avant.

        Si NiveauRecursion>N
                Renvoyer Minkowski(Tuple)
        FinSi
// Le Renvoyer, comme en C, fait arrêter le déroulement de la fonction. Donc, Ici, NiveauRecursion
<=N.
        Si D[NiveauRecursion].Taille=0
                renvoyer 0
        FinSi
// NiveauRecursion <=N et D[NiveauRecursion].Taille<>0
        Si NiveauRecursion>1
// Effectue le filtrage indiqué dans LPA.6, à ne pas faire si NiveauRecursion=1.
                Pour NumEns=NiveauRecursion..N
                        NumArete=1
                        TantQue NumArete<=D[NumEns].Taille
                                Si pas SurEnveloppInf(Tuple, D[NumEns][NumArete],
Donnee,NiveauRecursion)
                                        ExtraireAreteNumero(D[NumEns], NumArete)
                                Sinon
                                        NumArete+=1
                                FinSi
                        FinTantQue      // NumArete
                FinPour         // NumEns
        FinSi                           // NiveauRecursion >1

        DTemp : TypeGroupeDEnsemble
        TupleTemp : TypeEnsemble
        AreteTemp : TypeArete
        NumPlusPetitEnsemble : Entier

// Insertion de l'optimisation citée après LPA. Son principe est de prendre l'ensemble avec le moins
d'arêtes
// pour limiter le nombre de tours de boucle qui suivront. Ce qui donne :
        NumPlusPetitEnsemble= NiveauRecursion
        Pour NumEns=NiveauRecursion+1..N
```

```
                Si D[NumEns].Taille<NumPlusPetitEnsemble
                        NumPlusPetitEnsemble=NumEns
                FinSi
        FinPour
        PermuterEnsembles(D, NumPlusPetitEnsemble, NiveauRecursion)

// Boucle qui correspond à LPA.9. Nous sommes obligés de dupliquer les ensembles avant l'appel récursif, car // le programme modifie les ensembles d'entrée sur lesquels il travaille.
        TantQue D[NiveauRecursion].Taille >0
                TupleTemp=DupliquerEnsemble(Tuple)
                AreteTemp = ExtraireAreteNumero(D[NiveauRecursion], 1)
                AjouterAreteAEnsemble(TupleTemp,AreteTemp)
                DTemp=DupliquerGroupeAPartirDe(D, NiveauRecursion +1)
                VMTemp+=CalculerVolumeMixte(DTemp, TupleTemp, NiveauRecursion +1)
        FinTantQue                      // D[NiveauRecursion].Taille >0
        Renvoyer VMTemp
}
```